UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Programa de Pós-Graduação em Engenharia Elétrica

Thales Francisco Mota Carvalho

Técnicas de Inteligência Computacional Aplicadas ao Diagnóstico de Tuberculose

Belo Horizonte

2023

Thales Francisco Mota Carvalho

Técnicas de Inteligência Computacional Aplicadas ao Diagnóstico de Tuberculose

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Prof. Dr. Frederico Gadelha Guimarães
Co-orientador: Prof. Dra. Silvana Spíndola de Miranda
Co-orientador: Prof. Dr. Ricardo de Oliveira Duarte

Belo Horizonte
2023

Carvalho, Thales Francisco Mota.
C331t Técnicas de inteligência computacional aplicadas ao diagnóstico de tuberculose [recurso eletrônico] / Thales Francisco Mota Carvalho . - 2023.
1 recurso online (221 f. : il., color.) : pdf.

Orientador: Frederico Gadelha Guimarães.
Coorientadores: Silvana Spíndola de Miranda, Ricardo de Oliveira Duarte.

Tese (doutorado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Apêndices: f. 183-221.

Bibliografia: f. 174-182.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia elétrica - Teses. 2. Inteligência computacional - Teses. 3. Aprendizado profundo - Teses. 4. Tuberculose - Diagnóstico - Teses. . I. Guimarães, Frederico Gadelha. II. Miranda, Silvana Spíndola de. III. Duarte, Ricardo de Oliveira. IV. Universidade Federal de Minas Gerais. Escola de Engenharia. V. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pelo bibliotecário Reginaldo César Vital dos Santos CRB/6 2165
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FOLHA DE APROVAÇÃO

"TÉCNICAS DE INTELIGÊNCIA COMPUTACIONAL APLICADAS AO DIAGNÓSTICO DE TUBERCULOSE"

THALES FRANCISCO MOTA CARVALHO

Tese de Doutorado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Doutor em Engenharia Elétrica. Aprovada em 14 de dezembro de 2023. Por:

Prof. Dr. Frederico Gadelha Guimarães
DEE (UFMG) - Orientador

Prof. Dr. Ricardo de Oliveira Duarte
DELT (UFMG) - Coorientador

Prof. Dr. Silvana Spíndola de Miranda
Clínica Médica (UFMG) - Coorientadora

Prof. Dr. Karla Tereza Figueiredo Leite
IME (UERJ)

Prof. Dr. Domingos Alves
Medicina Social (USP)

Prof. Dr. Andrea Gomes Campos Bianchi
DECOM (UFOP)

Prof. Dr. Lida Jouca de Assis Figueredo
(FUNED)

Documento assinado eletronicamente por **Frederico Gadelha Guimaraes**, **Professor do Magistério Superior**, em 17/12/2023, às 20:36, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Silvana Spindola de Miranda**, **Professora do Magistério Superior**, em 18/12/2023, às 11:02, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Ricardo de Oliveira Duarte**, **Professor do Magistério Superior**, em 18/12/2023, às 11:12, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Domingos Alves**, **Usuário Externo**, em 19/12/2023, às 10:34, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Lida Jouca de Assis Figueredo**, **Usuário Externo**, em 19/12/2023, às 12:18, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Karla Tereza Figueiredo Leite**, **Usuária Externa**, em 08/01/2024, às 12:12, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Andrea Gomes Campos Bianchi**, **Usuária Externa**, em 08/01/2024, às 20:41, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2897905** e o código CRC **4A2EA6D5**.

Referência: Processo nº 23072.277055/2023-62 SEI nº 2897905

Com profunda afeição e gratidão, dedico este trabalho a Thácio Gabriel, meu filho, que ilumina meus dias, inspira minha felicidade e é a mais bela dádiva que já recebi.

AGRADECIMENTOS

Agradeço a Deus por me amparar nos momentos difíceis, me dar força interior para superar as dificuldades e por sempre estar comigo.

À Vívian, minha alma gêmea, minha esposa. Sua ajuda e apoio foram para mim de valor inestimável, e qualquer palavra que eu utilize seria insuficiente para agradecer-lhe com o devido merecimento. Tenho certeza que sem você eu não teria chegado até aqui. Entre as infinitas coisas que tenho que agradecer-lhe, muito obrigado pela amizade, amor, companheirismo, dedicação, lealdade, carinho, e pelo maior presente de todos, o nosso pequeno Thácinho. Ao meu filho, Thácio Gabriel, que foi uma fonte constante de alegria e inspiração. Sua presença inocente e feliz é um lembrete diário da beleza e simplicidade da vida. Cada pequeno gesto seu, cada sorriso, cada nova descoberta e cada momento, enriqueceram minha existência, alegrando, reenergizando e fortalecendo minha vida.

À minha mãe Dulce, por sempre estar ao meu lado com o seu amor eterno e incondicional. Obrigado por ter me educado, pelos conselhos, incentivos, afeto, amizade e por ter se dedicado e contribuído para meu crescimento. Ao meu irmão Tácio (*in memorian*), presente em meu coração, por me dá coragem para não desistir.

À minha família, a qual amo muito, pelo carinho e incentivo. Aos meus primos por todos os momentos que passamos juntos de pura amizade. À família da minha esposa, por ter me recebido de coração aberto e sempre torcerem por mim. Em especial, agradeço à Denise, Herculano e Bernardo, por nos receber e abrigar em seu lar em Belo Horizonte.

Ao Prof. Dr. Frederico Gadelha Guimarães, o qual expresso minha mais sincera gratidão e profundo respeito, que não apenas me orientou com maestria nesta jornada acadêmica, mas também se tornou um modelo inspirador de profissionalismo e dedicação. A paciência, confiança e sabedoria com as quais ele conduziu meu desenvolvimento foram fundamentais. Sua capacidade de oferecer orientação incisiva e enriquecida por sugestões perspicazes, foi essencial para a realização deste trabalho.

Aos meus coorientadores, Prof.ª Dr.ª Silvana Spíndola e Prof. Dr. Ricardo de Oliveira Duarte, pela atenção, valiosas sugestões, esforços, aperfeiçoamento do trabalho e por abrirem as portas dos seus laboratórios. Agradeço também a toda equipe do laboratório MINDS e do Grupo de Pesquisa em Micobacterioses da UFMG, em especial a Lida.

Aos membros da banca, Dr. Domingos Alves, Dr.ª Andrea G. C. Bianchi, Dr.ª Karla T. F. Leite e Dr.ª Lida J. A. Figueredo, por suas importantes contribuições e sugestões. Ao Dr. João V. Gomide, por compartilhar seu conhecimento e dados.

Aos meus amigos, especialmente Cleysinho e Vívian, Jhonny e Elizângela, pelo companheirismo e por estarem sempre disponíveis. Obrigado aos meus amigos da UFVJM, de forma especial, à Dicom e ao IECT - Janaúba por todo apoio e ajuda nesta trajetória.

“A persistência é o menor caminho do êxito”
Charles Chaplin

RESUMO

A tuberculose (TB) permanece como uma das doenças infecciosas mais mortais globalmente, necessitando de métodos de diagnóstico rápidos e precisos. A baciloscopia, uma técnica microscópica para identificar bacilos da TB, é prevalente devido à sua simplicidade e custo acessível, entretanto é uma abordagem cansativa, totalmente manual e necessita de treinamento intensivo do examinador. Neste sentido, este trabalho investiga a aplicação de técnicas de Aprendizado Profundo (DL, do inglês *Deep Learning*) para auxiliar o examinador e aprimorar a acurácia da baciloscopia. Para isso, foi conduzido uma avaliação extensiva de vários modelos de redes neurais convolucionais e métodos de detecção de objetos em diversos conjuntos de dados, visando compreender sua performance e aplicabilidade na detecção e contagem dos bacilos da TB. Em complemento, visando aprimorar a performance dos modelos, são propostas e avaliadas várias estratégias, incluindo criação de um novo conjunto de dados público, técnicas de aumento de dados, estratégias de particionamento de imagens e a implementação de um sistema de votação para facilitar a anotação de imagens para treinamento. O aumento de dados provou ser essencial para aumentar a robustez dos modelos, enquanto o particionamento de imagens melhorou o desempenho e possibilitou o tratamento de variações no tamanho das imagens, mantendo parâmetros de modelos padronizados. O sistema de votação proposto tem o potencial de ser tornar uma ferramenta valiosa para a criação de conjuntos de treinamento personalizados, vital para a generalização dos modelos em diferentes contextos. Também foi desenvolvido um protótipo de software de Diagnóstico Auxiliado por Computador, integrando as técnicas e modelos avaliados, visando auxiliar o examinador durante a contagem dos bacilos no diagnóstico da TB. Este protótipo representa um avanço significativo para a aplicação prática de técnicas de DL na baciloscopia, com potencial para acelerar e tornar mais preciso o processo de diagnóstico. Os experimentos realizados forneceram revelações sobre as possibilidades e limitações do uso de DL na detecção de bacilos da TB, sendo avaliado também o potencial de um diagnostico autônomo. Esta pesquisa contribui substancialmente para o campo, oferecendo uma análise detalhada do uso de DL no diagnóstico da TB via baciloscopia, e métodos que possam ser testados em ambientes reais. Em síntese, esta tese fornece um panorama aprofundado sobre a implementação de DL na detecção de bacilos da TB, ressaltando suas potencialidades, desafios e estabelecendo uma base sólida para pesquisas futuras e aplicações práticas.

Palavras-chave: Tuberculose, Diagnóstico, Segmentação de Imagem, Aprendizado Profundo.

ABSTRACT

Tuberculosis (TB) remains one of the deadliest infectious diseases globally, necessitating rapid and accurate diagnostic methods. Sputum smear microscopy, a microscopic technique to identify TB bacilli, is prevalent due to its simplicity and affordable cost. However, it is a laborious, entirely manual approach that requires intensive training of the examiner. In this context, this work investigates the application of Deep Learning (DL) techniques to assist the examiner and enhance the accuracy of sputum smear microscopy. For this, an extensive evaluation of various convolutional neural network models and object detection methods was conducted on diverse datasets, aiming to understand their performance and applicability in detecting and counting TB bacilli. In addition, aiming to improve the models performance, several strategies are proposed and evaluated, including creating a new public dataset, data augmentation techniques, image partitioning strategies, and implementing a voting system to facilitate the annotation of images for training. Data augmentation proved to be essential for increasing the models robustness, while image partitioning improved performance and allowed for handling variations in image size, maintaining standardized model parameters. The proposed voting system has the potential to become a valuable tool for creating customized training sets, vital for the generalization of models in different contexts. A Computer-Aided Diagnosis software prototype was also developed, integrating the assessed techniques and models, aiming to assist the examiner during the counting of bacilli in TB diagnosis. This prototype represents a significant advancement for the practical application of DL techniques in sputum smear microscopy, with the potential to accelerate and make the diagnostic process more precise. The experiments conducted provided insights into the possibilities and limitations of using DL in TB bacilli detection, also evaluating the potential for autonomous diagnosis. This research substantially contributes to the field, offering a detailed analysis of the use of DL in TB diagnosis via sputum smear microscopy, and methods that can be tested in real environments. In summary, this thesis provides an in-depth overview of the implementation of DL in TB bacilli detection, highlighting its potentialities, challenges, and establishing a solid foundation for future research and practical applications.

Keywords: Tuberculosis, Diagnosis, Image Segmentation, Deep Learning.

LISTA DE FIGURAS

Figura 1 – Coeficiente de incidência de TB no Brasil, de 2012 a 2022, por 100 mil habitantes. Fonte: Brasil (2023). 24
Figura 2 – (a) Coeficiente de incidência de TB nos estados do Brasil, em 2022, por 100 mil habitantes. (b) Coeficiente de mortalidade por TB nos estados do Brasil, em 2021 (dados extraídos em novembro/2022), por 100 mil habitantes. Adaptado de Brasil (2023). 25
Figura 3 – Exemplo bacilo tuberculose em forma de bastonetes. Fonte: Pinto (2018). 33
Figura 4 – Amostra do escarro de um paciente com tuberculose descolorado pelo método de *Ziehl-Neelsen*. Fonte: Campelo et al. (2001) 34
Figura 5 – Amostra da baciloscopia pelo método de *Ziehl-Neelsen* em um paciente com tuberculose ampliado em um microscópio. Fonte: Koneman et al. (2012) 34
Figura 6 – Representação do neurônio biológico. Fonte: Academy (2021). 36
Figura 7 – Representação do neurônio artificial. Fonte: Awad e Khanna (2015). . . 37
Figura 8 – ANN com três camadas. Fonte: Awad e Khanna (2015). 39
Figura 9 – Arquitetura da CNN *LeNet-5*. Fonte: LeCun et al. (1998). 41
Figura 10 – Exemplo de *data augmentation*, onde é aplicado operações sobre (a) uma imagem original, tais como: (b) reflexão, (c) rotação, (d) translação, (e) injeção de ruído, (f) alteração de escala, e (g) alteração de cor. . . 47
Figura 11 – A estrutura PRISMA para o processo de seleção dos estudos. 51
Figura 12 – Fluxo de trabalho básico de ML aplicado à classificação e localização do bacilo da TB usando DL e CV. Figura elaborada utilizando imagens contidas no conjunto de dados de Shah et al. (2017) 55
Figura 13 – (a) Exemplo de classificação de imagem onde cada objeto na imagem é classificado por um modelo de classificação em “TB Bacillus”(contém um bacilo da TB) ou “No TB Bacillus”(não contém um bacilo da TB). (b) Exemplo de um método de detecção de objetos localizando e classificando objetos pertencentes à classe “TB Bacillus”. Figura feita a partir de imagens do conjunto de dados de Shah et al. (2017). 56
Figura 14 – Exemplo de anotação de bacilos nos conjuntos de dados propostos por (a) Costa et al. (2014), (b) Quinn et al. (2016), (c) Shah et al. (2017), e (d) Trilaksana et al. (2020) (as imagens foram extraídas do conjunto de dados original ou artigos e redimensionadas para melhor visualização). 57

Figura 15 – Ilustração de como os dados podem ser separados para realizar o processo de treinamento e teste para avaliação de modelos gerados por métodos de ML. (a) Dividindo os dados em treinamento, validação e teste usando subconjuntos de dados independentes. (b) Validação cruzada *k-fold*, divisão dos dados em k subgrupos contendo um fold para teste e $k-1$ *folds* para treinamento. 61
Figura 16 – Ilustração da matriz de confusão para problemas de classificação de duas classes (positiva ou negativa). 61
Figura 17 – (a) Representação gráfica do cálculo da Interseção sobre União (IoU). (b) Exemplo do resultado do cálculo de IoU considerando B_{gt} (caixa colorida de verde) e B_p (caixa colorida de vermelho) (preparado utilizando uma imagem do conjunto de dados de Shah et al. (2017)). 64
Figura 18 – Etapas definidas para construção do conjunto de dados. 84
Figura 19 – Exemplos de fragmentos de uma lâmina da baciloscopia, em que são destacados em vermelho: (a) bacilos da tuberculose, (b) concentração de reagente, (c) artefatos. 86
Figura 20 – Suporte para capturar fotos do microscópio usando smartphone: (a) ilustração do suporte, (b) suporte sendo utilizado e (c) imagem capturada. 87
Figura 21 – Exemplo de imagens do conjunto de dados após o recorte realizado nas imagens com baica (a) e alta (b) densidade de bacilos da TB. 88
Figura 22 – Funcionalidade do Find TB Bacillus que permite ao especialista identificar e marcar a posição do bacilo em uma imagem. 89
Figura 23 – Processo da escolha dos bacilos (marcações) usados para compor o conjunto de dados, onde: (a) tem-se a foto original com o recorte a ser feito, (b) identificação dos bacilos pelo usuário 1 em vermelho, (c) identificação dos bacilos pelo usuário 2 em verde, (d) identificação dos bacilos pelo usuário 3 em azul, (e) combinação de todas as identificações, (f) ajuste das marcações para compor o conjunto de dados. 91
Figura 24 – (a) Arquivo CSV o qual encapsula as anotações referentes às imagens do banco de dados; (b) Representação gráfica de um plano cartesiano ilustrando as coordenadas armazenadas; e (c) Visualização da imagem *S4F10.jpg*, destacando as marcações contidas no arquivo CSV. 93
Figura 25 – Apresentação de uma imagem de cada uma das lâminas positivas que compõem o conjunto de dados. 94
Figura 26 – Geração de fragmentos que não possuem bacilos (classe negativa, caixa vermelha) a partir da posição/dimensão de um fragmento que contém bacilos (classe positiva, caixa verde). Figura feita a partir de imagens do conjunto de dados de Shah et al. (2017). 97

Figura 27 – Ilustração do processo de *data augmentation*, onde: (a) representa as posições dos bacilos (classe positiva, box verde) e as posições sem bacilos (classe negativa, box vermelho) em uma imagem do dataset-4; (b) 8 fragmentos de imagens extraídos da imagem original; (c) criação de 7 novas imagens a partir de cada fragmento extraído (aplicando operações de *data augmentation*), totalizando, assim, 64 imagens geradas para compor o conjunto de treinamento/teste. 98
Figura 28 – Ilustração das operações de *data augmentation* aplicada em uma imagem para gerar um conjunto de 8 imagens, sendo que (a) contém a imagem a original e (b) contém a imagem original e 7 novas modificadas. . . . 100
Figura 29 – Arquitetura construída usando TensorFlow, baseada no trabalho de Quinn et al. (2016). Imagem gerada pela função *tensorflow.keras.utils.plot_model* do TensorFlow. 102
Figura 30 – Arquitetura construída usando TensorFlow, baseada no trabalho de Panicker et al. (2018). Imagem gerada pela função *tensorflow.keras.utils.plot_model* do TensorFlow. 103
Figura 31 – Arquitetura construída usando TensorFlow, baseada no trabalho de Udegova, Shelomentseva e Chentsov (2021). Imagem gerada pela função *tensorflow.keras.utils.plot_model* do TensorFlow. 104
Figura 32 – Arquitetura construída usando TensorFlow, baseada no trabalho de Simon et al. (2019). Imagem gerada pela função *tensorflow.keras.utils.plot_model* do TensorFlow. 105
Figura 33 – Método proposto para detecção do bacilo da TB utilizando o K-*Means* para segmentação da imagem e CNN para classificação. 108
Figura 34 – Ilustração do espaço de cores HSV, onde: (a) representação do espaço HSV, e (b) intervalo de cores utilizado na filtragem realizada em Figura 33c. 109
Figura 35 – Representação gráfica que destaca algumas das imagens resultantes em cada fase do método proposto. 109
Figura 36 – Esquema ilustrativo da estratégia empregada para expandir um conjunto de dados para se tornar mais representativo, composto por quatro etapas, visando aprimorar a generalização dos modelos gerados pela CNN. . . . 111
Figura 37 – Esquema ilustrativo da estratégia empregada para treinar e avaliar os métodos de detecção. 114
Figura 38 – Ilustração do processo de particionamento das imagens adotado neste trabalho. 116
Figura 39 – Esquema ilustrativo da estratégia empregada para treinar e avaliar os métodos de detecção a partir da estrategia de particionamento das imagens proposto. 117

Figura 40 – Esquema ilustrativo da estratégia de realizar a detecção do bacilo da TB utilizando um sistema de consenso entre diferentes métodos de DL e uma curadoria feita por uma especialista. 121
Figura 41 – Módulos proposto para criação do software de auxílio ao diagnostico da TB. 124
Figura 42 – Esquema ilustrativo do fluxo de trabalho proposto para realizar o diagnóstico automático da TB através do uso de métodos de DL. . . . 130
Figura 43 – Matriz de confusão gerada a partir do diagnóstico realizado pelo processo de diagnóstico proposto. 151
Figura 44 – Histograma com a distribuição de bacilos identificados pelo método de DL em um determinada lâmina durante o Experimento 8.2. 152
Figura 45 – Teste realizado durante integração do conjunto, protótipo / smartphone / microscópio. 153
Figura 46 – Diagrama de caso de uso com as funcionalidades do protótipo construído. 158
Figura 47 – Protótipo do sistema CADx desenvolvido: Tela Inicial. 159
Figura 48 – Protótipo do sistema CADx desenvolvido: Menu de acesso às funcionalidades de carregar uma imagem ou uma fonte de imagem em tempo real. 160
Figura 49 – Protótipo do sistema CADx desenvolvido: Visualização da imagem obtida em um determinada fonte. 161
Figura 50 – Protótipo do sistema CADx desenvolvido: Opções de ajuste de cores da funcionalidade de visualização da imagem. 162
Figura 51 – Protótipo do sistema CADx desenvolvido: Visualização da imagem após análise do método de DL, (a) realçando os bacilos da TB identificados com um box verde, e (b) permitindo que o especialista altere a quantidade de bacilos contados. 163
Figura 52 – Protótipo do sistema CADx desenvolvido: Interface de edição das marcações realizadas na imagem. 164
Figura 53 – Protótipo do sistema CADx desenvolvido: Funcionalidade de adicionar imagens que representam um mesmo campo de visualização (subcampos). 165
Figura 54 – Protótipo do sistema CADx desenvolvido: Interface que permite acessar informações que são geradas durante análise das imagens. 166
Figura 55 – Protótipo do sistema CADx desenvolvido: Funcionalidade para o especialista definir o diagnóstico da lâmina que está sendo analisada. 167
Figura 56 – Protótipo do sistema CADx desenvolvido: Menu com opções de (a) salvar e (b) abrir um diagnóstico realizado. 168

Figura 57 – Protótipo do sistema CADx desenvolvido: (a) simulação da visualização de um campo de visão obtido pelo smartphone e enviado para o sistema, e (b) opção de zoom sento utilizado na visualização da imagem com os bacilos da TB detectados. Imagem obtida do conjunto de dados do Capítulo 4, sem recorte. 169
Figura 58 – Diagrama de caso de uso do Find TB Bacillus. 184
Figura 59 – Protótipo da interface para criar tarefa. 186
Figura 60 – Protótipo da interface para visualizar as tarefas criadas. 186
Figura 61 – Protótipo da interface para gerenciar a tarefa. 187
Figura 62 – Protótipo da interface para editar uma imagem e marcar os bacilos. . . 187
Figura 63 – Protótipo da interface para comparar as marcações dos bacilos. 188
Figura 64 – Protótipo da interface para o aluno realizar a tarefa. 188
Figura 65 – Protótipo da interface para realizar a marcação de bacilos em uma imagem. 189

LISTA DE TABELAS

Tabela 1 – Critérios para leitura e interpretação dos resultados da baciloscopia de escarro (BAAR: Bacilo Álcool-Ácido Resistente) (MS, 2022) 34
Tabela 2 – Base de dados de artigos que foram utilizados. 51
Tabela 3 – Artigos sobre técnicas de classificação na literatura. 73
Tabela 4 – Artigos sobre técnicas de segmentação e classificação na literatura. . . 75
Tabela 5 – Relação de lâminas selecionadas para capturar as imagens. 86
Tabela 6 – Relação entre lâminas e quantidade de imagens que cada especialista foi responsável por realizar a identificação dos bacilos. 91
Tabela 7 – Quantidades média (considerando os *5-fold* e o *data augmentation*) de fragmentos positivos (com bacilos) e negativos (sem bacilos) em cada conjunto de dados. 98
Tabela 8 – Quantidades de imagens em cada um dos *5-fold* de cada conjunto de dados (considerando o *data augmentation*). 100
Tabela 9 – Quantidades de imagens em cada conjunto de dados ao ser separado em treinamento, validação e teste (considerando o *data augmentation*). 100
Tabela 10 – Resoluções máxima, miníma e média (em pixels) das imagens encontradas nos quatro conjuntos de dados (valores representados em Largura x Altura). 114
Tabela 11 – Quantidades de imagens em cada um dos 5-*fold* de cada conjunto de dados após o particionamento das imagens (considerando o *data augmentation*). 117
Tabela 12 – Quantidades de imagens em cada conjunto de dados separados em treinamento, validação e teste, após o particionamento das imagens (considerando o *data augmentation*). 119
Tabela 13 – Quantidade de imagens separadas para treinamento, validação e teste após processamento do conjunto de dados. 127
Tabela 14 – Resultados obtidos usando o dataset-1 com as arquiteturas CNNs replicadas. 135
Tabela 15 – Resultados obtidos usando o dataset-2 com as arquiteturas CNNs replicadas. 135
Tabela 16 – Resultados obtidos usando o dataset-3 com as arquiteturas CNNs replicadas. 135
Tabela 17 – Resultados obtidos usando o dataset-4 com as arquiteturas CNNs replicadas. 135
Tabela 18 – Resultados obtidos em cada conjunto de dados ao utilizar o K-*means* + CNN-D. 137

Tabela 19 – Quantidades média (considerando os *5-fold*) de fragmentos positivos e negativos ampliados em cada conjunto de dados. 138
Tabela 20 – Resultados obtidos em cada conjunto de dados ao utilizar o K-means + CNN-D*. 139
Tabela 21 – Resultados obtidos em cada conjunto de dados com arquitetura CNN-D utilizando quantidade ampliada de fragmentos negativos (CNN-D*). . . 140
Tabela 22 – Resultados obtidos usando o dataset-1 com os métodos de detecção utilizados. 142
Tabela 23 – Resultados obtidos usando o dataset-2 com os métodos de detecção utilizados. 142
Tabela 24 – Resultados obtidos usando o dataset-3 com os métodos de detecção utilizados. 142
Tabela 25 – Resultados obtidos usando o dataset-4 com os métodos de detecção utilizados. 143
Tabela 26 – Resultados obtidos ao realizar o Experimento 5 (Exp. 5) e resultados sumarizados do Experimento 4 (Exp. 4) utilizando a Faster R-CNN. . 144
Tabela 27 – Resultados obtidos em cada conjunto de teste ao utilizando o método Faster R-CNN com partição das imagens. 146
Tabela 28 – Média obtida por cada método treinado ao processar os quatro conjuntos de testes, utilizando o método Faster com partição das imagens. 146
Tabela 29 – Resultados obtidos em cada conjunto de teste ao utilizando o método Faster R-CNN com partição das imagens em um sistema de votação. . 148
Tabela 30 – Média obtida ao processar os quatro conjuntos de testes, utilizando o método Faster com partição das imagens em um sistema de votação. . 148

LISTA DE ABREVIATURAS E SIGLAS

ANN	*Artificial Neural Network*
AUC	*Area Under the ROC Curve*
AVA	Ambiente Virtual de Aprendizagem
BAAR	Bacilo Álcool-ácido Resistente
BCG	Bacilo Calmette-Guérin
CADx	*Computer-Aided Diagnosis*
CNNs	Redes Neurais Convolucionais
CV	*Computer Vision*
DL	*Deep Learning*
Faster R-CNN	*Faster Region-Based Convolutional Network*
FSAF	*Feature Selective Anchor-Free*
HSV	*Hue-Saturation-Value*
IA	Inteligência Artificial
IoU	*Intersection over Union*
kNN	*K-Nearest Neighbors*
mAP	*Mean Average Precision*
ML	*Machine Learning*
Mtb	*Mycobacterium tuberculosis*
OMS	Organização Mundial da Saúde
PRISMA	*Reporting Items for Systematic Reviews and Meta-Analyses*
R-CNN	*Region based Convolutional Neural Networks*
RGB	*Red-Green-Blue*
ROIs	*Region of Interest*

RPNs	*Region Proposal Networks*
SGD	*Stochastic Gradient Descent*
SSD	*Single Shot MultiBox Detector*
SUS	Sistema Único de Saúde
SVM	*Support Vector Machines*
TB	Tuberculose
TRM-TB	Teste Rápido Molecular para Tuberculose
WHO	*World Health Organization*
YOLO	*You Only Look Once*
YOLOF	*You Only Look One-level Feature*
ZN	*Ziehl-Nielsen*

SUMÁRIO

1 Introdução 22
1.1 Objetivos 27
1.1.1 Objetivos Específicos 27
1.2 Contribuições 27
1.3 Estrutura do Trabalho 29
2 Referencial Teórico 30
2.1 Tuberculose 30
2.2 Diagnóstico da Tuberculose 31
2.2.1 Baciloscopia 32
2.3 Aprendizado de Máquina 35
2.3.1 Rede Neural Artificial 36
2.3.2 Aprendizado profundo 38
2.3.2.1 Redes Neurais Convolucionais 39
2.4 K-*Means Clustering* 41
2.5 Detecção de objetos 42
2.5.1 Faster R-CNN 42
2.5.2 YOLO 43
2.5.3 SSD 44
2.5.4 RetinaNet 45
2.5.5 FSAF 45
2.5.6 EfficientDet 46
2.6 *Data Augmentation* 46
3 Revisão Sistemática 49
3.1 Métodos 50
3.1.1 Estratégia de pesquisa 50
3.1.2 Critério de seleção 52
3.1.3 Estudos selecionados 52
3.2 Revisão da identificação do bacilo da Tuberculose por Aprendizado Profundo 53
3.2.1 Processo de classificação de imagem 53
3.2.1.1 Classificação de Imagem e Detecção de Objetos 55
3.2.2 Conjuntos de Dados Públicos 56
3.2.3 Estratégia e métricas para avaliação dos modelos 59
3.2.4 Técnicas de Deep Learning 64
3.2.4.1 Técnicas de classificação 64
3.2.4.2 Técnicas de Segmentação/Classificação 67

3.3 Discussão e recomendações sobre a revisão sistemática realizada 79
3.4 Conclusões sobre a revisão sistemática 82
4 Construção de um Novo Conjunto de Dados 84
4.1 Escolha das amostras 85
4.2 Captura das imagens 86
4.2.1 Controle e recorte dos tamanhos das imagens 87
4.3 Sistema para identificação manual do bacilo 88
4.4 Identificação dos bacilos 90
4.5 Tratamento dos dados obtidos 92
4.6 Disponibilização do conjunto de dados 92
5 Materiais e Métodos 95
5.1 Conjuntos de dados 96
5.1.1 Dados para treinamento/teste da CNN 97
5.1.2 Dados para treinamento/teste dos métodos de detecção 98
5.2 Replicação de arquiteturas CNNs propostas na literatura 100
5.2.1 Trabalhos selecionados 101
5.2.1.1 Arquiteturas 101
5.2.2 Implementação e avaliação 105
5.3 Detecção do bacilo utilizando K-*Means* e CNN 106
5.3.1 Processo de segmentação e classificação 106
5.3.2 Aperfeiçoamento dos dados de treinamento 110
5.3.3 Implementação e avaliação 111
5.4 Algoritmos de detecção de objetos 112
5.4.1 Estratégia de partição de imagens 114
5.4.2 Implementação e avaliação 117
5.5 Explorando treinamento e teste em diferentes conjuntos de dados 118
5.6 Anotação semi-automática 120
5.7 Identificação assistida do bacilo da tuberculose para diagnóstico 122
5.8 Explorando métodos de DL para diagnóstico automático 124
5.8.1 Dados disponíveis 125
5.8.2 Treinamento 126
5.8.3 Proposta de um processo de diagnóstico 127
5.8.4 Avaliação 129
6 Resultados e discussão 131
6.1 Experimento 1: Avaliação de modelos de CNN em múltiplos datasets . . . 132
6.2 Experimento 2: Detecção de bacilos de TB com K-*means* e CNN 136
6.3 Experimento 3: Aperfeiçoamento da CNN utilizando padrões negativos ampliados 137

6.4 Experimento 4: Comparação de métodos de detecção de objetos na identificação de bacilos de TB . . . 141
6.5 Experimento 5: Aplicação e avaliação da estratégia de partição de imagens com Faster R-CNN . . . 143
6.6 Experimento 6: Avaliação Transversal dos Modelos em Diferentes Conjuntos de Dados . . . 144
6.7 Experimento 7: Sistema de Votação para Combinação de Modelos na Detecção de Bacilos . . . 147
6.8 Experimento 8: Diagnóstico de Lâminas de Baciloscopia . . . 149
6.9 Software de auxílio ao diagnóstico da TB . . . 152
7 Conclusões e Trabalhos Futuros . . . 170
7.1 Considerações Finais . . . 170
7.2 Trabalhos Futuros . . . 172
REFERÊNCIAS . . . 174
Apêndice A Find TB Bacillus, um Ambiente Virtual de Aprendizagem 183
A.1 Funcionalidades do Find TB Bacillus . . . 184
A.2 Desenvolvimento . . . 186
Apêndice B Códigos-Fontes . . . 190
B.1 Treinamento e teste das CNNs implementadas . . . 190
B.2 Teste K-means + CNN . . . 199
B.3 Treinamento e teste com diferentes métodos de detecção . . . 210

1 Introdução

A tuberculose (TB) é uma infecção causada pelo bacilo *Mycobacterium tuberculosis* (Mtb). A forma mais comum da doença é a tuberculose pulmonar, que afeta os pulmões, mas o Mtb também pode infectar outros tecidos do corpo, causando a tuberculose extrapulmonar. Morfologicamente, o Mtb é classificado como um bacilo, caracterizando-se por sua forma alongada semelhante a bastonetes. A transmissão ocorre principalmente por via aérea, quando partículas infectantes são liberadas no ambiente através da tosse, fala ou espirro de indivíduos portadores de TB pulmonar ou laríngea ativa. Essas partículas, quando inaladas por pessoas saudáveis, podem levar à infecção e ao desenvolvimento da doença (CAMPELO et al., 2001).

De acordo com o Relatório Global da Tuberculose 2022, elaborado pela Organização Mundial da Saúde (OMS), em 2019, antes do advento da pandemia do COVID-19, a TB era a doença infecciosa mais mortal do mundo, matando mais de um milhão de pessoas a cada ano e afetando outros milhões (WHO, 2022). A TB é uma das dez principais causas de morte em todo o mundo e era a principal causa de morte por um único agente infeccioso (classificação acima de HIV/AIDS[1]). Estima-se que cerca de um quarto da população mundial esteja infectada com o bacilo Mtb, e o risco dessas pessoas desenvolverem TB é de 5-10% ao longo da vida. Segundo os dados divulgados da epidemia de TB, globalmente, estima-se que cerca de 10 milhões de pessoas adoeceram com TB em 2019, 5.8 milhões em 2020, e em 2021 estima-se que foram 6.4 milhões de doentes e 1.4 milhão de mortes por TB (WHO, 2022).

Por outro lado, é importante destacar que, globalmente, o número anual de pessoas relatadas como tendo acesso ao tratamento da TB cresceu, sendo que número acumulado de pessoas tratadas entre 2018 e 2021 foi de 26,3 milhões. O acesso ao tratamento preventivo da TB tem apresentado aumentos, de 1 milhão em 2015, para 2.2 milhões em 2018, 3.6 milhões em 2019 e uma redução para 3.5 milhões em 2021. Estima-se que desde 2000, o tratamento da TB evitou mais de 60 milhões de mortes. Entende-se que a TB é curável, cerca de 85% das pessoas que desenvolvem TB podem ser tratadas com sucesso com o uso de medicamentos por 6 meses (WHO, 2020a; WHO, 2022).

No Relatório Global da Tuberculose 2022, entre 2012 e 2019, observou-se um aumento global no número de pessoas recém-diagnosticadas com TB. Contudo, essa tendência reverteu-se a partir de 2020, período marcado por uma acentuada redução desses diagnósticos durante a pandemia de COVID-19. Apesar de uma recuperação parcial

[1] HIV é a sigla do termo inglês *Human Immunodeficiency Virus*, traduzido para português como vírus da imunodeficiência humana. AIDS é sigla do termo inglês *Acquired Immunodeficiency Dyndrome*, traduzido para português como síndrome da imunodeficiência adquirida.

em 2021, as taxas de novos diagnósticos de TB ainda permanecem inferiores aos níveis registrados em 2019. Apesar do aumento nas notificações de TB, observa-se uma grande lacuna (3.4 milhões) entre o número de pessoas recém-diagnosticadas e notificadas (6,4 milhões) e os 9.8 milhões de pessoas que se estima terem desenvolvido TB em 2021. Essa lacuna se deve a uma combinação de subnotificação de pessoas diagnosticadas com TB e subdiagnóstico, que ocorre se as pessoas com TB não têm acesso a cuidados de saúde ou não são diagnosticadas corretamente. Em 2021, um conjunto de 10 países somaram 75% da discrepância global entre a incidência prevista de TB e o número reportado de novos diagnósticos da doença. A Índia, Indonésia, Filipinas, Paquistão e Nigéria lideraram como os principais contribuintes, representando, respectivamente, 24%, 13%, 10%, 6,6% e 6,3% dessa lacuna (WHO, 2022).

Neste mesmo relatório, a OMS destaca os impactos da pandemia COVID-19 sobre o progresso recente na redução da epidemia de TB, mostrando que a pandemia COVID-19 ainda continua a ter um impacto prejudicial no acesso ao diagnóstico e tratamento da TB. Houve uma significativa redução global no número de diagnósticos de TB, caindo de 7.1 milhões em 2019 para 5.8 milhões em 2020, um nível não visto desde 2012. Índia, Indonésia e Filipinas foram responsáveis pela maior parte dessa redução em 2020, com uma recuperação parcial em 2021. Outros países com alta carga de TB também tiveram reduções consideráveis (WHO, 2022). Durante a pandemia, impactos negativos nos serviços essenciais de TB foram identificados, incluindo a realocação de recursos humanos e financeiros da TB para o combate à COVID-19, tais como, redução do número de unidades de saúde que oferecem atendimento ambulatorial e internação para pessoas com TB, diminuição na realização de testes diagnósticos, na notificação de casos de TB ativa, no monitoramento do tratamento, na coleta de dados e elaboração dos relatórios (WHO, 2020a).

Em razão do acima exposto, considera-se que é necessário melhorar o acesso ao diagnóstico de TB e o tratamento da TB para eliminar as lacunas entre a incidência e as notificações, buscando, assim, garantir que as pessoas sejam diagnosticadas corretamente e iniciem o regime de tratamento mais eficaz o mais cedo possível. Portanto, é fundamental, melhorar a detecção de TB, aumentar a confirmação bacteriológica e ampliar o uso de diagnósticos recomendados pela OMS (WHO, 2022).

Segundo o Manual de Recomendações para o Controle da Tuberculose no Brasil do Ministério da Saúde (BRASIL, 2019), a região das Américas representou cerca de 3,0% da carga mundial de TB em 2017, com 268 mil casos novos estimados, os quais estão localizados em nações como Brasil (33,0%), Peru (14,0%), México (9,0%) e Haiti (8,0%), sendo estes os países com a maior carga. Vale ressaltar que o Brasil faz parte da lista de 30 países da OMS com alta carga de TB e TB-HIV. Em contrapartida, destes 30 países com alta carga de TB, o Brasil está incluído na lista com altos níveis de tratamento em 2021, juntamente com China, Federação Russa e Tailândia (WHO, 2022).

Na Figura 1 é descrito o coeficiente de incidência de TB no Brasil nos anos de 2012 a 2022, por 100 mil habitantes. Estes dados foram divulgados pelo Boletim Epidemiológico de Tuberculose 2023 do Ministério da Saúde, o qual é definido como dados preliminares sujeitos a alteração (BRASIL, 2023). Percebe-se na Figura 1 que, entre os anos de 2012 e 2015, houve queda no coeficiente de incidência de TB no Brasil. No entanto, a partir de 2016 observou um aumento neste coeficiente, apenas até o ano de 2019. Pois, em 2020, constatou-se uma queda acentuada em relação a todos os anos anteriores. Fato que pode ser relacionado com a pandemia COVID-19, conforme estimativa da OMS (BRASIL, 2023). De acordo com WHO (2020a), na região das Américas, a incidência de TB está aumentando lentamente, devido a uma tendência de aumento no Brasil durante os anos de 2016 a 2019, o que pode ser constatado pela Figura 1.

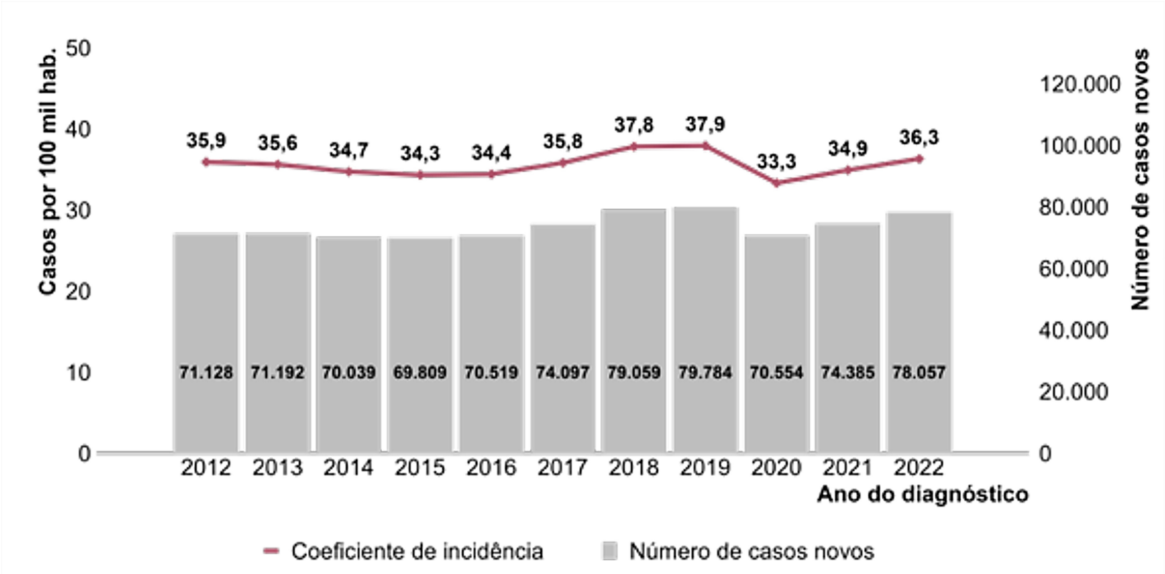


Figura 1 – Coeficiente de incidência de TB no Brasil, de 2012 a 2022, por 100 mil habitantes. Fonte: Brasil (2023).

A Figura 2a apresenta o coeficiente de TB nos estados do Brasil, no ano de 2022. Verifica-se que, em 2022, o número de casos notificados foi de 78.057 e os coeficientes de incidência variaram de 09 a 84 casos, por 100 mil habitantes, entre os estados do Brasil. Percebe-se pela Figura 2a que os três estados com maior coeficiente de incidência de TB foram Amazonas, Roraima e Rio de Janeiro, com valores de 84,1, 75,9 e 68,6, respectivamente, a cada 100 mil habitantes (BRASIL, 2023).

Conforme dados do Boletim Epidemiológico de Tuberculose 2023, o índice de óbitos causados por TB vinha apresentando uma diminuição gradual e lenta ao longo dos últimos vinte anos. Entretanto essa progressão positiva foi interrompida em 2021, em que ocorreram 5.072 mortes devido à doença, resultando em uma taxa média nacional de 2,38 mortes por cada 100 mil habitantes. Sendo que, Rio de Janeiro, Amazonas e Pernambuco foram os estados brasileiros com as maiores taxas de mortalidade por TB, conforme

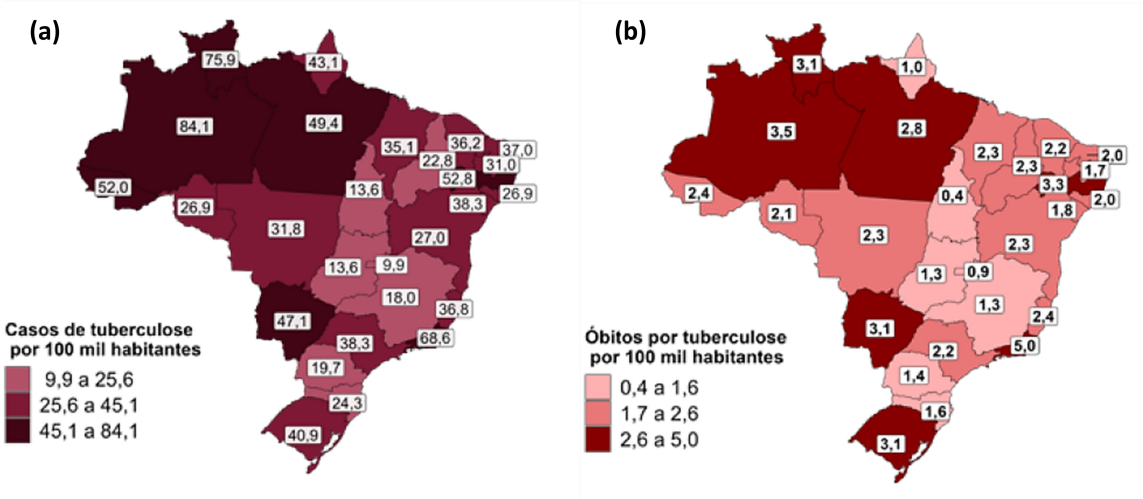


Figura 2 – (a) Coeficiente de incidência de TB nos estados do Brasil, em 2022, por 100 mil habitantes. (b) Coeficiente de mortalidade por TB nos estados do Brasil, em 2021 (dados extraídos em novembro/2022), por 100 mil habitantes. Adaptado de Brasil (2023).

ilustrado na Figura 2b. Um total de seis estados registraram um índice de mortalidade por TB acima da média nacional (BRASIL, 2023).

Diante do exposto, entende-se, que diagnosticar a TB de forma rápida e correta, ampliar a confirmação bacteriológica entre aqueles com diagnóstico de TB e tratar a TB de forma adequada, nos seus estados iniciais, são ações essenciais para o controle da doença e interrupção de sua cadeia de propagação (WHO, 2022).

A TB pulmonar pode ser identificada por meio da identificação do bacilo, técnica definida como baciloscopia, teste rápido molecular para tuberculose (TRM-TB) e pelo isolamento do bacilo por meio da cultura. A baciloscopia é uma técnica utilizada em muitos países para diagnóstico e controle (de pacientes em tratamento) da TB pulmonar. Esta técnica é relativamente rápida de ser executada e de baixo custo, porém ela possui baixa sensibilidade, é cansativa por ser totalmente manual e depende muito da experiência do observador (BRASIL, 2022). O TRM-TB apresenta o resultado em aproximadamente 100 minutos, sendo necessária somente uma amostra de escarro, entretanto o TRM-TB tem um custo considerável (WHO, 2020a). A cultura é um método de elevada especificidade e sensibilidade no diagnóstico da TB, considerado o método padrão de diagnóstico. Porém a cultura é limitada em função do crescimento lento das micobactérias, o que torna o diagnóstico demorado, bem como pela necessidade de estrutura física adequada do laboratório e qualificação da equipe técnica, levando de 4 a 8 semanas (BRASIL, 2022).

A baciloscopia do escarro ainda é a técnica mais utilizada em países em desenvolvimento, por ser um método simples e seguro, de baixo custo e de fácil manutenção, apesar de possuir baixa sensibilidade ao ser comparada com os testes TRM-TB e cultura (KHUTLANG et al., 2009)(PANICKER et al., 2018)(BRASIL, 2022). Devido a importância

da baciloscopia no diagnóstico da TB pulmonar, muitos autores da literatura têm proposto utilizar técnicas de processamento de imagem para realizar o diagnóstico automatizado da TB de forma rápida, eficiente e precisa. Além da avaliação de um radiologista ou pneumonologista, têm se destacado técnicas de Inteligência Artificial (IA), como Aprendizado de Máquina (ML, do inglês *Machine Learning*) e Aprendizado Profundo (ML, do inglês *Deep Learning*).

Tais técnicas vêm ganhando destaque por possuírem alta capacidade de predição de doenças e por auxiliarem o profissional de saúde a realizar diagnósticos mais precisos. Trabalhos que utilizam a visão computacional para identificar doenças pulmonares foram abordados por Felix et al. (2009) e Macedo et al. (2012). Métodos de ML também têm sido utilizados para fazer o diagnóstico de doenças mamárias (GONÇALVES et al., 2017). A Rede Neural Convolucional (CNN, do inglês *Convolutional Neural Network*) foi retratada por Neto e Hidalga (2017) para detecção de tumores cerebrais e abordada por Vergara (2018) para alterações cerebrais associadas à esquizofrenia. Métodos de DL têm sido retratados para identificação de convulsões epiléticas (SILVA et al., 2017) e ao diagnóstico de glaucoma (LIMA et al., 2019).

Ainda no contexto de aplicação de técnicas de IA, existem alguns estudos explorando formas de melhorar a acurácia e a eficiência da detecção e classificação do bacilo Mtb. Por exemplo, modelos baseados em CNN, como os propostos por Panicker et al. (2018) e Quinn et al. (2016), mostram alta acurácia em classificar imagens de esfregaços de escarro. Abordagens integradas, incluindo CNNs, foram exploradas por Simon et al. (2019) e Mithra e Emmanuel (2019), apresentando resultados promissores em termos de precisão e sensibilidade. Os trabalhos de Kuok et al. (2019) e El-Melegy, Mohamed e ElMelegy (2019) exploraram um método de detecção de objeto e a integração com CNN, reduzindo falsos positivos e elevando os índices de desempenho. Além disso, Sethi, Parmar e Suri (2018) expandiram as possibilidades de aplicação dessas técnicas, propondo um modelo otimizado para hardware dedicado capaz de identificar patógenos em uma variedade de amostras de doenças, incluindo a TB. Além destes, existem outros estudos que também focam na TB, na análise de imagens da baciloscopia (por diferentes colorações) ou em imagens de raio-x.

Considerando o impacto da TB em escala mundial, o propósito da OMS de reduzir a subnotificação e melhorar o acesso ao diagnóstico e tratamento de TB, juntamente com a possibilidade de se aplicar técnicas computacionais para auxiliar na identificação da TB, o presente trabalho tem como objetivo propor uma nova abordagem semi-automatizada que seja capaz de auxiliar o profissional de saúde no diagnóstico da TB por meio da baciloscopia.

Com o propósito de obter agilidade na detecção e diagnóstico da TB e deste modo contribuir para a redução de mortalidade pela doença, a principal questão de pesquisa

que pretende-se explorar é a seguinte: como abordagens de processamento de imagens e inteligência artificial podem ser utilizadas para desenvolver um software que auxilie no diagnóstico da TB, por meio da análise de imagens da baciloscopia? A resposta prática desta pergunta pode contribuir com uma implicação importante para aperfeiçoamento da baciloscopia, capaz de torná-la mais ágil e precisa.

1.1 Objetivos

O objetivo geral deste trabalho é explorar de forma sistemática o uso de aprendizado profundo para detectar o bacilo Mtb e desenvolver um protótipo de software que automatize parte do processo de diagnóstico da TB pela baciloscopia. O protótipo proposto auxiliará um especialista na tarefa de análise microscópica, onde é realizado a contagem dos bacilos manualmente. Desta forma, o tempo de diagnóstico e risco de falha humana serão reduzidos, fazendo com que os índices de eficiência da baciloscopia sejam aprimorados.

1.1.1 Objetivos Específicos

Para alcançar o objetivo geral, é necessário que sejam atingidos os objetivos específicos, listados a seguir:

- Entender os passos utilizados na técnica de baciloscopia e identificar quais são as limitações e dificuldades do ser humano em realizar esse procedimento manualmente;
- Analisar as soluções automatizadas propostas para auxiliar no diagnóstico da TB;
- Propor um novo conjunto de dados formado por imagens da baciloscopia para treinamento de métodos de ML;
- Analisar e implementar algoritmos de ML para identificar, destacar e contar os bacilos Mtb em imagens da baciloscopia;
- Realizar experimentos computacionais rigorosos para validar o desempenho dos algoritmos implementados em diferentes cenários de testes com distintos conjuntos de dados;
- Desenvolver um protótipo de apoio ao diagnóstico da TB a partir de imagens da baciloscopia.

1.2 Contribuições

Com o grande número de ocorrências de TB, surge a necessidade de aprimorar a forma e rapidez com que se diagnostica a doença. Além do mais, a baciloscopia ainda é o

método mais utilizado como primeira etapa para o diagnóstico da doença e controle de tratamento, porém é cansativo, totalmente manual e necessita de treinamento intensivo do examinador. Desta forma, justifica-se o desenvolvimento de novas abordagens que possam aprimorar a baciloscopia.

Neste sentido, esta tese apresenta uma série de contribuições significativas para o campo da detecção automática do bacilo Mtb via baciloscopia utilizando técnicas de DL. As contribuições podem ser categorizadas da seguinte forma:

Avaliação Abrangente de Modelos de DL: Uma avaliação detalhada e abrangente de diferentes arquiteturas de CNNs e variados métodos de detecção de objeto foi realizada, cobrindo uma variedade de conjuntos de dados e cenários diferentes. Esta avaliação permitiu identificar os modelos com melhor desempenho, destacando-se a Faster R-CNN, e proporcionou uma compreensão aprofundada de como esses modelos se comportam em diferentes condições.

Desenvolvimento e Otimização de métodos: Foram propostas e avaliadas diversas métodos para aprimorar o desempenho dos modelos de DL no contexto específico da detecção de bacilos Mtb. Isso incluiu a implementação de técnicas de aumento de dados para melhorar a robustez dos modelos, a aplicação de estratégias de particionamento de imagens para lidar com variações no tamanho das imagens e a introdução de um sistema de votação para auxiliar na anotação de imagens para treinamento.

Desenvolvimento de um processo para Análise Autônoma: Foi proposto um processo de diagnóstico para avaliar o desempenho de modelos de DL ao analisar de forma autônoma lâminas de baciloscopia. Este processo foi submetido a uma análise crítica, considerando tanto as métricas de desempenho quanto uma interpretação mais ampla dos resultados com base em uma matriz de confusão detalhada.

Protótipo de Software CADx: Um protótipo de software CADx foi desenvolvido com o objetivo de auxiliar no diagnóstico da TB, integrando as técnicas e métodos avaliados ao longo dos experimentos. Este protótipo representa um passo significativo em direção à implementação prática das técnicas de DL na análise de baciloscopia, potencialmente acelerando e aprimorando o processo de diagnóstico.

Contribuição para a Literatura e Futuros Trabalhos: Os resultados e conclusões obtidos por meio desta pesquisa contribuem significativamente para a literatura existente, oferecendo uma visão completa e detalhada do estado atual das técnicas de DL aplicadas ao diagnóstico da TB via baciloscopia. Além disso, foram delineadas diversas direções para trabalhos futuros, proporcionando um guia valioso para pesquisadores que desejam continuar explorando este campo.

Análise Crítica e Perspectivas Futuras: A tese não apenas apresenta uma série de experimentos e avaliações, mas também oferece uma análise crítica dos resultados,

destacando as limitações e desafios encontrados, bem como as oportunidades para melhorias futuras. Isso garante que os leitores tenham uma compreensão clara e completa do estado atual do campo e das possíveis direções para o futuro.

Em resumo, as contribuições desta tese abrangem desde a avaliação de modelos e o desenvolvimento de novos métodos até a proposta de implementação prática de soluções para o diagnóstico de TB, fornecendo um recurso valioso para a comunidade científica e médica e pavimentando o caminho para futuras inovações no campo.

1.3 Estrutura do Trabalho

O restante do trabalho está organizado da seguinte forma:

- O Capítulo 2 apresenta a definição do principais fundamento teóricos utilizados neste trabalho: a tuberculose e seu diagnóstico, a baciloscopia, aprendizado de máquina, principais métodos de aprendizado profundo e aumento de dados;
- O Capítulo 3 apresenta uma revisão sistemática realizada a partir de trabalhos que foram publicados na literatura para auxiliar o diagnóstico de doenças utilizando aprendizado profundo, os trabalhos relacionados ao diagnóstico da TB e o conjunto de dados existentes na área de reconhecimento de bacilos em imagens de baciloscopia por meio de técnicas computacionais;
- O Capítulo 4 apresenta a estrutura do novo conjunto de dados proposto, descrevendo os procedimentos de escolha das amostras, a captura das imagens, o sistema para identificação e tratamento dos bacilos e a disponibilização dos conjuntos de dados;
- O Capítulo 5 apresenta os materiais e métodos que são utilizados, demonstrando os conjunto de dados, as arquitetura CNNs utilizadas, os métodos a serem explorados, e as abordagens proposta para aprimorar a identificação automática dos bacilos;
- O Capítulo 6 apresenta os experimentos computacionais e análises dos resultados obtidos;
- Por fim, o Capítulo 7 apresenta as conclusões e os trabalhos futuros.

2 Referencial Teórico

2.1 Tuberculose

A TB é uma doença infectocontagiosa, comumente crônica, que atinge principalmente os pulmões, denominada TB pulmonar, porém outros órgãos podem ser afetados como: rins, ossos, olhos, entre outros, sendo estas denominadas como TB extrapulmonar (FILHO, 2001). A TB é causada por uma bactéria, que pertence a qualquer uma das sete espécies que integram o complexo *Mycobacterium tuberculosis*: *M. tuberculosis*, *M. bovis*, *M. africanum*, *M. canetti*, *M. microti*, *M. pinnipedi* e *M. caprae*. Este complexo refere-se ao grupo de espécies geneticamente muito semelhantes, mas com características fenotípicas diversas, as quais causam TB em humanos e animais. De todas as espécies do complexo, a *M. tuberculosis* é a mais estudada (FORRELLAD et al., 2013).

Acredita-se que a TB seja anterior ao próprio surgimento do homem. Pressupõe-se que ela ocorreu como uma doença endêmica entre os animais muito antes de afetar os humanos, uma vez que ossos de animais dos períodos paleolítico e neolítico[1] mostram evidências da doença, e a história registrada desde tempos muito antigos contém referências inconfundíveis aos seus efeitos no homem (STEELE; RANNEY, 1958). Além disso, historiadores encontraram a existência de TB em múmias egípcias que datam de 2.000 a 4.000 a.C., o que comprova sua existência milenar (DANIEL; BATES; DOWNES, 1994).

A palavra tuberculose vem do latim *tuberculum*, diminutivo de *tuber*, que significa tumor, referindo-se aos nódulos que a doença causa no pulmão. Em 24 de março de 1882, Robert Koch, bacteriologista alemão, apresentou o bacilo da TB à Sociedade Fisiológica de Berlim, por isso, a TB também é conhecida como Bacilo de Koch (KOCH et al., 1932).

A TB é uma doença de transmissão aérea e se instala a partir da inalação de aerossóis oriundos das vias aéreas, durante a fala, espirro ou tosse das pessoas com TB ativa (pulmonar ou laríngea), que lançam no ar partículas em forma de aerossóis contendo bacilos. O risco de transmissão é maior durante contatos prolongados em ambientes fechados e com pouca ventilação, uma vez que as gotículas microscópicas de saliva eliminadas, contendo o agente infeccioso, podem permanecer dispersos no ar durante horas. Assim, uma pessoa, ao respirar o ar contaminado, inala as micobactérias, que se instalam nos pulmões. É importante destacar que as formas extrapulmonares não são transmissíveis e ocorrem mais frequentemente em pessoas que vivem com o HIV, especialmente, aquelas com comprometimento imunológico (BRASIL, 2019).

[1] Período Paleolítico ou Idade da Pedra Lascada (do surgimento da humanidade até 8000 a.C.). Período Neolítico ou Idade da Pedra Polida (de 8000 a.C. até 5000 a.C.).

A TB é uma doença curável em praticamente todos os casos, o tratamento dura no mínimo seis meses, é gratuito e está disponível no Sistema Único de Saúde (SUS). Logo nas primeiras semanas do tratamento, o paciente se sente melhor e, por isso, o sucesso do tratamento da doença está diretamente associado à conscientização do paciente sobre a gravidade da doença, a necessidade de continuidade do tratamento prescrito e, principalmente, a importância da regularidade no uso dos medicamentos até o final, independentemente da melhora dos sintomas (BRASIL, 2019).

A prevenção da TB pode ser feita pela vacina bacilo Calmette-Guérin (BCG), ofertada no SUS. A vacina BCG protege a criança das formas mais graves da doença, como a TB miliar e a TB meníngea, mas tem pouca eficácia contra a TB pulmonar; deve ser administrada, prioritariamente, em crianças de 0 a 4 anos, com obrigatoriedade para menores de 1 ano de idade. Outras medidas de prevenção da TB incluem o diagnóstico precoce, oferta do tratamento da infecção latente e a busca ativa dos contatos que promovem a prevenção e a quebra da cadeia de transmissão da doença (BRASIL, 2019).

2.2 Diagnóstico da Tuberculose

Para o diagnóstico da TB pulmonar, no Brasil, são utilizados exames bacteriológicos/moleculares e também exame por imagem. Por meio dos exames bacteriológicos/moleculares, a TB pode ser identificada de três formas: (i) baciloscopia, (ii) teste rápido molecular para tuberculose (Xpert MTB RIF Ultra) e (iii) cultura para micobactéria. No exame por imagem, comumente é utilizada a radiografia de tórax. A radiografia não permite o diagnóstico definitivo de TB, pois em paciente sintomático respiratório (tosse com mais de 3 semanas), a sensibilidade é de 85% e a especifidade de 96% (WHO, 2021). Neste cenário, a radiografia de tórax, auxilia o profissional de saúde a utilizar outros exames confirmatórios (BRASIL, 2019).

A baciloscopia é uma técnica de pesquisa do bacilo álcool-ácido resistente (BAAR), pelo método de *Ziehl-Nielsen* (ZN), é utilizada em grande parte do país para diagnóstico e controle (de pacientes em tratamento) da TB pulmonar. Esta técnica é relativamente rápida de ser executada e de baixo custo, sendo recomendada pelo Ministério da Saúde, portanto, deve ser realizada por todo laboratório público de saúde e pelos laboratórios privados tecnicamente habilitados, de baixa complexidade. Entretanto, a baciloscopia depende muito da experiência do observador, logo, desde que seja executada corretamente em todas as suas fases, permite detectar de 60% a 80% dos casos de TB pulmonar em adultos. Isto é muito importante uma vez que que os casos com baciloscopia positiva são os maiores responsáveis pela manutenção da transmissão. Vale ressaltar que em crianças, a sensibilidade da baciloscopia é bastante diminuída pela dificuldade de obtenção de uma amostra com boa qualidade (BRASIL, 2008).

O teste rápido molecular Xpert MTB RIF Ultra para tuberculose (TRM-TB) é indicado para o diagnóstico de TB pulmonar e laríngea em adultos e adolescentes, em função de maior sensibilidade e especificidade em amostras respiratórias e extrapulmonares, quando em comparação à baciloscopia. O TRM-TB é um teste de amplificação de ácidos nucleicos utilizado para detecção de DNA dos bacilos do complexo *M. tuberculosis*, bem como mutações que conferem resistência à rifampicina. O TRM-TB apresenta o resultado em aproximadamente 100 minutos, sendo necessária somente uma amostra de escarro. A sensibilidade do TRM-TB em uma amostra clínica de adultos é de cerca de 85% e a sensibilidade do teste para resistência à rifampicina é de 95% (WHO, 2020b). Entretanto, o TRM-TB tem um custo considerável, encontra-se disponível em alguns municípios brasileiros, na rede pública de saúde, não está disponível na maioria dos laboratórios da rede privada (BRASIL, 2008), e cobre apenas 40% dos pacientes com TB que iniciam tratamento (WHO, 2020a).

A cultura é um método de elevada especificidade e sensibilidade no diagnóstico da TB, considerado o método padrão de diagnóstico. Nos casos pulmonares com baciloscopia negativa, a cultura do escarro pode aumentar em até 30% o diagnóstico bacteriológico da doença (BRASIL, 2019). Porém a cultura é limitada em função do crescimento lento das micobactérias, o que torna o diagnóstico demorado, bem como pela necessidade de estrutura física adequada do laboratório e qualificação da equipe técnica, levando de 4 a 8 semanas (FERREIRA et al., 2005). Os meios de cultura mais comumente utilizados são os sólidos à base de ovo, pois, são de menor custo e apresentam um baixo índice de contaminação, mas o tempo de detecção do crescimento bacteriano é lento, variando de 14 a 30 dias e podendo estender-se por até oito semanas (BRASIL, 2019). Além destes, existem os métodos que utilizam meios de cultura líquidos, os quais apresentam resultados mais rápido que os métodos sólidos, geralmente após 5 a 13 dias. Entretanto, pelo menos uma cultura em meio sólido deve ser realizada em paralelo, para garantia de isolamento de cepas de *Mycobacterium tuberculosis* que não crescem em outros meios, e por permitir a preservação da cepa para estudos posteriores (FILHO et al., 2004).

Em resumo, a baciloscopia do escarro continua sendo a técnica mais utilizada em nosso país, por ser um método simples e seguro, apesar de possuir baixa sensibilidade ao ser comparada com os testes TRM-TB e cultura. Devido a importância da baciloscopia no diagnóstico da TB pulmonar e a proposta deste trabalho de utilizar técnicas de processamento de imagem para realizar o diagnóstico automatizado de forma rápida, eficiente e precisa, a próxima seção detalha esta técnica.

2.2.1 Baciloscopia

O entendimento da baciloscopia é essencial, uma vez que se pretende automatizá-lo utilizando técnicas de inteligência computacional. Os principais métodos utilizados para

realizar a baciloscopia são a coloração de *Ziehl-Neelsen* e Auramina "O". O método *Ziehl-Neelsen* é a técnica mais utilizada para identificação do bacilo da TB, sendo o recomendado pelo Ministério da Saúde.

A bactéria *M. tuberculosis* é classificada morfologicamente como bacilo, devido ao seu formato em bastonete, podendo ser um formato reto ou ligeiramente curvo, como pode ser visto na Figura 3. Cada bacilo pode medir de 1μm de comprimento por 0,2μm de largura, podendo chegar a 10μm de comprimento por 0,6μm de largura (LOGAN, 2009).

Figura 3 – Exemplo bacilo tuberculose em forma de bastonetes. Fonte: Pinto (2018).

A coloração de ZN foi desenvolvida por Franz Ziehl, em 1882, e posteriormente alterada por Friedrich Neelsen. Neste método, detecta-se os microrganismos que são capazes de resistir à descoloração pela mistura álcool-ácido. Esta propriedade de resistência ao álcool-ácido está relacionada com a composição da parede celular das micobactérias, sendo muito importante para a análise por microscopia, pois permite que o bacilo assuma coloração rosada, facilitando a visualização do mesmo na amostra. Por este motivo, os bacilos são conhecidos como bacilo álcool-ácido resistente (BAAR) (PINTO, 2018).

Em síntese, na técnica de ZN, realiza-se um processo de descoloração e coloração em uma amostra clínica, como é mostrado na Figura 4. Utilizando o composto fucsina fenicada, cora-se todas as células bacterianas e outras estruturas presentes no esfregaço de vermelho. Em seguida é aplicado a mistura álcool-ácido para descorar todas as bactérias, transformando-as na cor azul do corante de contraste (azul de metileno). Entretanto, as bactérias que são resistentes a este processo de descoloração permanecem coradas de vermelho pela fucsina. Dessa forma, através do microscópio, as bactérias visualizadas coradas de vermelho são bacilos álcool-ácido resistente, e as bactérias visualizadas de azul são não resistentes ao álcool-ácido, como é demonstrado na Figura 5 (CAMPELO et al., 2001).

Um dos procedimentos realizados pela baciloscopia é a contagem dos bacilos na amostra através da observação do escarro em um microscópio. A contagem é realizada dividindo a amostra em pequenas partes, denominadas campos, sendo que a quantidade de bacilos em cada campo é registrada em uma planilha. Os dados registrados, são utilizados para diagnosticar a TB, considerando a proporção da quantidade de bacilos pelos campos observados, conforme Tabela 1. Esse procedimento pode ser considerado cansativo e passível de erro, uma vez que a contagem e divisão dos campos devem ser feitas manualmente e

sua acurácia muitas vezes depende da experiência do profissional que está analisando a amostra (KONEMAN et al., 2012).

De acordo com Ezembro, AZAM e CADIR (2012), a realização da baciloscopia no caso da TB pulmonar é importante, pois, detecta a maioria dos casos bacilíferos, limita a cadeia de transmissão, permite avaliar o sucesso ou falência do tratamento, além de ser um método simples, rápido e de baixo custo. No entanto, alguns dos fatores que determinam os bons resultados da baciloscopia são os cuidados na análise técnica, a interpretação correta dos resultados e ainda a emissão do resultado a tempo, normalmente em menos de 24 horas. Deve-se ressaltar que uma baciloscopia positiva (identificação do bacilo), na sua grande maioria trata-se do *M. tuberculosis*, porém outras microbactérias tem as mesmas características de serem álcool-ácido resistentes, tais como: *M. avium*, *M. assassina*, entre outras.

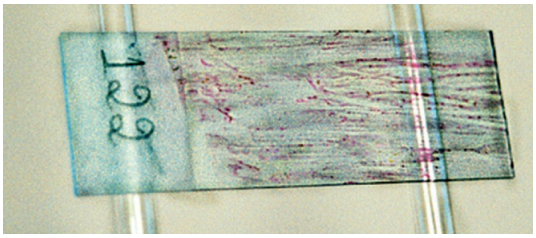


Figura 4 – Amostra do escarro de um paciente com tuberculose descolorado pelo método de *Ziehl-Neelsen*. Fonte: Campelo et al. (2001)

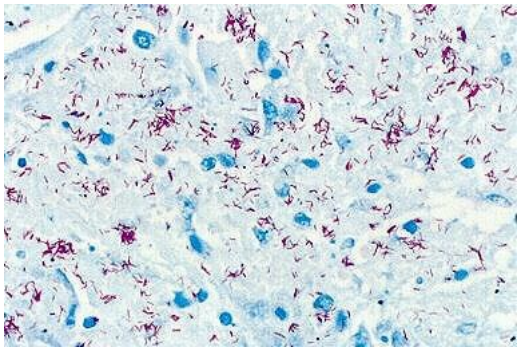


Figura 5 – Amostra da baciloscopia pelo método de *Ziehl-Neelsen* em um paciente com tuberculose ampliado em um microscópio. Fonte: Koneman et al. (2012)

Tabela 1 – Critérios para leitura e interpretação dos resultados da baciloscopia de escarro (BAAR: Bacilo Álcool-Ácido Resistente) (MS, 2022)

Leitura	**Resultado**
Não são encontrados BAAR em 100 campos	Relata-se o resultado como NEGATIVO
São encontrados de 1 a 9 BAAR, em 100 campos	Relata-se a quantidade de BAAR encontrada
São encontrados de 10 a 99 BAAR, em 100 campos	Relata-se o resultado como POSITIVO +
São encontrados em média de 1 a 10 BAAR por campo, nos primeiros 50 campos observados	Relata-se o resultado como POSITIVO ++
São encontrados em média mais de 10 BAAR por campo, nos primeiros 20 campos observados	Relata-se o resultado como POSITIVO +++

2.3 Aprendizado de Máquina

O aprendizado de máquina (ML, do inglês, *Machine Learning*) pode ser definido como métodos computacionais usando a experiência para melhorar o desempenho ou fazer previsões precisas. Vale ressaltar que, a experiência refere-se às informações anteriores disponíveis para se aprender, que normalmente são dados coletados e disponibilizados para análise. Estes dados podem estar na forma de conjuntos de treinamento com identificação humana ou outros tipos de informações obtidas por meio da interação com o ambiente, sendo que, sua qualidade e tamanho são essenciais para o sucesso das previsões feitas (MOHRI; ROSTAMIZADEH; TALWALKAR, 2018).

De acordo com Shalev-Shwartz e Ben-David (2014), ML refere-se à detecção automatizada de padrões significativos nos dados. Nas últimas décadas, ML tornou-se uma ferramenta comum em quase todas as tarefas que exigem extração de informações de grandes conjuntos de dados, tais como: os mecanismos de busca que aprendem a trazer os melhores resultados; o software anti-spam que aprende a filtrar as mensagens de e-mail; as transações com cartão de crédito que são protegidas por um software que aprende a detectar fraudes; câmeras digitais que aprendem a detectar rostos; aplicativos inteligentes de assistência pessoal em smartphones que aprendem a reconhecer comandos de voz; carros equipados com sistemas de prevenção de acidentes; etc. Além disso, ML é amplamente utilizado em aplicações científicas, como bioinformática, medicina e astronomia.

Como definido por Mohri, Rostamizadeh e Talwalkar (2018), o sucesso de um algoritmo de aprendizado de máquina depende dos dados usados. Dessa forma, entende-se que o ML está relacionado à análise de dados e estatísticas, ou seja, as técnicas de ML são métodos orientados por dados que combinam conceitos fundamentais em ciência da computação com ideias de estatística, probabilidade e otimização. São várias as abordagens de ML existentes que podem ser utilizadas para habilitar os algoritmos a aprenderem e se adaptarem, como por exemplo: árvores de decisão (*decision tree*), SVM (do inglês, *support vector machines*), kNN (do inglês, *k-nearest neighbors*), Rede Neural Artificial (ANN, do inglês *Artificial Neural Network*), Aprendizado Profundo (DL, do inglês, *Deep Learning*), entre outras.

Neste trabalho, utilizou-se a abordagem de aprendizado profundo, por meio de técnicas de Redes Neurais Convolucionais (CNN, do inglês, *Convolutional Neural Network*). Porém, antes de abordar sobre o aprendizado profundo e as técnicas de CNNs, faz-se necessário definir alguns conceitos de ANN úteis para a compreensão de CNN.

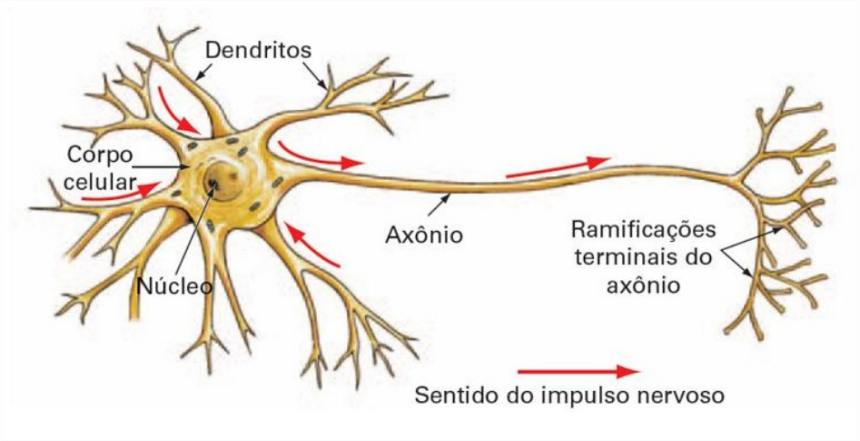


Figura 6 – Representação do neurônio biológico. Fonte: Academy (2021).

2.3.1 Rede Neural Artificial

Uma rede neural artificial é um modelo de computação inspirado na estrutura das redes neurais do cérebro. De forma semelhante ao cérebro, ANN consiste em um grande número de dispositivos computacionais básicos, os neurônios, que estão conectados uns aos outros em uma rede de comunicação complexa, por meio da qual o cérebro é capaz de realizar cálculos altamente complexos (SHALEV-SHWARTZ; BEN-DAVID, 2014).

Para compreender o funcionamento da ANN, é essencial entender alguns conceitos básicos referentes ao funcionamento do cérebro humano. Em vista disso, Academy (2021) definem o neurônio como a unidade básica do cérebro humano especializada na transmissão de informações. A Figura 6 apresenta um neurônio do sistema nervoso do corpo humano. Pode-se perceber que, este é constituído por três estruturas principais: (i) o corpo celular, responsável por coletar e combinar informações vindas de outros neurônios; (ii) os dendritos, os quais têm a função de receber informações oriundas de outros neurônios e conduzi-las até o corpo celular; e o axônio, no qual estão os nervos terminais que transmitem as informações para outros neurônios, transmissão chamada de sinapse. Dessa forma, são pelas sinapses que os neurônios se unem funcionalmente, formando uma enorme rede de comunicação, denominada rede neural (ACADEMY, 2021).

Baseado na estrutura e funcionamento da rede neural biológica, McCulloch e Pitts (1943) propuseram a rede neural artificial, a qual simula de maneira simplificada o funcionamento de um neurônio do sistema nervoso do corpo humano. A Figura 7 apresenta o modelo do neurônio artificial. Os impulsos elétricos provenientes de outros neurônios são representados pelos sinais de entrada. No modelo da ANN, $X_1, ..., X_n$ são os dados de entrada (*inputs*). A medida dos estímulos recebidos é representada por meio dos pesos sinápticos. No modelo da ANN, os pesos sinápticos (*weights*) são representados por $W_1, ..., W_n$. O corpo celular é representado por: a junção aditiva ($\sum$), que é o somatório dos estímulos de entrada multiplicado pelo seus pesos sinápticos; uma função de ativação

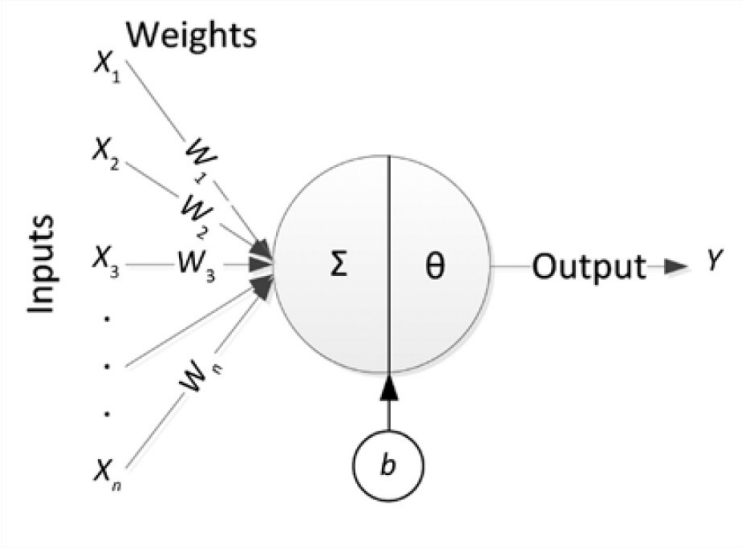


Figura 7 – Representação do neurônio artificial. Fonte: Awad e Khanna (2015).

(θ), que define com base nas entradas e pesos sinápticos, qual será a saída do neurônio; e a polarização (b), que tem como finalidade permitir um melhor ajuste da função de ativação (θ), com o intuito de aumentar o grau de liberdade desta função. A propagação da saída (*output*) é análogo ao axônio em um neurônio biológico, representado por Y (ACADEMY, 2021), (AWAD; KHANNA, 2015).

Conforme Awad e Khanna (2015), matematicamente, um neurônio artificial é representado por:

$$Y = \theta\left(\sum_{i}^{n} W_i X_i + b\right) \tag{2.1}$$

A Equação 2.1 pode ser modelada, usando uma forma de matriz:

$$Y = \theta\left(W.X + b\right) \tag{2.2}$$

onde $W = [W_1 W_2 ... W_n]$ e $X = \begin{bmatrix} \\ X_2 \\ \vdots \\ X_n \end{bmatrix}$

Dessa forma, as entradas X são multiplicadas pelos pesos sinápticos correspondentes W, gerando as entradas ponderadas, descrevendo assim, uma das bases matemáticas do funcionamento de uma ANN que é a multiplicação de matrizes (ACADEMY, 2021).

Segundo Awad e Khanna (2015), a função de ativação molda a saída, uma vez que o desempenho do modelo de ANN depende da escolha desta função. Múltiplas funções de ativação podem ser usadas, as mais utilizadas são:

$$\textbf{Limitador rígido} : \theta(a) = \begin{cases} 0, \text{se } a < 0, \\ 1, \text{se } a > 0 \end{cases} \tag{2.3}$$

$$\textbf{Função Linear de Saturação} : \theta(a) = \begin{cases} 0, \text{se } a < 0, \\ a, \text{se } 0 \leq a \leq 1, \\ 1, \text{se } a > 1 \end{cases} \tag{2.4}$$

$$\textbf{Função log-sigmóide} : \theta(a) = \frac{1}{1 + e^{-a}} \tag{2.5}$$

$$\textbf{Função sigmóide tangente hiperbólica} : \theta(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \tag{2.6}$$

A Figura 8 mostra uma ANN, definida por Awad e Khanna (2015) como uma associação de camadas em cascata de neurônios, cada uma com sua própria matriz de peso, vetor de polarização e vetor de saída. Percebe-se pela Figura 8 que uma camada de neurônios é formada por vários neurônios artificiais que operam em paralelo, com a saída da camada sendo a saída do vetor, que é formada pelas saídas individuais dos neurônios. De forma geral, um vetor constituído de N entradas e uma camada de M neurônios, tem-se que Wij é o peso da conexão, onde i representa o índice do neurônio em questão e j representa a entrada a qual o peso sináptico se refere; Y_i e b_i são a saída e a polarização associadas ao neurônio, respectivamente (AWAD; KHANNA, 2015).

Nota-se, ainda na Figura 8, que a ANN possui três camadas. A primeira camada em uma rede é chamada de camada de entrada, enquanto a última é chamada de camada de saída, e todas as camadas existentes entre as duas (entrada e saída) são referidas como camadas ocultas. Assim, a Figura 8 apresenta uma ANN com duas camadas ocultas (AWAD; KHANNA, 2015). É importante ressaltar que as camadas adjacentes são completamente conectadas, isto é, todos os neurônios de uma camada estão conectados a todos os neurônios da camada seguinte. E a quantidade de camadas e de neurônios necessários para o desempenho ótimo da ANN depende da complexidade da aplicação para a qual esta é projetada (ACADEMY, 2021). A próxima seção trata das redes neurais artificiais profundas e das camadas ocultas.

2.3.2 Aprendizado profundo

Aprendizado Profundo (DL, do inglês, *Deep Learning*) é uma sub-área do Aprendizado de Máquina, baseado no conceito de redes neurais artificiais, em que a informação é passada através de cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada. Cada camada é tipicamente um algoritmo simples e uniforme contendo

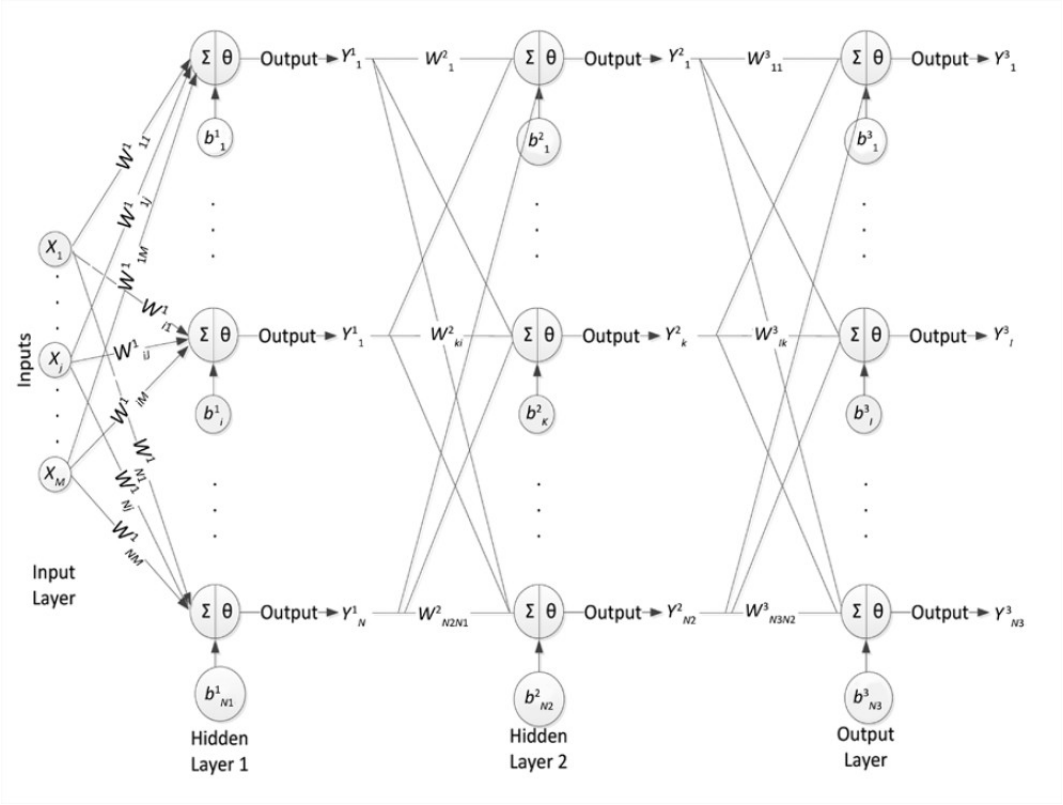


Figura 8 – ANN com três camadas. Fonte: Awad e Khanna (2015).

um tipo de função de ativação. O DL é responsável por avanços recentes em visão computacional, reconhecimento de fala, processamento de linguagem natural e reconhecimento de áudio (ACADEMY, 2021).

De acordo com Deng e Yu (2014), DL baseia-se em algoritmos de ML para aprender vários níveis de representação a fim de modelar relacionamentos complexos entre os dados, ou seja, é um conjunto de algoritmos que tentam aprender em vários níveis, correspondendo a diferentes níveis de abstração. Dessa forma, recursos e conceitos de nível superior são definidos em termos de recursos de nível inferior, e essa hierarquia de recursos é chamada de arquitetura profunda.

Dentro da área de DL, uma rede neural convolucional é uma classe de rede neural profunda, mais comumente aplicada para analisar imagens visuais (VALUEVA et al., 2020). Sendo assim, a próxima seção descreve uma rede neural convolucional.

2.3.2.1 Redes Neurais Convolucionais

Uma Rede Neural Convolucional (CNN, do inglês, *Convolutional Neural Networks*) é um algoritmo de aprendizado profundo que pode captar uma imagem de entrada, atribuir importância, isto é, pesos e vieses que podem ser aprendidos, a vários objetos da imagem e ser capaz de diferenciar um do outro. Além disso, o pré-processamento exigido em uma CNN é muito menor em comparação com outros algoritmos de classificação. Enquanto nos

métodos primitivos os filtros são feitos à mão, com treinamento suficiente, as CNNs têm a capacidade de aprender esses filtros (ACADEMY, 2021).

As primeiras CNNs, denominadas *Neocognitrons*, foram criadas em 1980 por Fukushima. Fukushima (1980) concebeu redes neurais com múltiplas camadas de agrupamento e convoluções que usava um design hierárquico e multicamadas, o qual permitiu ao computador aprender a reconhecer padrões visuais. Em 1989, o mecanismo de *backpropagation* foi proposto por Waibel et al. (1989) para o reconhecimento de fonemas em uma rede neural. Em 1989, LeCun et al. (1989) propuseram as primeiras CNNs que utilizam o *Backpropagation*, denominadas *ConvNets*, projetadas para o reconhecimento de dígitos manuscritos. Em 1998, outra CNN foi proposta por LeCun et al. (1998), denominada *LeNet*, a qual utilizou aprendizado baseado em gradiente. A partir deste trabalho, foram estabelecidas as redes convolucionais que têm sido utilizadas (ACADEMY, 2021).

Em uma arquitetura CNN, encontra-se quatro camadas básicas, que são: entrada, convolução, subamostragem, e totalmente conectada.

- **Camada de entrada:** tem a função de ler a imagem, isto é, processar os valores de pixel da imagem (O'SHEA; NASH, 2015).
- **Camada de convolução:** define uma operação para um filtro mapear as ativações de uma camada para a próxima, isto é, a camada de convolução gera um mapa de características da camada (SERRÃO et al., 2020b). Dessa forma, percebe-se que em cada camada de convolução, a CNN tenta entender os padrões básicos, como por exemplo, na primeira camada de convolução, a CNN tenta aprender padrões e bordas das imagens; na segunda camada, a CNN tenta entender a forma e a cor; e em uma camada final, denominada camada totalmente conectada, a CNN tenta classificar a imagem (ACADEMY, 2021).
- **Camada de subamostragem:** é usada após uma camada de convolução com a finalidade de reduzir o tamanho do mapa de características. Sendo assim, não há parâmetros a aprender nesta camada, uma vez que esta camada é usada apenas para reduzir o tamanho da dimensão da imagem (ACADEMY, 2021).
- **Camada totalmente conectada:** contém neurônios que estão diretamente conectados aos neurônios nas camadas adjacentes, sem estar conectado a nenhuma camada dentro delas. Isso é análogo ao modo como os neurônios são organizados nas formas tradicionais de ANN (O'SHEA; NASH, 2015).

A Figura 9 ilustra a arquitetura da CNN *LeNet-5* proposta por LeCun et al. (1998), onde é possível perceber a camada de entrada (*input*), as camadas de convoluções (*convolutions*), a camada de subamostragem (*subsampling*), a camada totalmente conectada (*full connection*) e a camada de saída (*output*).

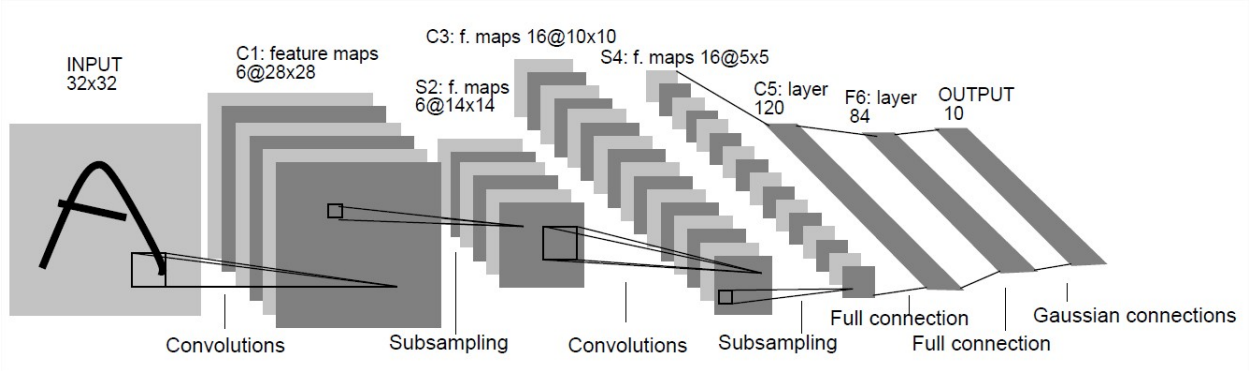


Figura 9 – Arquitetura da CNN *LeNet-5*. Fonte: LeCun et al. (1998).

2.4 K-Means Clustering

O K-*Means Clustering* é um método distintivo no campo da mineração de dados, caracterizado por sua abordagem de segmentação de um conjunto variado de dados em torno de pontos centrais, conhecidos como centroides. Esse método baseia-se nas características inerentes dos dados, proporcionando uma organização eficiente. No espectro dos algoritmos de ML, o K-*Means* é classificado como um algoritmo de aprendizado não supervisionado. Isso significa que, diferentemente dos métodos supervisionados, o K-*Means* opera sem um resultado alvo predefinido, permitindo a formação espontânea de agrupamentos baseados nas semelhanças entre os dados (SHALEV-SHWARTZ; BEN-DAVID, 2014).

O algoritmo K-*Means* foi inicialmente descrito por Hartigan (1975), com uma versão mais refinada apresentada por Hartigan e Wong (1979). Este algoritmo parte de uma abordagem estruturada para particionar um conjunto de dados em K grupos distintos (ou clusters). O desenvolvimento do K-*Means* envolve quatro etapas cruciais: (i) A escolha de K pontos iniciais atua como os centros dos clusters; (ii) cada ponto de dados é associado ao cluster cujo centroide é o mais próximo, baseando-se em medidas de distância; (iii) o ponto central de cada cluster é recalculado para refletir a posição média dos pontos atribuídos a ele; (iv) as etapas de atribuição e reajuste são repetidas iterativamente até que os centros dos clusters alcancem uma posição ideal, indicando a convergência do algoritmo (LIKAS; VLASSIS; J. Verbeek, 2003).

O K-*Means* é notoriamente reconhecido por sua simplicidade operacional e eficiência computacional, tornando-o adequado para uma ampla gama de aplicações em conjuntos de dados diversos. Contudo, sua aplicabilidade vem acompanhada de algumas limitações significativas. A necessidade de pré-determinar o número de clusters K e a sensibilidade do algoritmo às condições iniciais de seleção dos centroides podem influenciar significativamente os resultados finais, exigindo consideração cuidadosa e, em alguns casos, métodos complementares para otimizar a performance do algoritmo (FRäNTI; SIERANOJA, 2019).

2.5 Detecção de objetos

No contexto da detecção de objetos, existem métodos específicos de DL que combinam de maneira otimizada a seleção de áreas de interesse com a extração e classificação de características. Estes métodos representam um avanço significativo em visão computacional, oferecendo a capacidade de identificar e localizar objetos em imagens e vídeos com alta precisão. Estes métodos de DL geralmente seguem duas abordagens principais: detecção baseada em regiões e detecção em uma única etapa (REDMON et al., 2016a).

A detecção baseada em regiões, exemplificada por modelos como R-CNN (*Region-based Convolutional Neural Networks*) (GIRSHICK et al., 2014), Fast R-CNN (GIRSHICK, 2015) e Faster R-CNN (REN et al., 2015), envolve a proposição inicial de ROIs onde os objetos podem estar presentes, seguida pela classificação dessas regiões em categorias de objetos e ajuste das caixas delimitadoras. Por outro lado, métodos de uma única etapa, como YOLO (REDMON et al., 2016b) e SSD (LIU et al., 2016), realizam a classificação de objetos e a regressão das caixas delimitadoras simultaneamente, oferecendo detecção mais rápida, embora por vezes com precisão ligeiramente inferior em comparação com os métodos baseados em regiões. Ambas as abordagens têm sido aplicadas com sucesso em uma variedade de aplicações, desde a vigilância por vídeo até a análise de imagens médicas, demonstrando a versatilidade e eficácia dos métodos de DL na detecção de objetos.

A seguir, são apresentados de forma resumida alguns métodos de DL amplamente utilizados para realizar a detecção de objetos em imagens ou vídeos.

2.5.1 Faster R-CNN

O Faster R-CNN, um avanço significativo no campo da detecção de objetos, foi proposto por Ren et al. (2015), representando um salto qualitativo em relação ao seu antecessor, o Fast R-CNN. Este algoritmo inovador aborda especificamente o gargalo de desempenho associado à pesquisa seletiva ou às *bounding boxes* (caixas de borda) no Fast R-CNN. O Faster R-CNN introduz uma abordagem mais eficiente, substituindo a pesquisa seletiva por uma rede convolucional compacta, conhecida como Redes de Propostas de Região (*Region Proposal Networks* - RPNs). Essas RPNs são projetadas para gerar regiões de interesse, compartilhando camadas convolucionais com a rede de detecção, o que reduz significativamente o custo computacional adicional (REN et al., 2015).

O RPN, no cerne do Faster R-CNN, revoluciona a geração de propostas de regiões, e um Fast R-CNN subsequente é responsável pela detecção efetiva. A inclusão de caixas de âncora pelo Faster R-CNN aborda diretamente as variações de proporção e escala dos objetos, proporcionando uma melhoria notável na precisão e eficiência (REN et al., 2015).

O principal benefício do Faster R-CNN é aprimorar a eficiência computacional e a precisão em comparação com as versões anteriores. Este modelo consegue ser até 10 vezes mais rápido que o Fast R-CNN, mantendo uma precisão de detecção comparável em diversos conjuntos de dados, consolidando-se como um dos algoritmos de detecção de objetos mais precisos e eficientes disponíveis até a data (SACHAN, 2017).

Apesar dessas vantagens, o Faster R-CNN ainda enfrenta desafios no que diz respeito à aplicação em tempo real, principalmente devido à sua complexidade computacional. Embora seja mais rápido que o Fast R-CNN, a exigência computacional do Faster R-CNN permanece um obstáculo para seu uso em ambientes onde a velocidade de processamento é crucial (SACHAN, 2017).

2.5.2 YOLO

No avançado domínio da detecção de objetos, o algoritmo YOLO (*You Only Look Once*), proposto por Redmon et al. (2016b), representa um método revolucionário. Distanciando-se das abordagens tradicionais, o YOLO aborda a detecção de objetos como um singular problema de regressão, processando diretamente da imagem de entrada para aprender tanto as coordenadas das caixas delimitadoras quanto as classes dos objetos detectados (REDMON et al., 2016b). Esta abordagem única permite que o YOLO analise a imagem em sua totalidade, efetuando previsões em uma única passagem pela rede neural, fundamentando assim o seu nome emblemático.

Em termos operacionais, o YOLO divide a imagem em uma grade de dimensões SxS. Cada célula dessa grade é responsável por prever N caixas delimitadoras, cada uma com uma pontuação de confiança associada. Essa pontuação de confiança reflete não apenas a precisão da caixa delimitadora, mas também se ela efetivamente contém um objeto, independentemente de sua classe. Além disso, o YOLO calcula uma pontuação de classificação para cada caixa em relação a cada classe de treinamento. Combinando essas pontuações, é possível estimar a probabilidade de cada classe estar presente em uma caixa delimitadora específica, resultando em um total de $S \times S \times N$ caixas delimitadoras previstas por imagem (REDMON et al., 2016b).

A principal vantagem do YOLO reside em sua velocidade excepcional, tornando-o altamente adequado para aplicações em tempo real. Além disso, o YOLO demonstra uma notável capacidade de generalização para diferentes domínios, uma vez que processa a imagem completa de uma só vez. Portanto, essas informações ajudam a evitar falsos positivos. Contudo, existem desafios associados ao YOLO, particularmente em sua capacidade de detectar objetos pequenos e em situações de agrupamento. Historicamente, a precisão do YOLO tem sido geralmente inferior à do Faster R-CNN (SACHAN, 2017).

Por ser um método bastante popular, a Yolo conta com diferentes versões, oficiais e não oficiais. Como por exemplo: (i) YOLOv1 (REDMON et al., 2016b): introduziu a abordagem de visualização única para detecção de objetos, dividindo a imagem em regiões e prevendo caixas delimitadoras e probabilidades para cada região, é marcada pela sua rapidez, mas menos preciso em comparação com métodos concorrentes, com dificuldades em detectar objetos pequenos; (ii) YOLOv5 (JOCHER, 2020): não é uma continuação oficial das versões anteriores, mas uma implementação independente com foco na simplicidade e na facilidade de uso, é rápida, leve e fácil de implementar, com suporte para treinamento e inferência em várias plataformas; (iii) YOLOF(CHEN et al., 2021): destaca-se por sua arquitetura única que utiliza apenas um nível de recurso para a detecção de objetos, diferindo das abordagens multi-escala comuns em outras versões do YOLO, a qual busca um equilíbrio entre eficiência e em termos de velocidade e uma boa precisão, sendo especialmente benéfica para sistemas com recursos limitados ou para aplicações que exigem alta velocidade de processamento; dentre varias outras.

2.5.3 SSD

O SSD (*Single Shot MultiBox Detector*) é uma técnica de detecção de objetos que executa a extração de características e a predição de caixas delimitadoras em uma única passagem pela imagem, representando uma evolução na eficiência e precisão da detecção de objetos. Desenvolvido por Liu et al. (2016), o SSD distingue-se por sua capacidade de operar em múltiplas escalas, utilizando uma bateria de camadas convolucionais para detectar objetos em diferentes níveis de granularidade espacial (LIU et al., 2016).

A arquitetura do SSD permite que ele calcule um mapa de características abrangente, a partir do qual realiza predições de caixas delimitadoras utilizando caixas de âncora de proporções variadas. Essencialmente, o SSD aprende os deslocamentos das caixas de âncora em vez de prever as dimensões das caixas delimitadoras de forma direta. O uso estratégico de múltiplas camadas convolucionais permite ao SSD abordar objetos em diferentes escalas, tornando o algoritmo particularmente eficaz na detecção de objetos de vários tamanhos (LIU et al., 2016).

Para cada posição em uma camada de características, o SSD estabelece um conjunto predeterminado de caixas delimitadoras e suas respectivas pontuações de confiança para a detecção de classes de objetos. Ajustes finos são feitos por meio de regressão para alinhar as caixas delimitadoras com as formas reais dos objetos, culminando em uma detecção robusta que compreende uma variedade de tamanhos de objetos (LIU et al., 2016).

Uma das vantagens mais marcantes do SSD é a sua rapidez. Dispensando uma rede separada para a proposição de regiões de objetos e unificando todas as previsões em uma única etapa, o SSD opera a uma velocidade que favorece o uso em cenários de

tempo real. Contudo, essa velocidade pode ser alcançada às custas de uma menor precisão, particularmente na detecção de objetos menores, quando comparado com métodos mais complexos e computacionalmente intensivos como o Faster R-CNN (SACHAN, 2017).

2.5.4 RetinaNet

RetinaNet, introduzido por Lin et al. (2017b), é notável por abordar o desequilíbrio intrínseco entre classes na detecção de objetos usando a função de perda focal. Este modelo combina uma rede de características com um design específico de cabeça de detecção para lidar eficientemente com a detecção de objetos em diferentes escalas. O elemento central que diferencia a RetinaNet de outros detectores é sua função de perda focal, que foi projetada para resolver o problema de desequilíbrio de classe, colocando mais ênfase em objetos difíceis de detectar (LIN et al., 2017b).

A função de perda Focal modifica a perda de entropia cruzada padrão usada na classificação de objetos. Ela adiciona um fator de modulação à perda, reduzindo a contribuição para a perda de exemplos fáceis e aumentando a importância de corrigir exemplos mal classificados. Isso resulta em um treinamento mais eficaz, especialmente em cenários onde existem muitos exemplos de fundo fácil (contendo objetos comuns e repetidos) em comparação com exemplos de objetos raros ou difíceis (LIN et al., 2017b).

Além da função de perda Focal, a RetinaNet utiliza uma *Feature Pyramid Network* (FPN) para a extração de características. A FPN melhora a detecção de objetos em várias escalas, combinando informações de baixo nível com alta resolução e características de alto nível com contexto semântico mais rico. Isso torna a RetinaNet particularmente eficaz na detecção de objetos pequenos e em cenas com objetos de vários tamanhos (LIN et al., 2017a).

2.5.5 FSAF

O FSAF (*Feature Selective Anchor-Free*), proposto por Zhu, He e Savvides (2019), representa uma abordagem inovadora na detecção de objetos ao se desviar do uso tradicional de âncoras. O método é caracterizado por ser "*anchor-free*", ou seja, não depende de âncoras predefinidas para detectar objetos. Em vez disso, ele opera diretamente nas características extraídas da imagem, utilizando um módulo de seleção de características para escolher dinamicamente as camadas mais apropriadas da rede para a detecção de objetos em diferentes escalas (ZHU; HE; SAVVIDES, 2019).

O módulo de seleção de características do FSAF permite que a rede decida quais níveis de características usar para cada objeto, com base em seu tamanho e forma. Isso é alcançado por meio de um mecanismo de aprendizado que pondera a importância de

diferentes camadas de características para diferentes objetos. Como resultado, o FSAF pode adaptar-se automaticamente para otimizar a detecção de objetos variando em tamanho e aspecto (ZHU; HE; SAVVIDES, 2019).

Uma das principais vantagens do FSAF é sua abordagem de detecção direta. Ao eliminar a necessidade de âncoras, que muitas vezes precisam ser ajustadas manualmente e podem ser difíceis de otimizar, o FSAF simplifica o fluxo de detecção de objetos. Isso não apenas reduz a complexidade e o tempo de treinamento, mas também pode melhorar a acurácia em certos cenários, especialmente em imagens com uma distribuição variada de tamanhos de objetos. No entanto, o FSAF pode ser menos eficaz em cenários com objetos de formas e tamanhos altamente variáveis devido à ausência de âncoras (TIAN et al., 2019).

2.5.6 EfficientDet

EfficientDet, proposto por Tan, Pang e Le (2020), é um modelo de detecção de objetos focado na eficiência, equilibrando a precisão e a complexidade computacional. A inovação central do EfficientDet é sua estrutura de *backbone* eficiente, baseada na família de modelos EfficientNet (TAN; LE, 2019), e um mecanismo de fusão bidirecional de características, chamado BiFPN (*bi-directional feature pyramid network*). Este mecanismo permite uma agregação eficaz de informações de características em diferentes resoluções, melhorando a precisão da detecção (TAN; PANG; LE, 2020).

O BiFPN no EfficientDet melhora a forma como as características de diferentes escalas são fundidas e filtradas. Ao contrário do FPN tradicional, que combina características em uma única direção, o BiFPN permite múltiplas interações de ida e volta, garantindo que informações ricas em contexto de diferentes escalas sejam integradas de forma mais eficiente. Isso resulta em um desempenho melhorado, especialmente na detecção de objetos pequenos (TAN; PANG; LE, 2020).

Outro aspecto importante do EfficientDet é sua escalabilidade. O modelo pode ser sistematicamente dimensionado, ajustando a profundidade, a largura e a resolução das imagens, o que permite uma customização flexível para diferentes requisitos de desempenho e eficiência. Essa escalabilidade torna o EfficientDet adaptável a uma ampla gama de aplicações e dispositivos, desde dispositivos móveis de baixa potência até servidores de alto desempenho (TAN; PANG; LE, 2020).

2.6 Data Augmentation

Um problema comum na área de aprendizado de máquina é a dificuldade de se obter dados confiáveis para realizar o treinamento, sendo que muitas vezes o processo de

aquisição e processamento dos dados podem ser difíceis, complexos e custosos de serem realizados. Este problema se torna ainda maior no caso das CNNs, pois, atualmente, as CNNs mais eficazes são complexas e podem possuir milhões de parâmetro, exigindo assim uma grande quantidade de dados de treinamento (MIKOŁAJCZYK; GROCHOWSKI, 2018). Além disso, a falta de dados pode levar o conjunto de dados a possuir uma grande diferença entre a quantidade de observações em diferentes classes (classes desbalançadas), em que alguns casos impede o aprendizado e generalização dos métodos de ML.

Uma das maneiras de lidar com este problema é utilizar técnicas de aumento de dados, comumente chamada de *data augmentation*. Estas técnicas visam aplicar operações sobre os dados existentes para gerar novos dados. Ou seja, a partir uma única imagem é possível gerar diversas outras imagens (pertencentes a classe da imagem original). Levando assim, a um aumento na quantidade de dados disponíveis para treinamento. A Figura 10 exemplifica um processo simples de *data augmentation*, onde a partir de uma imagem são geradas varias outras.

Figura 10 – Exemplo de *data augmentation*, onde é aplicado operações sobre (a) uma imagem original, tais como: (b) reflexão, (c) rotação, (d) translação, (e) injeção de ruído, (f) alteração de escala, e (g) alteração de cor.

Existem diversas técnicas de *data augmentation* que podem ser aplicadas em imagens no contexto de métodos de ML, como por exemplo: transformações geométricas, transformações de espaço de cores, aplicação de filtros de kernel, mistura de imagens, apagamento aleatório, aumento de espaço de recursos, *adversarial training*, *GAN-based augmentation*, transferência de estilo, *meta-learning schemes*, entre outros (SHORTEN; KHOSHGOFTAAR, 2019). Dentre as técnicas mais utilizadas, pode-se destacar as abordagens baseadas em manipulações básicas de imagem, pois são facilmente aplicadas e podem levar a bons resultados. A seguir, são listadas algumas dessas manipulações:

- **Reflexão:** Inverter uma imagem no eixo horizontal ou eixo vertical (Exemplo de reflexão na vertical: Figura 10b);
- **Rotação:** Girar uma imagem no sentido horário ou anti-horário entre 1°e 359°(Exemplo de rotação em 90°anti-horário: Figura 10c);
- **Translação:** Deslocar uma imagens para a esquerda e/ou direita e/ou para cima e/ou para baixo (Exemplo de leve movimentação para direita: Figura 10d);

- **Injeção de ruído:** Injetar algum tipo de ruído na imagem, como por exemplo, inserção de valores aleatórios de pixeis e distorção (Exemplo de distorção: Figura 10e);
- **Escala:** Aplicar a operação de zoom in/out (aproximar/afastar) na imagem (Exemplo de zoom in: Figura 10f);
- **Alteração de cor:** Alterar padrões de cores, saturação, brilho, contrates e outras operações sobre os pixeis da imagem (Exemplo de alteração da tonalidade Figura 10g);

3 Revisão Sistemática

A baciloscopia do escarro ainda é a técnica mais utilizada em países em desenvolvimento, por ser um método simples e seguro, de baixo custo e de fácil manutenção, apesar de possuir baixa sensibilidade ao ser comparada com os testes TRM-TB e cultura (KHUTLANG et al., 2009)(PANICKER et al., 2018)(BRASIL, 2019). Devido a importância da baciloscopia no diagnóstico da TB pulmonar, muitos autores da literatura têm proposto utilizar técnicas de processamento de imagem para realizar o diagnóstico automatizado da TB de forma rápida, eficiente e precisa. Neste escopo, destacam-se as técnicas de Inteligência Artificial (IA), como Aprendizado de Máquina (ML, do inglês *Machine Learning*) e Aprendizado Profundo (DL, do inglês *Deep Learning*). Tais técnicas vêm ganhando destaque por possuírem alta capacidade de predição de doenças e por auxiliarem o profissional de saúde a realizar diagnósticos mais precisos.

O Aprendizado Profundo, sub-área da ML, é baseado no conceito de redes neurais artificiais, em que a informação é passada por meio de cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada. Tendo como grande vantagem a extração automática do mapa de características a partir de operações realizadas em camadas convolucionais. Comparado às técnicas tradicionais de visão computacional, o DL tem permitido avanços para obter maior precisão em tarefas como classificação de imagens e detecção de objetos (O'MAHONY et al., 2019). Tais avanços têm motivado diversos trabalhos a usarem DL para processar imagens da baciloscopia, sendo que diferentes estudos comparativos mostram que o DL supera técnicas tradicionais de ML (KANT; SRIVASTAVA, 2018)(EL-MELEGY et al., 2019)(CHITHRA; JAGATHEESWARI, 2020). No estudo realizado por Kotei e Thirunavukarasu (2022), os autores apontam DL como melhor técnica para criar sistemas eficazes para detecção de TB em imagens de esfregaço de escarro microscópico.

Desta forma, este capítulo tem como objetivo realizar uma revisão sistemática dos artigos encontrados na literatura que propõem técnicas de DL para auxiliar o profissional de saúde no diagnóstico da TB por meio da baciloscopia. Sendo assim, o foco é revisar artigos que utilizam DL para detectar/classificar imagens microscópicas de esfregaço de escarro usadas durante a baciloscopia. Para isso, as principais questões de pesquisa que pretende-se explorar são as seguintes: 1) Quais técnicas de DL estão sendo aplicadas para classificar bacilos da TB em imagens microscópicas de esfregaço de escarro (baciloscopia)? 2) Como essas técnicas são utilizadas? 3) Quais passos faltam para essas técnicas serem utilizadas no diagnóstico da TB na prática? As respostas destas perguntas podem contribuir com uma implicação importante para o aperfeiçoamento do uso da DL na baciloscopia, buscando torná-la mais ágil e precisa.

O presente capítulo é derivado no artigo científico intitulado *"A systematic review and repeatability study on the use of deep learning for classifying and detecting tuberculosis bacilli in microscopic images"* (CARVALHO et al., 2023), publicado na revista "Progress in Biophysics and Molecular Biology". A investigação veiculada nesse artigo integra, substancialmente, parte do trabalho desenvolvido ao longo desta tese de doutorado, proporcionando uma análise criteriosa e contextualizada do uso de técnicas de DL para a classificação e detecção de bacilos da tuberculose em imagens microscópicas. É importante destacar que o trabalho de Carvalho et al. (2023) discorre sobre um estudo de replicação que não é apresentado neste capítulo. Os resultados dessa replicação são, de certa forma, englobados pelo Capítulo 6.

3.1 Métodos

Esta seção apresenta os métodos, regras e critérios de pesquisa usados para recuperar os artigos relevantes para o estudo. Foram seguidos os procedimentos *Preferred Reporting Items for Systematic Reviews and Meta-Analyses* (PRISMA) (MOHER et al., 2009) para estabelecer os padrões de inclusão e exclusão. Os trabalhos recuperados estão dentro de um único domínio, a detecção automática de bacilos da TB usando técnicas de DL em imagens de esfregaços de escarro corados com *Ziehl-Neelsen* (ZN) obtidos usando um microscópio de campo claro.

A Figura 11 ilustra a estrutura PRISMA estabelecida para o processo de seleção de estudos.

3.1.1 Estratégia de pesquisa

Para garantir uma ampla busca de artigos relevantes, os artigos foram recuperados de várias bases de dados de publicações, como: Scopus (https://www.scopus.com/), IEEEXplore (https://ieeexplore.ieee.org/), Web of Science (www.webofscience.com), Science Direct (https://www.sciencedirect.com/), ACM digital library (https://dl.acm.org/), Springer Link (https://link.springer.com/), PubMed (https://www.ncbi.nlm.nih.gov/pubmed/) and Europe PMC (https://europepmc.org/). Em complemento, para ampliar a busca, também foram recuperados artigos a partir da ferramenta Google Scholar (https://scholar.google.com). As pesquisas foram realizadas nas nove bases de dados usando as seguintes palavras-chave de pesquisa: *tuberculosis*, *sputum*, e *deep learning*, publicados entre janeiro de 2000 e julho de 2022. É importante destacar que esta seleção temporal baseou-se em uma investigação preliminar de publicações relevantes, além de considerar a data em que esta revisão foi elaborada. Como as bases de dados possuem mecanismos e configurações de buscas diferentes, a parametrização de busca e palavras-chave utilizadas são apresentadas na Tabela 2. Devido a diversidade e quantidade de artigos encontrados na ferramenta Google Scholar,

apenas os 50 primeiros artigos apresentados na busca foram avaliados (ordenados pela relevância).

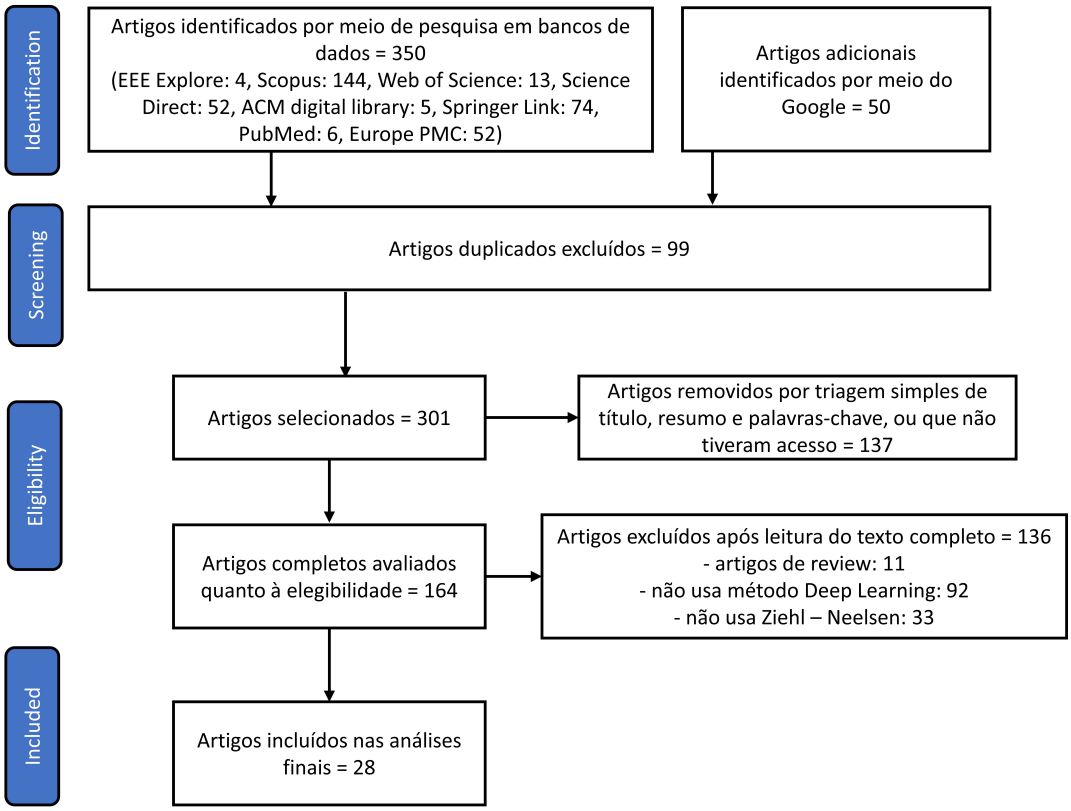


Figura 11 – A estrutura PRISMA para o processo de seleção dos estudos.

Tabela 2 – Base de dados de artigos que foram utilizados.

Base de dados	**Query de pesquisa**
IEEE Explore	(“All Metadata”:deep learning) AND (“All Metadata”:sputum) AND (“All Metadata”:tuberculosis)
Scopus	(ALL (deep AND learning) AND TITLE (tuberculosis) AND ALL (sputum))
Web of Science	deep learning (All Fields) AND sputum (All Fields) AND tuberculosis (Title)
Science Direct	deep learning sputum; and Title field = tuberculosis
ACM digital library	[All: deep learning] AND [Title: tuberculosis] AND [All: sputum]
Springer Link	deep AND learning AND “sputum”; and Title Contains field = tuberculosis
PubMed	((deep learning) AND (tuberculosis[Title]) AND (sputum))
Europe PMC	(“deep learning” AND TITLE:“tuberculosis” AND “sputum”))
Google Scholar	“deep learning” “tuberculosis” “sputum Smear”

3.1.2 Critério de seleção

Um total de 400 artigos foi recuperado a partir das bases de dados utilizadas. Após esta extensa busca, artigos duplicados foram identificados e removidos (99 artigos foram excluídos nesta etapa). Em seguida, títulos, resumos e palavras-chave foram examinados para realizar a primeira triagem, eliminando artigos fora do escopo deste estudo (132 artigos foram excluídos nesta etapa). Os artigos que se mostraram relevantes, e aqueles nos quais não foi possível identificar o foco do estudo na etapa anterior, permaneceram para uma segunda etapa de triagem, onde as seções sobre métodos e conclusões dos artigos foram analisadas (141 artigos foram excluídos). Após esta etapa, os textos remanescentes foram lidos na íntegra para total compreensão. Nesta última etapa, uma análise meticulosa foi realizada para verificar a relevância de cada estudo (28 artigos foram analisados).

Todos os artigos que satisfizeram as seguintes condições foram incluídos:

- Utilizaram pelo menos uma técnica de aprendizado profundo como classificador. Artigos que utilizaram apenas estratégias de segmentação de imagem ou técnicas clássicas de ML (ou seja, Redes Neurais, SVM, Random Forest, etc.) foram excluídos.
- Realizaram classificação e/ou detecção do bacilo da tuberculose em imagens de esfregaço de escarro coradas pelo ZN, obtidas usando um campo claro. Artigos em que técnicas de DL foram aplicadas em diferentes modalidades de imagem, tais como cultura, raio-x e RM, foram excluídos.
- Artigos publicados entre janeiro de 2000 e julho de 2022.
- Artigos completos escritos em inglês, publicados em periódicos, conferências ou capítulos com mais de quatro páginas.

3.1.3 Estudos selecionados

Como ilustrados na Figura 11, após as etapas de filtragem e análise meticulosa de cada artigo, os 28 artigos que atenderam aos critérios estabelecidos foram selecionados para esse estudo. Os artigos selecionados são detalhados nas Seções 3.2.4.1 e 3.2.4.2, e sumarizados nas Tabelas 3 e 4. Em complemento, a Seção 3.2 também apresenta conceitos importantes para compreender o procedimento de classificação e detecção de imagens no contexto de ML e DL.

3.2 Revisão da identificação do bacilo da Tuberculose por Aprendizado Profundo

3.2.1 Processo de classificação de imagem

Visão computacional (CV, do inglês *Computer vision*) é uma área da computação, na qual aplicações e transformações são realizadas em imagens, tais como detecção e extração de dados, reconhecimento e classificação de objetos, buscando alcançar interpretações, análises de formas e conclusões, a partir dos dados de imagem (FORSYTH; PONCE, 2002). Métodos de ML têm um papel importante na visão computacional, auxiliando na tarefa de reconhecimento de padrões e classificação/previsão de objetos. Tais métodos têm sido explorados para processar imagens médicas (i.e., esfregaço de escarro, raio-x, ressonância magnética) para o desenvolvimento de novas maneiras de auxiliar no diagnóstico da TB.

Observando os artigos que utilizam técnicas de DL selecionadas para este estudo, verificou-se que os autores adotam o processo básico de treinamento e teste de modelos de ML. O fluxo de trabalho descrito abaixo, apresentado na Figura 12, resume o processo geralmente adotado para construir modelos capazes de identificar o bacilo da TB em imagens de esfregaço de escarro, utilizando métodos de ML supervisionados:

- *Criação do conjunto de dados*: Após obter as imagens de esfregaço de escarro de TB usando um microscópio (Figura 12a), o processo de anotação é realizado, onde um ou mais especialistas identificam as posições dos bacilos na imagem (Figura 12b). Essas posições permitem a criação de um conjunto de dados contendo vários fragmentos de imagem (Figura 12c), que são utilizados para compor dois conjuntos de dados, treinamento e teste (Figura 12d). Nesta etapa, também é comum realizar o *data augmentation*, uma operação que expande o conjunto de dados realizando operações (ex.: rotação, recorte, espelhamento, entre outras) em imagens existentes para gerar novas imagens.

- *Treinamento*: A partir do conjunto de dados de treinamento, um método de DL é usado para extrair milhares de características relevantes (atributos) de fragmentos de imagem (Figura 12e) e para realizar o treinamento de um modelo de classificação (Figura 12f). O modelo obtido após o treinamento (Figura 12g) é submetido a uma avaliação de desempenho (Figura 12h), onde várias métricas são utilizadas para medir a performance de classificação correta de bacilos da TB. Essa avaliação pode ser realizada com o conjunto de dados de treinamento ou com um terceiro conjunto de dados (conjunto de validação). Se o modelo não apresentar uma boa generalização dos dados, obtendo uma taxa de acerto baixa, é necessário realizar um novo treinamento (Figura 12i). Nesse novo treinamento, ajustes de parâmetro e/ou

arquitetura do método de DL geralmente são realizados, buscando assim obter um melhor desempenho.

- *Teste*: Após o treinamento, o modelo é submetido a uma nova avaliação denominada teste (Figura 12j). Nessa etapa, cada imagem do conjunto de dados de teste (imagens desconhecidas pelo modelo, ou seja, independentes de treinamento) é submetida ao método de DL com o modelo treinado a fim de ser classificada. Ou seja, cada imagem receberá um rótulo de previsão, por exemplo, "TB bacillus"ou "no TB bacillus"(Figura 12m). Comparando o rótulo obtido na classificação, com o rótulo conhecido (real) em cada imagem classificada, é possível avaliar o desempenho do modelo em classificar novas imagens (Figura 12n). Para isso, a matriz de confusão é gerada e as métricas apresentadas na Seção 3.2.3 são calculadas. Se o modelo não atingir o desempenho esperado, novos treinamentos podem ser realizados, como ajustes de parâmetros e/ou arquitetura do método de DL (Figura 12o). Se os resultados obtidos durante o treinamento e teste forem satisfatórios, o modelo está pronto para prever novos fragmentos (Figura 12p).

- *Predição* (inferência): Até aqui, este fluxo de trabalho apresenta as etapas para treinar um modelo capaz de classificar fragmentos de imagem que contêm ou não contêm bacilos. No entanto, na prática, é interessante ter uma estratégia para encontrar os bacilos em uma imagem obtida sob o microscópio. Existem diferentes estratégias para fazer isso, uma das quais é utilizar métodos de visão computacional (como segmentação de imagem e/ou busca por regiões de interesse) para dividir a imagem e gerar pequenos fragmentos de imagem (Figura 12q e Figura 12r). Cada um dos fragmentos obtidos é submetido ao método de DL com o modelo treinado (Figura 12s) para ser classificado como "TB bacillus"ou "no TB bacillus"(Figura 12t). Por fim, os fragmentos classificados como "TB bacillus"são destacados na imagem que deu origem aos fragmentos (Figura 12u).

Entende-se que esse fluxo de trabalho é a base para estudos que usam ML ou DL, mas ainda assim, podem existir variações dependendo da pesquisa e dos métodos utilizados. Por exemplo, existem métodos de DL capazes de realizar as operações de localização e classificação simultaneamente. Esses métodos não usam fragmentos de imagem para treinamento, mas imagens inteiras junto com uma lista de anotações das posições de cada bacilo em cada imagem. Dessa forma, esses métodos são capazes de detectar as posições dos bacilos dentro de uma imagem obtida a partir do microscópio.

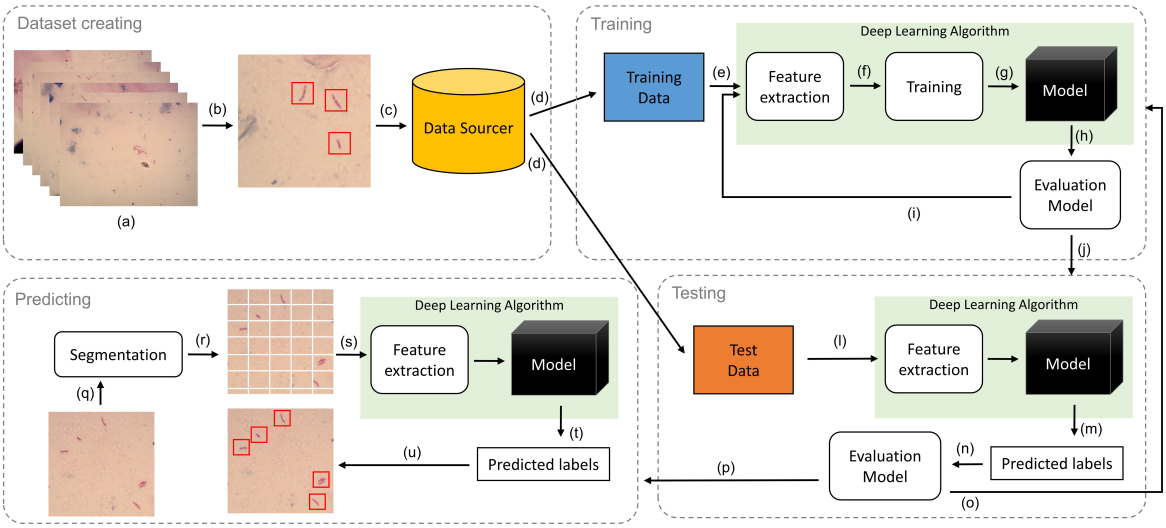


Figura 12 – Fluxo de trabalho básico de ML aplicado à classificação e localização do bacilo da TB usando DL e CV. Figura elaborada utilizando imagens contidas no conjunto de dados de Shah et al. (2017) .

3.2.1.1 Classificação de Imagem e Detecção de Objetos

Dentro do domínio da visão computacional, pode-se categorizar as técnicas de DL em: técnicas de classificação de imagem e técnicas de detecção/segmentação de objetos. Técnicas de classificação de imagem (ou classificação de objeto), como a Rede Neural Convolucional, são usadas para classificar um objeto específico em uma imagem. Esses tipos de métodos usam modelos com ajuste de peso para classificar um objeto em uma das n classes nas quais o modelo foi treinado. Ou seja, a partir de um treinamento realizado com um conjunto de dados contendo várias imagens de n classes, um modelo capaz de classificar uma nova imagem em uma das classes previstas é gerado. No DL, métodos como CNN são capazes de realizar extração de características para reconhecimento de padrões e classificação automaticamente, alcançando milhares de características avaliadas. Por exemplo, vários estudos treinaram uma CNN para classificar pequenos fragmentos de imagem em duas classes: contém bacilos da TB e não contém bacilo. A Figura 13a ilustra o processo de entrada e saída deste exemplo de classificação, onde cada uma das nove imagens de entrada é classificada/rotulada em uma de duas classes.

Diferente de realizar apenas a classificação de imagem, técnicas de detecção/ segmentação de objetos visam localizar (detectar) objetos em uma imagem dada e, além disso, definir (classificar) a qual categoria cada objeto pertence. A Figura 13b ilustra o resultado da detecção e classificação de bacilos da TB em um campo de visão sob o microscópio. Três passos básicos podem ser estabelecidos para construir um fluxo de trabalho tradicional de detecção de objetos: seleção de região de interesse, extração de características e classificação. A seleção da Região de Interesse (ROI, do inglês *Region of Interest*) geralmente é realizada por algoritmos de visão computacional, que podem

ser baseados em binarização, limiarização e técnicas de janela deslizante, entre outros. A extração de características e classificação podem seguir o fluxo já mostrado na Figura 12.

Por outro lado, existem técnicas de detecção DL capazes de realizar os três passos básicos deste fluxo de trabalho simultaneamente. Para isso, elas geralmente usam algumas estratégias de segmentação baseadas em regiões ou baseadas em grades de imagem, associadas a uma rede de extratores de características e classificação. Entre essas técnicas, destacam-se: YOLO (You Only Look Once) (REDMON et al., 2016b), R-CNN (Regions with Convolutional Neural Networks) (GIRSHICK et al., 2014), Fast R-CNN (Fast Region-based Convolutional Network) (GIRSHICK, 2015), Faster R-CNN (Faster Region-Based Convolutional Network) (REN et al., 2015), Mask R-CNN (HE et al., 2017), SSD (Single Shot Detector) (LIU et al., 2016), RetinaNet (LIN et al., 2017b), entre outros. Por exemplo, usando um desses métodos, a fonte de dados da Figura 12 poderia ser formada por imagens similares à Figura 12a (em vez de fragmentos) junto com uma lista de posições de cada bacilo em cada imagem. Além disso, os passos q, r, s, e t da Figura 12 podem ser combinados para serem realizados em uma única fase, detecção+classificação.

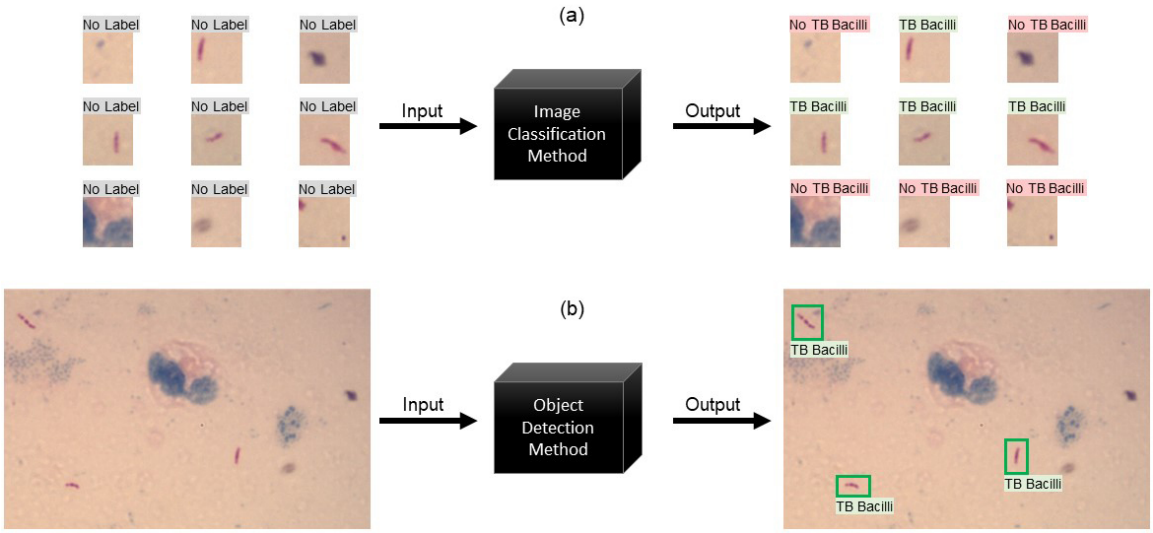


Figura 13 – (a) Exemplo de classificação de imagem onde cada objeto na imagem é classificado por um modelo de classificação em "TB Bacillus"(contém um bacilo da TB) ou "No TB Bacillus"(não contém um bacilo da TB). (b) Exemplo de um método de detecção de objetos localizando e classificando objetos pertencentes à classe "TB Bacillus". Figura feita a partir de imagens do conjunto de dados de Shah et al. (2017).

3.2.2 Conjuntos de Dados Públicos

Um dos desafios enfrentados na tarefa de classificação de imagens é a construção de conjuntos de dados representativos, sem ruído, que contenham padrões suficientes para permitir o aprendizado e a generalização dos métodos de ML. Esse desafio é ainda maior para as técnicas de DL, uma vez que essas técnicas requerem um grande número

de imagens para gerar modelos capazes de classificar imagens com bom desempenho (HALEVY; NORVIG; PEREIRA, 2009; SUN et al., 2017; SHORTEN; KHOSHGOFTAAR, 2019).

Os conjuntos de dados usados para treinar métodos de ML para classificar o bacilo da TB são formados por imagens obtidas por meio do microscópio. Geralmente, cada imagem é obtida capturando um campo de observação de uma lâmina (amostra). Em cada imagem, um ou mais especialistas realizam o processo de anotação, ou seja, identificam as posições dos bacilos (ou outros artefatos) na imagem. Essas posições fornecem pequenos fragmentos de imagem contendo o bacilo da TB, que são usados para extrair características e padrões que permitem que o processo de aprendizado ocorra. A Figura 14 ilustra o resultado deste processo de identificação de bacilos em imagens de diferentes bancos de dados.

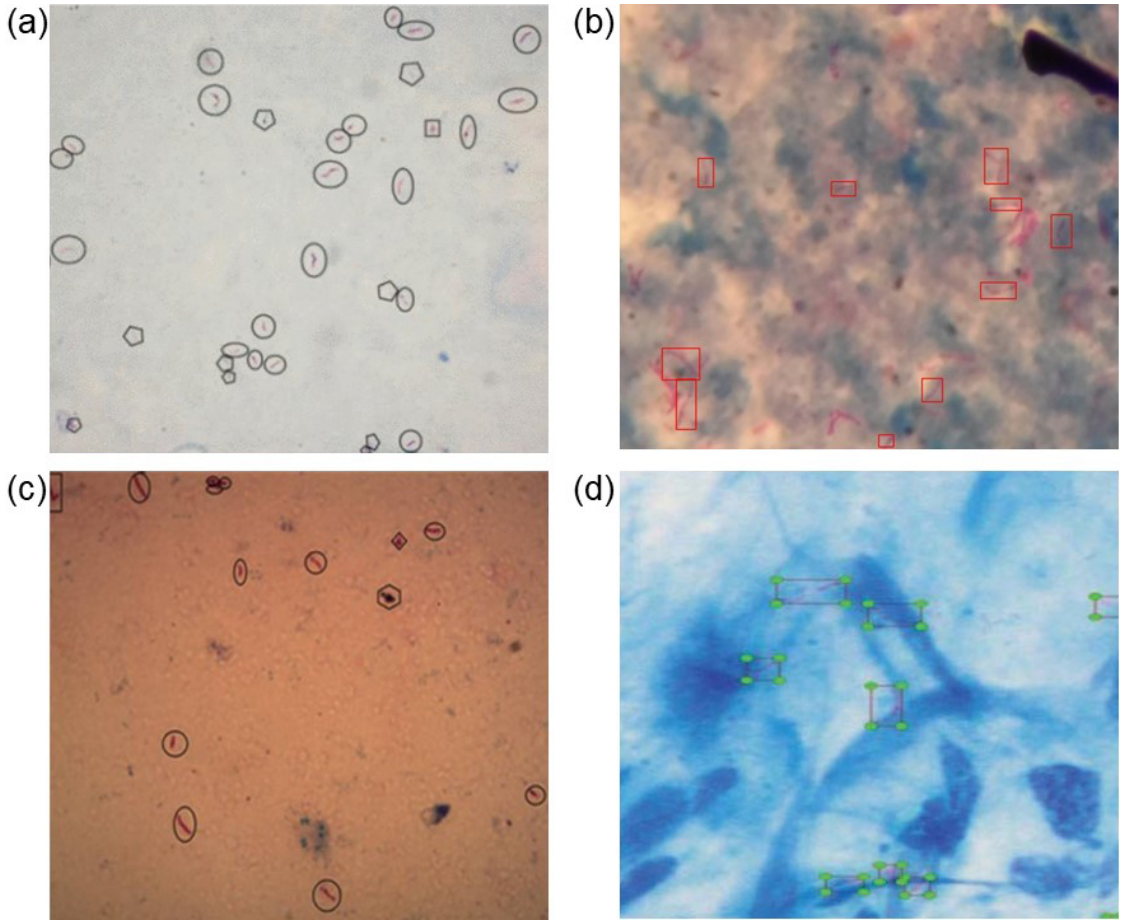


Figura 14 – Exemplo de anotação de bacilos nos conjuntos de dados propostos por (a) Costa et al. (2014), (b) Quinn et al. (2016), (c) Shah et al. (2017), e (d) Trilaksana et al. (2020) (as imagens foram extraídas do conjunto de dados original ou artigos e redimensionadas para melhor visualização).

Nos artigos selecionados neste estudo, identificou-se uma grande diferença nas quantidades de imagens dos conjuntos de dados que os autores usaram. Em geral, os conjuntos de dados descritos variaram de dezenas a milhares de imagens, e na maioria dos

artigos, a anotação foi realizada por mais de um especialista. É interessante observar o processo e o número de especialistas (patologistas) envolvidos na construção do conjunto de dados, pois isso pode aumentar a confiabilidade dos dados e, consequentemente, melhorar o desempenho das técnicas de DL. As Tabelas 3 e 4 apresentam mais detalhes sobre os conjuntos de dados utilizados em cada artigo.

Na maioria dos artigos, os conjuntos de dados utilizados são privados, ou seja, não são disponibilizados para outros pesquisadores. No entanto, identificou-se nesta revisão quatro conjuntos de dados que os autores propõem disponibilizar as imagens e suas anotações, são eles: Costa et al. (2014), Quinn et al. (2016), Shah et al. (2017), e Trilaksana et al. (2020). A Figura 14 apresenta alguns exemplos de imagens anotadas em cada um desses conjuntos de dados.

Costa et al. (2014) propôs dois conjuntos de dados de imagens obtidas de esfregaços de escarro. O primeiro contém 1200 imagens em diferentes focos sem anotação e o segundo contém 120 imagens anotadas. Conforme apresentado pelos autores, o segundo conjunto de dados é recomendado para ser usado no treinamento de métodos de ML a fim de classificar o bacilo da TB. Onde cada uma das 120 imagens foi analisada por um especialista, que identificou/registrou as posições dos bacilos. Durante essa anotação, foram registrados fragmentos contendo (Figura 14a): bacilo verdadeiro (círculo), bacilo duvidoso (polígono), e bacilos agrupados (retângulo). Este conjunto de dados consiste em imagens coletadas de 10 campos em 12 lâminas de diferentes pacientes, usando uma câmera de 10 megapixels acoplada a um microscópio. Para maior diversidade, o conjunto de dados foi formado por imagens categorizadas em dois grupos, que têm baixa e alta densidade de fundo (contendo uma forte presença de contracoloração com solução de azul de metileno no fundo). As imagens desses dois grupos podem ser categorizadas como imagens que têm alta e baixa densidade de bacilos. Conforme relatado pelos autores, o conjunto de dados está disponível em: http://www.tbimages.ufam.edu.br/.

No trabalho desenvolvido por Quinn et al. (2016), três bases de dados foram propostas que podem ser usadas para treinar métodos de ML. Cada base de dados é composta por imagens usadas para diagnosticar malária, tuberculose e ovos de parasitas intestinais. A base de dados formada por imagens de tuberculose foi obtida capturando 928 imagens de amostras frescas de escarro tingidas com ZN, observadas sob um microscópio com magnificação de x1000. Essas imagens foram capturadas usando um smartphone acoplado ao microscópio com um suporte criado em uma impressora 3D. Segundo os autores, diferentes especialistas analisaram cada imagem e anotaram um total de 3734 objetos, identificados com caixas delimitadoras (conforme ilustrado na Figura 14b). Conforme relatado pelos autores, o conjunto de dados está disponível em: http://air.ug/microscopy.

Para colaborar na criação de métodos de ML que possam ajudar a melhorar a sensibilidade da detecção manual por microscopia manchada por *Ziehl–Neelsen*, Shah et al.

(2017) propôs o *Ziehl–Neelsen Sputum Smear Microscopy Image Database* (ZNSM-iDB). O ZNSM-iDB tem milhares de imagens obtidas de 10 lâminas diferentes, sendo capturadas por três modelos diferentes de microscópios/câmeras. As imagens neste conjunto de dados estão organizadas em sete categorias, e as imagens na categoria "*Manually segmented bacilli in a viewfield*"podem ser usadas para treinar métodos de ML. Em cada imagem desta categoria, os seguintes fragmentos foram identificados manualmente (Figura 14d): um único bacilo (círculo ou oval), bacilos ocluídos (quadrado ou retângulo), estruturas vermelhas não classificadas (diamante) e artefatos (hexágono). Considerando as imagens obtidas nos três microscópios, um total de 300 imagens foram anotadas manualmente no ZNSM-iDB. Conforme relatado pelos autores, o conjunto de dados está disponível em: http://14.139.240.55/znsm.

Trilaksana et al. (2020) também propôs um conjunto de dados consistindo de imagens de esfregaços de escarro. De acordo com os autores, este conjunto de dados visa auxiliar pesquisadores na tarefa de detectar bacilos de TB usando aprendizado profundo. O conjunto de dados proposto contém 350 imagens obtidas de imagens de lâminas de esfregaço de escarro tingidas com TB positiva. Para garantir uma maior variabilidade, as imagens no banco de dados são compostas por esfregaços de escarro normais e super coloridos. A magnificação usada no microscópio é de 1000 magnificações. O processo de anotação foi realizado por um especialista, em que durante este processo, 3 classes de objetos foram anotadas, que são: *TBBacteria*, *NonTBbacteria* e *Stainresidues*. Ao final da análise de todas as imagens, foi obtido um total de 3102 bactérias de TB com variações de forma e intensidade de cor. Os autores não fornecem um link direto para baixar ou solicitar dados no artigo.

3.2.3 Estratégia e métricas para avaliação dos modelos

Existem duas estratégias clássicas de ML para avaliar modelos de classificação gerados por métodos de aprendizado supervisionado: (i) realizar treinamento, validação e teste usando conjuntos de dados independentes e/ou (ii) realizar validação cruzada (ARLOT; CELISSE et al., 2010). Essas técnicas são amplamente utilizadas, e todos os artigos analisados neste trabalho usam pelo menos uma dessas duas técnicas. Essas técnicas são descritas abaixo:

- Treinamento/Teste: Nesta estratégia, são realizadas duas etapas, na etapa de treinamento usa-se um método de ML para gerar um modelo preditivo e na etapa de teste o modelo gerado é avaliado usando diferentes métricas (por exemplo, acurácia, precisão, sensibilidade, entre outros). Para isso, o conjunto de dados pode ser dividido em três subconjuntos (aleatoriamente): conjunto de treinamento, conjunto de validação e conjunto de teste. Durante a etapa de treinamento, o processo de aprendizado e

criação do modelo é realizado a partir dos dados do conjunto de treinamento (WANG et al., 2020). Nesta mesma etapa, os dados de validação podem ser usados para avaliar e ajustar diferentes pesos e hiperparâmetros do modelo. Após o término da etapa de treinamento e com o modelo criado, o conjunto de teste é usado para avaliar o desempenho de classificação, verificando assim a capacidade de generalização do modelo e seu desempenho na classificação de padrões não vistos no treinamento (WANG et al., 2020). Em geral, os dados são divididos em 70%, 10% e 20%, para treinamento, validação e teste, respectivamente. A Figura 15a ilustra essa separação de dados.

- Validação Cruzada (*k-fold cross-validation*): nesta técnica, o conjunto de dados é dividido em k subconjuntos de tamanho igual (*k-fold*), e em cada iteração, um *fold* é usado como dados de teste e os outros *folds* $(k-1)$ são usados como dados de treinamento (NTI; NYARKO-BOATENG; ANING, 2021). A avaliação de desempenho de um método de ML é realizada a partir das médias calculadas de cada métrica observada (por exemplo, acurácia, precisão, sensibilidade, entre outros) durante cada uma das k repetições (treinamento/teste) realizadas. A maioria dos trabalhos de ML na literatura que usa essa estratégia adota um k=10 (NTI; NYARKO-BOATENG; ANING, 2021; REFAEILZADEH; TANG; LIU, 2009). No entanto, dependendo do tamanho do conjunto de dados ou do método de ML escolhido, pode ter um alto custo computacional e longo tempo de execução, e pode ser necessário adotar valores mais baixos para k ou outra estratégia de avaliação. A Figura 15b ilustra essa separação de dados. A maioria dos trabalhos que usam DL (inclusive os selecionados para este estudo) adota a primeira estratégia em vez da validação cruzada *k-fold* devido ao custo computacional de realizar múltiplas repetições.

Existem várias maneiras de avaliar o desempenho dos métodos de ML, onde as métricas de avaliação de desempenho comumente aceitas como medidas da qualidade de classificação são construídas a partir de uma matriz de confusão (SOKOLOVA; JAPKOWICZ; SZPAKOWICZ, 2006). Nesta matriz, o número de exemplos corretamente e incorretamente reconhecidos para cada classe é registrado. A Figura 16 apresenta uma matriz de confusão para classificação binária, na qual: Verdadeiro Positivo (VP ou TP, do inglês *True Positive*) indica o número de exemplos da classe positiva corretamente classificados como classe positiva; Verdadeiro Negativo (VN ou TN, do inglês *True Negative*) indica o número de exemplos da classe negativa corretamente classificados como classe negativa; Falso Positivo (FP) indica o número de exemplos da classe negativa que foram erroneamente classificados como classe positiva; Falso Negativo (FN) indica o número de exemplos da classe positiva que foram erroneamente classificados como classe negativa.

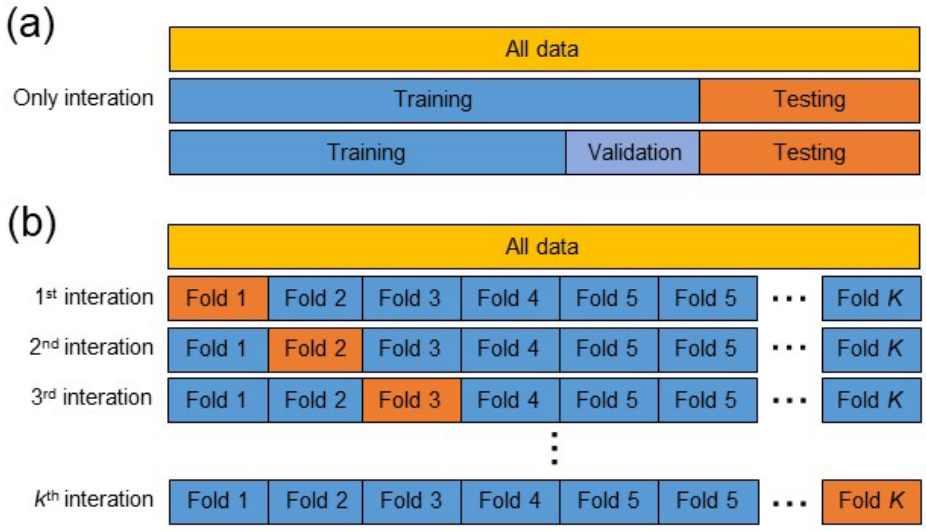


Figura 15 – Ilustração de como os dados podem ser separados para realizar o processo de treinamento e teste para avaliação de modelos gerados por métodos de ML. (a) Dividindo os dados em treinamento, validação e teste usando subconjuntos de dados independentes. (b) Validação cruzada *k-fold*, divisão dos dados em k subgrupos contendo um fold para teste e $k-1$ *folds* para treinamento.

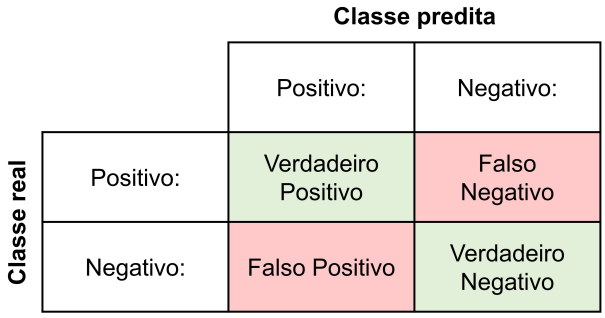


Figura 16 – Ilustração da matriz de confusão para problemas de classificação de duas classes (positiva ou negativa).

A partir da matriz de confusão, é possível obter as seguintes métricas que são amplamente utilizadas para avaliar o desempenho dos métodos de ML:

- Acurácia: Permite calcular a proporção de respostas corretas, por exemplo, um modelo que classifica corretamente a classe de 91 exemplos de um total de 100 exemplos terá uma acurácia de 0,91 ou 91%. Em problemas com classes desbalanceadas, o valor da acurácia pode levar a uma conclusão errada sobre o modelo, pois pode causar uma superestimação do desempenho. A acurácia pode ser calculada pela Equação 3.1.

$$Acurácia = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

- Sensibilidade: Também conhecida como *recall*, revocação ou Taxa de Verdadeiros Positivos, permite obter a proporção de verdadeiros positivos, ou seja, a capacidade

do modelo de classificar corretamente os exemplos da classe positiva. A sensibilidade pode ser calculada pela Equação 3.2.

$$Sensibilidade = \frac{TP}{TP + FN} \tag{3.2}$$

- Especificidade: Também conhecida como Taxa de Verdadeiros Negativos, permite obter a proporção de verdadeiros negativos, ou seja, a capacidade do modelo de classificar corretamente os exemplos da classe negativa. A especificidade pode ser calculada pela Equação 3.3.

$$Especificidade = \frac{TN}{TN + FP} \tag{3.3}$$

- Precisão: Calcula a proporção de amostras que foram corretamente classificadas como positivas. A precisão pode ser calculada pela Equação 3.4.

$$Precisão = \frac{TP}{TP + FP} \tag{3.4}$$

- F-*measure*: Também conhecida como F1-Score, pode ser interpretada como uma média ponderada que combina precisão e recall, onde valores altos de F-*measure* mostram um baixo número de falsos positivos e um baixo número de falsos negativos. A F-*measure* pode ser calculada pela Equação 3.5.

$$F - measure = 2 * \frac{Precisão * Recall}{Precisão + Recall} \tag{3.5}$$

Além destas métricas, a Área Sob a Curva ROC (AUC, do inglês *Area Under the ROC Curve*) também é utilizada. A métrica AUC permite avaliar o quanto o modelo é capaz de diferenciar as classes do problema. Esta é uma medida amplamente recomendada e utilizada para avaliar modelos de ML, e estudos como Bradley (1997), Ling, Huang e Zhang (2003), Huang e Ling (2005) mostram formalmente como AUC pode contribuir para uma avaliação mais rigorosa do que algumas outras métricas.

As métricas apresentadas acima são amplamente utilizadas para avaliar o desempenho de métodos de DL, visando classificar a classe de um dado objeto como certa ou errada. Por exemplo, classificar corretamente um fragmento de imagem que contém ou não um bacilo de TB. Contudo, existem outras duas métricas que auxiliam na tarefa de avaliar técnicas de DL que realizam detecção de objetos, que são: *Intersection over Union* (IoU) e *mean Average Precision* (mAP). Essas métricas possibilitam avaliar, por exemplo, as técnicas que detectam os bacilos em uma imagem obtida de um campo de leitura de microscopia de esfregaço de escarro. Uma breve descrição destas métricas é apresentada a seguir:

- IoU: Baseada no Índice de Jaccard (coeficiente de similaridade para dois conjuntos de dados), a IoU calcula a área de sobreposição entre a caixa delimitadora predita (B_p) e a caixa delimitadora verdadeira (caixa delimitadora *ground-truth*, B_{gt}), dividida pela área de união entre as duas caixas delimitadoras (PADILLA; NETTO; SILVA, 2020), conforme mostrado na Equação 3.6 e ilustrado pela Figura 17a. Se B_p está completamente fora da área de B_{gt}, temos $IoU = 0$. Se a sobreposição é total, tem-se o valor de $IoU = 1$. Assim, quanto maior a sobreposição, maior o valor de IoU. Na tarefa de detecção de objetos, o valor de IoU pode ser utilizado como um limiar (t) para classificar se a predição realizada é um verdadeiro positivo ou um falso positivo (onde, $IoU \geq t$ a detecção é considerada correta, $IoU < t$ a detecção é considerada incorreta). Para ilustrar este conceito, a Figura 17b apresenta três caixas delimitadoras verdes indicando as posições dos bacilos de TB (B_{gt}) e quatro caixas delimitadoras vermelhas representando o resultado de uma técnica de DL para a detecção de objetos (B_p). Neste exemplo, adotamos $t = 0.5$, portanto podemos considerar que a técnica de DL teve uma classificação correta e três classificações erradas.

$$J(B_p, B_{gt}) = IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{3.6}$$

- *Average Precision*: é uma métrica importante que permite avaliar o trade-off entre precisão e *recall* (sensibilidade) para diferentes valores de confiança associados às caixas delimitadoras geradas por uma detecção de objetos (PADILLA; NETTO; SILVA, 2020). Essa métrica é adotada na maioria das competições como uma das métricas para avaliar detecções e ranquear equipes. O valor de AP é obtido pela área sob a curva precisão x *recall* (PR Curve), gerada pelas detecções realizadas em cada classe. Existem diferentes formas de calcular a área sob a PR Curve, como a interpolação de 11 pontos AP e todos os pontos de interpolação AP. A interpolação de 11 pontos foi escolhida para ser utilizada no PASCAL VOC Challenge (EVERINGHAM et al., 2015; PARK; KIM, 2020). Além disso, para medir o desempenho de modelos multi-classe, ou seja, modelos capazes de detectar objetos de duas ou mais classes diferentes, pode-se utilizar a métrica média AP (mAP). O mAP é basicamente obtido a partir da média dos APs calculados para cada classe. O trabalho de Padilla, Netto e Silva (2020) detalha e exemplifica o cálculo de AP e mAP.

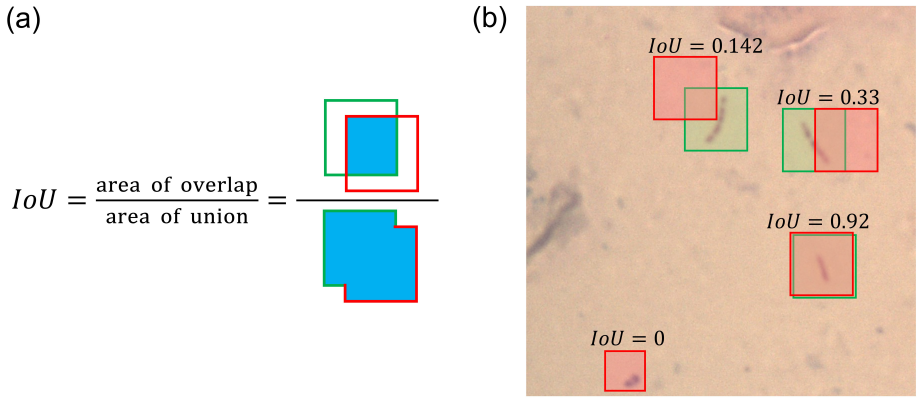


Figura 17 – (a) Representação gráfica do cálculo da Interseção sobre União (IoU). (b) Exemplo do resultado do cálculo de IoU considerando B_{gt} (caixa colorida de verde) e B_p (caixa colorida de vermelho) (preparado utilizando uma imagem do conjunto de dados de Shah et al. (2017)).

3.2.4 Técnicas de Deep Learning

Nesta seção, é apresentado os artigos encontrados durante a revisão da literatura, realizada em nove bases de dados, seguindo os procedimentos estabelecidos por PRISMA.

Os artigos foram agrupados em duas subseções para melhor compreensão. A Seção 3.2.4.1 apresenta artigos que propuseram técnicas de classificação para bacilos de TB em fragmentos de imagem. A Seção 3.2.4.2 apresenta os artigos que propuseram técnicas para detectar bacilos de TB em imagens de um campo de observação de microscópio. Todos os artigos apresentados estão no domínio de uso de técnicas de DL para processar imagens obtidas em microscopia de esfregaço de escarro usando o método ZN.

3.2.4.1 Técnicas de classificação

Zurac et al. (2022) propuseram um método automático para identificar bacilos álcool-ácido resistentes (BAAR) usando CNN, que são treinadas para processar imagens de lâminas inteiras (WSIs) e indicar a localização dos BAAR. O classificador baseado em caminhos para detecção de BAAR é baseado na arquitetura RegNetX4. O conjunto de dados de treinamento utilizado tem 510 WSIs (110 positivas e 400 negativas), e mais de 260.000 *patches* positivos e mais de 700 milhões de *patches* negativos de 64 × 64 pixels foram selecionados. O conjunto de dados de validação tem 60 WSIs (23 positivas e 37 negativas), totalizando 570 WSIs. As WSIs foram escaneadas com Aperio GT450, ampliação de 40x. Neste experimento, o método baseado em IA obteve resultados melhores em comparação aos patologistas. Quando o patologista realizou o diagnóstico com assistência da IA, ele obteve resultados ainda melhores (superiores aos da IA sozinha). O teste clínico mostrou 98,33% de acurácia, 95,65% de sensibilidade e 100% de especificidade para o método assistido por IA.

Momeny et al. (2022) propuseram um método para classificar imagens de TB usando CNN generalizada. A CNN foi aprimorada com base no método Greedy AutoAugment (GAA), mistura de *pooling*, *dropout* e normalização de lote. Os autores propuseram uma arquitetura de CNN baseada em experimentos realizados usando cinco configurações. A Configuração 1 consiste em cinco camadas de convolução, sete camadas de ativação ReLU, três camadas de *pooling* e três camadas totalmente conectadas. O modelo de CNN da Configuração 1 foi aprimorado usando normalização de lote (Configuração 2), *pooling* misto (Configuração 3), ativação paramétrica ReLU - PReLU (Configuração 4) e GAA e *dropout* (Configuração 5). O conjunto de dados com 1547 imagens foi dividido em um conjunto de treinamento (70%), conjunto de validação (10%) e conjunto de teste (20%). Após o uso do método GAA para aumentar o número de imagens, obteve-se 7.546 instâncias negativas e 3.283 instâncias positivas no conjunto de dados de treinamento. O modelo mostra os melhores resultados em todas as métricas, sensibilidade (85%), especificidade (97%), precisão (92%), e F-*Measure* (89%).

Udegova, Shelomentseva e Chentsov (2021) realizou uma otimização da arquitetura CNN para detecção de bacilo de TB. Os autores construíram um algoritmo experimental para otimizar o modelo original de CNN, incluindo a otimização da dimensão do modelo, aumento de dados, ajuste dos parâmetros do modelo e melhoria da regularização. Os experimentos foram realizados em 400 imagens. A melhor arquitetura mostrou a precisão com o menor erro de teste em comparação com outros modelos experimentais: sensibilidade (0,9858), especificidade (0,9843), precisão (0,9850) e loss (0,445).

Muyama, Nakatumba-Nabende e Mudali (2021) aplicou modelos de DL baseados em duas CNNs pré-treinadas: VGGNet e GoogLeNet Inception V3 para diagnosticar TB a partir de 148 imagens microscópicas de esfregaço de escarro coradas por ZN de dois conjuntos de dados diferentes (Quinn et al. (2016) e ZNSM-iDB de Shah et al. (2017)). Os resultados obtidos sem usar aumento de dados são para a CNN VGGNet: acurácia (0,800), precisão (0,795) e sensibilidade (0,808), e para a CNN GoogLeNet Inception v3: acurácia (0,722), precisão (0,754) e sensibilidade (0,808).

Ibrahim et al. (2021) propuseram um método de detecção automatizado usando a CNN AlexNet pré-treinada em 3 conjuntos de experimentos A, B e C. Eles compararam a detecção de TB usando os modelos de CNN AlexNet com o resultado real fornecido pelo microbiologista e analisaram inconsistências entre modelos de rede e humanos. Os conjuntos de dados são divididos em 70% para treinamento e 30% para teste, e no experimento A: 124 imagens para treinamento e 54 para testes; experimento B: 524 imagens para treinamento e 371 para teste; experimento C: 530 imagens para treinamento e 159 para teste. Os resultados obtidos foram no experimento A: 98,15% de acurácia, 96,77% de sensibilidade e 100% de especificidade; experimento B: 98,09% de acurácia, 98,59% de sensibilidade e 97,67% de especificidade; experimento C: 98,73% de acurácia, 98,59%

de sensibilidade e 98,84% de especificidade. O resultado do experimento C superou o microbiologista empregado em termos de precisão, sensibilidade e especificidade.

Rachmad, Chamidah e Rulaningtyas (2020) desenvolveu um sistema de classificação de TB usando a arquitetura CNN ResNet-101, combinando Máquina de Vetor de Suporte (SVM, do inglês *support-vector machine*) como classificador, para ajudar patologistas clínicos a completarem sua tarefa de identificar bactérias TB e produzir os melhores resultados de classificação. O experimento foi realizado em 1266 fragmentos de imagem, consistindo em 633 imagens de bactérias TB e 633 imagens de bactérias não-TB. As imagens foram obtidas do conjunto de dados Shah et al. (2017). Os resultados são uma acurácia de 97,6%, uma precisão de 97,9%, uma sensibilidade de 97,4% e F1-Score de 97,6%.

Samuel e Kanna (2019) propuseram um sistema de detecção de TB composto por dois subsistemas: (i) um sistema de aquisição de dados e (ii) um sistema de reconhecimento. No sistema de aquisição de dados, uma estrutura com um microscópio motorizado é projetada e desenvolvida para automatizar a aquisição de todos os campos de visão (FOVs). Após a aquisição de todos os FOVs, os dados são passados para o sistema de reconhecimento. No sistema de reconhecimento, o método de *transfer learning* é implementado customizando o modelo Inception V3 DeepNet. Este modelo aprende com os pesos pré-treinados do Inception V3 e classifica os dados usando SVM a partir do conhecimento transferido. Eles usaram o conjunto de dados ZNSM-iDB e seu próprio conjunto de dados digital microscópico adquirido. Um total de 1242 imagens foram obtidas (620 são bacilos e 622 são não-bacilos). Usando validação cruzada de 5-fold, seu modelo atinge uma acurácia de 95,05% e uma AUC de 95,05%.

López et al. (2017) apresentaram um método para a classificação automática de manchas de microscopia de campo claro usando versões RGB, R-G e em escala de cinza como entradas de um modelo CNN para identificar TB. Três diferentes arquiteturas de CNN foram implementadas, gerando 3 modelos diferentes. Eles usaram o conjunto de dados de Costa et al. (2014), aplicando um aumento de dados, resultando em 29.310 *patches* (50% dos *patches* com bacilo e os outros 50% sem bacilo). Nos testes de classificação de *patches*, cada modelo alcançou uma AUC superior a 96% e o melhor resultado foi alcançado usando a versão de entrada R-G, o Modelo 3 que alcançou uma AUC de 99%. Além disso, na versão de entrada R-G, todos os três modelos alcançaram uma acurácia superior a 97%.

Quinn et al. (2016) avaliaram o desempenho de CNNs profundas em três diferentes tarefas de microscopia: diagnóstico de malária em esfregaços de sangue espesso, TB em amostras de esfregaço de escarro e ovos de parasitas intestinais em amostras de fezes. Entre essas análises, as imagens de TB foram obtidas a partir de escarro e coradas com o corante ZN. O conjunto de dados consiste em 3734 objetos em 928 imagens, dividindo em

50% para o conjunto de treinamento e 50% para o conjunto de teste. Os resultados da classificação do bacilo TB foram de 99% para AUC e 93% para AP.

3.2.4.2 Técnicas de Segmentação/Classificação

Angayarkanni et al. (2022) propuseram uma nova arquitetura DL baseada em pixel para geração de máscaras (SegZNet) e para segmentação, as imagens pré-processadas e as máscaras correspondentes são alimentadas na arquitetura UNet. Eles utilizam dois datasets: (i) *Open-Source dataset* disponível in Kaggle, que tem 1265 imagens positivas; e (ii) dataset de Ma et al. (2017), que tem 1735 imagens. Utilizando métodos de *data augmentation* obtiveram um conjunto de 4000 amostras, divididos em conjunto de treinamento com 3040 imagens, de validação com 800 imagens e teste limitado a 100 imagens do dataset 2. Para a classe positiva os autores apresentaram 98,5%, 91,53% e 100% para a acurácia, precisão e sensibilidade, respectivamente, com IOU de 0.8728.

Vanitha, Angayarkanni e Sathishkumar (2021) analisaram e avaliaram diversas técnicas de pré-processamento e segmentação para determinar as melhores técnicas para a segmentação de bacilos. Para a comparação de técnicas de segmentação, eles apresentam algoritmos clássicos como VIZ, *Threshold Otsu*, *Canny Edge*, Grafo de Felsenszwalb, *Watershed*, e algoritmos supervisionados como o UNet. Eles usam dois conjuntos de dados: (1) conjunto de dados de código aberto disponível no Kaggle, que tem 1265 imagens positivas; e (2) o conjunto de dados de Panicker et al. (2018), que tem 1735 imagens. Utilizando métodos de aumento de dados, ele obteve um conjunto de 4000 amostras, dividido em um conjunto de treinamento: de 3040 imagens, um conjunto de validação: de 800 imagens, e um conjunto de teste limitado a 100 imagens do conjunto de dados 2. Entre todos os experimentos realizados, concluiu-se que o UNet apresentou o melhor desempenho, obtendo uma acurácia de 0,99, um Dice Score de 0,988, uma sensibilidade de 0,99, um PPV de 0,98 e um índice de Jaccard de 0,985.

Reist et al. (2021) propuseram um método para detecção automatizada do bacilo da TB usando a arquitetura RetinaNet. Esta arquitetura consiste numa rede base chamada *Feature Pyramid Network*. Os autores selecionaram o *backbone* ResNet-152 porque apresentou melhor desempenho. Um conjunto de 910 imagens disponíveis no Kaggle foi usado para avaliar este método, no qual os dados foram separados aleatoriamente em um conjunto de treinamento (70%), conjunto de validação (10%) e conjunto de teste (20%). Os resultados alcançados são uma acurácia de 67,1%, uma sensibilidade de 86,56% e uma F-score de 75,61%. Durante o treinamento, foi observado um mAP máximo de 0,795.

Winarno, Bilhaq e Agoes (2021) propuseram uma arquitetura melhorada do *framework Deeply Supervised Object Detector* (DSOD) para a detecção de bacilos de TB. A arquitetura DSOD é um método DL para detecção de objetos baseado na estrutura

livre de região proposta. Essa abordagem permite que o treinamento convirja sem usar o modelo de pré-treinamento para iniciação. Eles usaram o conjunto de dados TBDB de Trilaksana et al. (2020), com imagens adicionais retiradas do conjunto de dados de Shah et al. (2017). Um total de 600 imagens foram preparadas para o conjunto de dados, divididas em 90 imagens para teste e 510 imagens para treinamento e validação. Os experimentos foram realizados usando a plataforma Caffe, tendo como resultado um AP de 72,32%.

Chithra e Jagatheeswari (2020) propõem uma técnica, denominada *adaptive fractional crow* (AFC-Deep CNN), para detecção de gravidade e identificação do nível de infecção usando imagens de esfregaço de escarro. A técnica proposta segue cinco etapas, que envolvem pré-processamento, segmentação, extração de características, redução de dimensionalidade e classificação, usando *fractional crow search-based deep* CNN (FC-SVNN), para o diagnóstico de TB. Eles usaram o conjunto de dados ZNSM-iDB com 75 imagens, onde 50 imagens são bacilos e 25 são imagens não bacilos. No experimento de *severity analysis*, o método proposto obteve melhores resultados que técnicas como SVM, rede neural, e outras. Os resultados baseados na validação cruzada, valores de 0,935, 0,92 e 0,08 foram obtidos para acurácia, TPR e FPR, respectivamente.

Trilaksana et al. (2020) propõem um banco de dados de imagens de esfregaço de escarro com dados coletados por especialistas indonésios. Este banco de dados contém 3 classes, que são: TBBacteria, NonTBbacteria e Stainresidues e contém 350 imagens com 3.102 bactérias de TB com variação de forma e intensidade de cor. As imagens foram dividias em 276 imagens para treinamento e validação e 72 imagens para teste. Além disso, eles propõem um método DL para detecção de bacilos da TB utilizando a arquitetura PeleeNet. Os experimentos foram realizados na plataforma Caffe. Os resultados do mAP para cada categoria são: TBBacteria fornece acurácia de 26,87%, NonTNBacteria 26,82%, e Stainresidues não é detectado.

Swetha, Sankaragomathi e Thangamalar (2020) realizam o pré-processamento com técnicas de redução de ruídos e modificação de intensidade, utilizam a segmentação e uma CNN para classificar a presença do bacilo da TB como leve, moderado e grave, em imagens de escarro baseadas no método ZN. Para segmentação, são utilizados métodos como *Channel Area Thresholding* (CAT) que é o método de *multilevel-thresholding*. Para classificação, é utilizada uma CNN combinada aos extratores de características *Histogram of Oriented Gradients* (HOG) e *Speed up Robust Feature* (SURF). O conjunto de dados é formado por trinta imagens de escarro com condições únicas de TB como TB leve, moderada e grave. O classificador CNN proposto obteve desempenho de 99,5%, 94,7% e 99%, de acurácia, sensibilidade e especificidade, respectivamente.

Serrão et al. (2020a) propõem um método para realizar a segmentação e identificação de bacilos, associando modelos CNNs e uma abordagem de imagem em mosaico. Para avaliação usaram um conjunto de dados de imagens validado por três especialistas.

O banco de dados consistiu em 5.000 imagens em mosaico, sendo 60% para o conjunto de treinamento, 20% para o conjunto de validação e 20% para o conjunto de teste. Foram avaliadas três arquiteturas CNNs e três métodos de otimização em cada arquitetura. A melhor arquitetura CNN apresentou os seguintes valores de acurácia: 99,6%, precisão: 99,3%, sensibilidade: 99,9%, F1-*score*: 99,6%.

Yang et al. (2020) desenvolveram um processo baseado em ML para classificar imagens WSIs coradas com ZN. Esse processo inclui dois modelos CNNs (Inception-V3) para reconhecer blocos contendo micobactérias álcool-ácido resistentes (AFB, do inglês *acid-fast mycobacteria*) e classificá-los em AFB-positivas ou AFB-negativas, gerando um escore de mapa de probabilidade de AFB. Também é incluído um modelo de l*ogistic regression* (LR) para classificar as lâminas com base em características de mapas de probabilidade AFB montados a partir dos resultados de classificação baseados em blocos pela CNN. Eles usaram ainda uma técnica de aprendizagem ativa para melhorar o desempenho do método proposto. Nos experimentos foram usadas 167 lâminas sendo que 51 (30,5%) são positivas e 116 (69,5%) são negativas. A estratégia 10-*fold cross-validation* foi utilizada. As duas CNNs obtiveram os valores: sensibilidade de 98.72 e 98.75, especificidade de 99,35 e 98,05, acurácia de 99,04 e 98,48 e F-score de 99,03% e 98,75%, respectivamente. No final do processo, os modelos LR aprenderam características para classificar os WSIs em classes AFB-positivas ou AFB-negativas e os melhores resultados obtidos foram: sensibilidade (87,13%), especificidade (87,62%) e F1 (80,18%).

Simon et al. (2019) propõem uma *shallow* CNN com camada LSTM (*Long Short-Term Memory units*) para detectar o bacilo da TB, a partir das imagens microscópicas. A detecção é realizada a partir da classificação de *patches* gerados na divisão da imagem original (combinado com a técnica *non-maximum suppression*). O experimento foi realizado em 1265 imagens consistindo em 43,7% de *patches* positivos. O conjunto de dados é dividido em 50% para treinamento e 50% para teste. Os resultados são AUC de 0,99, AP de 0,99, acurácia de 96,29%, precisão de 0,96, sensibilidade de 0,96 e F1-Score de 0,96.

Mithra e Emmanuel (2019) apresentam um sistema automático de detecção do bacilo da TB. Inicialmente, o pré-processamento é realizado usando filtragem de mediana adaptativa. Em seguida, um algoritmo de limiarização multinível, chamado *Channel Area Thresholding* (CAT), é proposto para segmentação de imagens bacterianas. As características extraídas, a partir das técnicas *Location-oriented histogram* (LOH) e *Speed up robust feature* (SURF), são utilizadas em uma *Deep belief network* (DBN) para identificação de TB, bem como para contagem de bactérias totais presentes na imagem para verificação do nível de infecção. Eles utilizaram o data set ZNSM-iDB Shah et al. (2017), com 500 imagens, sendo 275 para treinamento e 225 para testes. Os resultados alcançados são: acurácia de 97,55, sensibilidade de 97,86, e especificidade de 98,23.

Hu et al. (2019) propõem uma estratégia de construção de conjuntos de dados baseada na partição não sobreposta de sub-grafos. Para avaliar esse método, usaram três modelos de CNNs bem conhecidos (Inception v3, ResNet, DenseNet) em um conjunto de dados de 2.630 imagens de baciloscopia de escarro, divididas em 1.421 para treinamento, 609 para validação e 600 para teste. Além disso, eles também remontaram os mapas de probabilidade prevista dos sub-gráficos em uma imagem de slide inteira (WSI) para exibir visualmente os resultados previstos. Apontando assim, as áreas que provavelmente possuem bacilos da TB. Os resultados do experimento obtiveram melhores desempenhos com a CNN Inception v3 (0,984 de acurácia, 0,984 de precisão, 0,984 de sensibilidade e 0.984 de F1 Score).

Kuok et al. (2019) propõem um sistema de identificação de TB, sem nenhum mecanismo de pré-processamento. Na fase de detecção, os candidatos positivos ou negativos são gerados e, em seguida, submetidos para fase de classificação de candidatos usando CNNs. Eles utilizaram 19.234 imagens no total, onde 1.186 contêm TB e 18.048 não contém. Nos experimentos, são comparados o desempenho de detecção entre os métodos *Faster Region-based Convolutional Neural Network* (Faster R-CNN), refined-faster-R-CNN (RFRCNN) e SVM baseados em pixels. Também são comparados *single-member* CNN, ensemble CNN, e deep CNN como classificadores para avaliar o desempenho da classificação. A melhor taxa de detecção do sistema proposto foi de cerca de 86% de sensibilidade (Faster R-CNN). Ao avaliar os métodos utilizados na segunda fase (classificação), a Ensemble CNN obteve valores de 94,80%, 98,06% e 91,65%, para G-mean, sensibilidade, especificidade, respectivamente.

El-Melegy, Mohamed e ElMelegy (2019) propõem uma abordagem DL para a detecção e classificação de bacilos de TB em imagens microscópicas coradas com ZN. A abordagem proposta é baseada na estrutura da *Faster Region-based Convolutional Neural Network* (Faster R-CNN). Eles utilizaram o conjunto de dados público ZNSM-iDB (SHAH et al., 2017). Após aplicar métodos de *data augmentation* o número de imagens microscópicas se tornou 1500 imagens, divididas em 80% para treinamento e 20% para teste. Ao realizar a comparação entre a Faster R-CNN, SVMs, k-*Nearest Neighbo*r (kNN), *Linear Discriminant Analysis* (LDA) e *Quadratic Discriminant Analysis* (QDA), a Faster R-CNN obteve os melhores resultados (sensibilidade: 98,3%, precisão: 82,6%, F-score: 89,7%).

El-Melegy et al. (2019) utilizam a estrutura *Faster-Region-based Convolutional Neural Network* (Faster R-CNN) para detecção de bacilos da TB. Para reduzir a alta taxa de falsos positivos do Faster-RCNN, eles adicionam um estágio adicional que consiste em uma CNN que classifica as caixas delimitadoras encontradas pelo Faster-R-CNN em bacilos de TB reais ou falsos. Isso aumenta o desempenho de detecção da abordagem geral. Eles utilizaram o conjunto de dados público ZNSM-iDB (SHAH et al., 2017), foram

utilizadas 500 imagens com 2500 (bacilos) objetos, divididos em 80% do conjunto de dados usado para treinamento e 20% para teste. Testes comparativos entre SVM, KNN, QDA, LDA, Faster R-CNN, e Faster R-CNN+CNN, foram realizados. Os melhores resultados alcançados foram para Faster R-CNN+CNN, com sensibilidade: 98,4%, precisão: 85,1%, F-score: 91,2%.

Panicker et al. (2018) propõem um método que realiza a detecção de TB, por binarização de imagens e posterior classificação das regiões detectadas utilizando uma CNN. O método proposto é implementado em duas etapas. Na primeira etapa, utilizam uma abordagem de segmentação simples para classificar o primeiro plano e o plano de fundo das imagens, usando o método de *Otsu*. O primeiro plano consiste em bacilos únicos, bacilos sobrepostos e outros artefatos. Os objetos segmentados em primeiro plano são então entregues a uma CNN, que classifica os objetos em bacilos e não bacilos. Eles utilizaram o conjunto de dados proposto por Costa et al. (2014), 120 imagens divididas em 80% para treinamento e 20% para teste. Para validação, foram utilizadas 20% das imagens do conjunto de treinamento. Nos experimentos realizados com a CNN, obteve-se acurácia de 98,88%. Resultados experimentais de detecção+classificação de TB mostram que o algoritmo proposto atinge 97,13% de sensibilidade, 78,4% de precisão e 86,76% de F-score.

Kant e Srivastava (2018) propõem um método em cascata para o diagnóstico de TB baseada em DL. Duas CNNs são treinadas para classificar *patch* de imagens, onde a primeira classifica em "bacillus negative"ou "bacillus suspect", e a segunda classifica em "bacillus negative"e "bacillus positive". Dessa forma, a segunda CNN permite diferenciar entre verdadeiros positivos e falsos positivos da primeira CNN. Os fragmentos de imagens de entrada da primeira CNN são obtidos por uma estratégia de detecção de *patchwise*, em que são obtidos pequenos *pacths* (20x20 pixels) em uma imagem do campo de visão microscópica. Utilizam o data-set ZNSM-iDB (SHAH et al., 2017), dividido em 60% para treinamento, 20% para validação e 20% para testar o método. Os resultados alcançados são sensibilidade de 83,78% , precisão de 67,55% e F1-Score de 74.79%.

Xiong et al. (2018) construíram um modelo CNN, denominado tuberculosis AI (TB-AI), para reconhecer o bacilo da TB. Para detectar os bacilos, fragmentos de 32×32 pixels são obtidos pelo método de janela deslizante (em uma WSI) com um passo de 16 para serem classificados pela CNN. O conjunto de dados é formado por um total de 246 amostras, dos quais 45 casos (30 positivos e 15 negativos) foram para o conjunto de treinamento e 201 casos (108 positivos e 93 negativos) foram para o conjunto de testes. No conjunto de treinamento eram 96.530 *patches* em amostras positivas. Após realizar o *data augmentation* foram obtidas 578.191 *patches* com bacilos. Da mesma forma, foram obtidas 2.510.307 amostras negativas. Eles compararam o diagnóstico de TB-IA com o resultado fornecido por patologistas humanos, analisaram inconsistências entre IA e

humano e ajustaram o protocolo para obter previsões mais precisas. Ao final dos ajustes e testes realizados, o TB-AI alcançou 97,94% de sensibilidade e 83,65% de especificidade.

Sethi, Parmar e Suri (2018) propõem um modelo DL baseado na arquitetura SqueezeNet que é capaz de reconhecer características de patógenos em três imagens de amostras de doenças diferentes: (a) Malária em esfregaços de sangue espessos, (b) Tuberculose em amostras de escarro e (c) Infecção parasitária intestinal nas amostras de fezes. Métodos de compressão são aplicados ao modelo treinado para otimizar o consumo de energia, espaço de memória e a complexidade computacional, permitindo que o modelo gerado seja utilizado em hardware dedicado (Raspbery Pi + Myriad VPU). Foi utilizado o conjunto de dados proposto por Quinn et al. (2016). A CNN proposta obteve uma AUC maior que 0,99 para todos os 3 conjuntos de dados.

Tabela 3 – Artigos sobre técnicas de classificação na literatura.

Referência	**Ano**	**Técnica de Classificação**	**Objetivo**	**Conjunto de Dados**	**Métricas de Avaliação (%)**
Zurac et al. (2022)	2022	Arquitetura RegNetX4	Propõe um método automático para identificar AFB usando CNN.	570 WSIs, dividindo em conjunto de treinamento: 510 WSIs e conjunto de validação: 60 WSIs.	Acurácia: 0,9833 Sensibilidade: 0,9565 Especificidade: 1,0
Momeny et al. (2022)	2022	Arquitetura CNN	Propõe um método para classificar imagens de TB usando uma CNN generalizada. A CNN é aprimorada com base no método GAA, pooling misto, dropout e normalização de batch.	1547 imagens, dividindo em conjunto de treinamento (70%), conjunto de validação (10%) e conjunto de teste (20%).	Sensibilidade: 0,85 Especificidade: 0,97 Precisão: 0,92 F-*Measure*: 0.89
Udegova, Shelomentseva e Chentsov (2021)	2021	Arquitetura CNN	Construiu um algoritmo experimental para otimizar o modelo original de CNN, incluindo otimização de dimensão do modelo, aumento de dados, ajuste de parâmetros do modelo e melhoria da regularização.	400 imagens	Sensibilidade: 0,9858 Especificidade: 0,9843 Acurácia: 0,9850 Loss: 0,445
Muyama, Nakatumba-Nabende e Mudali (2021)	2021	Arquiteturas de CNNs pré-treinadas: VGGNet e GoogLeNet Inception v3.	Aplicar modelos de DL baseados em duas CNNs pré-treinadas para diagnosticar TB a partir de 148 imagens microscópicas de esfregaço de escarro coradas pelo método ZN, de dois conjuntos de dados diferentes.	148 imagens, dividindo em conjunto de treinamento: 88, conjunto de validação: 30 e conjunto de teste: 30.	VGGNet: acurácia (0,800), precisão (0,795), e sensibilidade (0,808). GoogLeNet Inception v3: acurácia (0,722), precisão (0,754), sensibilidade (0,808).
Ibrahim et al. (2021)	2021	Arquitetura de CNN pré-treinada: AlexNet	Usar a AlexNet pré-treinada em 3 conjuntos de experimentos A, B e C; realizar validação cruzada de 10-fold; comparar detecção de TB usando modelos AlexNet com o resultado real fornecido pelo microbiologista; e analisar as inconsistências entre os modelos de rede e humanos.	Experimento A: 124 imagens (conjunto de treinamento) e 54 (conjunto de teste); Experimento B: 524 imagens (conjunto de treinamento) e 371 (conjunto de teste); Experimento C: 530 imagens (conjunto de treinamento) e 159 (conjunto de teste).	Experimento A: acurácia: 0,9815, sensibilidade: 0,9677, especificidade: 1,0 Experimento B: acurácia: 0,9809, sensibilidade: 0,9859, especificidade: 0,9767; Experimento C: acurácia: 0,9873, sensibilidade: 0,9859, especificidade: 0,9884.

Rachmad, Chamidah e Rulaningtyas (2020)	2020	Arquitetura CNN ResNet-101 usando SVM como classificador	Desenvolver um sistema de classificação de TB usando a arquitetura CNN ResNet-101 combinando SVM como classificador.	1266 imagens (633 bactérias TB e 633 bactérias não-TB), dividindo em conjunto de treinamento: 866 e conjunto de teste: 380.	Acurácia: 0,976 Precisão: 0,979 Sensibilidade: 0,974 F1 score: 0,976
Samuel e Kanna (2019)	2019	Modelo Inception V3 DeepNet da arquitetura CNN + SVM	Propor um sistema de detecção de TB composto por dois subsistemas: um sistema de aquisição de dados FOVs e um sistema de classificação. Este modelo aprende a partir dos pesos pré-treinados do Inception V3 e classifica os dados usando SVM.	Conjunto de dados ZNSM-iDB e seu próprio conjunto de dados digitais microscópicos. Total de 1242 imagens (620 são bacilos e 622 não são bacilos).	Acurácia: 0,9505 AUC: 0,9505
López et al. (2017)	2017	Arquitetura CNN.	Propor um método para classificação automática de patches de imagem usando versões RGB, R-G e em escala de cinza como entradas de três modelos de CNNs para identificar TB.	Conjunto de dados de Costa et al. (2014).	AUC: 0,99 Acurácia: 0,99
Quinn et al. (2016)	2016	Arquitetura CNN	Propor um método para avaliar o desempenho da CNN profunda em três diferentes tarefas de microscopia: diagnóstico de malária em esfregaços de sangue espesso, TB em amostras de esfregaço de escarro e ovos de parasitas intestinais em amostras de fezes.	3,734 objetos em 928 imagens, dividindo em 50% para o conjunto de treinamento e 50 % para o conjunto de teste.	AUC: 0,99 AP: 0,93

Tabela 4 – Artigos sobre técnicas de segmentação e classificação na literatura.

Referência	**Ano**	**Técnica de Segmentação**	**Técnica de Classificação**	**Objetivo**	**Conjunto de Dados**	**Métricas de Avaliação** (%)
Angayarkanni et al. (2022)	2022	Arquitetura CNN SegZNet e arquitetura CNN UNet	Arquitetura CNN UNet	Propor uma nova arquitetura DL baseada em pixels para a geração de máscaras (SegZNet) e propor uma arquitetura baseada em CNN para identificar e segmentar a região de interesse (UNet).	Conjunto de treinamento: 3040 imagens, conjunto de validação: 800 imagens, e conjunto de teste limitado a 100 imagens.	IOU: 0.8728 Acurácia: 98,5 Precisão: 0,9153 Sensibilidade: 1,0
Vanitha, Angayarkanni e Sathishkumar (2021)	2021	Arquitetura CNN UNet	Arquitetura CNN UNet	Apresentar uma análise comparativa de vários métodos de segmentação supervisionados e não supervisionados para a segmentação automatizada de bacilos de TB em imagens de esfregaço de escarro.	Conjunto de treinamento: 3040 imagens, conjunto de validação: 800 imagens e conjunto de teste limitado a 100 imagens.	Acurácia: 0.99, Dice: 0,988 Sensibilidade: 0.99 VPP: 0,98 Jaccard: 0,985
Reist et al. (2021)	2021	Arquitetura RetinaNet	Arquitetura RetinaNet	Propor um método para a detecção automatizada do bacilo usando a arquitetura RetinaNet.	910 imagens: conjunto de treinamento (70%), conjunto de validação (10%), e conjunto de teste (20%)	Acurácia: 0,671 Sensibilidade: 0,8656 F-score: 0,7561
Winarno, Bilhaq e Agoes (2021)	2021	Arquitetura DSOD	Arquitetura DSOD	Propor uma arquitetura melhorada do framework Deeply Supervised Object Detector (DSOD) para detecção de bacilos de TB.	Trilaksana et al. (2020) e Shah et al. (2017). 600 imagens, divididas em conjunto de teste: 90, treinamento e validação: 510.	Acurácia AP: 0,7232
Chithra e Jagatheeswari (2020)	2020	Técnica de limiarização adaptativa	Fractional crow search-based deep CNN (FC-SVNN)	Propor a técnica AFC-Deep CNN para detecção de gravidade e identificação do nível de infecção usando imagens de esfregaço de escarro. A técnica segue cinco passos: pré-processamento, segmentação, extração de características, redução de dimensionalidade e classificação de TB.	Conjunto de dados ZNSM-iDB com 75 imagens: 50 imagens são de bacilos e 25 imagens são de não-bacilos.	Acurácia: 0,935 TPR: 0,92 FPR: 0,08

Trilaksana et al. (2020)	2020	Arquitetura PeleeNet	Arquitetura PeleeNet	Propor um banco de dados de imagens de esfregaço de escarro com dados coletados por especialistas da Indonésia. Além disso, propor um método de DL para a detecção de bacilos de TB.	3 classes: TBBacteria, NonTBbacteria e Stainresidues. 350 imagens com 3.102 bactérias TB, divididas em 276 para treinamento e validação, e 72 para teste.	Acurácia para TBBacteria: 0,2687, Acurácia para NonTNBacteria: 0,2682, Stainresidues não é detectado.
Swetha, Sankaragomathi e Thangamalar (2020)	2020	Channel Area Thresholding (CAT).	Arquitetura CNN combinada com outros extratores de características.	Realizar pré-processamento com técnicas de redução de ruído e modificação de intensidade, usar segmentação e uma CNN para classificar a presença do bacilo de TB como leve, moderada e severa em imagens de escarro.	30 imagens com condições de TB como leve, moderada e severa.	Acurácia: 0,995 Sensibilidade: 0,947 Especificidade: 0,99
Serrão et al. (2020a)	2020	Arquitetura CNN associada a uma abordagem de mosaico de imagem.	Arquitetura CNN	Propor um método para realizar a segmentação e identificação de bacilos, associando modelos de CNN e uma abordagem de imagem em mosaico.	5.000 imagens em mosaico: 60% para o conjunto de treinamento, 20% para o conjunto de validação e 20% para o conjunto de teste.	Acurácia: 0,996 Precisão: 0,993 Sensibilidade: 0,949 Especificidade: 0,996
Yang et al. (2020)	2020	Arquitetura CNN (Inception-V3).	Regressão logística (LR)	Propor um processo formado por CNNs e LR para classificar WSIs de tecidos corados com ZN em classes AFB-positivas ou AFB-negativas.	167 lâminas: 51 (30.5%) são positivas e 116 (69.5%) são negativas.	Valores dos modelos LR: Sensibilidade: 0,8713 Especificidade: 0,8762 F1: 0,8018
Simon et al. (2019)	2019	Dividir a imagem em patches	Arquitetura CNN rasa com camada LSTM	Propor uma CNN rasa com camada LSTM (Long Short-Term Memory units) para detectar o bacilo da TB.	1265 imagens (43.7% positivas), divididas em 50% para conjunto de treinamento e 50% para conjunto de teste.	AUC: 0,99 AP: 0,99 Acurácia: 0,9629 Precisão: 0,96 Sensibilidade: 0,96 F1-Score: 0,96
Mithra e Emmanuel (2019)	2019	Channel Area Thresholding (CAT)	Rede de crença profunda (DBN)	Apresentar um sistema automático para identificação e contagem do número de bacilos de TB presentes nas imagens.	ZNSM-iDB, com 500 imagens, 275 para o conjunto de treinamento e 225 para o conjunto de teste.	Acurácia: 0,9755 Sensibilidade: 0,9786 Especificidade: 0,9823

Hu et al. (2019)	2019	Estratégia baseada em Partição de Subgrafos	Inception v3	Propor uma estratégia de construção de conjunto de dados baseada na partição não sobreposta de subgrafos; classificar usando CNNs; detectar bacilos de TB usando mapas de subgrafos em WSI.	2.630 imagens, dividindo em 1.421 para o conjunto de treinamento, 609 para o conjunto de validação e 600 para o conjunto de teste.	Acurácia: 0,984 Precisão: 0,984 Sensibilidade: 0,984 F1-Score: 0,984
Kuok et al. (2019)	2019	Faster R-CNN	Arquitetura CNN	Propor um sistema de identificação de TB em duas etapas (detecção e classificação), sem qualquer mecanismo de pré-processamento.	19.234 imagens no total, 1.186 contêm TB e 18.048 não contêm TB.	Sensibilidade: 0,86 (Faster R-CNN) G-mean: 0,9480 (CNN) Sensibilidade: 0,9806 (CNN) Especificidade: 0,9165 (CNN)
El-Melegy, Mohamed e ElMelegy (2019)	2019	Faster R-CNN	Faster R-CNN	Propor uma abordagem de DL para a detecção e classificação de bacilos de TB em imagens microscópicas coradas com ZN.	Conjunto de dados ZNSM-iDB: 1500 imagens, dividindo em 80% para o conjunto de treinamento e 20% para o conjunto de teste.	Sensibilidade: 0,983 Precisão: 0,826 F-score: 0,897
El-Melegy et al. (2019)	2019	Faster R-CNN	Arquitetura CNN	Reduzir a alta taxa de falsos positivos do Faster-R-CNN incluindo uma etapa adicional consistindo de uma CNN para classificar as caixas delimitadoras encontradas pelo Faster-R-CNN em bacilos de TB reais ou falsos.	Conjunto de dados ZNSM-iDB: 500 imagens com 2500 bacilos, dividindo em 80% para o conjunto de treinamento e 20% para o conjunto de teste.	Sensibilidade: 0,984 Precisão: 0,851 F-score: 0,912

Panicker et al. (2018)	2018	Método Otsu	Arquitetura CNN	Propor um método que realiza a detecção de TB, por binarização de imagens e subsequente classificação das regiões detectadas usando uma CNN.	Conjunto de dados Costa et al. (2014): 120 imagens, dividindo em 80% para o conjunto de treinamento e 20% para o conjunto de teste; para validação, 20% do conjunto de treinamento.	Sensibilidade: 0,9713 Precisão: 0,784 F-score: 0,8676
Kant e Srivastava (2018)	2018	Detecção por fragmentos de imagem	Arquitetura CNN	Propor um método que combina duas CNNs para identificar bacilos de TB em fragmentos de imagem.	Conjunto de dados ZNSM-iDB, dividindo em 60% para o conjunto de treinamento, 20% para o conjunto de validação e 20% para o conjunto de teste.	Sensibilidade: 0,8378 Precisão: 0,6755 F-score: 0,7479
Xiong et al. (2018)	2018	Arquitetura CNN com janela deslizante	Arquitetura CNN	Propor um modelo de CNN, chamado inteligência artificial para tuberculose (TB-AI), para a detecção automática do bacilo da TB.	246 amostras: 45 (30 positivas e 15 negativas) para o conjunto de treinamento e 201 (108 positivas e 93 negativas) para o conjunto de teste.	Sensibilidade: 0,9794 Especificidade: 0,8365
Sethi, Parmar e Suri (2018)	2018	Janela deslizante de tamanho fixo	Arquitetura CNN SqueezeNet	Propor a implementação de um sistema de suporte à microscopia diagnóstica para estudo em: (a) Malária em esfregaços grossos de sangue, (b) Tuberculose em amostras de esfregaço de escarro e (c) Infecção parasitária intestinal em amostras de fezes.	Conjunto de dados Quinn et al. (2016).	AUC: 0,99 para os 3 conjuntos de dados.

3.3 Discussão e recomendações sobre a revisão sistemática realizada

Neste estudo, foram encontrados 28 artigos na literatura que propuseram o uso de DL para auxiliar na identificação de bacilos da TB em imagens microscópicas de esfregaços de escarro (microscopia de esfregaço). Tais artigos foram encontrados após uma seleção cuidadosa realizada em nove bases de dados. Ao relacionar os processos adotados nestes trabalhos com a metodologia de uso (treinamento/teste) das técnicas de ML/DL, sintetizamos o processo de classificação e detecção do bacilo da TB em um fluxo de trabalho ilustrativo (Figura 12). Com este fluxo de trabalho, é possível responder à pergunta de como as técnicas de DL podem ser usadas para identificar o bacilo da TB.

Construir este fluxo de trabalho a partir dos artigos analisados permitiu a identificação de pontos importantes que devem ser considerados ao usar DL. Primeiramente, é importante ter um conjunto de dados diversificado e robusto para realizar o treinamento, pois isso pode contribuir para gerar modelos capazes de obter bons resultados na classificação de bacilos em diferentes imagens, que podem ser obtidas em diferentes cenários (por exemplo, diferentes microscópios, câmeras, resoluções e amostras). Apesar desta importante questão, existem poucos conjuntos de dados públicos, levando muitos autores a desenvolver seus próprios conjuntos de dados privados. Nota-se que isso dificulta a reprodutibilidade e o desenvolvimento de novas técnicas. Por exemplo, apenas Quinn et al. (2016) e Shah et al. (2017) forneceram endereços eletrônicos para acesso direto aos seus conjuntos de dados (sem necessidade de fazer quaisquer solicitações). Verificou-se que quinze artigos utilizaram os conjuntos de Quinn et al. (2016) e/ou Shah et al. (2017). Por outro lado, artigos de Costa et al. (2014) e/ou Trilaksana et al. (2020) requerem um pedido para acessar os conjuntos de dados. Verificou-se que seis trabalhos utilizaram os conjuntos de Costa et al. (2014) e/ou Trilaksana et al. (2020). Finalmente, 9 artigos propuseram seus próprios conjuntos de dados privados.

Outro ponto importante é o uso de diferentes métricas e estratégias para avaliar os modelos gerados. Não há uma única métrica que permita concluir quão bem o modelo gerado se apresenta. Portanto, é importante avaliar modelos utilizando várias métricas. Por exemplo, alguns artigos apresentaram apenas a acurácia de seu modelo, no entanto, sabe-se que essa métrica pode ter um viés em conjuntos de dados desbalanceados. Outros artigos discutem seus resultados com base em algumas métricas de comparação, tornando difícil a interpretação do desempenho do modelo. Por outro lado, artigos como Simon et al. (2019) e mais alguns, permitem ao leitor avaliar os resultados relatados a partir de diferentes perspectivas ao usar várias métricas de desempenho, principalmente aquelas relatadas na Seção 3.2.3. Vale ressaltar que a métrica AP (ou mAP) é altamente recomendada para avaliar métodos que realizam detecção de objetos Padilla, Netto e Silva (2020), Park e Kim (2020), no entanto, poucos artigos a utilizaram.

Além das estratégias de avaliação clássicas apresentadas na Seção 3.2.3, um experimento importante a ser realizado é a comparação do desempenho do modelo de DL com profissionais que realizam o diagnóstico de TB. Observou-se que a maioria dos estudos não apresentou uma forma de utilizar essas técnicas em um laboratório, mas apenas demonstrou como o DL pode se sair bem na classificação do bacilo da TB. Devido à complexidade deste tipo de experimento, homem vs máquina, poucos artigos, como Ibrahim et al. (2021) e Zurac et al. (2022), fizeram alguma comparação ou interação entre a técnica de DL proposta e o patologista.

O fluxo de trabalho apresentado é uma generalização dos passos necessários para criar e usar modelos DL para classificar bacilos de TB. Além de ajudar a responder uma das perguntas estabelecidas, também pode auxiliar novos pesquisadores a entender como usar técnicas de DL e projetar seus experimentos para propor novas abordagens que podem ajudar no diagnóstico de TB.

Após entender o processo de uso do DL, buscou-se responder quais técnicas de DL estão sendo aplicadas para classificar bacilos de TB em imagens microscópicas de esfregaços de escarro. Analisando os artigos selecionados, observou-se que as CNNs têm sido a técnica de DL mais explorada para identificar o bacilo da TB em fragmentos de imagem. Diferentes autores apresentaram arquiteturas de CNN com bom desempenho (especificidades e sensibilidade próximas a 99%), além de mostrarem que as CNNs são capazes de superar várias técnicas clássicas de ML.

No entanto, ao olhar para os artigos que abordavam a detecção de bacilos de TB em uma imagem de um campo de visão do microscópio, encontrou-se uma maior variedade de métodos propostos.

Em geral, poucos trabalhos foram encontrados que usavam técnicas específicas de DL para detectar e classificar objetos simultaneamente (por exemplo, UNet, Faster R-CNN, entre outros). A maioria dos artigos apresentou soluções mistas que usam técnicas clássicas de visão computacional associadas a técnicas de DL para localizar o bacilo da TB em imagens de esfregaços de escarro. Notou-se novamente, nessas abordagens mistas, as CNNs se destacam pelo número de artigos que as utilizam e apresentam bons resultados.

Foi observado que vários artigos que usam métodos de detecção de bacilos de TB em uma imagem de um campo de visão do microscópio (Tabela 3) têm um desempenho um pouco pior do que os métodos que realizam apenas a classificação (Tabela 4). Esse fenômeno pode estar associado à existência de bacilos que não foram anotados (na parte inferior da imagem), que podem confundir o modelo e/ou aumentar erroneamente a quantidade de FP. No entanto, são esses métodos que podem realmente ajudar no diagnóstico, ou seja, ajudar os patologistas a localizar os bacilos em cada campo observado sob o microscópio.

Como pode ser visto nos artigos revisados, todos os autores apresentaram diferentes métodos que se saíram bem na tarefa de reconhecer um bacilo de TB. No entanto, poucos artigos, como Samuel e Kanna (2019), exploraram o conceito de desenvolver uma interface/sistema que integra o microscópio, o patologista e algumas técnicas de DL para realizar juntos o diagnóstico de TB.

Finalmente, após ler todos os artigos citados e experiência previa em um laboratório de diagnóstico de TB (com tentativas empíricas de aplicar esses métodos na prática), foi identificado que ainda há um longo caminho a percorrer para que essas técnicas sejam usadas no laboratório. Porque, embora existam vários trabalhos que abordem o uso de técnicas de DL no diagnóstico de TB, e que proponham diferentes técnicas que apresentam resultados promissores, são poucos os trabalhos que exploraram tópicos como:

- desenvolvimento e teste de sistemas com boa usabilidade que permitam ao patologista interagir com a técnica de DL durante o diagnóstico. Assim permitindo avaliar o quanto essas técnicas podem realmente ajudar no momento do diagnóstico;
- criação de um processo de ponta a ponta que permita a elaboração de uma sequência de etapas que possam facilitar o uso de DL no diagnóstico de TB. Em outras palavras, estabelecer as fases iniciais de como as imagens poderiam ser capturadas até as fases finais de como o diagnóstico seria emitido com a interação entre o patologista e a técnica de DL;
- criar conjuntos de dados mais diversos com imagens obtidas de diferentes fontes. Além de realizar testes unificando diferentes conjuntos de dados. Portanto, permitindo uma avaliação de como as técnicas de DL poderiam ser treinadas em imagens de um laboratório e serem usadas em outro laboratório;
- maior transparência no sentido de compartilhar códigos-fonte e conjuntos de dados. Para permitir que os pesquisadores testem diferentes soluções em diferentes cenários. Além de facilitar o desenvolvimento de novas técnicas ou melhorias nos conceitos já propostos;
- maior padronização das métricas e estratégias de treinamento/teste usadas para avaliar os modelos. Assim facilitando a comparação entre os trabalhos e uma melhor interpretação dos resultados obtidos;
- expandir incentivos e parcerias entre grupos multidisciplinares para realizar experimentos e desenvolver um sistema de Diagnóstico Auxiliado por Computador (CADx, do inglês *Computer-Aided Diagnosis*) baseado em DL para a realização de esfregaços microscópicos de escarro. A quantidade de artigos encontrados pode refletir como este assunto ainda é pouco explorado.

Explorar cada vez mais esses tópicos são etapas que precisam ser realizadas para o desenvolvimento de uma plataforma/software que possa de fato ser testado e aprimorado até o ponto de ser utilizado por profissionais que realizam diagnósticos de TB em seus laboratórios.

3.4 Conclusões sobre a revisão sistemática

Este estudo teve como objetivo realizar uma revisão sistemática e reproduzir alguns trabalhos na literatura, buscando responder às perguntas: 1) Quais técnicas de DL estão sendo aplicadas para classificar os bacilos de TB em imagens microscópicas de esfregaços de escarro? 2) Como essas técnicas são utilizadas? 3) Quais etapas estão faltando para que essas técnicas sejam utilizadas no diagnóstico da TB na prática? Para isso, realizou-se uma busca cuidadosa em nove bases de dados, onde foi encontrado 28 artigos na literatura que propunham o uso de DL para auxiliar na identificação de bacilos de TB em imagens microscópicas de esfregaços de escarro.

Após analisar cada artigo, percebeu-se que a principal técnica utilizada para classificar os bacilos de TB em pequenos fragmentos de imagem era a CNN. Onde vários trabalhos propuseram diferentes arquiteturas e estratégias para usar CNN como um sistema de suporte ao diagnóstico de TB. Em geral, essas arquiteturas são combinadas com técnicas clássicas de visão computacional ou outro processamento de imagem para localizar os bacilos de TB em imagens obtidas sob o microscópio. Todos os artigos apresentam bons resultados, mas observou-se que poucos estudos exploraram técnicas de DL capazes de realizar detecção+classificação de maneira unificada (por exemplo, Faster R-CNN, UNet, Yolo, Mask R-CNN, entre outros) e poucos experimentos compararam o desempenho de DL vs Patologista.

Buscando responder à segunda pergunta, foi resumido em um fluxo de trabalho todas as etapas necessárias para usar DL para identificar o bacilo da TB. Sendo descrito etapas como criação de conjuntos de dados, treinamento/validação, teste e identificação (inferência) de bacilos em imagens microscópicas de esfregaços de escarro. Além disso, apresentou-se as principais métricas, estratégias de treinamento/teste e bancos de dados públicos que podem ser usados na construção de modelos DL para identificar o bacilo da TB. Também observou-se a necessidade do desenvolvimento e disponibilização de mais bancos de dados públicos, para incentivar o desenvolvimento de novas técnicas de DL e permitir que benchmarks mais robustos sejam propostos e realizados.

Após todas as análises e aprofundando no tema em questão, observou-se que ainda há vários passos a serem dados para que tais técnicas sejam utilizadas na prática em laboratórios. Destacando principalmente a necessidade de realizar mais experimentos

que comparem o desempenho de DL vs Patologistas e o desenvolvimento de CADx que possam ser baixados, testados e validados por diferentes pesquisadores.

Uma das possíveis fraquezas desta revisão pode estar relacionada às limitações nos termos usados para pesquisar os bancos de dados, a seleção de apenas textos escritos em inglês e o foco restrito em técnicas de DL. No entanto, buscou-se estabelecer critérios que permitia encontrar artigos relevantes que possam colaborar para melhorar a microscopia de esfregaços de escarro usando técnicas de DL. As informações apresentadas e as análises realizadas reforçam o grande potencial que as técnicas de DL podem ter para auxiliar no diagnóstico da TB.

Finalmente, com o desenvolvimento de novos CADx, os patologistas poderiam contar com a ajuda de uma IA para analisar imagens microscópicas de esfregaços de escarro, para que o diagnóstico de TB possa ser mais rápido e preciso.

4 Construção de um Novo Conjunto de Dados

Conforme apresentado no Capítulo 3, identificar conjuntos de dados prontamente disponíveis na literatura não é uma tarefa trivial, mesmo considerando-se a expressiva quantidade de estudos que recorrem a técnicas de visão computacional e aprendizado de máquina para a classificação do bacilo da TB. Diante deste panorama, um dos propósitos centrais desta pesquisa é propor um conjunto de dados robusto, composto por imagens provenientes da baciloscopia, e que foi validado por um grupo de especialistas na área. Almeja-se que este repositório de imagens possa não apenas auxiliar na consecução do objetivo primordial deste trabalho, que é a criação de uma ferramenta de apoio ao diagnóstico da TB, mas também servir como recurso valioso para outros pesquisadores interessados em desenvolver e refinar técnicas de detecção e classificação do bacilo, visto que ficará disponível de forma gratuita.

A Figura 18 ilustra as seis etapas definidas para a construção desse novo conjunto de dados, são elas: (i) escolher as lâminas com amostras de escarro, (ii) capturar imagens de cada lâmina, (iii) construir um sistema colaborativo para identificação manual do bacilo em fotos, (iv) realizar a identificação/marcação dos bacilos nas fotos por uma equipe especializada, (v) realizar o tratamento dos dados e (vi) disponibilizar o conjunto de dados. Com essas etapas, foi possível obter um conjunto de dados confiável e altamente representativo.

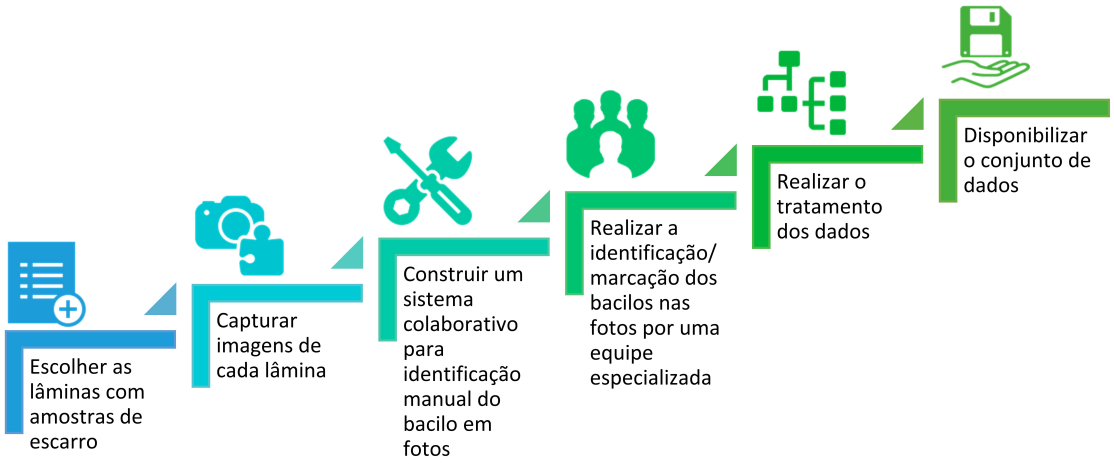


Figura 18 – Etapas definidas para construção do conjunto de dados.

As próximas seções detalham como cada etapa foi executada. Todas as atividades foram realizadas a partir da parceria entre o Programa de Pós-Graduação em Engenharia Elétrica, Laboratório de Pesquisa em Micobactérias do FM/HC/UFMG/Brasil e Ambulatório de tuberculose do Hospital das Clínicas da Universidade Federal de Minas Gerais.

4.1 Escolha das amostras

O desempenho de um método de aprendizado de máquina está diretamente relacionado ao conjunto de dados utilizados na fase de treinamento, onde um conjunto robusto e confiável pode melhorar significativamente a capacidade de identificação desses métodos. Sobre essa perspectiva, é importante escolher amostras da baciloscopia que levem a obter imagens do bacilo em diferentes tamanhos, cores e formatos, visando, assim, ter um conjunto variado e representativo de imagens do bacilo da TB.

Algumas definições foram estabelecidas para escolha das amostras, são elas:

- Utilização de lâminas de escarro corada pelo método de *Ziehl-Neelsen* (centrifugadas) sobre microscópio de luz branca de fundo;
- Escolha de 8 lâminas diagnosticadas como positivo +++ e/ou ++ (que foram numeradas de 1 a 8);
- Escolha de 2 lâminas diagnosticadas como negativo (que foram numeradas de 9 a 10);
- Observação de 50 campos de cada lâmina selecionada, sendo registrado uma foto para cada campo com o foco ajustado para o centro do campo;
- Utilização de lâminas com campos: sem bacilos, poucos bacilos e muitos bacilos;
- Uma ou mais lâminas que possuem campos com poucos e muitos artefatos;
- Uma ou mais lâminas que possuem campos com artefatos com alta concentração de fucsina.

Com estas definições, busca-se obter um conjunto variado de imagens que permitem representar os bacilos em diferentes estados. Para ilustrar a variabilidade de tamanhos, cores e formatos do bacilos da TB, é apresentado na Figura 19 fragmentos de imagens de lâminas da baciloscopia.

É importante ressaltar que esse trabalho faz parte do Projeto intitulado: AÇÕES INOVADORAS PARA AVALIAÇÃO DE MÉTODOS DIAGNÓSTICO DA TUBERCULOSE E TUBERCULOSE RESISTENTE, aprovado pelo Comitê de Ética da Universidade Federal de Minas Gerais, número CAAE: 95206318.3.0000.5149. Sendo que, as assinaturas do Termo de Consentimento Livre e Esclarecido foi solicitado para a utilização das amostras clínicas. A Tabela 5 apresenta o registro das lâminas selecionadas.

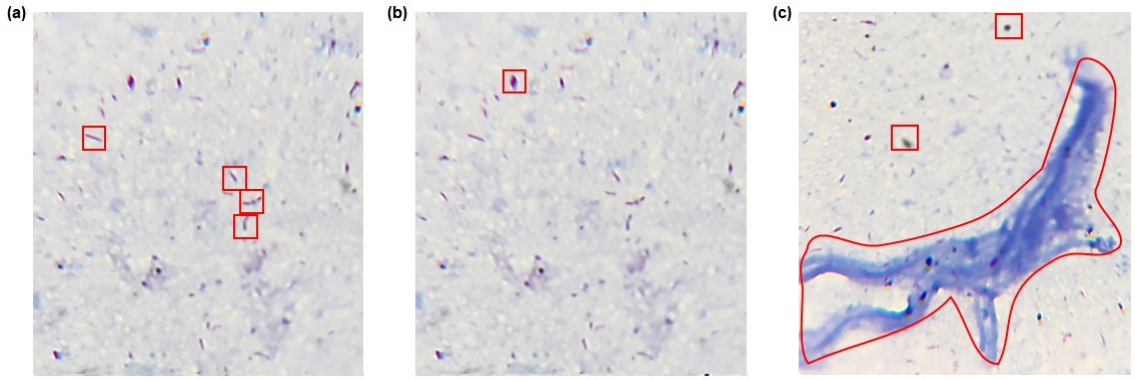


Figura 19 – Exemplos de fragmentos de uma lâmina da baciloscopia, em que são destacados em vermelho: (a) bacilos da tuberculose, (b) concentração de reagente, (c) artefatos.

Tabela 5 – Relação de lâminas selecionadas para capturar as imagens.

Numeração adotada	**Diagnóstico**
Lâmina 1	Positivo ++
Lâmina 2	Positivo +++
Lâmina 3	Positivo +++
Lâmina 4	Positivo ++
Lâmina 5	Positivo +++
Lâmina 6	Positivo ++
Lâmina 7	Positivo ++
Lâmina 8	Positivo ++
Lâmina 9	Negativo
Lâmina 10	Negativo

4.2 Captura das imagens

Uma vez que o software desenvolvido utiliza a câmera de um smartphone, visão computacional e aprendizado de máquina para realizar a segmentação e classificação do bacilo, é importante que as imagens do conjunto de dados sejam nítidas, com alta resolução e capturadas pela câmera de um smartphone. Para atender estes requisitos, foi utilizado o smartphone Samsung A50, com as seguintes especificações: 4GB RAM; armazenamento de 64gb; câmera traseira tripla com 25Mp+5Mp+8Mp, resolução máxima de 5774 x 4330 pixel com Estabilização Digital, autofoco, HDR; tela de 6.4”, 1080x2340 pixel, 403ppi, super AMOLED. Atualmente, esse smartphone é considerado um modelo intermediário e com preço aproximado de R$ 1600,00. A câmera e tela deste smartphone possuem atributos que colaboram para capturar imagens de boa qualidade e identificar de forma visual onde estão localizados os bacilos em um campo da lâmina.

A captura das imagens foi realizada utilizando um microscópio EX30 Series Biological Microscope[1]. Para acoplar o smartphone ao microscópio, foi utilizado um suporte confeccionado em polímero ABS de alta qualidade e resistência, impresso em uma impressora 3D (adquirido em loja on-line). A Figura 20 apresenta: (a) imagens do suporte, (b) o Samsung A50 acoplado ao microscópio e (c) imagem capturada.

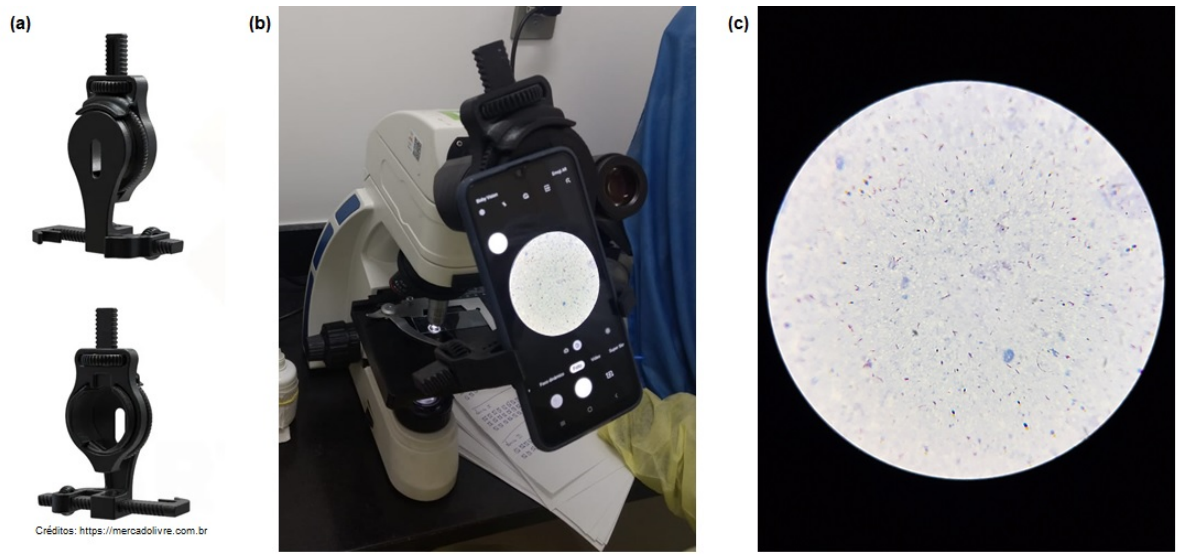


Figura 20 – Suporte para capturar fotos do microscópio usando smartphone: (a) ilustração do suporte, (b) suporte sendo utilizado e (c) imagem capturada.

Conforme estabelecido na seção anterior, foram capturadas 50 imagens para cada lâmina. Com o intuito de assegurar uma diversidade ampla nas imagens, estas não foram obtidas sequencialmente, conforme a leitura convencional em uma baciloscopia. Em vez disso, em cada lâmina, foram localizadas áreas que apresentam uma variedade distinta de bacilos e artefatos presentes nas amostras.

4.2.1 Controle e recorte dos tamanhos das imagens

Em lâminas caracterizadas por uma elevada concentração de bacilos, não é incomum encontrar campos de visão repletos de centenas de bacilos da TB. Esta densidade exacerbada pode acarretar complicações no processo de identificação desses micro-organismos nas imagens, tornando a análise mais meticulosa e, consequentemente, mais demorada. Adicionalmente, outro obstáculo na identificação precisa dos bacilos concerne à nitidez das imagens. Conforme discutido na seção precedente, o foco é invariavelmente centralizado, o que potencialmente compromete a clareza dos bacilos situados nas extremidades do campo de visão, dificultando sua identificação.

[1] Especificações técnicas: https://sios.net.au/microscopy/microscopes/upright-microscopes/upright-biological/ex30-biological-microscope

Neste sentido, para facilitar e padronizar as imagens obtidas, foi estabelecido um processo de recorte seguindo duas regras: Imagens que apresentam uma concentração reduzida de bacilos foram recortadas para formar um quadrado centralizado no campo de observação (Figura 21a); Imagens que apresentam uma concentração maior de bacilos foram recortadas para formar um quadrado menor ($\approx 1/5$ da largura da imagem) no centro do campo de observação (Figura 21b).

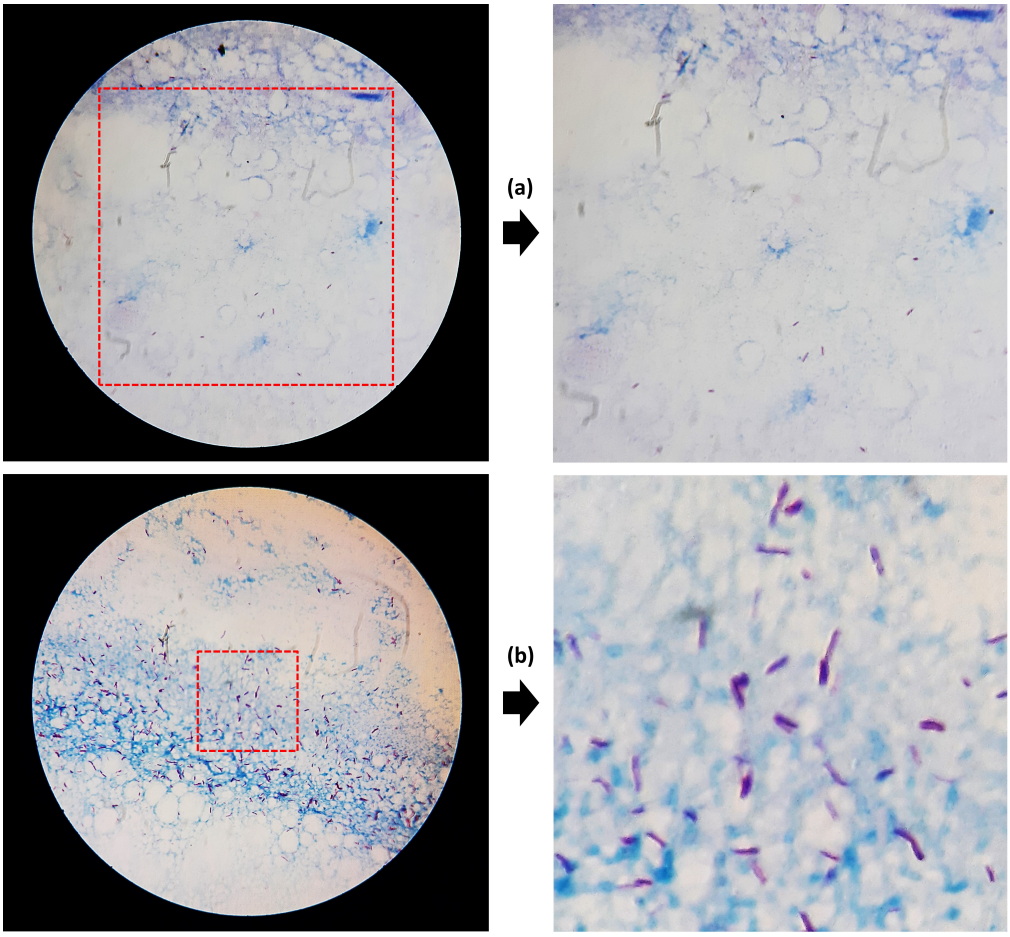


Figura 21 – Exemplo de imagens do conjunto de dados após o recorte realizado nas imagens com baica (a) e alta (b) densidade de bacilos da TB.

4.3 Sistema para identificação manual do bacilo

Uma das etapas mais importantes para criação do conjunto de dados é a identificação/mapeamento do bacilo nas imagens obtidas das lâminas da baciloscopia, em que um especialista identifica e registra onde estão localizadas as posições dos bacilos em várias imagens. Para facilitar esta tarefa foi desenvolvido um sistema web (acessível de qualquer lugar) que permite um ou mais especialistas registrarem de forma colaborativa a posição dos bacilos nas imagens obtidas. Esse sistema foi chamado de Find TB Bacillus.

Seguindo as etapas propostas, após obter as imagens, estas foram armazenadas e organizadas no Find TB Bacillus. Em seguida, uma ferramenta do sistema que permite localizar e selecionar de forma manual os bacilos da TB foi disponibilizado ao especialista, conforme ilustrado na Figura 22. Esta ferramenta conta com recursos simples para realizar esta tarefa, tais como: selecionar a área que contém um bacilo, ampliar uma área da foto, ajustar brilho, saturação e contrate da imagem apresentada ao usuário.

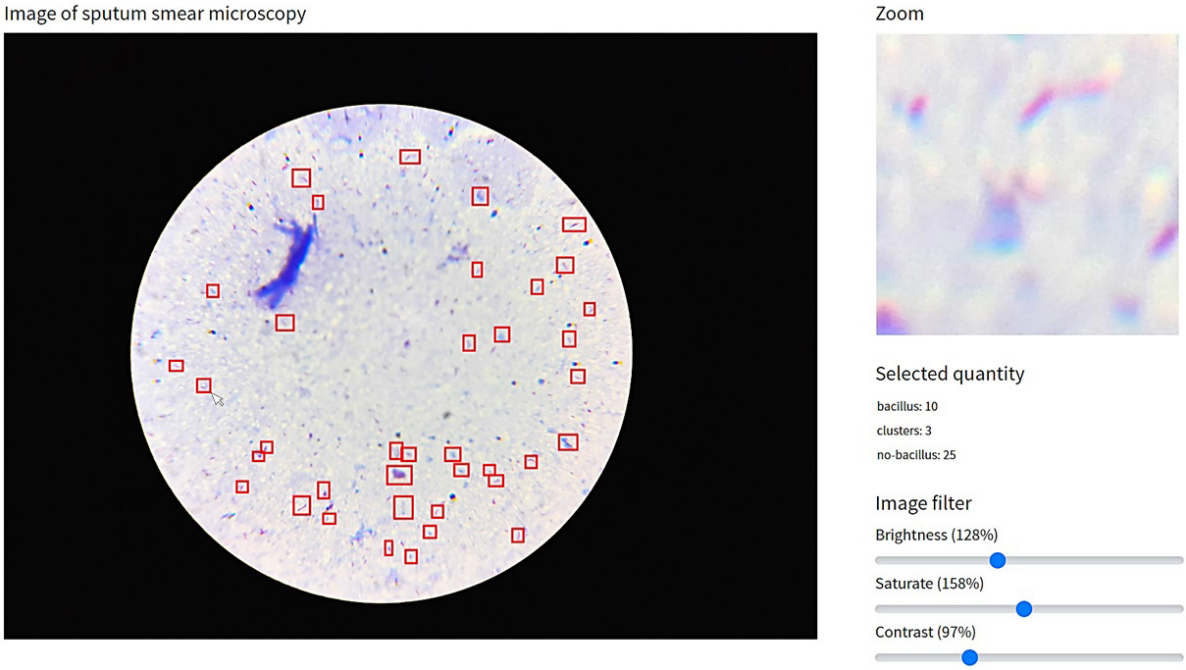


Figura 22 – Funcionalidade do Find TB Bacillus que permite ao especialista identificar e marcar a posição do bacilo em uma imagem.

Após os especialistas marcarem a posição dos bacilos nas imagens, o sistema permite exportar estas informações para arquivos no formato JPEG e JSON, que podem ser facilmente convertidas para um formato que os algoritmos de aprendizado de máquina possam interpretá-las. Vale ressaltar que um especialista não consegue visualizar o registro de outro especialista, somente o administrador do sistema possui esta permissão. Com essa forma prática de mapear os bacilos e exportar estes dados, a construção do conjunto de dados fica simplificada.

Durante os testes da versão alpha do Find TB Bacillus, identificou-se a possibilidade de transformá-lo em um Ambiente Virtual de Aprendizagem (AVA), onde professores podem disponibilizar tarefas de identificação do bacilo aos seus alunos. A concepção, aprimoramento e desenvolvimento do Find TB Bacillus como uma AVA é apresentada no Apêndice A.

4.4 Identificação dos bacilos

Conforme apresentado na Seção 3.2.2, uma das etapas mais importantes para criar o conjunto de dados é a identificação/mapeamento manual do bacilo nas imagens obtidas, processo conhecido como anotação (ou seja, localizar e marcar as posições dos bacilos em uma imagem). Compreende-se que estas posições são extremamente importantes para extrair as características do bacilo e realizar o treinamento dos métodos de aprendizado de máquina. Vale ressaltar que esta é uma etapa sensível e que o registro de falsos-positivos pode comprometer diretamente o desempenho dos métodos de aprendizado de máquina.

Em experimentos preliminares para definir o método de anotação das imagens, foi identificado que o processo de marcar cada um dos bacilos nas imagens é uma tarefa cansativa e demorada. Neste sentido, considerando o tempo de marcação, disponibilidade dos especialistas, e viabilidade técnica, foi estabelecido que seriam anotadas 25 imagens (campos) de cada lâmina selecionada. As demais 25 imagens disponíveis serão reservadas para uma futura anotação.

Para identificar os bacilos e aglomerados de bacilos (clusters), foi formada uma equipe de 7 especialistas, composta por profissionais do Laboratório de Pesquisa em Microbactérias da Faculdade de Medicina da Universidade Federal de Minas Gerais, do Hospital Júlia Kubitschek - Fundação Hospitalar do Estado de Minas Gerais (Fhemig) e da Fundação Ezequiel Dias (Funed). Estes profissionais são altamente qualificados e realizam constantemente a baciloscopia. Cada especialista acessou separadamente as imagens selecionadas e realizou as marcações que identificam as posições dos bacilos, como já ilustrado na Figura 22.

Para otimizar a identificação de bacilos nas imagens, dada a complexidade técnica e a disponibilidade dos especialistas, optou-se por uma estratégia colaborativa. Cada conjunto, constituído por 25 imagens provenientes de uma única lâmina, foi submetido à análise e anotação por uma equipe composta por três especialistas, conforme delineado na Tabela 6. Esta estrategia visa assegurar uma identificação abrangente dos bacilos e minimizar possíveis omissões. Desse modo, o conjunto final de dados engloba os bacilos identificados mediante a união das anotações de todos os especialistas envolvidos no processo. A título ilustrativo, a Figura 23 apresenta a seleção de bacilos efetuada por três especialistas distintos, nos quais cada um é representado por uma cor diferenciada (Figura 23b: vermelho, Figura 23c: verde, e Figura 23d: azul). A compilação das marcações, provenientes dos três especialistas, constitui a anotação final da imagem, conforme exemplificado na Figura 23f.

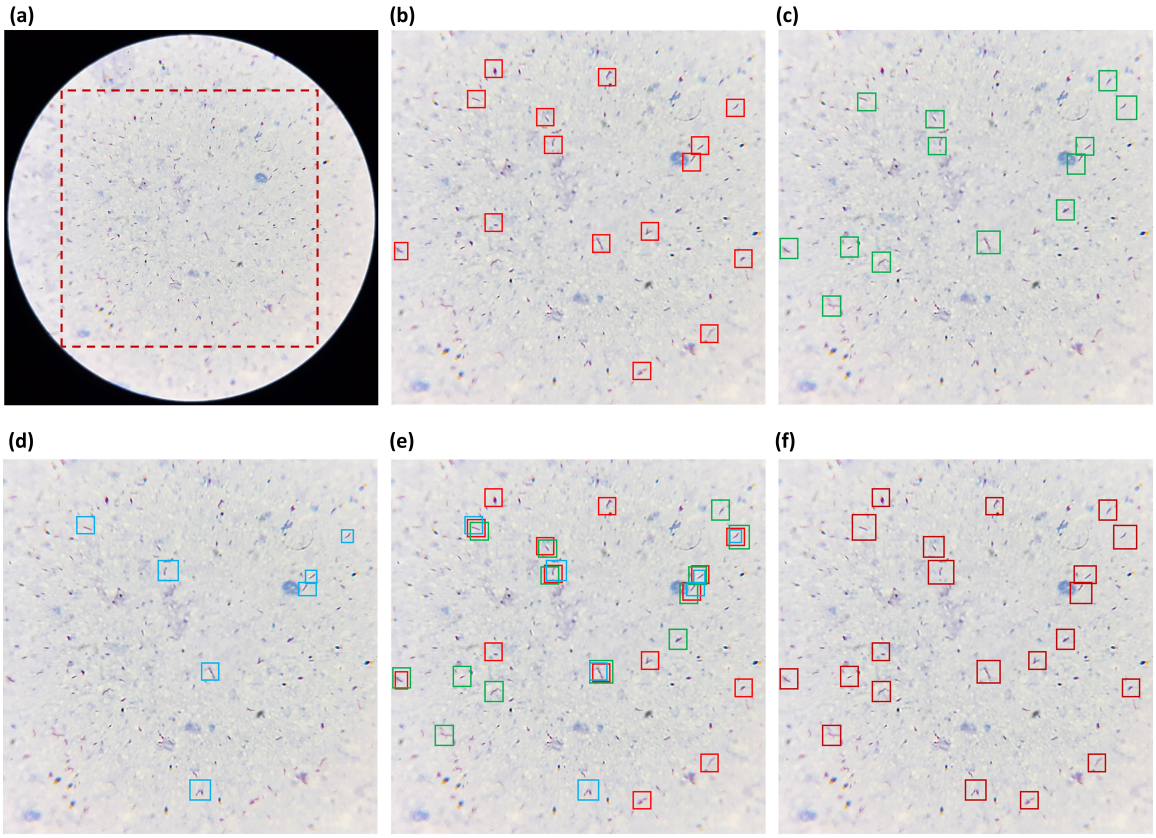


Figura 23 – Processo da escolha dos bacilos (marcações) usados para compor o conjunto de dados, onde: (a) tem-se a foto original com o recorte a ser feito, (b) identificação dos bacilos pelo usuário 1 em vermelho, (c) identificação dos bacilos pelo usuário 2 em verde, (d) identificação dos bacilos pelo usuário 3 em azul, (e) combinação de todas as identificações, (f) ajuste das marcações para compor o conjunto de dados.

Tabela 6 – Relação entre lâminas e quantidade de imagens que cada especialista foi responsável por realizar a identificação dos bacilos.

Especialistas	**Lâminas**	**Quantidade**
Especialista 1	Lâminas 1 e 2	50 imagens
Especialista 2	Lâminas 1 e 2	50 imagens
Especialista 3	Lâminas 1 e 2	50 imagens
Especialista 4	Lâminas 3, 4, 5, 6, 7 e 8	150 imagens
Especialista 5	Lâminas 3 e 4	50 imagens
Especialista 6	Lâminas 3, 4, 5, 6, 7 e 8	150 imagens
Especialista 7	Lâminas 5, 6, 7 e 8	100 imagens

4.5 Tratamento dos dados obtidos

Para facilitar o uso do conjunto de dados, todos os dados coletados foram tratados e padronizados, de acordo com os seguintes procedimentos:

1. As coordenadas (marcações do bacilo) obtidas após análise dos especialistas foram comparadas, e as marcações que se sobreporam foram unificadas em uma única marcação (como demostrado na Figura 23), utilizando a técnica *Non-Maximum Suppression*.

2. A nomenclatura dos arquivos das imagens foram padronizadas seguindo o padrão $S_iF_k.jpg$, onde i representa a numeração da lâmina e k a ordem da imagem capturada. Por exemplo, a imagem *S3F14.jpg* se refere a 14º imagem capturada da lâmina 3.

3. O registro das coordenadas dos bacilos foram armazenadas e padronizadas em um arquivo CSV, conforme ilustrado na Figura 24.

4. Criação de uma documentação e scripts em Python que demonstrem como realizar a leitura e interpretação dos arquivos disponibilizados.

Tendo em vista manter as imagens o mais fiel possível ao obtido na etapa de captura das imagens, nenhum ajuste de cor ou de resolução foram realizados. Sendo aplicado apenas o recorte da imagem, conforme apresentado na Subseção 4.2.1.

4.6 Disponibilização do conjunto de dados

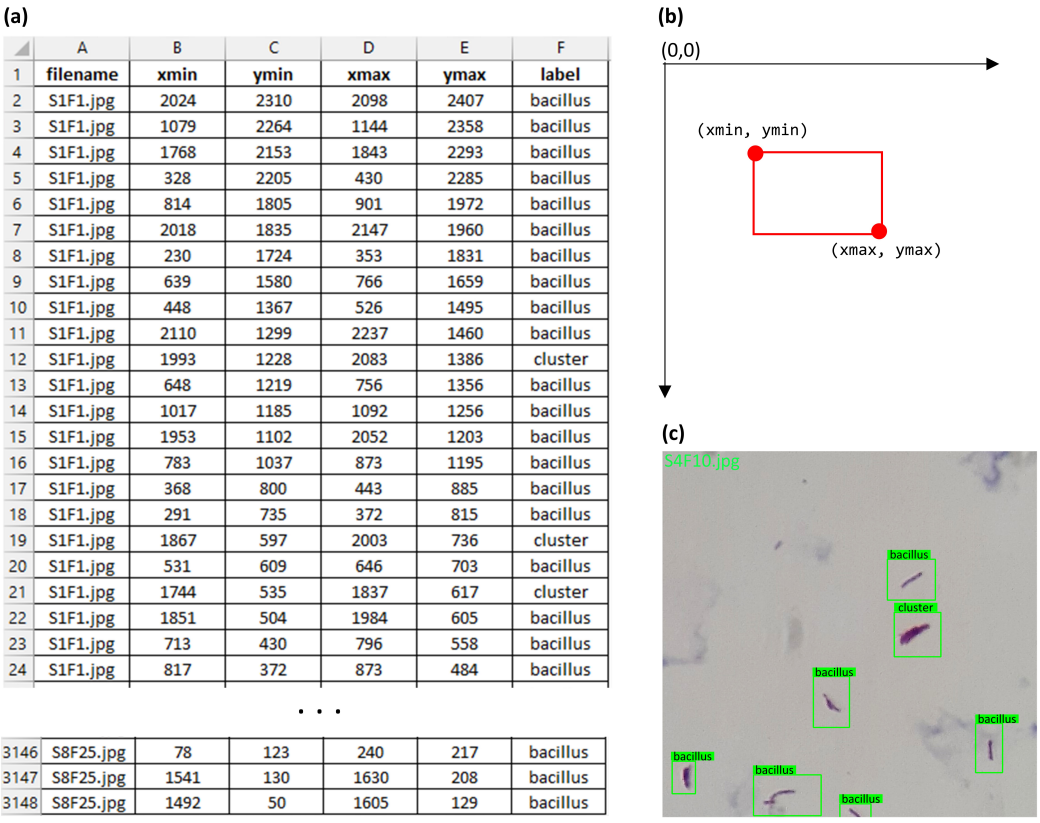
Um dos problemas identificado durante o estudo da literatura relacionada ao uso de técnicas de processamento de imagem e ML para identificação e classificação do bacilo da TB foi encontrar conjunto de dados disponíveis para download. Como apresentando no Capítulo 3 não foi possível obter muitos dos conjuntos de dados utilizados em alguns artigos, seja por não obter resposta dos autores, ou pelo site com os dados estar off-line, ou os autores não terem disponibilizados seus dados.

Para evitar tais problemas e disponibilizar de forma transparente e redundante, o download desse conjunto de dados está disponível em duas diferentes fontes, são elas: site do Laboratório Minds: (https://minds.eng.ufmg.br/) e em um servidor web próprio (http://tmserver.com.br/ufmg/TuberculosisDataSet).

Por fim, após a captura das imagens, o trabalho de anotação e o tratamento de dados, o conjunto de dados criado e disponibilizado neste trabalho pode ser resumido pelas seguintes características:

- Microscópio utilizado para captura das imagens: EX30 Series Biological Microscope;

- Câmera utilizada para captura das imagens: 25Mp - Smartphone Samsung A50;
- Quantidade de lâminas usadas: 10;
- Quantidade total de imagens obtidas: 500;
- Quantidade de especialistas que participaram do processo de anotação: 7;
- Quantidade de imagens anotadas: 200;
- Quantidade de imagens de lâminas negativas: 100;
- Quantidade de bacilos identificados: 2941;
- Quantidade de clusters identificados: 204;
- Tamanho total do conjunto de dados: 258 MB.



	A	B	C	D	E	F
1	filename	xmin	ymin	xmax	ymax	label
2	S1F1.jpg	2024	2310	2098	2407	bacillus
3	S1F1.jpg	1079	2264	1144	2358	bacillus
4	S1F1.jpg	1768	2153	1843	2293	bacillus
5	S1F1.jpg	328	2205	430	2285	bacillus
6	S1F1.jpg	814	1805	901	1972	bacillus
7	S1F1.jpg	2018	1835	2147	1960	bacillus
8	S1F1.jpg	230	1724	353	1831	bacillus
9	S1F1.jpg	639	1580	766	1659	bacillus
10	S1F1.jpg	448	1367	526	1495	bacillus
11	S1F1.jpg	2110	1299	2237	1460	bacillus
12	S1F1.jpg	1993	1228	2083	1386	cluster
13	S1F1.jpg	648	1219	756	1356	bacillus
14	S1F1.jpg	1017	1185	1092	1256	bacillus
15	S1F1.jpg	1953	1102	2052	1203	bacillus
16	S1F1.jpg	783	1037	873	1195	bacillus
17	S1F1.jpg	368	800	443	885	bacillus
18	S1F1.jpg	291	735	372	815	bacillus
19	S1F1.jpg	1867	597	2003	736	cluster
20	S1F1.jpg	531	609	646	703	bacillus
21	S1F1.jpg	1744	535	1837	617	cluster
22	S1F1.jpg	1851	504	1984	605	bacillus
23	S1F1.jpg	713	430	796	558	bacillus
24	S1F1.jpg	817	372	873	484	bacillus
...						
3146	S8F25.jpg	78	123	240	217	bacillus
3147	S8F25.jpg	1541	130	1630	208	bacillus
3148	S8F25.jpg	1492	50	1605	129	bacillus

Figura 24 – (a) Arquivo CSV o qual encapsula as anotações referentes às imagens do banco de dados; (b) Representação gráfica de um plano cartesiano ilustrando as coordenadas armazenadas; e (c) Visualização da imagem *S4F10.jpg*, destacando as marcações contidas no arquivo CSV.

A Figura 25 ilustra algumas das imagens e suas anotações que compõem esse conjunto de dados.

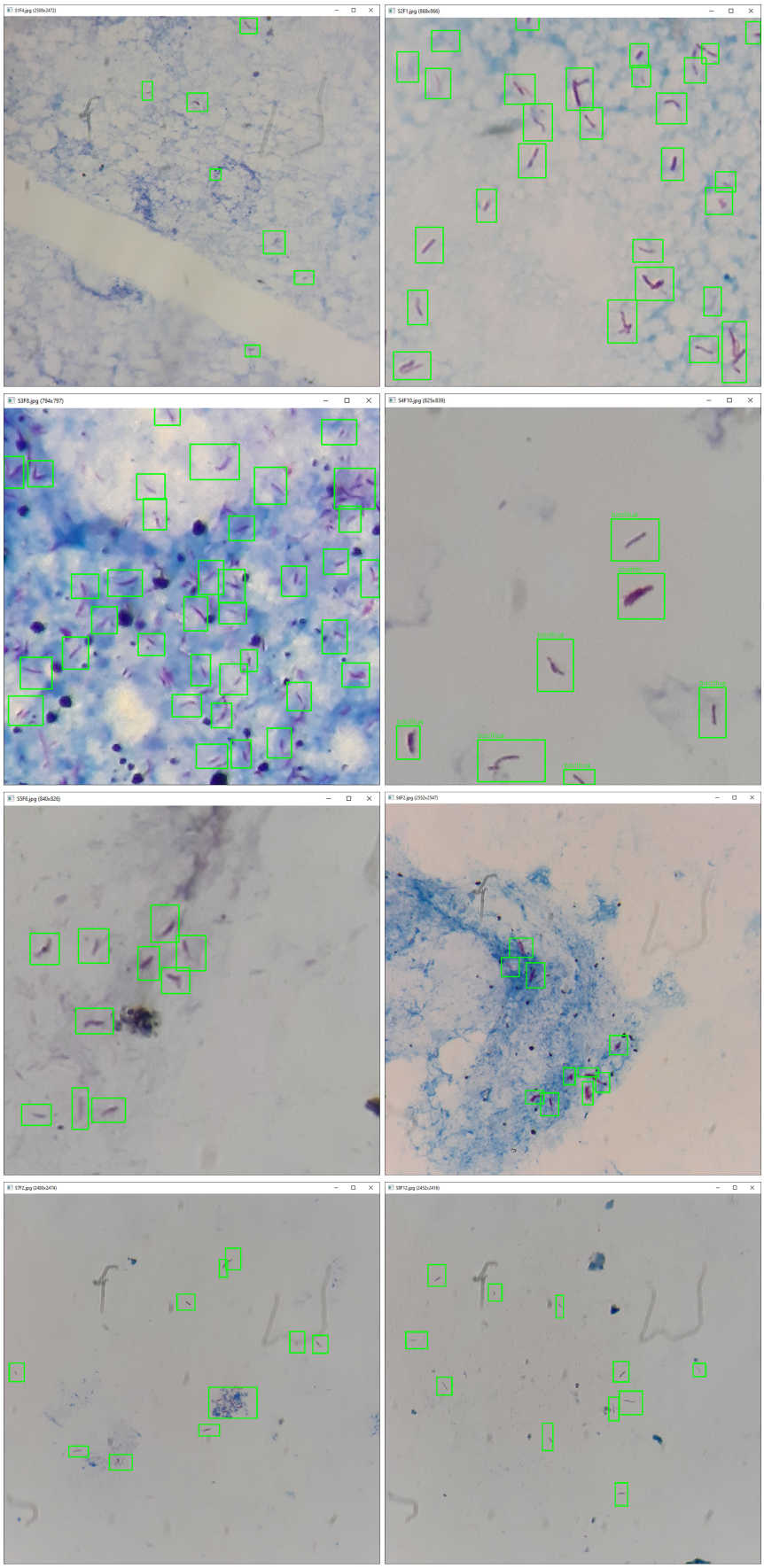


Figura 25 – Apresentação de uma imagem de cada uma das lâminas positivas que compõem o conjunto de dados.

5 Materiais e Métodos

Para viabilizar o uso prático de DL no processamento de imagens oriundas da baciloscopia, visando auxiliar no diagnóstico da TB, observa-se a necessidade de uma investigação mais aprofundada, do desenvolvimento de Sistema de Diagnóstico Auxiliado por Computador (CADx, do inglês *Computer-Aided Diagnosis*) e da ampliação da transparência processual. Neste contexto, este capítulo objetiva apresentar as abordagens metodológicas adotadas no presente estudo. Além de explorar variados recursos computacionais com o propósito de desenvolver e avaliar modelos de ML para a identificação do bacilo da TB em imagens adquiridas via baciloscopia pelo método ZN.

Antecipa-se que, por meio dos métodos empregados, aliados a um rigor científico e transparência, possam ser proporcionados novas percepções e a implementação de uma estratégia robusta para a detecção e quantificação dos bacilos da TB. Almeja-se, ademais, que essa estratégia robusta constitua a fundação para o desenvolvimento de um protótipo de sistema, equipado com uma interface intuitiva e usuário-amigável, permitindo, assim, a aplicação prática do DL em um laboratório dedicado ao diagnóstico de TB.

A Seção 5.1 aborda os conjuntos de dados utilizados, oferecendo um panorama sobre suas origens, características e o método como foram preparados para serem utilizados no contexto desta pesquisa. A Seção 5.2 apresenta um estudo de replicação de arquiteturas de CNNs descritas na literatura, destacando suas características e a padronização adotada na implementação e parametrização das redes. A Seção 5.3 propõe uma estratégia conjunta do algoritmo K-*Means*, empregado como método de segmentação, e CNN, utilizada como método de classificação, para detecção do bacilo, incorporando uma subseção exclusivamente dedicada ao refinamento dos dados para treinamento do modelo. A Seção 5.4 centraliza-se na aplicação de algoritmos de detecção de objetos amplamente reconhecidos na literatura. Adicionalmente, são propostas estratégias de partição de imagens para melhorar o desempenho da tarefa de detecção dos bacilos e permitir que os métodos se ajustem a diferentes tamanhos de imagens.

Posteriormente, a Seção 5.5 aborda a metodologia adotada para avaliar o desempenho dos métodos de DL ao lidar com a heterogeneidade de diferentes conjuntos de dados. Isto faz-se necessário, uma vez que a exploração entre treinamento e teste em diferentes conjuntos de dados é um experimento pouco explorado na literatura. A Seção 5.6 propõe uma estratégia para realizar a anotação automática dos dados que não foram anotados no conjunto de dados proposto no Capítulo 4, levando em consideração os modelos gerados pela Seção 5.5. Para tentar viabilizar o uso de métodos de DL, a Seção 5.7 apresenta a proposta de desenvolvimento de um protótipo de sistema de diagnóstico auxiliado por CADx para auxiliar na leitura das lâminas na baciloscopia. Reunindo as melhores técnicas

e métodos descritas nas seções anteriores para proporcionar uma ferramenta prática e eficiente para auxiliar profissionais no diagnóstico da TB. Por fim, a Seção 5.8 é proposto um processo para analisar como os métodos de DL se comportariam em um cenário de diagnóstico automatizado a partir da detecção e contagem dos bacilos da TB.

A cada seção, é construído um caminho lógico e científico que transita desde o dado bruto até uma possível aplicação na prática médica, pautando-se sempre em rigor científico. Buscando contribuir de maneira significativa para entender a complexidade, e avaliar o desempenho, dos métodos de DL para processar imagens da baciloscopia.

5.1 Conjuntos de dados

Conforme apresentado na Seção 3.2.2, foi identificado quatro trabalhos que propuseram conjuntos de dados que podem ser usados em estudos de identificação de bacilos de TB usando métodos de ML. Ao acessar os endereços eletrônicos disponíveis para download dos dados ou entrando em contato com os autores, foi possível baixar apenas os conjuntos de dados de Costa et al. (2014), Quinn et al. (2016), e Shah et al. (2017). Esses três conjuntos de dados juntamente com o proposto no Capítulo 4 foram usados para treinar e testar as os métodos de DL explorados neste trabalho.

Ao acessar o conjunto de dados proposto por Costa et al. (2014), encontram-se 120 imagens contendo anotações como bacilo verdadeiro, bacilo duvidoso e bacilos agrupados. Acessando o conjunto de dados proposto por Quinn et al. (2016), encontram-se 1265 imagens com suas respectivas anotações com as posições dos bacilos de TB. Acessando o conjunto de dados de Shah et al. (2017), encontram-se 300 imagens contendo anotações separadas em três grupos (MS-1, MS-2 e MS-3, relacionados ao tipo de microscópio/câmera que capturou a imagem). Nestas imagens, foram anotados fragmentos contendo um único bacilo, bacilos ocluídos, estruturas vermelhas não classificadas e artefatos.

Neste estudo, foram utilizadas todas as imagens e fragmentos anotados fornecidos por Costa et al. (2014) e Quinn et al. (2016). No conjunto de dados de Shah et al. (2017), foram usadas as imagens dos grupos MS-1 e MS-2, de maneira semelhante ao que foi feito em outros trabalhos Kant e Srivastava (2018), El-Melegy et al. (2019), El-Melegy, Mohamed e ElMelegy (2019). Já no conjunto de dados proposto no Capítulo 4 todas as 200 imagens anotadas foram utilizadas. Os fragmentos anotados como verdadeiro bacilo, único bacilo, bacilos ocluídos e bacilos agrupados (clusters) foram considerados como uma única classe de bacilo verdadeiro. Por simplicidade, os conjuntos de dados de Costa et al. (2014), Quinn et al. (2016), Shah et al. (2017) e o proposto no Capítulo 4 foram nomeados como dataset-1, dataset-2, dataset-3 e dataset-4, respectivamente.

5.1.1 Dados para treinamento/teste da CNN

Para gerar modelos de classificação usando CNN, é necessário ter um conjunto de dados contendo fragmentos que têm bacilos de TB (classe positiva) e fragmentos que não têm bacilos de TB (classe negativa). O dataset-1, dataset-2, dataset-3 e dataset-4 possuem mais de 8.000, 9.000, 1.700 e 3.000 fragmentos de imagem de classe positiva, respectivamente. Como não há anotação para os fragmentos negativos, para cada fragmento contendo bacilo, buscou-se gerar um fragmento (da mesma dimensão) que não contenha um bacilo (que não intersecta com nenhuma região anotada). A escolha deste fragmento foi realizada priorizando encontrar uma área próxima ao bacilo anotado, não sendo possível, uma região aleatória foi escolhida. A Figura 26 exemplifica esse processo. Esta abordagem procura prevenir a seleção de regiões contendo bacilos não anotados para compor o conjunto de fragmentos negativos em imagens com alta densidade de bacilos, baseando-se na premissa de que, ao identificar um bacilo em determinada área, é provável que o especialista tenha também reconhecido e anotado os bacilos adjacentes.

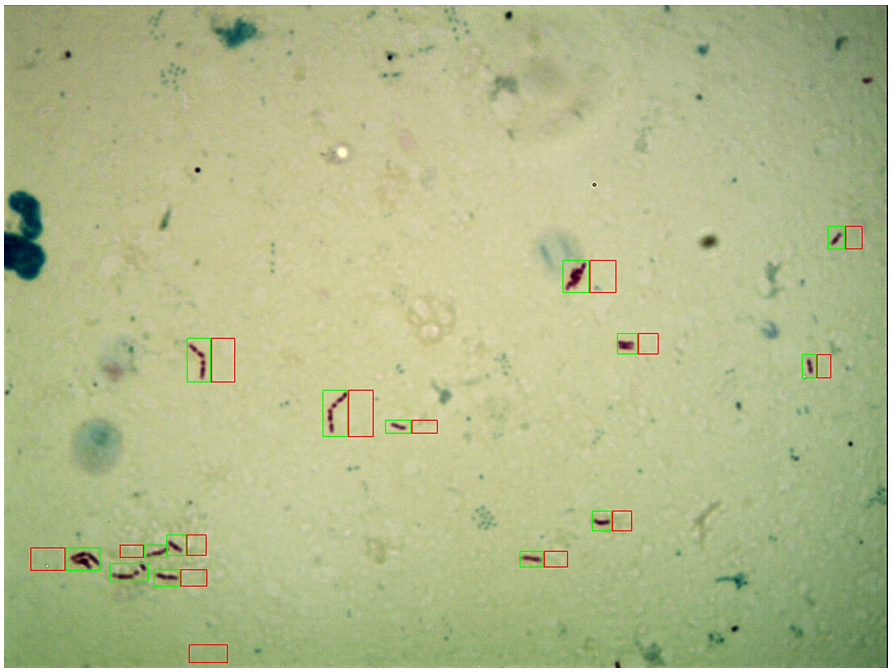


Figura 26 – Geração de fragmentos que não possuem bacilos (classe negativa, caixa vermelha) a partir da posição/dimensão de um fragmento que contém bacilos (classe positiva, caixa verde). Figura feita a partir de imagens do conjunto de dados de Shah et al. (2017).

Para avaliar as arquiteturas de CNN, optou-se por realizar a validação cruzada em 5 partes (*5-fold cross-validation*), sendo que 10% do conjunto separado para treinamento foi utilizado como conjunto de validação em cada *fold*. Em complemento, foi realizado o *data augmentation* para aumentar o número de fragmentos e proporcionar uma melhor generalização dos métodos utilizados (como realizado pela maioria dos trabalhos apresentados). A técnica de *data augmentation* usada neste estudo consiste em aplicar as

operações de rotação de 90°, 180° e 270°, e inverter horizontalmente, gerando, assim, sete novos fragmentos para cada fragmento de imagem original (como ilustrado na Figura 27). Essa técnica é utilizada antes da separação dos dados pela validação cruzada. Conforme mencionado anteriormente, as imagens foram divididas em cinco partes. No entanto, o número de fragmentos não segue necessariamente essa proporção, uma vez que existem imagens (campo de visão) com mais e menos bacilos. A Tabela 7 mostra o número médio de fragmentos (considerando os *5-fold*) que compõem cada conjunto de dados após o *data augmentation*.

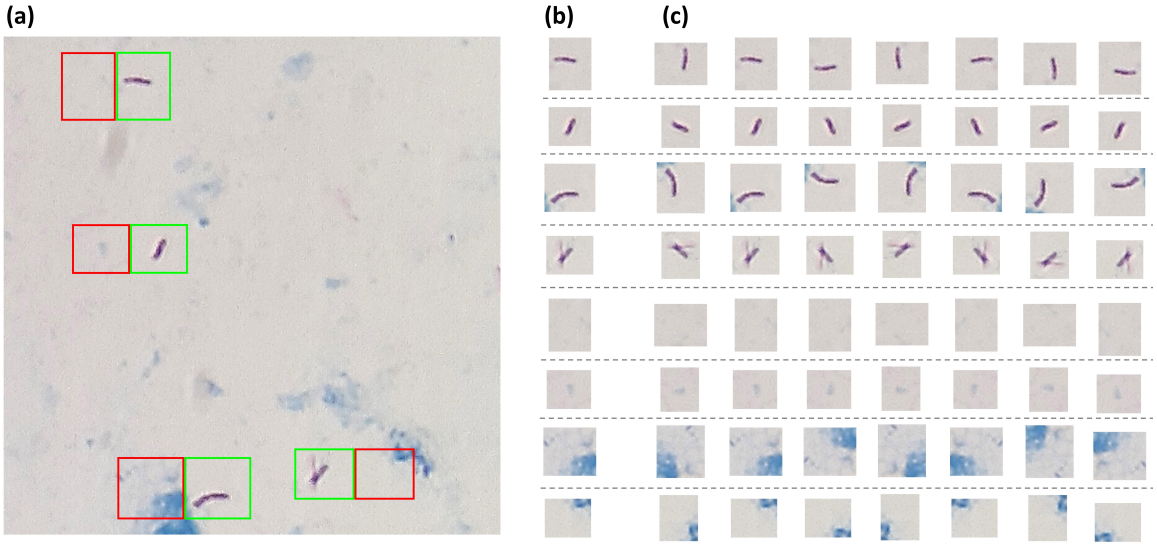


Figura 27 – Ilustração do processo de *data augmentation*, onde: (a) representa as posições dos bacilos (classe positiva, box verde) e as posições sem bacilos (classe negativa, box vermelho) em uma imagem do dataset-4; (b) 8 fragmentos de imagens extraídos da imagem original; (c) criação de 7 novas imagens a partir de cada fragmento extraído (aplicando operações de *data augmentation*), totalizando, assim, 64 imagens geradas para compor o conjunto de treinamento/teste.

Tabela 7 – Quantidades média (considerando os *5-fold* e o *data augmentation*) de fragmentos positivos (com bacilos) e negativos (sem bacilos) em cada conjunto de dados.

	dataset-1	**dataset-2**	**dataset-3**	**dataset-4**
Treino	55.506 positivos 55.506 negativos	56.739 positivos 56.739 negativos	10.149 positivos 10.149 negativos	17.934 positivos 17.934 negativos
Validação	3.330 positivos 3.330 negativos	6.397 positivos 6.397 negativos	1.314 positivos 1.314 negativos	2.194 positivos 2.194 negativos
Teste	14.709 positivos 14.709 negativos	15.784 positivos 15.784 negativos	2.866 positivos 2.866 negativos	5.032 positivos 5.032 negativos

5.1.2 Dados para treinamento/teste dos métodos de detecção

Diferente do apresentado anteriormente, o desenvolvimento e a avaliação de métodos para detecção de objetos requerem a utilização de imagens que retratam o campo

de visão integral, ao invés de apenas fragmentos que contenham ou não bacilos. Nesse contexto, é necessário utilizar as imagens completas contendo bacilos da TB, extraídas dos conjuntos de dados disponíveis. O dataset-1, dataset-2, dataset-3 e dataset-4 contêm, respectivamente, 120, 1218, 185 e 200 imagens com bacilos. Adicionalmente, além das imagens, é preciso contar com as anotações dessas imagens, que determinam as localizações dos bacilos por meio de caixas delimitadoras, a fim de treinar e testar os modelos.

Considerando os experimentos propostos nas seções subsequentes para avaliar os métodos de detecção do bacilo da TB, optou-se por utilizar técnicas diferentes de avaliação em experimentos distintos, conforme suas especificidades. Nesse sentido, os conjuntos de dados dataset-1, dataset-2, dataset-3 e dataset-4 foram subdivididos da seguinte maneira:

- *5-fold cross-validation*: De forma semelhante ao apresentado Subseção 5.1.1, para treinar e testar os métodos de detecção, adotou-se a estratégia de validação cruzada em 5 partes (*5-fold cross-validation*), onde 10% do conjunto destinado ao treinamento foi alocado como conjunto de validação em cada *fold*. Essa proporção de divisão dos dados baseia-se nas práticas observadas nos trabalhos discutidos na Seção 3 e na literatura. No entanto, a subdivisão dos dados e as operações de *data augmentation* foram executadas considerando as imagens integrais dos datasets (campo de visão), em não aos fragmentos. A Figura 28 ilustra as operações de *data augmentation* realizadas em uma imagem, enquanto a Tabela 8 apresenta a quantidade de imagens que compõem cada parte da validação cruzada.

- Treinamento/Validação/Teste: considerando as estratégias apresentadas na Subseção 3.2.3, este estudo também optou em alguns casos por usar uma abordagem de divisão convencional de dados para treinamento, validação e teste no contexto dos métodos de detecção. Assim sendo, o conjunto de dados foi subdividido da seguinte forma: 70% das imagens foram destinadas ao treinamento dos modelos, 10% para a fase de validação, e os 20% restantes foram reservados para avaliação da eficácia do modelo em dados não explorados previamente. Essa proporção de divisão dos dados baseia-se nas práticas observadas nos trabalhos discutidos na Seção 3 e na literatura. As operações de *data augmentation* também foram realizadas nesses dados (conforme Figura 28). A Tabela 9 apresenta a distribuição das imagens entre os conjuntos de treinamento, validação e teste em cada conjunto de dados.

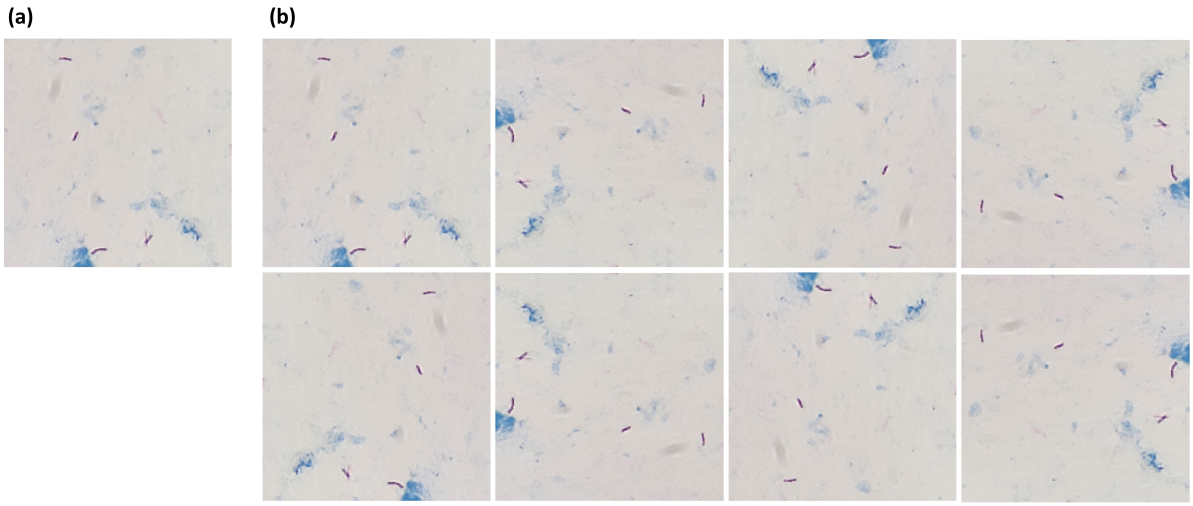


Figura 28 – Ilustração das operações de *data augmentation* aplicada em uma imagem para gerar um conjunto de 8 imagens, sendo que (a) contém a imagem a original e (b) contém a imagem original e 7 novas modificadas.

Tabela 8 – Quantidades de imagens em cada um dos *5-fold* de cada conjunto de dados (considerando o *data augmentation*).

	dataset-1	**dataset-2**	**dataset-3**	**dataset-4**
Treino	688	6.992	1.064	1.152
Validação	80	784	120	128
Teste	192	1.944	296	320

Tabela 9 – Quantidades de imagens em cada conjunto de dados ao ser separado em treinamento, validação e teste (considerando o *data augmentation*).

	dataset-1	**dataset-2**	**dataset-3**	**dataset-4**
Treino	672	6.816	1.032	1.120
Validação	88	968	144	152
Teste	192	1.952	296	320

5.2 Replicação de arquiteturas CNNs propostas na literatura

Como abordado na Capítulo 3, foi observado que poucos estudos fornecem os códigos-fonte e/ou conjuntos de dados utilizados para classificar o bacilo da TB. Este fenômeno contribui negativamente para que diferentes autores possam comparar suas técnicas e propor melhores estratégias para melhorar o estado da arte. Além disso, observou-se que os trabalhos que apresentaram melhor desempenho são aqueles baseados em CNN. Buscando entender o potencial e as limitações das técnicas de DL, desenvolver um protótipo de um CADx para baciloscopia, e entender o quão distantes essas técnicas estão de serem utilizadas em laboratórios, foi selecionado alguns trabalhos para serem reproduzidos neste trabalho. Dada a diversidade de métodos e trabalhos, e quão difícil pode ser reproduzi-los fielmente, focou-se em implementar/reproduzir arquiteturas CNNs. Além disso, os códigos-fonte implementados são fornecidos para incentivar futuras pesquisas na área.

5.2.1 Trabalhos selecionados

As arquiteturas CNNs propostas nos trabalhos de Quinn et al. (2016), Panicker et al. (2018), Simon et al. (2019), e Udegova, Shelomentseva e Chentsov (2021) foram selecionadas para serem reproduzidas e testadas neste trabalho. Tais trabalhos foram escolhidos porque respeitam os seguintes critérios estabelecidos:

- Possuir detalhes sobre os parâmetros de cada camada da CNN proposta: permite reproduzir o método de maneira similar ou igual àquela estabelecida pelos autores.;
- Deve ser possível reproduzir a arquitetura proposta usando a biblioteca TensorFlow V. 2.4.1 (com Keras V.2.4) (ABADI et al., 2015): estabelecer um único framework previne que diferenças entre bibliotecas disponíveis interfiram nos resultados obtidos. Permitindo, assim, avaliar a robustez da arquitetura proposta;
- Não combinou diferentes arquiteturas ou métodos: métodos híbridos frequentemente possuem implementações mais complexas, o que pode dificultar a reprodução do artigo de maneira confiável. Isso poderia levar a uma interpretação errônea dos resultados e impedir uma comparação justa;
- Não usou pesos pré-treinados: permite evitar a escolha de pesos diferentes dos autores originais, que consequentemente poderiam gerar modelos com desempenho inferior aos propostos.

Tais critérios visam facilitar a implementação dos métodos propostos, realizar os experimentos computacionais em um tempo computacionalmente viável e permitir que a reprodução/comparação dos trabalhos seja realizada da maneira mais justa e igualitária possível.

Vale mencionar que, sem ter acesso aos códigos-fonte originais dos trabalhos selecionados, é difícil reproduzi-los com precisão. Dessa forma, analisando as descrições das arquiteturas fornecidas e utilizando os recursos do TensorFlow, buscou-se aproximar o máximo possível das arquiteturas CNNs propostas.

5.2.1.1 Arquiteturas

Considerando os trabalhos selecionados, as camadas das arquiteturas apresentadas pelos autores podem ser descritas ou resumidas como:

- Quinn et al. (2016): usaram redes com quatro camadas ocultas: (i) Camada de convolução: 7 filtros de tamanho 3×3; (ii) Camada de pooling: max-pooling, fator 2; (iii) Camada de convolução: 12 filtros de tamanho 2×2 e (iv) Camada totalmente conectada, com 500 unidades ocultas.

- Panicker et al. (2018): propuseram uma arquitetura final que consiste em 2 camadas de convolução com 32 mapas de filtro (3 × 3) seguidos por 3 camadas de convolução com 64 filtros (3 × 3) e finalmente 1 camada de convolução com 128 filtros (3 × 3). A saída desta série de camadas de convolução é alimentada em uma camada totalmente conectada com 128 neurônios com ativação *Rectified Linear Unit* (ReLU) e, finalmente, um neurônio para classificação binária (com uma função sigmoid). Cada camada de convolução é seguida por camadas de normalização, ativação e *max pooling*.

- Simon et al. (2019): propuseram uma arquitetura CNN reduzida com uma camada de convolução com 7 mapas de filtro (3 × 3) seguida pela camada de *pooling* (2 × 2) acoplada com uma camada LSTM.

- Udegova, Shelomentseva e Chentsov (2021): usaram uma arquitetura que consiste em 3 camadas de convolução com 32 mapas de filtro, onde os dois primeiros são seguidos por uma camada de *max-pooling* e o terceiro é seguido por uma camada de *pooling* médio. Então, existem quatro etapas de *dropout* (com probabilidade 0,5; 0,5; 0,4; 0,3), onde cada uma é seguida por uma camada totalmente conectada (512; 254; 64 e 2 unidades ocultas).

Usando TensorFlow, visou-se chegar o mais próximo possível das arquiteturas descritas nos trabalhos selecionados. As Figuras 29, 30, 32 e 31 ilustram as arquiteturas implementadas.

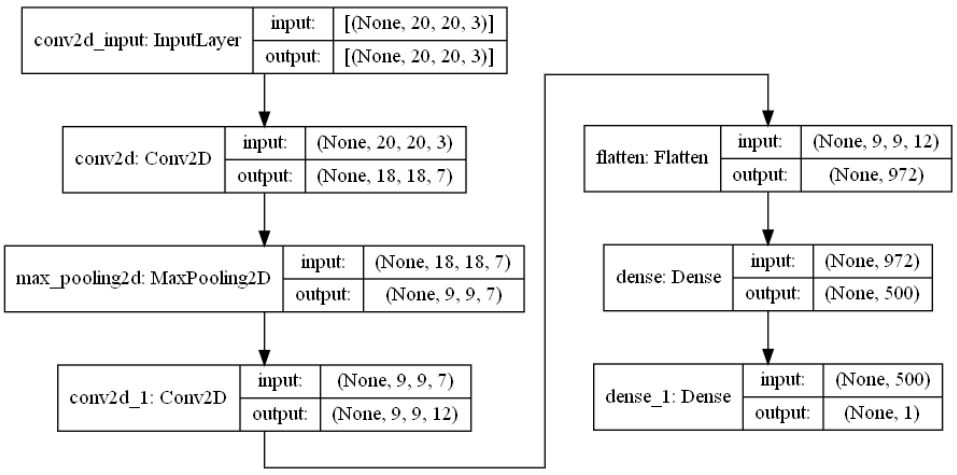


Figura 29 – Arquitetura construída usando TensorFlow, baseada no trabalho de Quinn et al. (2016). Imagem gerada pela função *tensorflow.keras.utils.plot_model* do TensorFlow.

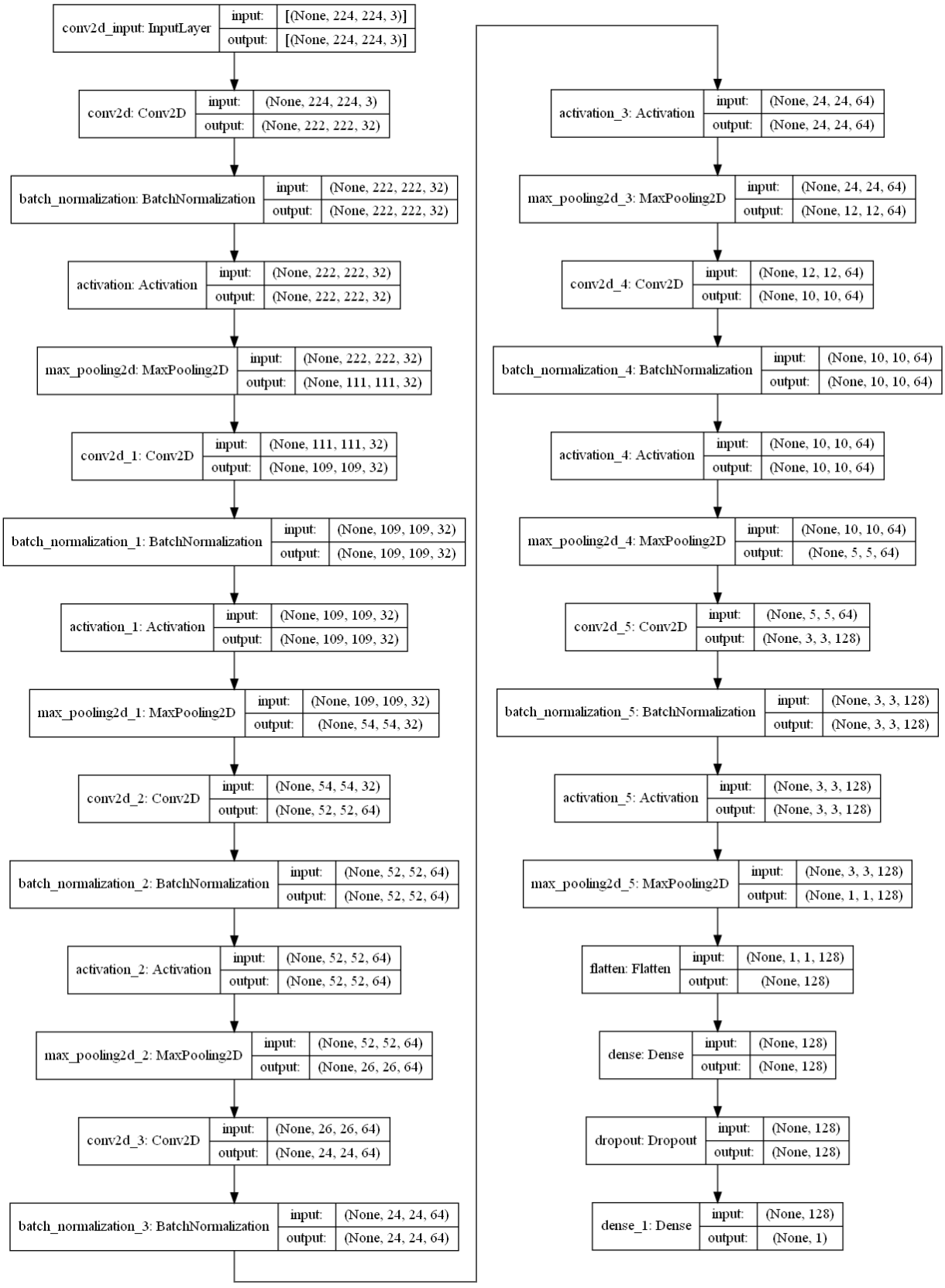


Figura 30 – Arquitetura construída usando TensorFlow, baseada no trabalho de Panicker et al. (2018). Imagem gerada pela função *tensorflow.keras.utils.plot_model* do TensorFlow.

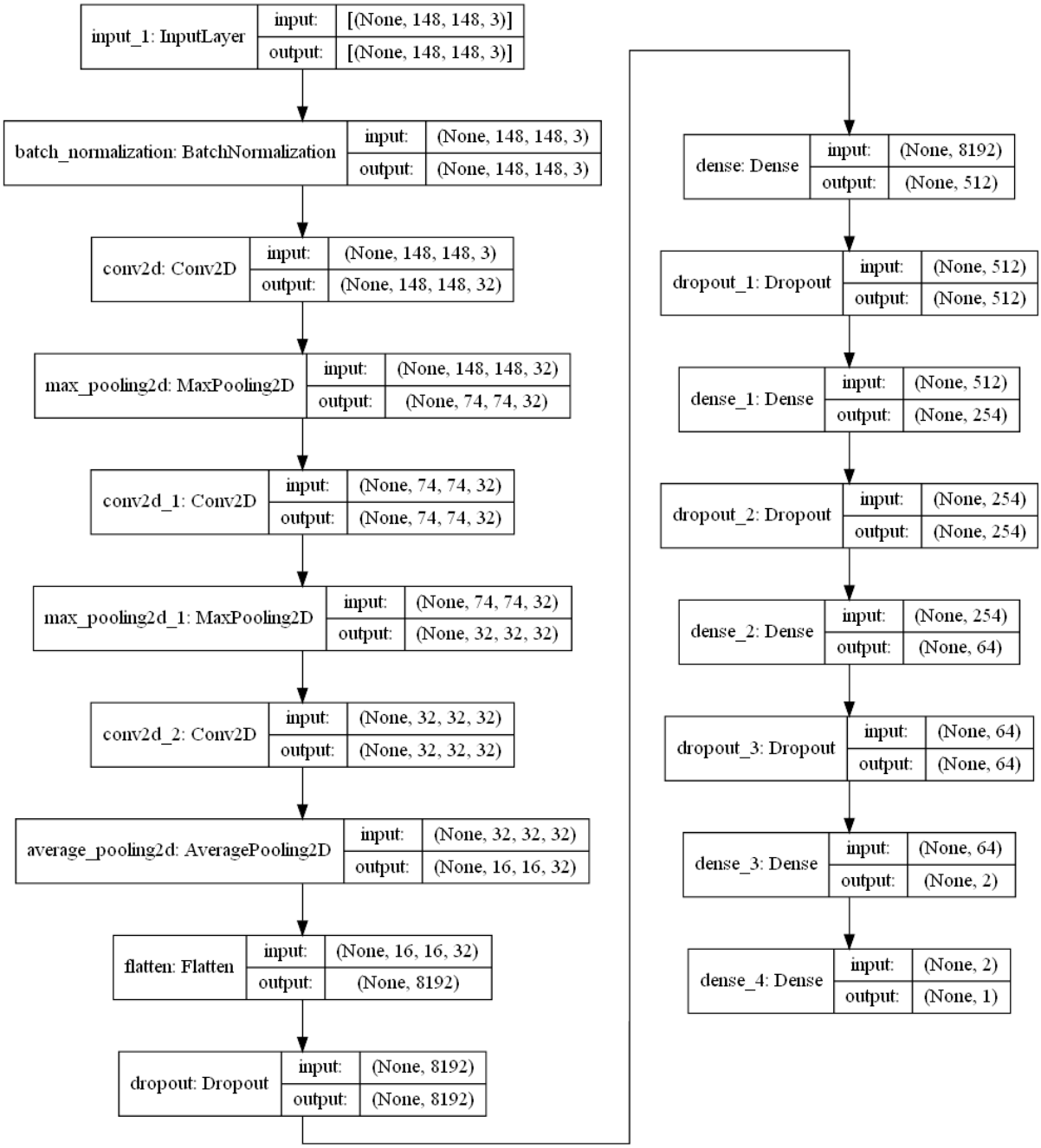


Figura 31 – Arquitetura construída usando TensorFlow, baseada no trabalho de Udegova, Shelomentseva e Chentsov (2021). Imagem gerada pela função *tensorflow.keras.utils.plot_model* do TensorFlow.

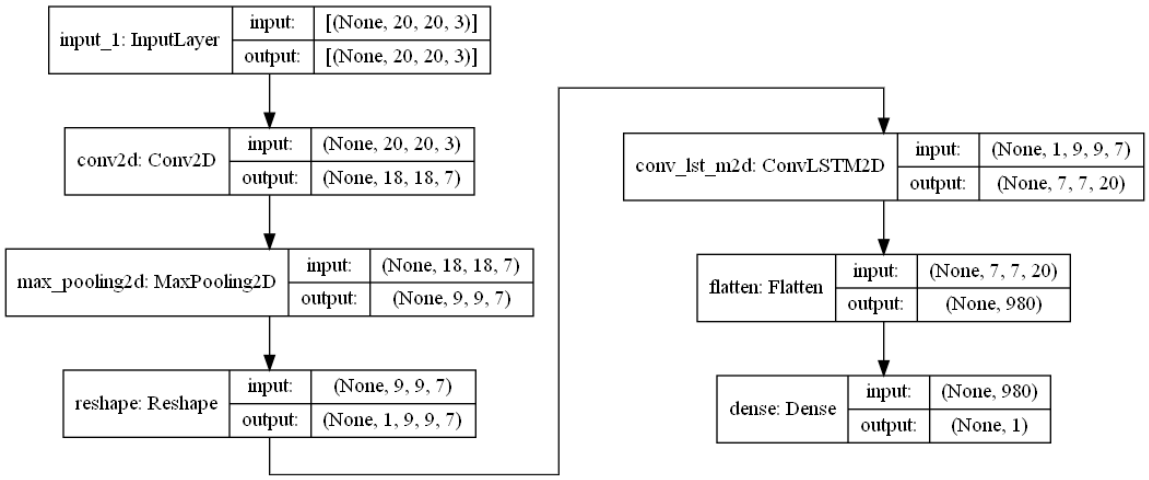


Figura 32 – Arquitetura construída usando TensorFlow, baseada no trabalho de Simon et al. (2019). Imagem gerada pela função *tensorflow.keras.utils.plot_model* do TensorFlow.

5.2.2 Implementação e avaliação

Cada autor propôs diferentes estratégias para definir alguns parâmetros de treinamento. Portanto, para obter uma comparação mais justa, foi estabelecido uma lista de parâmetros a serem usados por todas as CNNs. Eles são: (i) o número de épocas igual a 100; (ii) lotes de dados de tamanho igual a 25; (iii) Gradiente Descendente Estocástico (SGD, do inglês *Stochastic Gradient Descent* - SGD) como otimizador, com taxa de aprendizado = 0,001 e momento = 0,9; (iv) passos por época igual ao tamanho dos dados de entrada; e (v) *seed* para realizar o treinamento embaralhando a ordem da imagem definida como 42. Para as outras configurações, foram utilizados os parâmetros padrão do TensorFlow. As imagens de entrada da CNN foram redimensionadas para o tamanho definido por cada autor antes do treinamento. Mais informações podem ser conferidas no código-fonte disponível no Apêndice B.1.

A implementação das CNNs foi realizada utilizando Python V.3.8.5, com TensorFlow V.2.4.1 (ABADI et al., 2015), Keras V.2.4.0 (CHOLLET et al., 2015), Scikit-learn V.1.0.2 (PEDREGOSA et al., 2011), entre outras. Essas bibliotecas foram instaladas a partir da plataforma Anaconda (ANACONDA, 2020).

Para avaliar as arquiteturas implementadas, foi realizado o processo de *5-fold cross-validation*, conforme descrito na Seção 5.1.1. Após treinar o modelo, o seu desempenho na classificação foi mensurado usando as métricas apresentadas na Seção 3.2.3. Cada arquitetura foi testada em cada um dos conjuntos de dados disponíveis. O resultado dessas avaliações são apresentadas na Seção 6.1.

5.3 Detecção do bacilo utilizando K-Means e CNN

A literatura apresenta evidências da robustez e alta acurácia alcançada pelas CNNs no processo de identificação do bacilo em fragmentos obtidos de imagens da baciloscopia. Contudo, apesar de sua eficácia na classificação de fragmentos de imagem previamente conhecidos e delimitados em testes *in silico*, essas redes não são diretamente utilizadas na detecção, segmentação e contagem do bacilo em imagens de grandes dimensões que retratam o campo de visão da baciloscopia. Dessa forma, torna-se imperativo introduzir uma fase preliminar de segmentação, afim de identificar Regiões de Interesse (ROIs, od inglês *Regions of Interest*), onde a presença do bacilo é provável ou possível. Como explicado na Seção 3.2.1, Figura 12.

A técnica de agrupamento K-*Means* pode ser uma estratégia nesse cenário, pois, tem a capacidade de subdividir a imagem em segmentos ou clusters, mediante a similaridade dos pixels, permitindo, assim, o isolamento de áreas potenciais para a detecção do bacilo. Ao implementar o K-*Means* como método preliminar de segmentação, não apenas facilita-se a identificação de regiões onde o bacilo está presente, mas também viabiliza-se a avaliação do desempenho subsequente da CNN. Isso ocorre ao isolar estas regiões e proporcionar um contexto focado e menos suscetível a ruídos e interferências para a atuação da rede neural.

Neste contexto, esta seção tem como foco a minuciosa descrição da implementação, integração e avaliação dos métodos K-*Means* e CNNs. O objetivo principal é realizar uma avaliação criteriosa das CNNs em um cenário específico de detecção do bacilo da TB, utilizando imagens provenientes da baciloscopia. Além disso, busca-se avaliar a viabilidade da integração dessa estratégia no protótipo de um sistema CADx proposto.

5.3.1 Processo de segmentação e classificação

No contexto da segmentação de imagens, o algoritmo K-*Means* é uma ferramenta eficaz para criar máscaras que diferenciam objetos ou regiões específicas, segmentando pixels com base em suas características no espaço de cor. Esse método torna-se particularmente relevante ao considerar métodos de coloração como o ZN, em que os bacilos da TB são coloridos em um vermelho brilhante que contrasta fortemente com o fundo azul. Ao empregar o K-*Means* para segmentar a imagem com base nas intensidades dos canais de cor, é possível identificar, de maneira distinta, tonalidades pixels que podem pertencer aos bacilos e ao fundo, devido à diferença marcante em suas características de cores (cromáticas). Diante desse contexto, aptou-se usar o K-*Means* como técnica de segmentação, propondo, assim, uma método de detecção do bacilo da TB com CNN.

O procedimento de detecção compreendido pela segmentação usando o K-*Means* e pela classificação via CNN, proposto neste estudo, é delineado em fases sequenciais (como ilustrado no fluxograma da Figura 33):

- Inicialmente, uma imagem que representa um campo de visão (Figura 33a) é submetida a uma operação que intensifica sua saturação, com o objetivo de realçar a coloração dos bacilos (Figura 33b);
- Com o objetivo de minimizar a quantidade de informações ou objetos na imagem e, consequentemente, diminuir o tempo computacional para o processamento da imagem, é aplicada uma filtragem por cor (Figura 33c). Essa filtragem, realizada no espaço de cores *Hue-Saturation-Value* (HSV), define intervalos de cores que correspondem à tonalidade vermelha/rosa/roxo que o bacilo da TB adquire durante o processo de coloração da baciloscopia. Esses intervalos foram estabelecidos mediante observações manuais e são ilustrados na Figura 34;
- Subsequente às etapas iniciais de processamento da imagem, emprega-se o método K-*Means* (disponível na biblioteca OpenCV (BRADSKI, 2000) para determinar 5 centroides, representando 5 cores *Red-Green-Blue* (RGB) (ou seja, $K = 5$) (Figura 33d);
- Cada centroide gera uma nova imagem conforme as seguintes sub-etapas: (i) binarização, utilizando como *threshold* o valor RGB associado ao centroide (Figura 33e); (ii) detecção de borda, empregando os métodos *Canny*, *dilate* e *findContours* da biblioteca OpenCV (Figura 33f);
- Para cada uma das imagens previamente gerada, o método *findContours* da biblioteca OpenCV é utilizado para obter as coordenadas dos objetos presentes na imagem (Figura 33h), possibilitando, assim, a obtenção dos fragmentos de ROIs da imagem;
- Tendo os fragmentos de imagem em mãos, uma filtragem por tamanho é aplicada, na qual o fragmento não pode exceder 0,5% da área total da imagem (Figura 33i);
- Cada fragmento de imagem criado é, então, submetido a um modelo gerado por uma CNN, previamente treinado para identificar bacilos da TB (Figura 33k);
- No decorrer do procedimento de detecção de objetos, pode ocorrer sobreposição de fragmentos quando os centroides apresentam cores similares. A fim de prevenir a identificação e contagem duplicada de um bacilo, o algoritmo *Non-Maximum Suppression* é aplicado (Figura 33l);
- Finalmente, as coordenadas dos fragmentos gerados e classificados como bacilos pela CNN são utilizadas para destacar a localização dos bacilos na imagem que representa o campo de visão da baciloscopia.

Devido à extensão de parâmetros, as bibliotecas e funções necessárias para realização das operações apresentadas anteriormente foram omitidas no texto. Entretanto, tais informações podem ser conferidas no código-fonte disponível no Apêndice B.2.

A Figura 35 ilustra de forma concisa e visual esse processo, onde: (a) representa a imagem de entrada, exemplificando um campo de visão; (b) exibe a imagem resultante após o processamento, que inclui o aumento de saturação e a filtragem por cor; (c, d, e) mostram as imagens resultantes após a aplicação do K-*Means* com $K = 2$ e binarização, utilizando os valores dos centroides como *threshold*; (f) destaca, por meio de caixas em vermelho, os elementos identificados na imagem a partir das técnicas de detecção de bordas e objetos; (g) fragmentos de imagem obtidos das imagens segmentadas na etapa anterior, os quais foram filtrados pelo tamanho; (h) etapa em que os fragmentos são classificados utilizando CNN e união das caixas que se interceptam; finalmente, (i) apresenta a imagem resultante do processo de detecção, na qual os fragmentos classificados como bacilos são destacados por caixas em verde na imagem de entrada.

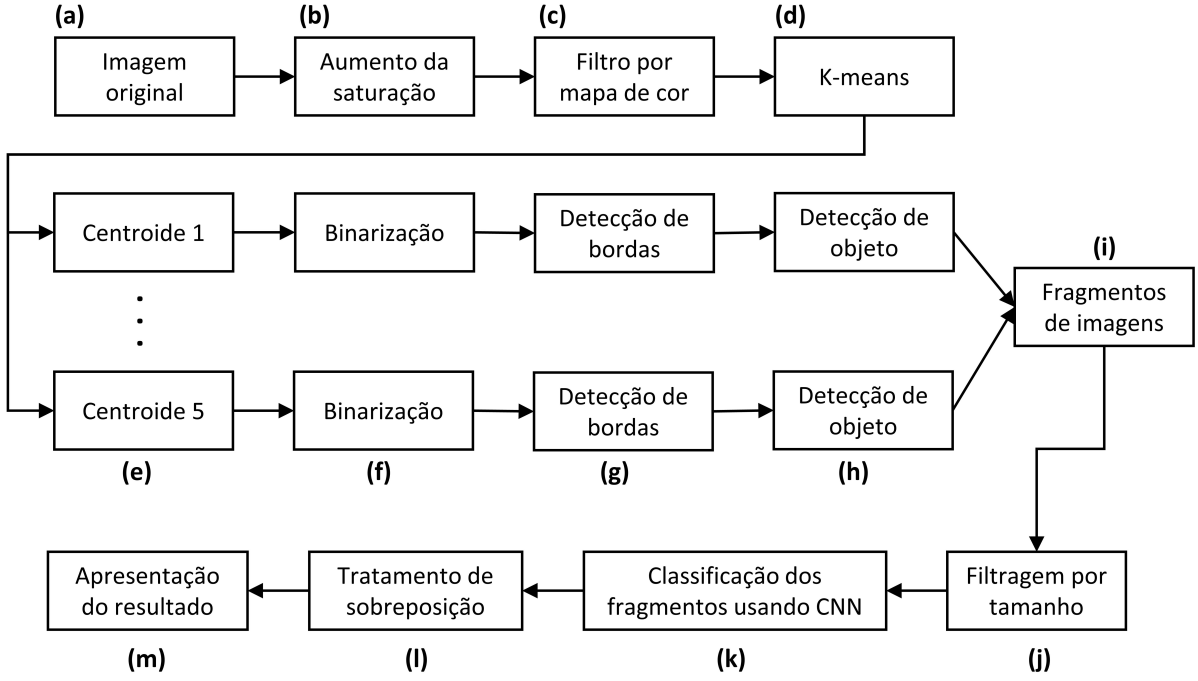


Figura 33 – Método proposto para detecção do bacilo da TB utilizando o K-*Means* para segmentação da imagem e CNN para classificação.

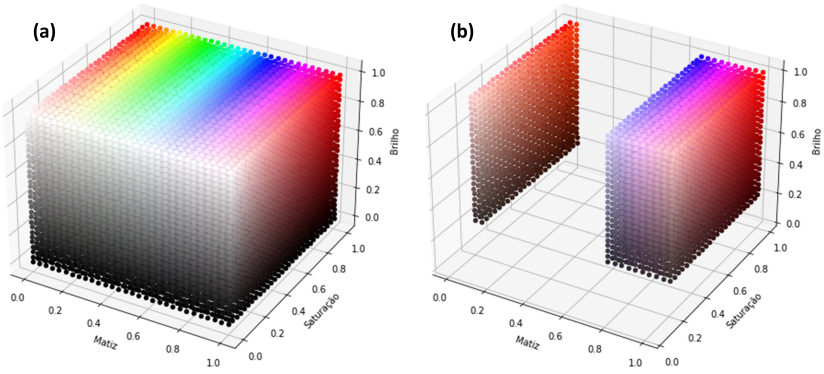


Figura 34 – Ilustração do espaço de cores HSV, onde: (a) representação do espaço HSV, e (b) intervalo de cores utilizado na filtragem realizada em Figura 33c.

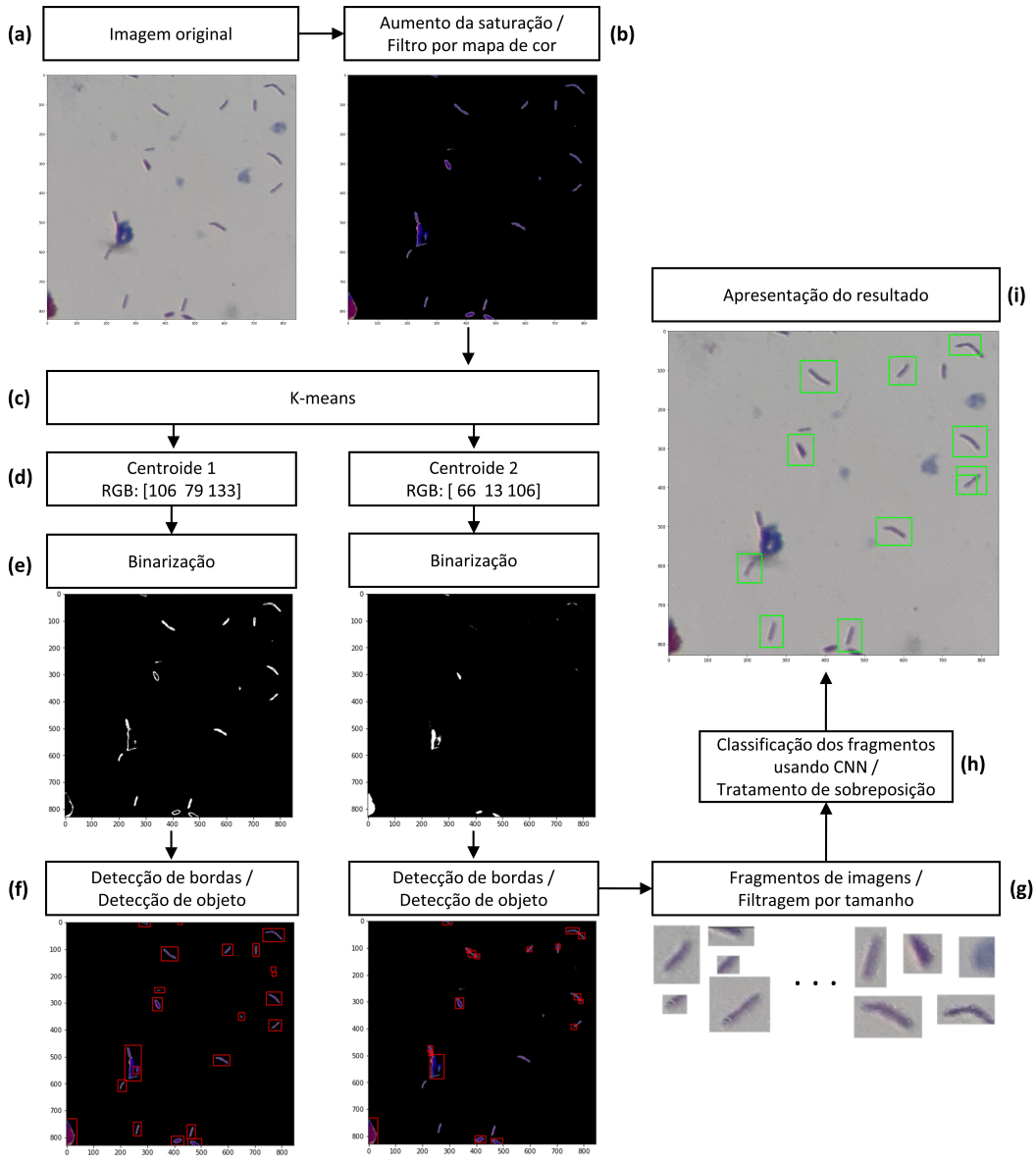


Figura 35 – Representação gráfica que destaca algumas das imagens resultantes em cada fase do método proposto.

5.3.2 Aperfeiçoamento dos dados de treinamento

Durante os ensaios preliminares que envolveram a aplicação do algoritmo K-*Means* e CNN, observou-se uma tendência de superestimação dos modelos gerados pelas CNNs. Essa tendência foi identificada como sendo resultado da representação inadequada da classe negativa (não-bacilo) nos conjuntos de dados. A análise aprofundada desse comportamento está detalhada nas Subseções 6.2 e 6.3. Para mitigar essa questão, propõe-se uma estratégia que envolve a identificação e expansão de padrões negativos relevantes, seguida pelo retreinamento dos modelos.

A estratégia utilizada para criar o conjunto de dados representativo que objetiva melhorar a generalização dos modelos gerados pela CNN é composta por quatro etapas, são elas (conforme ilustrado na Figura 36):

- No primeiro treinamento, a etapa inicial compreende um procedimento padrão de treinamento. Nesse processo, o conjunto de dados é dividido em conjuntos distintos para treinamento/validação e teste (Figura 36a). Os fragmentos de imagens provenientes do conjunto de treinamento/validação são extraídos (Figura 36b) e utilizados para gerar o "Modelo CNN", um modelo elaborado para classificar imagens em "bacilos"e "não-bacilos"(Figura 36c, d). É relevante ressaltar que, nessa fase, os fragmentos classificados como positivos e negativos são gerados conforme detalhado na Subseção 5.1.1;

- A fase de Identificação de Falsos Positivos, realizada subsequente, se dedica à identificação dos padrões (fragmentos de imagem) que o modelo não é capaz de classificar corretamente. Para tal, as imagens do campo de visão pertencentes ao conjunto de treinamento/validação (Figura 36e) são processadas pelo K-*Means* (Figura 36f), com o objetivo de extrair as ROIs (Figura 36g). Esses fragmentos de interesse são, então, classificados pelo "Modelo CNN", construído na fase anterior (Figura 36h). Subsequentemente, todos os fragmentos erroneamente classificados como bacilos (falsos positivos) são separados para próxima etapa (Figura 36i).

- Na fase do segundo treinamento, os fragmentos de imagem do conjunto de treinamento/validação inicial (Figura 36b) são unificados aos identificados como falsos positivos (Figura 36i), originando um novo conjunto de treinamento/validação, mais representativo e robusto (Figura 36j). Utilizando esse conjunto, um novo treinamento é executado (Figura 36l), resultando na criação de um modelo de classificação aprimorado, denominado "Modelo CNN*"(Figura 36m);

- A avaliação do modelo gerado, "Modelo CNN*", é realizada utilizando-se as imagens de campo de visão separados no conjunto de dados de teste (Figura 36a, n). Para tal, cada imagem do conjunto é submetida ao processamento via K-*Means* para extração

das ROIs (Figura 36o), e cada fragmento gerado (Figura 36p) é posteriormente classificado pelo refinado "Modelo CNN*" (Figura 36q). Por fim, para quantificar a eficácia do "Modelo CNN*", as posições dos fragmentos categorizados como "bacilo" e "não bacilos" são confrontadas com as anotações do conjunto de teste, e as métricas detalhadas na Seção 3.2.3, então, são calculadas (Figura 36r).

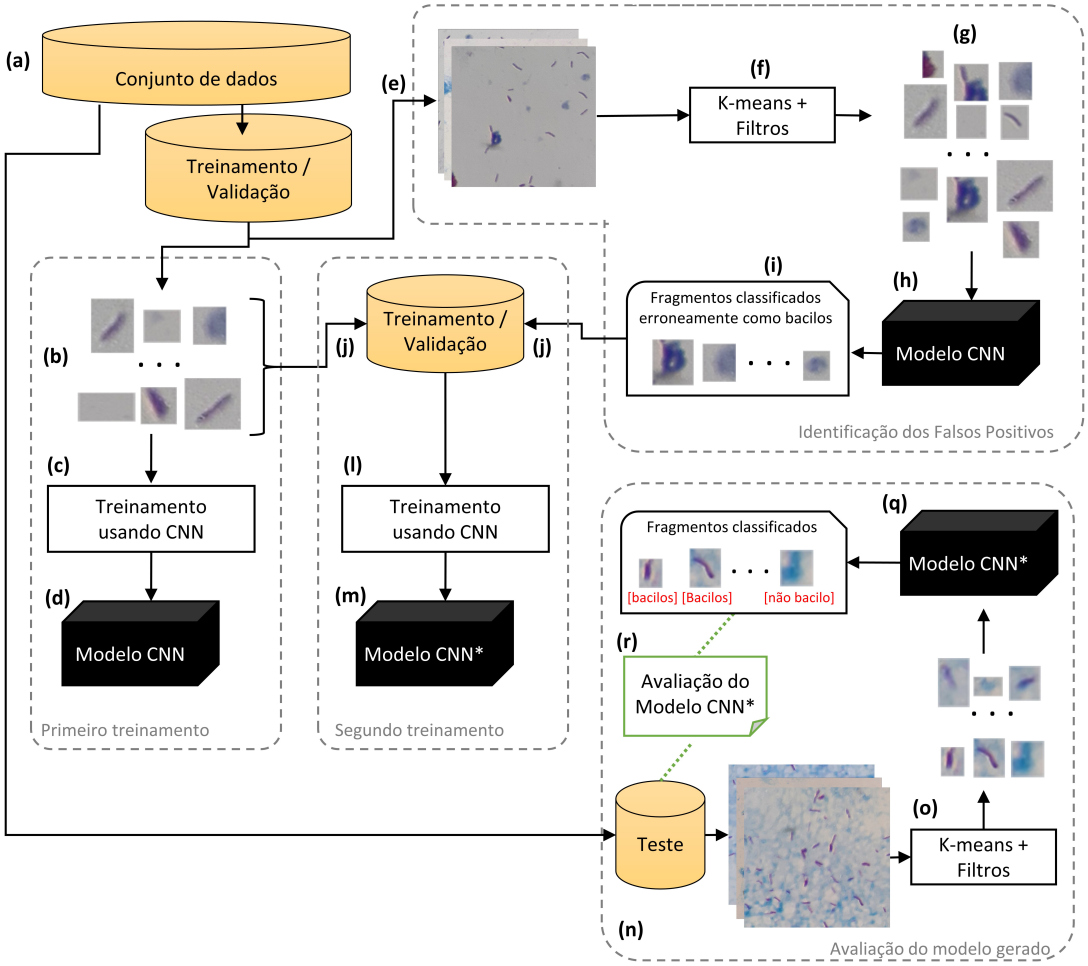


Figura 36 – Esquema ilustrativo da estratégia empregada para expandir um conjunto de dados para se tornar mais representativo, composto por quatro etapas, visando aprimorar a generalização dos modelos gerados pela CNN.

Por meio deste fluxo de trabalho proposto, viabiliza-se a expansão dos conjuntos de dados, tornando-os mais representativos e, consequentemente, aprimora-se o desempenho das CNNs na tarefa de detecção do bacilo da TB.

5.3.3 Implementação e avaliação

Conforme discutido anteriormente, diversos parâmetros foram ajustados empiricamente para a implementação dos métodos propostos nesta seção. Os códigos-fonte, juntamente com os parâmetros utilizados, são detalhadamente apresentados no Apêndice

B.2. Vale ressaltar que, em todos os experimentos conduzidos, adotou-se $K = 5$ para o algoritmo K-*Means*, uma vez que testes empíricos indicaram que este valor que oferece o melhor *tradeoff* entre custo computacional e desempenho de classificação.

No método descrito, os fragmentos resultantes da segmentação da imagem mediante o uso do K-*Means* são submetidos à classificação por uma CNN, a qual é treinada para categorizar as imagens como "bacilo"ou "não bacilo". Contudo, não foi especificada a arquitetura que é empregada na CNN. Neste contexto, com o objetivo de avaliar a eficácia da integração do K-*Means* com a CNN, é adotada a arquitetura do modelo que mostrar melhor desempenho nos quatro conjuntos de dados disponíveis, conforme avaliação proposta na Seção 5.2.

Dentre as principais linguagens e bibliotecas utilizadas na implementação do método proposto destaca-se: Python V.3.8.5, com TensorFlow V.2.4.1 (ABADI et al., 2015), Keras V.2.4.0 (CHOLLET et al., 2015), Scikit-learn V.1.0.2 (PEDREGOSA et al., 2011), Pillow V.9.0.1 (CLARK, 2022), OpenCV V.4.5.4 (BRADSKI, 2000), entre outras. Essas bibliotecas foram instaladas a partir da plataforma Anaconda (ANACONDA, 2020).

Para avaliar o K-*Means* juntamente com a CNN, foi realizado o processo de *5-fold cross-validation*, conforme descrito na Seção 5.1.1. Após treinar o modelo, o seu desempenho na classificação foi mensurado usando as métricas apresentadas na Seção 3.2.3. Cada estratégia apresentada foi testada em cada um dos conjuntos de dados disponíveis. Da mesma forma, para avaliar o K-*Means* juntamente com a estratégia de aprimorar o treinamento da CNN, foi utilizado o *5-fold cross-validation*, entretanto os conjuntos de dados apresentados na Seção 5.1.1 foram ampliados, conforme descrito na Subseção 5.3.2. O resultado dessas avaliações são apresentadas nas Seções 6.2 e 6.3.

5.4 Algoritmos de detecção de objetos

Conforme apresentado na Seção 3.2.1, diversas estratégias evidenciadas na literatura englobam a utilização de uma técnica destinada à segmentação de imagem, concomitantemente a outra destinada à classificação. Contudo, existem certos métodos de DL que possuem a capacidade de executar ambas as tarefas de maneira simultânea. Estes métodos têm sido objeto de extensivos estudos no contexto do diagnóstico médico, abrangendo áreas como neurologia, pneumologia, patologia digital, mamografia, cardiologia e imagens radiográficas, entre outras. É importante destacar que a acurácia desses métodos pode, em determinadas instâncias, ultrapassar a de clínicos experientes, conforme evidenciado por (LITJENS et al., 2017), (LIU et al., 2019) e (RAHMAT; ISMAIL; ALIMAN, 2019).

Nesta seção, apresentam-se sete abordagens de detecção que foram treinadas para realizar a detecção do bacilo da TB nos conjuntos de dados disponíveis. Tais métodos

pertencem a uma classe de algoritmos dedicadas à detecção de objetos em imagens (e vídeos) na área de DL e visão computacional amplamente utilizados na literatura. Os seguintes métodos foram usado:

- Faster R-CNN (*Region-based Convolutional Neural Networks*);
- YOLOF (*You Only Look One-level Feature*);
- YoloV5 (*YOLO version 5*);
- FSAF (*Feature Selective Anchor-Free Module for Single-Shot Object Detection*);
- SSD (*Single Shot MultiBox Detector*);
- EfficientDet;
- RetinaNet.

Além de serem amplamente utilizados, estes métodos foram selecionados neste estudo por estarem disponíveis em um mesmo framework e compartilharem a mesma base de bibliotecas em seu desenvolvimento (como pode ser visto na Subseção 5.4.2), fornecendo, assim, uma comparação mais justa e precisa. Em complemento, tais métodos visam realizar a detecção de objetos de forma rápida, característica importante no desenvolvimento de um CADx a ser usado de forma integrada ao microscópio.

Levando em consideração o processo de classificação de imagem apresentado na Seção 3.2.1, para treinar e avaliar os métodos de detecção, foram adotados os seguintes passos: (i) dividir os dados em conjuntos de treinamento/validação e teste (Figura 37a); (ii) utilizar as imagens que contêm os campos de visão e suas respectivas anotações (Figura 37b) para conduzir o treinamento do método de DL (Figura 37c), com o objetivo de desenvolver um modelo de detecção (Figura 37d); (iii) na etapa de teste, processar as imagens do conjunto de teste (Figura 37e) com o modelo treinado para detectar os bacilos (Figura 37f); e (iv) utilizar as anotações do conjunto de teste e as posições das regiões identificadas como bacilos pelo modelo para avaliar a acurácia do método (Figura 37g). Cada um desses passos foi executado para cada método mencionado nesta seção.

Dada a pertinência desses métodos no cenário de detecção de objetos e a escassa literatura que os comparam na identificação do bacilo da TB, é indispensável conduzir novos experimentos computacionais. Ainda que trabalhos, como Kuok et al. (2019), Reist et al. (2021) e Angayarkanni et al. (2022), apresentem métodos como Faster-RCNN, RetinaNet e UNet no contexto de detecção do bacilo da TB, o presente estudo propõe testes mais abrangente (em diversos conjuntos de dados) e rigorosos (sob mesmas condições/parâmetros), com objetivo de explorar aprofundadamente a aplicação de DL no apoio ao diagnóstico da TB.

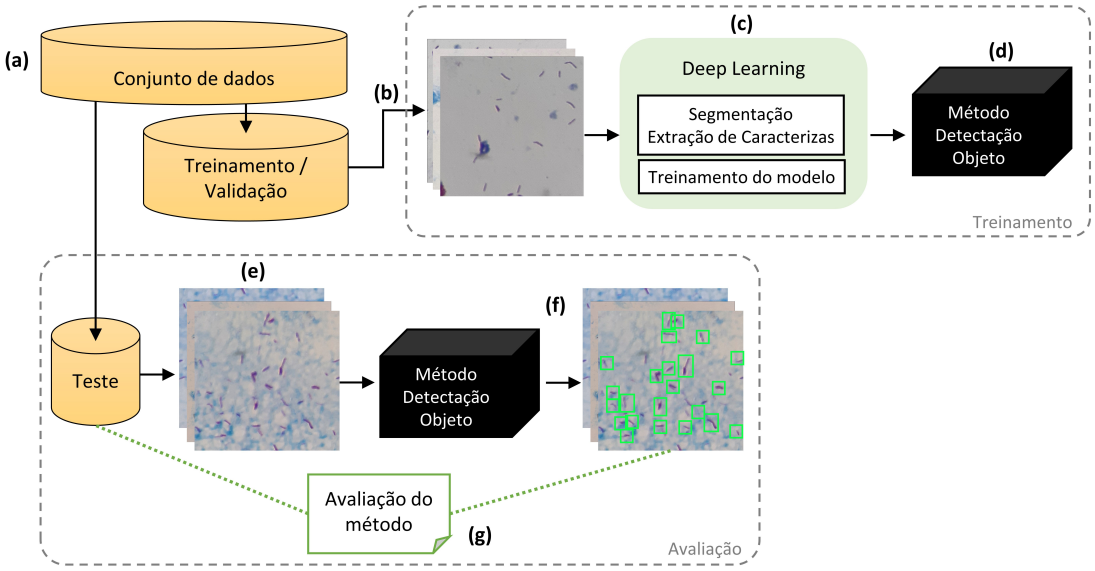


Figura 37 – Esquema ilustrativo da estratégia empregada para treinar e avaliar os métodos de detecção.

5.4.1 Estratégia de partição de imagens

Durante experimentos realizados com os métodos de detecção de objeto, identificou-se que a variabilidade de tamanhos das imagens podem comprometer o desempenho da rede e também levar a uma comparação injusta. Além disso, em um cenário real de utilização, é de suma importância que a variabilidade das resoluções seja levada em consideração. Neste sentido, este trabalho propõe uma estratégia de partição das imagens para minimizar os impactos dessa variabilidade e permitir também a padronização de alguns parâmetros de treinamento.

Para ilustrar essa variabilidade, a Tabela 10 apresenta as resoluções, máxima, miníma e média das imagens encontrada nos quatro conjuntos de dados utilizados. Também são apresentadas as resoluções média dos box que indicam as posições dos bacilos nas imagens.

Tabela 10 – Resoluções máxima, miníma e média (em pixels) das imagens encontradas nos quatro conjuntos de dados (valores representados em Largura x Altura).

	dataset-1	**dataset-2**	**dataset-3**	**dataset-4**
Imagem com maior resolução	2816 x 2112	1632 x 1224	2592 x 1944	2560 x 2557
Imagem com menor resolução	2816 x 2112	1632 x 1224	800 x 600	794 x 797
Resolução média das imagens	2816 x 2112	1632 x 1224	1375 x 1043	1647 x 1641
Resolução média dos boxs	81 x 81	66 x 67	45 x 42	78 x 85

Um dos impactos causados por essa variabilidade é a dificuldade em definir o parâmetro relativo ao tamanho de entrada da imagem, usados pelos métodos citados. Esse parâmetro tem a função de redimensionar a imagem inserida nas redes, ou seja, realizar um ajuste para uniformizar os atributos e pesos inerentes às redes de classificação. O valor

definido para esse parâmetro possui uma correlação direta com a eficiência dos métodos, e seu ajuste pode ser desafiador. Uma escolha de valor elevado pode implicar um custo computacional significativo, tanto em termos de tempo quanto de memória gráfica, o que pode inviabilizar o treinamento. Além disso, em imagens com alta dimensão, alguns métodos podem enfrentar desafios ao tentar identificar objetos de tamanho reduzido. Em contrapartida, a adoção de valores inferiores pode comprimir excessivamente a imagem, resultando na perda de características cruciais do bacilo, essenciais para que os métodos extraíam atributos relevantes que viabilizem a generalização dos modelos.

Dada a variabilidade e a complexidade de determinar um valor ótimo para este parâmetro, que seja aplicável de forma generalizada a todos os conjuntos de dados, foi aplicado uma estratégia para particionar as imagens. Esta abordagem visa reduzir os impactos associados à definição de um valor único para o referido parâmetro, reduzir o custo computacional, e assegurar que o fluxo de trabalho de detecção dos bacilos seja adaptável a qualquer dimensão de imagem. Neste sentido, o particionamento de uma imagem é efetuado dividindo-a em n segmentos, em que o valor de n é determinado pela Equação 5.1, onde *width* e *height* representam as dimensões de largura e altura da imagem, respectivamente, e *factor* é um coeficiente utilizado como limitador do tamanho das novas imagens a serem geradas. Ao segmentar a imagem em n partes, certos critérios foram estabelecidos: (i) O número de segmentos gerados verticalmente deve ser equivalente ao número gerado horizontalmente; (ii) Os segmentos devem possuir dimensões uniformes, em consonância com a proporção determinada pela divisão da imagem pelo *factor*; (iii) As dimensões de cada segmento devem ser acrescidas em 5%, levando em conta tanto a largura quanto a altura da imagem, resultando em uma intersecção entre os segmentos. A Figura 38 exemplifica este processo, para uma determinada imagem com dimensões 2540 x 2528 (Figura 38a) é calculado sua divisão em $n = 4$ segmentos, observando os critérios supracitados (Figura 38b, com $factor = 2000$), dando origem a quatro novas imagens (Figura 38c).

$$n = \left(\left\lceil \frac{\max(\text{width}, \text{height})}{\text{factor}} \right\rceil \right)^2 \tag{5.1}$$

É importante salientar, que o valor que determina a quantidade de partições realizadas na imagem foi estabelecido por meio de testes empíricos. Nesse processo, variados valores para o parâmetro de entrada e quantidade de partições foram avaliados.

Utilizando esse estratégia de partição, o fluxo de trabalho de treinamento e avaliação dos métodos de detecção foi ajustado para os seguintes passos: (i) dividir os dados em conjuntos de treinamento/validação e teste (Figura 39a); (ii) particionar as imagens de treinamento/validação (Figura 39b); (iii) utilizar as imagens particionadas e suas respectivas anotações (Figura 39c) para conduzir o treinamento do método de DL

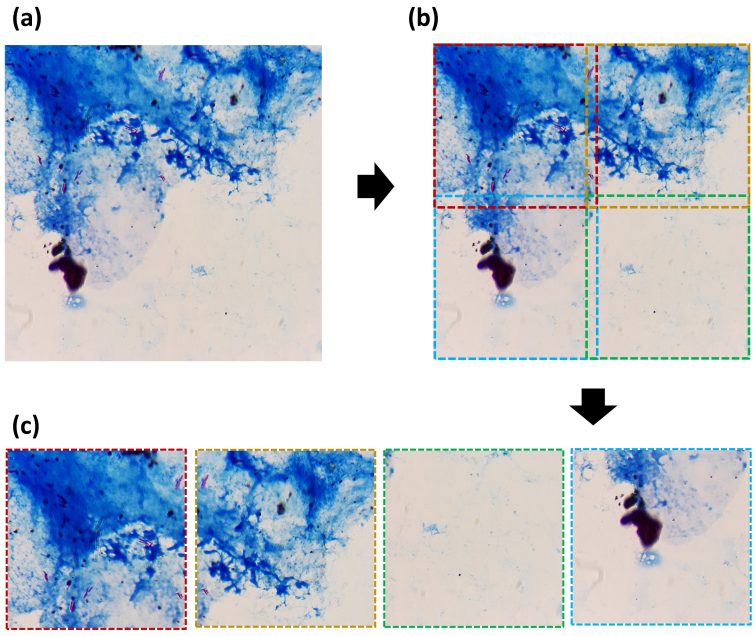


Figura 38 – Ilustração do processo de particionamento das imagens adotado neste trabalho.

(Figura 39d), com o objetivo de desenvolver um modelo de detecção (Figura 39e); (iv) na etapa de teste, realizar o particionamento de cada imagem (Figura 39f, g); (v) processar cada parte da imagem particionada (Figura 39g) com o modelo treinado para detectar os bacilos (Figura 39h); (vi) juntar as imagens particionadas e seus resultados da detecção em uma única imagem novamente (Figura 39i, j, k, l); e (vii) utilizar as anotações do conjunto de teste e as posições das regiões identificadas como bacilos pelo modelo para avaliar a acurácia do método (Figura 39m). Cada um desses passos foi executado para cada método mencionado nesta seção.

Para realizar o treinamento utilizando esse particionamento de imagens, é necessário alterar os conjuntos de treinamento/validação. Assim, quaisquer imagens do conjunto de dados que excedam o tamanho máximo estabelecido serão subdivididas conforme a proporção predefinida. No entanto, o conjunto de teste não requer modificações prévias, dado que o fluxo de trabalho foi ajustado para particionar as imagens no momento anterior à detecção e, subsequentemente, reagrupá-las após a etapa de classificação. Nesta perspectiva, adotando-se os critérios de particionamento definidos nesta seção, a estratégia de divisão de dados *5-fold cross-validation*, juntamente com o *data augmentation* discutidos na Seção 5.1.2, os totais de imagens integrantes de cada conjunto de dados, após o particionamento (considerando $factor = 2000$), são detalhados na Tabela 11.

Devido ao significativo custo computacional associado à avaliação do desempenho desta estratégia, nem todos os métodos selecionados foram testados. Decidiu-se avaliar exclusivamente o método que apresentou o melhor desempenho nos ensaios realizados sem

o particionamento das imagens. Esta abordagem objetiva assegurar uma avaliação justa e objetiva dentro de um período de execução factível.

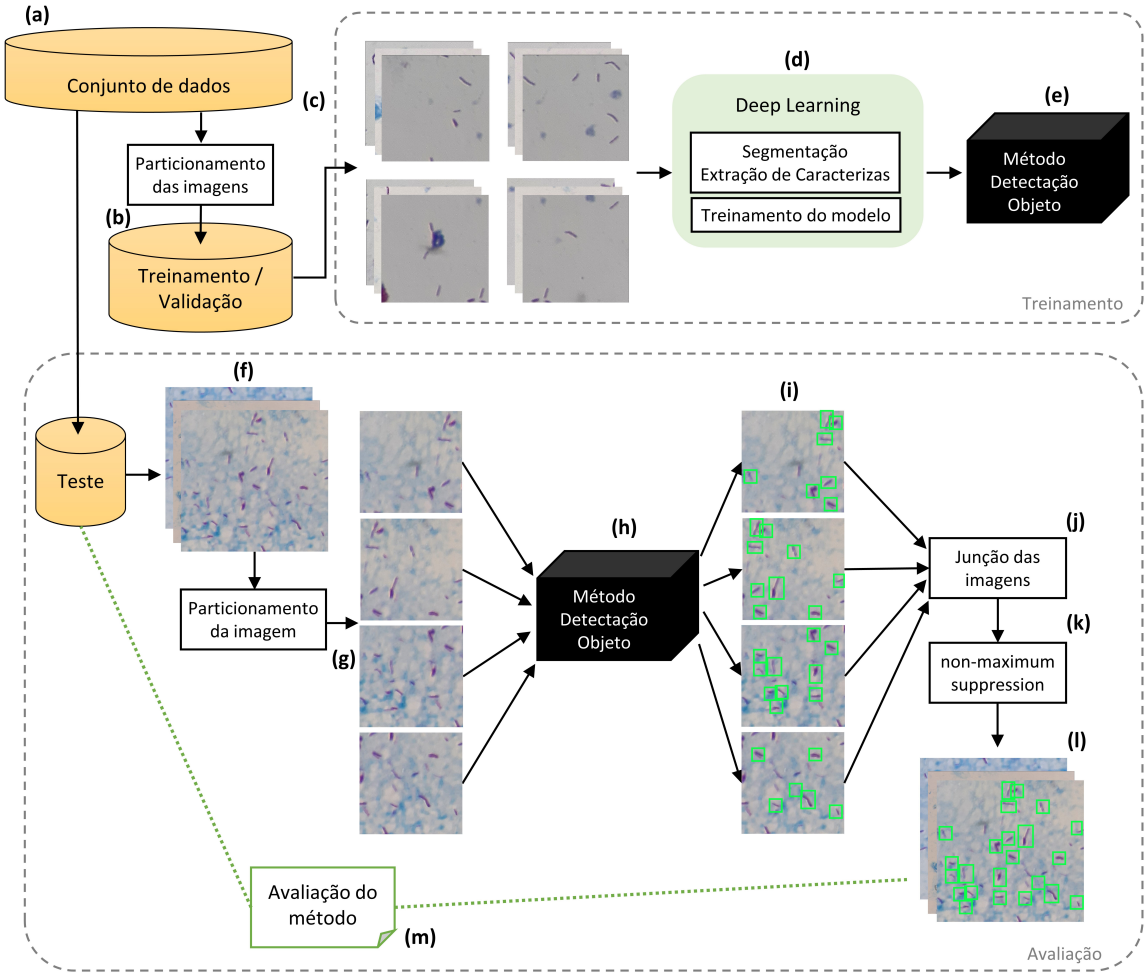


Figura 39 – Esquema ilustrativo da estratégia empregada para treinar e avaliar os métodos de detecção a partir da estrategia de particionamento das imagens proposto.

Tabela 11 – Quantidades de imagens em cada um dos 5-*fold* de cada conjunto de dados após o particionamento das imagens (considerando o *data augmentation*).

	dataset-1	**dataset-2**	**dataset-3**	**dataset-4**
Treino	2.472	6.992	1.760	2.584
Validação	344	784	192	280
Teste	192	1.944	296	320

5.4.2 Implementação e avaliação

Dentre as principais linguagens e bibliotecas utilizadas na implementação do método proposto destaca-se: Python V.3.8.5, Scikit-learn V.1.0.2 (PEDREGOSA et al., 2011), Pillow V.9.0.1 (CLARK, 2022), OpenCV V.4.5.4 (BRADSKI, 2000), entre outras. Essas bibliotecas foram instaladas a partir da plataforma Anaconda (ANACONDA, 2020).

Além das mencionadas, o framework predominante nesta parte do trabalho foi o IceVision (VÁZQUEZ; HASSAINIA, 2020). Este framework se destaca por disponibilizar uma gama de modelos pré-treinados de alta qualidade de Torchvision, MMDetection da Open MMLab, YOLOv5 da Ultralytic e EfficientDet de Ross Wightman. O IceVision promove um fluxo de trabalho integrado para DL, facilitando a integração com bibliotecas de alto desempenho e intuitivas, como PyTorch-Lightning e Fastai. Dentre os métodos disponíveis neste framework, destacam-se: Faster R-CNN, YOLOF, YoloV5, FSAF, SSD, EfficientDet, RetinaNet.

De maneira geral, optou-se pelo uso dos modelos pré-treinados e parâmetros padrões disponibilizados pelo framework IceVision. No entanto, durante avaliações empíricas com o objetivo de encontrar um bom *tradeoff* entre eficiência computacional e desempenho de classificação, realizaram-se ajustes em determinados parâmetros. Dentre os quais se destacam: *epochs* = 50; *batch size* = 5; IoU = 0.1; e, *presize* e *image size* de 512 e 384, respetivamente (responsáveis pelo redimensionamento das imagens de entrada). Além disso, o valor de *factor* na Equação 5.1 foi definido para 2000. Detalhes dos ajustes, parâmetros e demais implementações realizadas estão disponíveis no código-fonte apresentado no Apêndice B.3.

Para avaliar os métodos selecionados, com e sem a abordagem de partição de imagens, foi realizado o processo de *5-fold cross-validation*, conforme descrito na Seção 5.1.2. Após treinar os métodos, o desempenho na detecção do bacilo da TB foi mensurado usando as métricas apresentadas na Seção 3.2.3. Cada método apresentado foi testado em cada um dos conjuntos de dados disponíveis. O resultado dessas avaliações são apresentadas nas Seções 6.4 e 6.5.

5.5 Explorando treinamento e teste em diferentes conjuntos de dados

Poucos trabalhos na literatura abordam à detecção do bacilo da TB por meio de DL utilizando conjuntos de dados de treinamento e teste obtidos em diferentes contextos, como laboratórios ou países distintos. A importância da seleção criteriosa de conjuntos de dados de treinamento e teste é crucial para a eficácia dos modelos, com a expectativa de que imagens obtidas de um único laboratório e equipamento propiciem resultados com maior acurácia. Contudo, estudos que considerem imagens provenientes de diferentes laboratórios e países são escassos e necessitam de uma análise mais detalhada. A busca por compreender o desempenho desses métodos em cenários heterogêneos e mais representativos da realidade é essencial. Adicionalmente, tal investigação é crucial para avaliar a viabilidade de treinar modelos baseados em um conjunto de dados específico, objetivando o desenvolvimento de um sistema CADx que possa ser usado em diferentes laboratórios.

Para conduzir este experimento, adotou-se uma estratégia de divisão dos conjuntos de dados em Treinamento/Validação/Teste, utilizando as proporções de 70%, 10% e 20%, respectivamente. Adicionalmente, após a divisão dos dados, empregou-se a estratégia de particionamento de imagens, conforme detalhado na Seção 5.4.1, e as operações de *data augmentation*. Isso permite que se treine um modelo com um conjunto de dados e o teste com outro, utilizando métodos DL. A Tabela 12 mostra a quantidade de imagens após a divisão de cada conjunto e processamento das imagens.

Tabela 12 – Quantidades de imagens em cada conjunto de dados separados em treinamento, validação e teste, após o particionamento das imagens (considerando o *data augmentation*).

	dataset-1	**dataset-2**	**dataset-3**	**dataset-4**
Treino	2.472	6.816	1.760	2.584
Validação	344	968	192	280
Teste	192	1.952	296	320

Considerando a separação dos dados em treinamento, validação e teste para cada um dos conjuntos de dados:

- dataset-1: composto por imagens capturadas na Amazônia, Brasil;
- dataset-2: composto por imagens capturadas na Uganda;
- dataset-3: composto por imagens capturadas na Índia;
- dataset-4: composto por imagens capturadas em Minas Gerais, Brasil.

os seguintes experimentos podem ser delineados:

- Para cada um dos dataset-1, dataset-2, dataset-3, e dataset-4, treinar o método de detecção usando os seus respectivos conjunto de treinamento/validação e avaliá-los em cada conjunto de teste dos datasets-1, 2, 3 e 4.

Em outras palavras, o procedimento ilustrado na Figura 39 foi repetido 16 vezes, alternando os conjuntos de treinamento/validação e de teste para cada dataset.

Novamente, optou-se por utilizar apenas o método que demonstrou o melhor desempenho nos testes descritos na Seção 5.4. Essa escolha tem como objetivo empregar a técnica que se destacou na detecção do bacilo da TB em um conjunto de dados específico, a fim de investigar o comportamento deste método em avaliações transversais com diferentes conjuntos de dados. Os resultados desse experimento são apresentadas na Seção 6.6.

5.6 Anotação semi-automática

No Capítulo 4, foi enfatizado o desafio de anotar as imagens do conjunto de dados proposto neste trabalho. Este obstáculo é evidente em outros trabalhos, por exemplo, Shah et al. (2017) disponibilizam um subconjunto de 300 imagens anotadas, porém, em seu trabalho outras centenas de imagens também foram capturadas e disponibilizadas, mas não receberam anotações. Diante disso, torna-se essencial considerar estratégias que realizem essa anotação, ou que, ao menos, auxiliem os especialistas nesse processo.

A variabilidade na forma de capturar as imagens do campo de observação, juntamente com as diferenças na qualidade, resulta em uma heterogeneidade notável entre os conjuntos de dados. Para abordar esse desafio e propor um fluxo de trabalho que auxilie na anotação semi-automática, adaptável a diferentes conjuntos de dados, adotou-se uma estratégia de consenso.

No processo apresentado na Figura 40, foi estabelecido um sistema de consenso, onde, vários métodos são utilizados para detectar o bacilo da TB e somente quando há uma identificação unânime entre eles é que o bacilo é considerado real. O processo envolve as seguintes etapas: (i) a imagem de entrada (Figura 40a) é inicialmente particionada em n partes de tamanhos idênticos (Figura 40b); (ii) cada parte é avaliada individualmente pelos métodos disponíveis (Figura 40c), denotados como M1, M2, M3 e M4, treinados com o dataset-1, dataset-2, dataset-3 e dataset-4, respectivamente; (iii) os métodos identificam potenciais bacilos em cada parte (Figura 40d, e, f, g); (iv) uma etapa de consenso é então executada para cada parte da imagem correspondente, em que apenas os bacilos identificados de forma unânime por todos os métodos são considerados como detecções válidas (Figura 40h); (v) as localizações dos bacilos confirmados são destacadas na imagem original (Figura 40i); (vi) em seguida, um especialista realiza uma curadoria manual, adicionando ou removendo marcações de bacilos para refinar a anotação (Figura 40j); (vii) a etapa final resulta em uma imagem cuidadosamente anotada (Figura 40k).

Para avaliar a eficácia do sistema de consenso proposto utilizando os métodos de DL, foi empregada uma abordagem semelhante a Seção 5.5. Especificamente, os métodos de detecção (M1, M2, M3 e M4) foram treinados com o método que obteve melhor desempenho nos testes descritos na Seção 5.4. Posteriormente, a avaliação foi realizada utilizando os conjuntos de testes correspondentes de cada conjunto de dados. Os resultados desse experimento são apresentadas na Seção 6.7.

Para estimar os benefícios proporcionados por esta abordagem em um situação prática, é proposto um experimento em menor escala. Isto foi realizado com o objetivo de mensurar o tempo economizado no processo de anotação, e a eficácia na identificação dos bacilos. A investigação foi estruturada em dois experimentos/cenários distintos:

- Cenário A: Um especialista deve anotar manualmente um conjunto de imagens sem qualquer assistência da estratégia proposta. O conjunto consiste em 12 imagens da Lâmina 1 e 13 imagens da Lâmina 2 (lâminas utilizadas na construção do conjunto de dados proposto - Seção 4.1). O tempo necessário para completar essa tarefa será cronometrado.

- Cenário B: O mesmo especialista deve anotar um conjunto diferente de imagens, desta vez com o auxílio da estratégia de anotação semi-automática proposta. Este outro conjunto é composto por 13 imagens da Lâmina 1 e 12 imagens da Lâmina 2. Assim como no Cenário A, o tempo gasto será cronometrado para comparação.

Nos dois cenários, as anotações dos especialista e da estratégia proposta são comparadas, para quantificar a acurácia e a confiabilidade da anotação realizada pelo método.

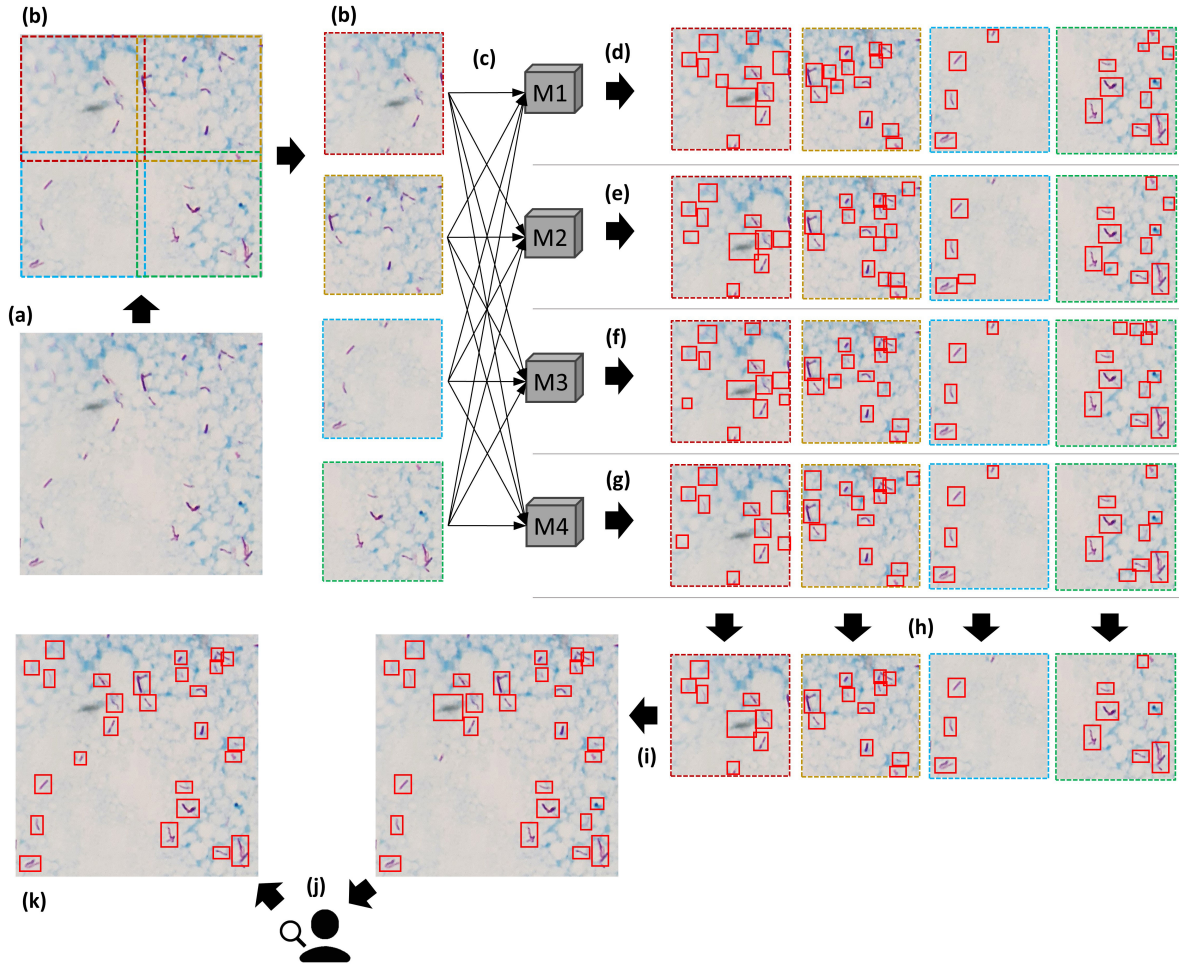


Figura 40 – Esquema ilustrativo da estratégia de realizar a detecção do bacilo da TB utilizando um sistema de consenso entre diferentes métodos de DL e uma curadoria feita por uma especialista.

5.7 Identificação assistida do bacilo da tuberculose para diagnóstico

Um dos principais objetivos deste trabalho é construir um protótipo de um CADx capaz de auxiliar parte do processo de diagnóstico da TB pela baciloscopia (método ZN). Para esta finalidade, é proposto a utilização de técnicas de visão computacional e de DL para agilizar a fase da baciloscopia que envolve a busca e contagem dos bacilos no microscópio. Ainda que essa fase não seja complexa de ser realizada por um especialista treinado e certificado, tal procedimento requer muito treinamento, atenção e acurácia, podendo se tornar cansativo e passível de erro. Logo, semi-automatizar essa tarefa com um software de apoio ao especialista, pode agilizar e aumentar a acurácia da baciloscopia.

O desenvolvimento deste protótipo foi realizado de forma incremental, ou seja, suas necessidades/funcionalidades foram identificadas e implementadas ao longo do seu desenvolvimento. A cada ciclo de desenvolvimento (incremento ou versão do sistema), foram realizadas atividades como: especificação e levantamento de requisitos, desenvolvimento, apresentação do protótipo, e avaliação. Essa abordagem de desenvolvimento foi baseada nas características dos modelos de desenvolvimento incremental e prototipação apresentados em Sommerville (2011). Esses modelos destacam-se por permitir implementações mais rápidas de um software, facilitando retornos contínuos dos usuários, avaliações frequentes das funcionalidades e análises de desempenho.

Durante o primeiro ciclo de desenvolvimento, percebeu-se a necessidade de estruturar o sistema em três módulos principais: (i) módulo de detecção; (ii) módulo de registro; e (iii) módulo de interface. Adicionalmente, desenvolveu-se uma interface específica para a captação de imagens do microscópio por meio de smartphones. No segundo ciclo, foram identificadas funcionalidades primordiais para o registro de informações, como a contagem de bacilos por campo visual e sua média correspondente. Também houve um aprimoramento na visualização das imagens, tanto as originais quanto as processadas pelo método de DL. Nos ciclos de desenvolvimento subsequentes, o foco centralizou-se na refinamento da interface do sistema. Além disso, foram incorporadas funcionalidades secundárias que auxiliam no processo de baciloscopia e efetuadas melhorias contínuas no módulo de detecção.

A Figura 41 ilustra o processo de interação do sistema, usuário e fonte de captura das imagens. Os tópicos a seguir detalham o funcionamento básico proposto para cada módulo e a interação esperada do usuário com o software, são eles:

- Inicialmente, o especialista insere e ajusta a lâmina da baciloscopia no microscópio, posicionando a lente no campo onde será iniciada a análise (Figura 41a);
- Uma vez na posição certa, a imagem do campo é capturada por um dispositivo de captura (como um smartphone). Essa imagem é enviada para o módulo de detecção

e para a tela do computador (interface do usuário), onde será visualizada pelo especialista (Figura 41b);

- Ao receber a imagem, o módulo de detecção utiliza a estrategia apresentada na Seção 5.4.1 para detectar os bacilos, seguindo os passos:
  - particionamento da imagem (Figura 41c);
  - detecção dos bacilos em cada imagem particionada usando um método de DL (Figura 41d, e);
  - processamento do resultado da detecção (Figura 41f) para realizar a junção das imagens e destacar as posições dos bacilos no campo de observação (Figura 41g).
- Após o processamento do módulo de detecção (Figura 41h), o especialista pode visualizar a imagem contendo a localização dos bacilos juntamente com a quantidade de bacilos identificados pelo método na interface do usuário (Figura 41i);
- Ao finalizar a detecção de um campo de observação, o módulo de detecção envia o resultado para o módulo de registro, o qual armazenará todos os dados obtidos, tais como posição dos bacilos, data/hora e imagem utilizada para detecção (Figura 41j);
- Em seguida, o especialista deve ajustar no microscópio outro campo da lâmina (Figura 41a), reiniciando assim todas as etapas da (Figura 41a) até (Figura 41i);
- Após finalizar a análise dos campos da lâmina, o especialista pode visualizar um relatório do sistema. Este relatório é gerado pelo módulo de registro, o qual compila todos os dados obtidos e apresenta informações relevantes para o diagnóstico da TB (Figura 41k). Informações como quantidade de bacilos identificados, quantidade de campos analisados, proporção de bacilos por campo, recomendação de diagnóstico (negativo, positivo +, ++ ou +++), dentre outras informações, serão exibidas na interface do usuário para apreciação do especialista (Figura 41l).

Essa modularização oferece robustez e adaptabilidade, visando garantir que cada componente possa ser refinado ou alterado sem alterar os demais. Por exemplo, essa estrutura facilita a substituição ou atualização do método de detecção e contagem do bacilo, facilitando a incorporação de novas estratégias de ML e incentivando contribuições de outros pesquisadores à ferramenta. Além da modularização, identificou-se a importância da estrutura do sistema ter a capacidade de aceitar múltiplos formatos de entrada de imagens. Essa flexibilidade torna o sistema capaz de se adaptar a variados cenários. Ele pode se conectar à câmera de um smartphone, seja via Wi-Fi ou cabo USB, interagir com uma câmera USB (formato comumente empregado em microscópios equipados com

câmeras) ou ainda importar imagens de um diretório específico, como as oriundas de scanners de lâminas.

Na Seção 6.9, dedicada aos resultados, são detalhadas as funcionalidades incorporadas ao protótipo e à interface projetada, seguindo a estrutura e fluxo de acordo com os módulos propostos. Durante os ciclos realizados, nas fases de especificação e levantamento de requisitos, diversas funcionalidades foram identificadas e subsequentemente incorporadas durante o estágio de desenvolvimento. Na seção de resultados, é fornecido uma descrição aprofundada de cada uma dessas funcionalidades, ilustradas com imagens do protótipo construído.

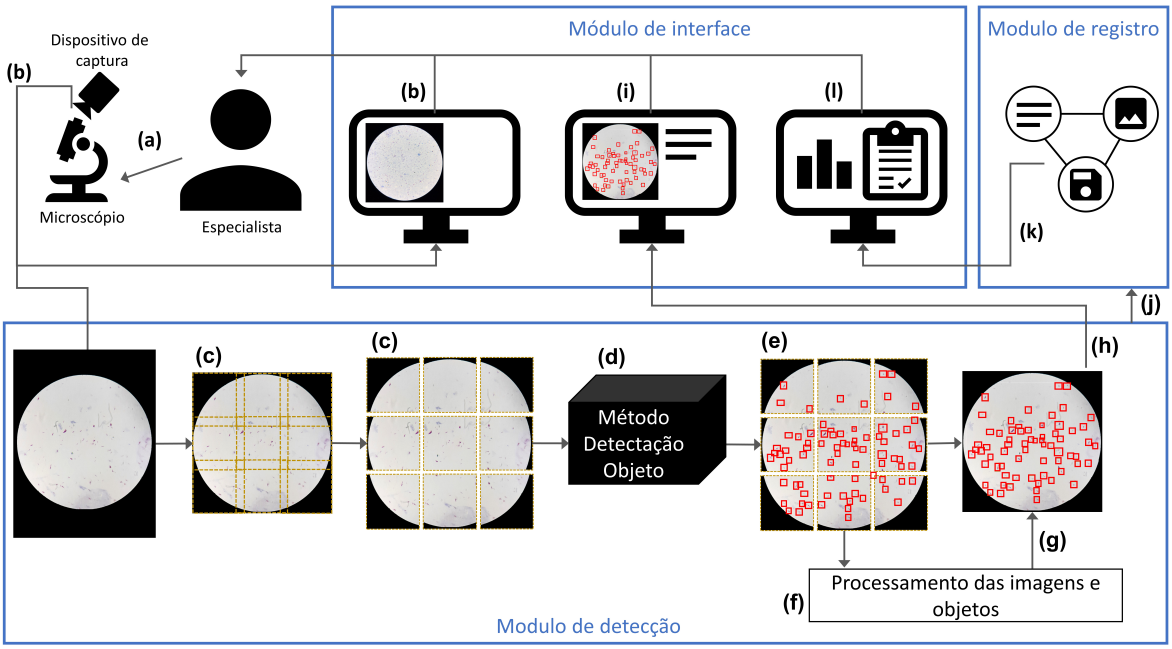


Figura 41 – Módulos proposto para criação do software de auxílio ao diagnostico da TB.

O desenvolvimento de todos os módulos foi realizado com a linguagem de programação Python sobre o sistema operacional Ubuntu 21.04. Várias bibliotecas e frameworks foram utilizadas no desenvolvimento, onde-se destacam: TensorFlow, Keras, Scikit-learn, PIL (Pillow - Python Imaging Library), OpenCV, IceVision, e PyQt, dentre outras.

5.8 Explorando métodos de DL para diagnóstico automático

Embora o foco principal deste estudo seja explorar e avaliar o potencial dos métodos de DL na detecção do bacilo da TB, bem como apresentar um protótipo de sistema CADx para facilitar o processo de diagnóstico, surge uma questão intrigante: como esses métodos se comportariam em um cenário de diagnóstico automatizado?

Infelizmente, os conjuntos de dados disponíveis até o momento não são suficientes para fornecer uma base de dados robusta que permita explorar integralmente essa questão. Os conjuntos atuais possuem limitações que impedem uma análise conclusiva sobre a viabilidade e eficácia de um sistema autônomo. Uma vez que as informações disponíveis são insuficientes para avaliar o desempenho do método no contexto de uma análise integral da lâmina, é possível apenas obter uma avaliação geral da capacidade de identificação do bacilo. Para realizar esse tipo de experimento mais abrangente, é necessário um conjunto de dados que possua algumas características, como: maior quantidade de imagens por lâmina; informações que correlacionam as imagens à lâmina específica de origem; imagens que representem um campo visual completo do microscópio; e os resultados do diagnóstico para cada lâmina.

Neste sentido, uma colaboração foi estabelecida com o projeto de pesquisa intitulado "Contagem Automática de Bacilos Álcool Ácido Resistentes para o Diagnóstico da Tuberculose". Este projeto, registrado sob o número de processo 441541/2018-6, sob a orientação do professor João Victor Gomide, foi aprovado no âmbito da Chamada Pesquisas De Inovação Em Saúde 2018, com financiamento do Conselho Nacional de Desenvolvimento Científico e Tecnológico. Adicionalmente, o projeto conta com a parceria do Serviço de Doenças Bacterianas e Fúngicas - DECD/DIOM - Diagnóstico de Micobactérias - Fundação Ezequiel Dias – FUNED. Esse projeto se destaca por possuir um conjunto de dados extenso e robusto, composto por milhares de imagens obtidas a partir de diversas lâminas.

As subseções seguintes apresentam, resumidamente, algumas informações sobre os dados adquiridos através desta colaboração, bem como o delineamento do fluxo de trabalho proposto para que o método de DL execute o diagnóstico ao analisar as imagens da baciloscopia.

5.8.1 Dados disponíveis

Através de reuniões e ao ter acesso ao conjunto de dados elaborado pelo projeto citado anteriormente, as seguintes características e informações pertinentes foram levantadas:

- Existem 362 diretórios no conjunto de dados, sendo que cada uma deles contém as imagens referentes a uma lâmina específica;
- As imagens foram obtidas por um Microscópio Zeiss AXIOLAB 5, Câmera AXIOCAM 105, resolução de 5 Mb;
- Em média, cada lâmina possui aproximadamente 410 imagens registradas;
- As amostras das imagens são oriundas diretamente do escarro e foram coradas pelo método de Ziehl-Neelsen, não centrifugadas;

- Para cada campo visual observado pelo microscópio, foram capturadas quatro imagens, estrategicamente posicionadas de modo a formar um mosaico. Esse arranjo visava cobrir a totalidade do campo de visão, minimizando distorções e perdas de informação decorrentes do ajuste de foco;
- As imagens foram capturadas de forma sequencial, respeitando as orientações de leitura do Ministério da Saúde.
- Para cada lâmina, dois especialistas realizaram diagnósticos independentes, analisando as lâminas diretamente sob o microscópio. A anotação dos dados, portanto, está associada a cada lâmina, e os diagnósticos foram categorizados como negativo, positivo +, positivo ++ ou positivo +++;

5.8.2 Treinamento

Visto que este conjunto de dados não conta com anotações específicas dos bacilos nas imagens e levando em consideração as conclusões obtidas nos experimentos realizados no Capitulo 6, que indicam a necessidade de treinar os métodos de DL com imagens análogas àquelas que serão avaliadas para diagnóstico, tornou-se necessário proceder com a anotação de algumas imagens deste conjunto de dados.

Desta forma, com o objetivo de criar um conjunto de treinamento robusto e eficaz, as especialistas encarregadas da análise das lâminas selecionaram cuidadosamente um conjunto de 505 imagens, consideradas representativas e promissoras para o treinamento dos modelos. Essas imagens foram escolhidas devido à sua diversidade em termos de quantidade de bacilos, variações nas tonalidades tanto dos bacilos quanto do fundo, e dentre outras características. A partir da seleção cuidadosa das imagens, foi empregada a estratégia delineada na Seção 5.7 para realizar uma anotação automatizada inicial. Em sequência, uma especialista da FUNED conduziu uma curadoria meticulosa dessas imagens, utilizando-se do sistema descrito na Seção 4.3.

Concluído esse processo de anotação, que resultou na identificação de 6.755 entre bacilos e clusters, procedeu-se à divisão das imagens em três conjuntos distintos, seguindo a distribuição: 70% para treinamento, 10% para validação e 20% para teste. É importante destacar que, que após a divisão, foi aplicado a estratégia de partição de imagens proposta na Seção 5.4.1 e incorporado imagens sem bacilos (originárias de lâminas com diagnóstico negativo). A Tabela 13 exibe a quantidade de imagens em cada subconjunto do banco de dados para o treinamento do método de DL. Ressalta-se que, mantendo o padrão adotado anteriormente, tanto bacilos quanto clusters foram classificados sob a categoria única de “bacilo”.

Tabela 13 – Quantidade de imagens separadas para treinamento, validação e teste após processamento do conjunto de dados.

	Imagens com bacilos	Imagens sem bacilos
Treino	8976	6392
Validação	1288	1592
Teste	800	-

Após a etapa de anotação, iniciou-se a fase de treinamento e teste do método de DL, visando produzir um modelo apto a identificar e quantificar os bacilos da TB. Este novo conjunto de dados foi então utilizado para treinar o método que demonstrou o melhor desempenho dentre os descrito na Seção 5.4. Após o treinamento, este método é, subsequentemente, aplicado no processo de diagnóstico detalhado na seção seguinte.

5.8.3 Proposta de um processo de diagnóstico

O processo de diagnóstico do bacilo da TB a partir de métodos de DL é uma tarefa desafiadora, que requer a atenção a detalhes específicos e adaptações para assegurar a acurácia e confiabilidade dos resultados. Para isso, é proposto um fluxo de trabalho personalizado de análise para lidar com essa tarefa, incorporando etapas como detecção de bacilos por meio de métodos de DL, aplicação de regras de filtragem, identificação de possíveis aglomerações de bacilos (clusters) e a definição de regras para o diagnóstico com base na contagem de bacilos.

Neste fluxo de trabalho, destaca-se uma particularidade em relação às regras para determinar os diagnósticos negativos e positivos. Geralmente, a abordagem padrão seria adotar o método de diagnóstico baseada na quantidade de bacilos, conforme definido na Tabela 1, estabelecida pelo Ministério da Saúde. No entanto, foi realizado alguns ajustes nessas regras para lidar com características associadas à análise de imagens por meio de métodos de DL. Isso inclui o enfrentamento de desafios como a presença de falsos negativos e/ou uma contagem mais densa de bacilos. Além disso, outra peculiaridade reside no fato de que um campo de visão é representado por quatro imagens distintas no conjunto de dados disponível, o que requer ajustes adicionais nos cálculos da proporção de bacilos por campo.

A Figura 42 ilustra o processo de diagnóstico proposto, apresentando cada etapa do fluxo de trabalho, desde a detecção inicial do bacilo até a definição final do diagnóstico. Cada etapa é detalhada a seguir:

- Inicialmente, um conjunto de imagens (≥ 400), todas pertencentes a uma mesma lâmina, são processadas pelo método de DL, configurado com um *threshold* de 0,1 ($T = 0,1$) (Figura 42a,b);

- Posteriormente, com as posições e os scores de confiança (probabilidade) associados a cada bacilo identificado (fragmento de imagem) em mãos, realiza-se uma primeira checagem. Em que, verifica-se se há pelo menos um bacilo com valor de probabilidade superior a 0,85 (*threshold* ajustado para $T \geq 0.85$, conforme a Figura 42c). Se esse critério não for atendido, a lâmina é classificada como "Negativo". Caso contrário, o processo avança para a próxima etapa;

- Na etapa subsequente (Figura 42d), os bacilos com probabilidade acima de 0,85 são submetidos a um filtro de cores, semelhante ao empregado na estratégia k-*means* (Figura 33c). Esse filtro verifica se o fragmento apresenta uma tonalidade dentro da faixa de cores vermelha/rosa/roxo. Se pelo menos um fragmento passar por esse filtro, segue-se para a próxima fase. Do contrário, a lâmina é classificada como "Negativo";

- Posteriormente, uma filtragem adicional é realizada para excluir todos os bacilos com probabilidade inferior a 0,5, ajustando o *threshold* para $T \geq 0,5$ (Figura 42e);

- Considerando que os clusters são importantes para o diagnóstico, devido ao seu peso significativo na contagem de bacilos, é proposto uma estratégia para ampliar essa contagem, levando em consideração os maiores fragmentos identificados (Figura 42f). Neste abordagem, cada fragmento cuja área exceda a média calculada a partir de todos os fragmentos, somada a um incremento de 50%, é subdividido em i novos fragmentos, equivalentes a i bacilos adicionais. O valor de i é determinado pelo quociente inteiro da divisão da área do fragmento pela menor área identificada em todas as imagens.

- Levando em conta os bacilos identificados e a ampliação da contagem realizada na etapa anterior, calcula-se as seguintes variáveis (Figura 42f):

  - μ_score: média dos scores atribuídos a cada bacilo pelo método de DL;
  - N_80: total de bacilos nas primeiras 80 imagens (total de bacilos equivalente aos 20 primeiros campos de visão);
  - N_200: total de bacilos nas primeiras 200 imagens (total de bacilos equivalente aos 50 primeiros campos de visão);
  - N_400: total de bacilos nas primeiras 400 imagens (total de bacilos equivalente aos 100 primeiros campos de visão);
  - M_80: média de bacilos por imagem nas 80 primeiras imagens ($N80/20$)(média de bacilos equivalente aos 20 primeiros campos de visão);
  - M_200: média de bacilos por imagem nas 200 primeiras imagens ($N200/50$)(média de bacilos equivalente aos 50 primeiros campos de visão);
  - NT: número total de bacilos considerando todas as imagens;

- Finalmente, com base nessas contagens, define-se o diagnóstico para a lâmina, seguindo as regras:
  - Se $\mu_score < 0,85$, o diagnóstico é "*To Check*"(explicado posteriormente);
  - Se $M_80 > 10$, o diagnóstico é "Positivo +++";
  - Se $M_200 \geq 1$, o diagnóstico é "Positivo ++";
  - Se $N_400 \geq 10$, o diagnóstico é "Positivo +";
  - Se $N_400 \geq 1$ *OU* $NT > 0$, o diagnóstico é "*To Check*";
  - Se $NT = 0$, o diagnóstico é "Negativo";

Como pode ser visto, além das classes de diagnóstico normalmente utilizadas ("Negativo", "Positivo +", "Positivo ++", "Positivo +++"), foi proposto uma nova classe de diagnóstico chamada de "*To Check*". Esta categoria foi concebida com o intuito de sinalizar situações em que se faz necessária a intervenção de um especialista para avaliação direta da lâmina, dada a incerteza que pode ser gerada ao analisar lâminas com poucos bacilos ou com baixa probabilidade. A introdução da classe "*To Check*"visa minimizar potenciais erros que poderiam surgir em cenários reais, garantindo uma segunda opinião qualificada para os casos mais complexos ou duvidosos. Além disso, antecipa-se que esta abordagem possa desempenhar um papel crucial na otimização do tempo e dos recursos do especialista, agilizando a análise das lâminas de baciloscopia e garantindo que a atenção do profissional seja direcionada para os casos que de fato requerem sua expertise.

Dessa forma, o processo proposto não só tenta manter a acurácia e a confiabilidade do diagnóstico, mas também introduz um mecanismo colaborativo, assegurando que os casos incertos sejam submetidos à avaliação humana, promovendo uma interação entre a inteligência artificial e o especializado.

5.8.4 Avaliação

Para validar a abordagem metodológica descrita na seção anterior, optou-se por realizar testes em um conjunto aleatório de 100 lâminas. É importante destacar que a seleção de todas as lâminas disponíveis não foi viável devido ao tempo computacional para efetuar essa análise. Para cada lâmina escolhida, as imagens correspondentes foram processadas através do fluxo de trabalho definido para realizar o diagnóstico (Figura 42), resultando na coleta dos diagnósticos finais de cada lâmina.

Posteriormente, os resultados coletados foram confrontados com os diagnósticos realizados pelos especialistas. Esse processo permitiu a construção de uma matriz de confusão para análise do comportamento do método. O resultado dessa avaliação é detalhado e discutido na Seção 6.8.

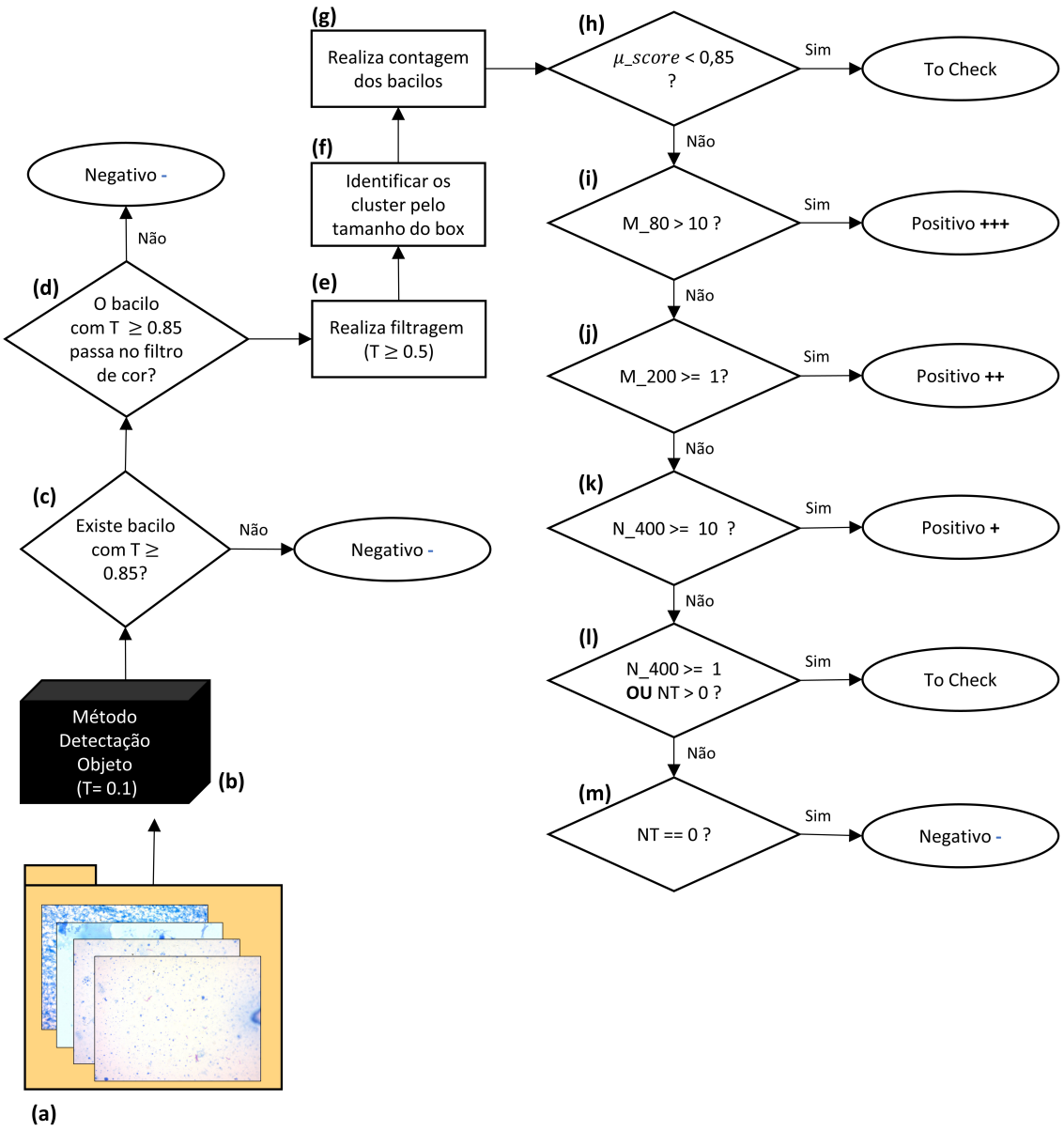


Figura 42 – Esquema ilustrativo do fluxo de trabalho proposto para realizar o diagnóstico automático da TB através do uso de métodos de DL.

6 Resultados e discussão

Este capítulo é dedicado à apresentação e à discussão sobre os resultados alcançados nos experimentos realizados, focados na avaliação de diversas estratégias de DL para identificação do bacilo da TB, conforme discutido no Capítulo 5. Ao longo de oito experimentos computacionais, explorou-se como os métodos de DL se comportam em variadas situações, aumentando progressivamente nossa compreensão sobre o seu desempenho em cenários distintos. As seções são estruturadas de maneira a gradualmente expandir o entendimento, abordando desde a aplicação de modelos de CNN em múltiplos conjuntos de dados, estratégias mais complexas e específicas, até chegar na proposta de um protótipo de um CADx que possa ser testado no contexto do diagnóstico da TB.

Inicialmente, na Subseção 6.1 (Experimento 1), é avaliado o desempenho de diferentes modelos de CNNs em uma variedade de conjuntos de dados, replicando alguns trabalhos da literatura. A Subseção 6.2 (Experimento 2) foca na combinação do K-*means* com um classificador baseado em CNN, para avaliar o desempenho das CNNs no contexto de detecção dos bacilos de TB. No Experimento 3, detalhado na Subseção 6.3, busca-se otimizar o desempenho da CNN por meio da ampliação dos dados de treinamento.

A Subseção 6.4 (Experimento 4) apresenta uma comparação abrangente entre diferentes métodos de detecção de objetos propostos na literatura, que foram ajustado neste trabalho para identificar os bacilos de TB. Na sequência, a Subseção 6.5 (Experimento 5) explora a aplicação da estratégia de partição de imagens em conjunto com o método Faster R-CNN. Após testar diversos métodos de DL e verificar qual obteve melhor desempenho, é realizado na Subseção 6.6 (Experimento 6) uma avaliação transversal dos modelos em diferentes conjuntos de dados, testando a capacidade de generalização dos modelos. O Experimento 7, descrito na Subseção 6.7, introduz um sistema de votação para a combinação de modelos na detecção de bacilos, buscando auxiliar na tarefa de anotação de imagens para treinamento.

Na Subseção 6.8, é realizado o Experimento 8, voltado para avaliar uma estratégia proposta para entender o uso dos métodos de DL no contexto de um diagnóstico automatizado, sem interferência humana na contagem do bacilo. Por fim, a Subseção 6.9 apresenta um protótipo de software CADx desenvolvido com propósito de auxiliar no diagnóstico da TB, integrando as técnicas e os métodos avaliados ao longo dos experimentos.

Todos os experimentos detalhados neste capítulo foram conduzidos em uma máquina configurada para atender demandas intermediárias de processamento associadas aos métodos de DL. O computador utilizado para tal fim possui um processador AMD Ryzen 9 5900X (12 núcleos/24 threads), 64 GB de memória RAM e uma GPU Nvidia

GeForce RTX 3060 12GB RAM. Além de outros componentes que permitem conduzir a uma operação estável e eficiente.

6.1 Experimento 1: Avaliação de modelos de CNN em múltiplos datasets

Nesta seção, são avaliadas as quatro arquiteturas de CNNs, selecionadas conforme Seção 5.2, em múltiplos conjuntos de dados. Uma etapa crucial para o entendimento da eficácia dessas arquiteturas em contextos variados e a identificação do modelo mais adequado para a tarefa de classificação de bacilos da TB. A necessidade de explorar diferentes conjuntos de dados contribui para avaliar como a heterogeneidade dos dados pode influenciar no desempenho dos modelos gerados por diferentes arquiteturas.

O Experimento 1, estruturado em quatro sub-experimentos, segue a abordagem 5-*fold cross-validation* para prover maior confiabilidade dos resultados obtidos e minimizar o risco de *overfitting*. Cada experimento aborda um conjunto de dados específico, permitindo avaliar a generalizabilidade e consistência do desempenho das diferentes arquiteturas. A separação dos dados foi realizada conforme Seção 5.1. Assim sendo, para simplificar, nomeou-se as arquiteturas de Quinn et al. (2016), Panicker et al. (2018), Simon et al. (2019) e Udegova, Shelomentseva e Chentsov (2021) como CNN-A, CNN-B, CNN-C e CNN-D, respectivamente. Os experimentos realizados com cada um dos quatro datasets foram nomeados de Experimentos 1.1, Experimentos 1.2, Experimentos 1.3 e Experimentos 1.4, respectivamente.

No Experimento 1.1, foi utilizado o dataset-1, juntamente com a abordagem 5-*fold cross-validation*, para avaliar os modelos gerados pelas arquiteturas CNNs que foram implementadas. Devido ao volume considerável de fragmentos de imagem envolvidos, o tempo total requerido para o treinamento/teste (nos 5-*fold*) das CNNs foi de aproximadamente 71 horas. Após os treinamentos e testes realizados em cada uma das cinco divisões do 5-*fold*, foi realizado o cálculo da média das métricas de avaliação empregadas. Os resultados são mostrados na Tabela 14.

Como pode ser visto neste experimento, os resultados obtidos em cada CNN foram próximos aos relatados pelos autores originais, mesmo utilizando alguns parâmetros e conjuntos de dados diferentes. Por exemplo, na métrica acurácia (adotada por Panicker et al. (2018), Udegova, Shelomentseva e Chentsov (2021), Simon et al. (2019)) e AUC (adotada por Quinn et al. (2016)), a diferença foi de aproximadamente 0,03. Observando os resultados, é identificado que a CNN-B apresentou o melhor desempenho. Isso pode ter ocorrido pelo fato de que a CNN-B foi proposta (e aprimorada) no conjunto de dados proposto por Costa et al. (2014). Vale ressaltar que as outras CNNs também mostraram bons resultados, obtendo valores superiores a 0,95 em todas as métricas avaliadas.

O Experimento 1.2 foi realizado de forma semelhante ao primeiro, no entanto, o dataset-2 foi utilizado. O tempo total de treinamento/teste com este conjunto de dados foi de aproximadamente 80 horas. Os resultados são apresentados na Tabela 15. Neste experimento, os valores encontrados também estavam próximos aos valores relatados pelos autores originais, de aproximadamente 0,02 (considerando acurácia apresentada em Panicker et al. (2018), Udegova, Shelomentseva e Chentsov (2021), Simon et al. (2019) e AUC por Quinn et al. (2016)). Entre as CNNs testadas, não há nitidamente uma arquitetura que se destaca. Entretanto, as CNN-B e CNN-D apresentaram desempenhos similares, evidenciando um empate nas métricas de acurácia, F-*measure* e AUC. Sendo que a CNN-D obteve uma ligeira superioridade ao avaliar a especificidade e precisão. No entanto, observa-se que todas as CNNs tiveram boa generalização, e em todas as métricas avaliadas, todas as CNNs obtiveram valores superiores a 0,96.

No Experimento 1.3, o treinamento e teste das CNNs implementadas foram realizados usando o dataset-3. Neste experimento, a CNN-D obteve melhores resultados em praticamente todas as métricas quando comparada com as demais CNNs, como pode ser visto na Tabela 16). Além disso, todas as CNNs obtiveram valores superiores a 0,97 e próximos aos valores relatados pelos autores originais. É interessante observar que praticamente todos os resultados obtidos no dataset-3 foram superiores aos obtidos nos demais conjuntos de dados. Isso pode estar relacionado a três razões: (i) a forma como as imagens foram capturadas, como o tipo de amostra, câmera e microscópio utilizado, e/ou (ii) o número de imagens de fragmentos usadas no treinamento e teste, e/ou (iii) o método adotado para anotar as imagens. O tempo total de treinamento/teste foi de aproximadamente 13 horas.

Finalmente, conduziu-se o Experimento 1.4, empregando o dataset-4 em um procedimento semelhante aos experimentos já realizados. Ao analisar os resultados apresentados na Tabela 17, percebe-se que, mais uma vez, a CNN-D superou as demais arquiteturas. A superioridade da CNN-D nos experimentos realizados talvez possa ser atribuída ao fato de que essa arquitetura foi concebida por meio de experimentos de otimização automatizada, visando refinar suas camadas para a tarefa especializada de classificação de bacilos da TB. Na Tabela 17, também é observado que todas as CNNs valores acima de 0,90 em todas as métricas avaliadas. O tempo total de treinamento/teste foi de aproximadamente 20 horas.

É importante ressaltar que todas as CNNs alcançaram bons resultados, atingindo alta sensibilidade e especificidade, métricas importantes a serem consideradas em um cenário de diagnóstico médico. Em adição, todos os resultados obtidos se aproximam dos apresentados pelos autores, evidenciando a robustez dos modelos gerados. Em outras palavras, os modelos gerados poderiam ser usados no desenvolvimento de softwares que ajudam no diagnóstico da TB, conforme relatado por todos os autores. Entretanto, para realizar uma avaliação mais abrangente e precisa da capacidade das CNNs em identificar

os bacilos da TB, é imprescindível conduzir testes que se concentrem na detecção do bacilo, ao invés de se limitar à classificação de fragmentos isolados, como foi o caso dos experimentos descritos nesta seção.

Tabela 14 – Resultados obtidos usando o dataset-1 com as arquiteturas CNNs replicadas.

CNN	Acurácia	Sensibilidade	Especificidade	Precisão	F-*measure*	AUC
CNN-A	0,9664 (±0,0067)	0,9607 (±0,0183)	0,9721 (±0,0097)	0,9719 (±0,0094)	0,9661 (±0,0072)	0,9664 (±0,0067)
CNN-B	**0,9740** (±0,0075)	0,9725 (±0,0136)	**0,9756** (±0,0080)	**0,9754** (±0,0080)	**0,9739** (±0,0077)	**0,9740** (±0,0075)
CNN-C	0,9554 (±0,0053)	0,9573 (±0,0184)	0,9536 (±0,0133)	0,9540 (±0,0118)	0,9554 (±0,0057)	0,9555 (±0,0053)
CNN-D	0,9666 (±0,0081)	**0,9741** (±0,0154)	0,9592 (±0,0161)	0,9600 (±0,0148)	0,9668 (±0,0081)	0,9666 (±0,0081)

Tabela 15 – Resultados obtidos usando o dataset-2 com as arquiteturas CNNs replicadas.

CNN	Acurácia	Sensibilidade	Especificidade	Precisão	F-*measure*	AUC
CNN-A	0,9789 (±0,0022)	0,9782 (±0,0037)	**0,9797** (±0,0049)	**0,9796** (±0,0048)	0,9789 (±0,0022)	0,9789 (±0,0022)
CNN-B	**0,9790** (±0,0029)	**0,9830** (±0,0017)	0,9749 (±0,0069)	0,9751 (±0,0066)	**0,9790** (±0,0028)	**0,9790** (±0,0029)
CNN-C	0,9709 (±0,0034)	0,9645 (±0,0081)	0,9774 (±0,0039)	0,9771 (±0,0037)	0,9707 (±0,0036)	0,9709 (±0,0034)
CNN-D	**0,9790** (±0,0036)	0,9796 (±0,0030)	0,9783 (±0,0070)	0,9784 (±0,0068)	**0,9790** (±0,0036)	**0,9790** (±0,0036)

Tabela 16 – Resultados obtidos usando o dataset-3 com as arquiteturas CNNs replicadas.

CNN	Acurácia	Sensibilidade	Especificidade	Precisão	F-*measure*	AUC
CNN-A	0,9896 (±0,0031)	0,9888 (±0,0031)	0,9904 (±0,0073)	0,9905 (±0,0071)	0,9896 (±0,0031)	0,9896 (±0,0032)
CNN-B	0,9821 (±0,0106)	**0,9907** (±0,0031)	0,9736 (±0,0232)	0,9745 (±0,0216)	0,9824 (±0,0101)	0,9821 (±0,0106)
CNN-C	0,9851 (±0,0044)	0,9857 (±0,0032)	0,9846 (±0,0074)	0,9846 (±0,0074)	0,9851 (±0,0043)	0,9851 (±0,0044)
CNN-D	**0,9905** (±0,0031)	0,9903 (±0,0013)	**0,9906** (±0,0055)	**0,9906** (±0,0055)	**0,9905** (±0,0031)	**0,9905** (±0,0031)

Tabela 17 – Resultados obtidos usando o dataset-4 com as arquiteturas CNNs replicadas.

CNN	Acurácia	Sensibilidade	Especificidade	Precisão	F-*measure*	AUC
CNN-A	0,9525 (±0,0057)	0,9603 (±0,0205)	0,9448 (±0,0220)	0,9463 (±0,0191)	0,9528 (±0,0055)	0,9525 (±0,0057)
CNN-B	0,9208 (±0,0445)	0,9077 (±0,1018)	0,9339 (±0,0280)	0,9335 (±0,0221)	0,9166 (±0,0538)	0,9208 (±0,0445)
CNN-C	0,9452 (±0,0096)	0,9568 (±0,0137)	0,9337 (±0,0216)	0,9356 (±0,0195)	0,9458 (±0,0091)	0,9452 (±0,0096)
CNN-D	**0,9665** (±0,0039)	**0,9664** (±0,0098)	**0,9667** (±0,0149)	**0,9669** (±0,0139)	**0,9665** (±0,0036)	**0,9665** (±0,0039)

6.2 Experimento 2: Detecção de bacilos de TB com K-means e CNN

Conforme discutido na seção anterior, todas as arquiteturas de CNN testadas exibiram um desempenho notável na tarefa de classificar fragmentos de imagens, distinguindo entre aqueles que contêm bacilos da TB e os que não contêm. No entanto, no âmbito do desenvolvimento de um sistema CADx e do entendimento das limitações de tais métodos, é importante avaliar o desempenho das CNNs em um cenário mais abrangente, englobando tanto a detecção quanto a classificação. Portanto, para assegurar a eficácia e a aplicabilidade das CNNs para diagnóstico da TB, é crucial conduzir experimentos que simulem a detecção dos bacilos em imagens de lâminas de microscópio.

Nesse contexto, foi realizado o Experimento 2, o qual emprega a técnica de segmentação de imagens por meio do método K-*means*, seguida pela classificação das Regiões de Interesse (ROIs) identificadas, utilizando uma CNN. Para isso, adotou-se o método detalhado na Seção 5.3. Visando avaliar a eficácia desse método, foi utilizado a abordagem 5-*fold cross-validation*, sendo utilizado as mesmas distribuições dos dados em cada um dos *folds* gerados para treinamento/validação/teste nos Experimentos 1.1 até 1.4.

Considerando a inviabilidade de testar todas as arquiteturas em função do tempo computacional necessário e dos objetivos específicos deste trabalho, decidiu-se conduzir o Experimento 2 exclusivamente com a arquitetura CNN-D, proposta por Udegova, Shelomentseva e Chentsov (2021). Esta escolha se justifica pelo fato de que a CNN-D demonstrou melhor desempenho na maioria dos testes realizados anteriormente, além de ter sido proposta em um estudo que focou em realizar experimentos de otimização da arquitetura para o contexto da baciloscopia.

Os resultados do Experimento 2 estão dispostos na Tabela 18, e a análise dos mesmos revela um desempenho insatisfatório da CNN no âmbito da detecção. As métricas de precisão, sensibilidade, F-*measure* e AP atingiram, em média, valores de 0,4261, 0,8800, 0,5658 e 0,4670, respectivamente. Ao analisar esses resultados, é possível perceber que o método K-*means* foi capaz de identificar uma grande quantidade dos fragmentos que contêm bacilos, conforme indicado pelos valores elevados de sensibilidade. Entretanto, os baixos valores de precisão apontam para uma dificuldade da CNN em diferenciar eficazmente os fragmentos que possuem bacilos e aqueles que não possuem, resultando em um número significativo de falsos positivos. Considerando que foram utilizados os modelos já treinados na seção anterior, e a mesma distribuição dos dados, o tempo total para realizar esse experimento foi de aproximadamente 4 horas.

É importante destacar que as métricas utilizadas nos experimentos de detecção diferem das empregadas nos experimentos de classificação. Essa diferença se deve ao fato de que, no contexto de detecção, o cálculo do valor para a classe Verdadeiro Negativo (TN, do inglês *True Negative*) não é aplicável. Isso acontece porque qualquer fragmento originado

do fundo da imagem (região sem bacilos) que não seja classificado como bacilo poderia ser considerado um TN, resultando em um valor excessivamente elevado e comprometendo, assim, a análise de métricas que incorporam o TN em seu cálculo.

Tabela 18 – Resultados obtidos em cada conjunto de dados ao utilizar o K-*means* + CNN-D.

Dataset	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
dataset-1	0,3323 (±0,1244)	0,9576 (±0,0047)	0,4808 (±0,1334)	0,5031 (±0,1105)
dataset-2	0,3896 (±0,0391)	0,7866 (±0,0153)	0,5200 (±0,0359)	0,3708 (±0,0236)
dataset-3	0,4849 (±0,0487)	0,9178 (±0,0062)	0,6330 (±0,0405)	0,5171 (±0,0353)
dataset-4	0,4975 (±0,0527)	0,8578 (±0,0398)	0,6292 (±0,0524)	0,4770 (±0,0619)

Os resultados obtidos podem indicar uma possível superestimação do modelo gerado pela CNN, sugerindo que o desempenho notável observado na fase de classificação de fragmentos não se traduziu efetivamente para a aplicação prática na detecção de bacilos. Esta discrepância nos resultados poderia ser atribuída a diversos fatores, como por exemplo, a natureza intrínseca dos dados ou o método empregado na construção do conjunto de dados. A avaliação de fragmentos isolados em um cenário controlado, como na fase de classificação, fornece um ambiente favorável para o modelo, mas pode não refletir as condições mais amplas encontradas durante a detecção de bacilos em imagens que representam o campo de visão. Isso ressalta a importância de conduzir avaliações abrangentes em diferentes etapas e contextos, garantindo, assim, que o modelo não só alcance um alto desempenho em cenários idealizados, mas também seja robusto e confiável quando aplicado em condições reais e variadas.

6.3 Experimento 3: Aperfeiçoamento da CNN utilizando padrões negativos ampliados

Reconhecendo as limitações e desafios identificados nos resultados dos testes de detecção, optou-se por investigar a influência do conjunto de treinamento e teste, mais especificamente a composição dos fragmentos negativos, no comportamento e precisão do modelo. Neste sentido, em sequência aos experimentos e análises sobre o desempenho das CNNs, a seção atual dedica-se ao Experimento 3, avaliar a estratégia delineada na Seção 5.3.2. A qual se propõe melhorar o desempenho do modelo a partir da ampliação dos conjuntos de treinamento/validação com padrões negativos mais representativos.

Adotando o método delineado na Figura 36, visando ampliar o conjunto de padrões negativos empregados no treinamento de um novo modelo de CNN, procedeu-se com a classificação dos conjuntos de treinamento/validação de cada *fold*, pertencentes a cada conjunto de dados, por meio dos modelos gerados durante o Experimento 2, usando a estratégia k-*means* juntamente com CNN. Ou seja, o modelo gerado por um conjunto de

treinamento/validação foi usado para classificar os próprios conjuntos de treinamento/-validação (Figura 36a-h). Gerando assim a identificação e coleta de todos os fragmentos equivocadamente classificados como bacilos — Falsos Positivos (Figura 36i). Na sequência, esses fragmentos foram integrados aos conjuntos de treinamento/validação, proporcionando um novo conjunto de dados ampliado (Figura 36j). A quantidade de fragmentos negativos incorporados a cada conjunto de dados, após o processo de ampliação, é detalhadamente apresentada na Tabela 19. É importante destacar que, durante esse processo, os conjuntos de teste permaneceram inalterados, assegurando, assim, sua integridade e garantindo que permanecessem completamente isolados dos dados utilizados para treinamento.

Tabela 19 – Quantidades média (considerando os *5-fold*) de fragmentos positivos e negativos ampliados em cada conjunto de dados.

	dataset-1	**dataset-2**	**dataset-3**	**dataset-4**
Treino	55.506 positivos 192.880 negativos	56.739 positivos 136.559 negativos	10.149 positivos 20.740 negativos	17.934 positivos 34.655 negativos
Validação	3.330 positivos 12.764 negativos	6.397 positivos 16.124 negativos	1.314 positivos 2.749 negativos	2.194 positivos 3.795 negativos
Teste	14.709 positivos 14.709 negativos	15.784 positivos 15.756 negativos	2.866 positivos 2.866 negativos	5.032 positivos 5.032 negativos

Posteriormente, iniciou-se o processo de treinamento da arquitetura CNN-D com os conjuntos treinamento/validação com os padrões negativos ampliados (nomeado como CNN-D*)(Figura 36j-m). Esta etapa foi conduzida seguindo a abordagem 5-*fold cross-validation*, análogo à aplicada no Experimento 1, visando manter a consistência e comparabilidade dos resultados. Os desempenhos médios alcançados pela CNN-D* em cada um dos conjuntos de dados são apresentados na Tabela 21. Por fim, de posse dos novos modelos, o Experimento 2 foi refeito, ou seja, o k-*means* juntamente com CNN foram usados para detectar os bacilos da TB em imagens do campo de visão usando a CNN-D*(Figura 36n-r). O tempo total de treinamento da CNN-D* e detecção com o K-*means* + CNN, em todos os conjuntos de dados, foram de aproximadamente 72 horas e 4 horas, respectivamente. A Tabela 20 apresenta os resultados das novas detecções realizadas.

A partir da análise dos resultados, nota-se uma melhora no desempenho do método. Em termos de precisão, sensibilidade, F-*measure* e AP, as médias alcançadas foram, respectivamente, 0,7844, 0,7024, 0,7198 e 0,6124. Ao comparar os valores apresentados nas Tabelas 18 e 20, percebe-se um aumento expressivo na precisão e uma diminuição na sensibilidade. Apesar de algumas implicações potencialmente negativas deste cenário, é importante destacar que o F-*measure* apresentou um aumento significativo, indicando que, ao balancear precisão e sensibilidade, o desempenho geral do método mostrou avanços. Até certo ponto, é plausível esperar tal comportamento, ao elevar a precisão, o modelo se torna mais criterioso na detecção de bacilos, o que pode acarretar na não identificação de alguns deles, refletindo na diminuição da sensibilidade.

Esses resultados reforçam a hipótese de que existe, de fato, uma superestimação nos modelos gerados pelas CNNs, que pode ser atribuída à falta de representatividade adequada da classe negativa. Adicionalmente, a possível presença de bacilos não anotados pode ser um fator que contribui para a redução da precisão do modelo, uma vez que, durante a avaliação, eles são erroneamente categorizados como Falsos Positivos, em vez de Verdadeiros Positivos. Este fenômeno destaca a necessidade crítica de avaliar tais métodos em cenários mais amplos e desafiadores, utilizando abordagens que gerem um volume maior de fragmentos para classificação, a exemplo do método k-*means*. A adoção de métodos mais eficientes para a obtenção de ROIs pode até mascarar essa superestimação, entretanto, é crucial considerar que tal abordagem pode acarretar implicações negativas, especialmente quando aplicada ao contexto real de diagnóstico da TB.

Tabela 20 – Resultados obtidos em cada conjunto de dados ao utilizar o K-means + CNN-D*.

Dataset	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
dataset-1	0,7783 (±0,0783)	0,9168 (±0,0124)	0,8394 (±0,0422)	0,7868 (±0,0456)
dataset-2	0,7269 (±0,0642)	0,3765 (±0,2188)	0,4710 (±0,1751)	0,3116 (±0,2112)
dataset-3	0,8202 (±0,0632)	0,7582 (±0,0401)	0,7859 (±0,0355)	0,6812 (±0,0414)
dataset-4	0,8121 (±0,0381)	0,7579 (±0,0358)	0,7830 (±0,0234)	0,6699 (±0,0362)

Tabela 21 – Resultados obtidos em cada conjunto de dados com arquitetura CNN-D utilizando quantidade ampliada de fragmentos negativos (CNN-D*).

Dataset	**Acurácia**	**Sensibilidade**	**Especificidade**	**Precisão**	**F-*measure***	**AUC**
dataset-1	0,9668 (±0,0064)	0,9633 (±0,0207)	0,9702 (±0,0122)	0,9702 (±0,0115)	0,9665 (±0,0069)	0,9668 (±0,0064)
dataset-2	0,9347 (±0,0148)	0,9938 (±0,0019)	0,8758 (±0,0300)	0,8893 (±0,0241)	0,9385 (±0,0132)	0,9348 (±0,0147)
dataset-3	0,9608 (±0,0111)	0,9940 (±0,0011)	0,9277 (±0,0219)	0,9324 (±0,0192)	0,9621 (±0,0104)	0,9608 (±0,0110)
dataset-4	0,9437 (±0,0053)	0,9706 (±0,0084)	0,9169 (±0,0124)	0,9211 (±0,0110)	0,9451 (±0,0051)	0,9437 (±0,0052)

6.4 Experimento 4: Comparação de métodos de detecção de objetos na identificação de bacilos de TB

Os experimentos anteriores concentraram-se na análise do potencial e das limitações das CNNs, uma técnica frequentemente empregada na literatura para detecção do bacilo da TB. Nesta seção, com o objetivo de aprofundar a exploração do uso de DL no diagnóstico da TB, é realizado o Experimento 4. Neste experimento, métodos relevantes na literatura de detecção de objetos são avaliados, especificamente, no contexto da identificação do bacilo da TB, conforme detalhado na Seção 5.4. Os experimentos realizados com cada um dos quatro conjunto de dados, usando os sete métodos selecionados, foram nomeados de Experimentos 4.1, Experimentos 4.2, Experimentos 4.3 e Experimentos 4.4. A técnica de 5-*fold cross-validation* foi empregada para conduzir o treinamento e a avaliação dos métodos usados.

No Experimento 4.1, os métodos selecionados foram treinados para identificar os bacilos presentes nas imagens do dataset-1. Os resultados referentes ao desempenho de cada abordagem são apresentados na Tabela 22. Observa-se que o método SSD apresenta os melhores resultados, seguido pelo FSAF. Por outro lado, as variantes da YOLO e a EfficientDet não demonstraram a mesma eficácia, refletindo em uma generalização inferior dos dados. Ao realizar uma análise comparativa com a abordagem de detecção implementada pelo K-*means* + CNN-D*, nota-se que os métodos avaliados neste experimento não alcançaram resultados tão promissores com o dataset-1. Esses resultados ficaram aquém das expectativas, especialmente quando se considera o histórico de sucesso desses métodos em detectar variados tipos de objetos em diferentes contextos, conforme reportado nos trabalhos de seus respectivos autores. Talvez uma das justificativas para esse desempenho possa estar relacionado aos parâmetros padrões utilizados ou as características intrínsecas do dataset-1. Neste experimento, o tempo total de treinamento e teste foi de aproximadamente 26 horas.

Por outro lado, nos Experimentos 4.2, 4.3 e 4.4, que foram realizados com os datasets-2, 3 e 4, respectivamente, os métodos de detecção exibiram um desempenho mais consistente e homogêneo entre os conjuntos de dados. Conforme demonstrado nos resultados apresentados nas Tabelas 23, 24 e 25, a Faster R-CNN destacou-se entre as abordagens, superando não apenas os demais métodos de detecção, mas também o desempenho do método K-*means* + CNN-D* nestes conjuntos de dados. Nestes experimentos, alguns métodos apresentaram baixo desempenho, tais como EfficientDet e YOLOF, os quais o valor do F-*meansure* foram abaixo de 0,54. O tempo total de treinamento e teste para os Experimentos 4.2, 4.3 e 4.4 foram de aproximadamente 193, 30, e 34 horas, respectivamente.

É interessante observar que os resultados apresentados mostram uma tendência comparável àquela observada nos Experimentos 2 e 3, onde há uma relação inversa entre

precisão e sensibilidade. Os métodos que exibiram altas taxas de precisão não foram necessariamente os mesmos que apresentaram elevada sensibilidade, e vice-versa. Quando um modelo adota uma postura mais rigorosa na identificação de bacilos, ele pode acabar por não reconhecer alguns deles, resultando em uma queda na sensibilidade.

Por fim, ao analisar os resultados como um todo, observa-se que o método Faster R-CNN obtive o melhor resultado geral. Tendo em média valores de 0,8288, 0,8003, 0,7914 e 0,7097 para precisão, sensibilidade, F-*measure* e AP, respectivamente, na detecção dos bacilos da TB nos quatro conjuntos de dados.

Tabela 22 – Resultados obtidos usando o dataset-1 com os métodos de detecção utilizados.

Método	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
EfficientDet	0,9748 (±0,0172)	0,0543 (±0,0119)	0,1025 (±0,0213)	0,0524 (±0,0106)
Faster R-CNN	0,9467 (±0,0296)	0,4851 (±0,0380)	0,6410 (±0,0388)	0,4678 (±0,0422)
FSAF	0,9790 (±0,0083)	0,4954 (±0,0613)	0,6554 (±0,0547)	0,4879 (±0,0593)
RetinaNet	0,9660 (±0,0096)	0,4071 (±0,0278)	0,5723 (±0,0276)	0,3971 (±0,0283)
SSD	**0,9838** (±0,0061)	**0,4961** (±0,0483)	**0,6580** (±0,0416)	**0,4888** (±0,0445)
YOLOF	0,2310 (±0,2982)	0,0023 (±0,0044)	0,0045 (±0,0086)	0,0016 (±0,0031)
YOLOv5	0,9829 (±0,0088)	0,3202 (±0,0631)	0,4793 (±0,0718)	0,3156 (±0,0610)

Tabela 23 – Resultados obtidos usando o dataset-2 com os métodos de detecção utilizados.

Método	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
EfficientDet	**0,9581** (±0,0022)	0,1154 (±0,0176)	0,2055 (±0,0281)	0,1116 (±0,0173)
Faster R-CNN	0,7334 (±0,0214)	**0,9387** (±0,0124)	**0,8231** (±0,0092)	**0,7826** (±0,0143)
FSAF	0,9004 (±0,0153)	0,6261 (±0,0452)	0,7372 (±0,0251)	0,5937 (±0,0364)
RetinaNet	0,9078 (±0,0033)	0,5002 (±0,0224)	0,6447 (±0,0179)	0,4707 (±0,0214)
SSD	0,9223 (±0,0071)	0,4463 (±0,0260)	0,6010 (±0,0235)	0,4269 (±0,0243)
YOLOF	0,8768 (±0,0146)	0,3873 (±0,0209)	0,5370 (±0,0220)	0,3571 (±0,0217)
YOLOv5	0,9318 (±0,0049)	0,6217 (±0,0272)	0,7455 (±0,0200)	0,6032 (±0,0273)

Tabela 24 – Resultados obtidos usando o dataset-3 com os métodos de detecção utilizados.

Método	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
EfficientDet	**0,9460** (±0,0306)	0,1739 (±0,0392)	0,2920 (±0,0580)	0,1668 (±0,0400)
Faster R-CNN	0,8289 (±0,0374)	**0,9356** (±0,0143)	**0,8784** (±0,0206)	**0,8278** (±0,0278)
FSAF	0,9296 (±0,0226)	0,7259 (±0,0516)	0,8136 (±0,0299)	0,7010 (±0,0467)
RetinaNet	0,9285 (±0,0225)	0,5314 (±0,0365)	0,6748 (±0,0273)	0,5040 (±0,0331)
SSD	0,9314 (±0,0218)	0,4877 (±0,0497)	0,6381 (±0,0394)	0,4639 (±0,0415)
YOLOF	0,8910 (±0,0313)	0,3869 (±0,0403)	0,5379 (±0,0373)	0,3497 (±0,0356)
YOLOv5	0,9354 (±0,0173)	0,7405 (±0,0415)	0,8258 (±0,0268)	0,7153 (±0,0389)

Tabela 25 – Resultados obtidos usando o dataset-4 com os métodos de detecção utilizados.

Método	Precisão	Sensibilidade	F-*measure*	AP
EfficientDet	**0,9659** (±0,0089)	0,1960 (±0,0604)	0,3217 (±0,0809)	0,1912 (±0,0603)
Faster R-CNN	0,8060 (±0,0162)	**0,8419** (±0,0279)	**0,8232** (±0,0157)	**0,7604** (±0,0258)
FSAF	0,9469 (±0,0246)	0,5374 (±0,1475)	0,6717 (±0,1138)	0,5197 (±0,1373)
RetinaNet	0,9332 (±0,0260)	0,4575 (±0,1085)	0,6059 (±0,0918)	0,4415 (±0,1032)
SSD	0,9445 (±0,0136)	0,5281 (±0,0686)	0,6746 (±0,0548)	0,5118 (±0,0671)
YOLOF	0,8819 (±0,0197)	0,3756 (±0,0919)	0,5202 (±0,0909)	0,3344 (±0,0818)
YOLOv5	0,9488 (±0,0121)	0,5523 (±0,0655)	0,6954 (±0,0513)	0,5335 (±0,0599)

6.5 Experimento 5: Aplicação e avaliação da estratégia de partição de imagens com Faster R-CNN

Embora métodos como a Faster R-CNN possuam vantagens como lidar com objetos de pequena dimensão (NGUYEN et al., 2020; LIU et al., 2021), a identificação do bacilo, dado seu formato e tamanho reduzido em relação ao tamanho total da imagem, configura-se como uma tarefa complexa. Essa complexidade é acentuada não só para os especialistas humanos, mas se torna exponencialmente mais desafiadora para os métodos baseados em aprendizado profundo. Essa realidade fica evidente ao analisar os resultados dos Experimentos 4 em contraposição às dimensões médias das imagens apresentadas na Tabela 10. Observa-se que no dataset-1, que possui um tamanho médio de imagem superior aos demais, a SSD conseguiu ser o método com melhores resultados. Em contraste, nos datasets-2, 3 e 4, que apresentam imagens com dimensões médias um pouco menores e similares entre si, a Faster R-CNN obteve os resultados mais satisfatórios.

Neste contexto, o Experimento 5 é proposto para avaliar a estrategia de partição das imagens, descrita detalhadamente na Seção 5.4.1. A qual busca minimizar os efeitos adversos causados pela variabilidade no tamanho das imagens e lidar com a heterogeneidade dos dados que podem se manifestar em diversos contextos laboratoriais reais. Para a realização deste experimento, optou-se pelo uso do método Faster R-CNN, aplicado a detecção das imagens previamente particionadas (conforme etapa apresentada na Figura 39h.

O Experimento 5 foi conduzido utilizando-se os mesmos *folds* separados no Experimento 4, visando proporcionar uma análise comparativa mais justa e equilibrada. Este experimento não foi realizado com o dataset-2, pois as dimensões das imagens presentes neste conjunto de dados são inferiores ao fator estabelecido para a divisão das imagens. Consequentemente, a implementação do processo de particionamento não provocaria alterações significativas nos resultados obtidos, resultando em uma situação praticamente idêntica àquela observada no Experimento 4, onde as imagens foram processadas sem a etapa de partição. Para realizar o treinamento e teste neste experimento, um tempo total de aproximadamente 40 horas foi gasto.

Os resultados do Experimento 5 são apresentados na Tabela 26, juntamente com os resultados sumarizados dos Experimentos 4 realizados com a Faster R-CNN. Uma análise comparativa entre os dois experimentos revela que a técnica de partição de imagens proporcionou uma melhoria significativa no desempenho, especialmente no que tange à sensibilidade e AP. O ganho mais expressiva pode ser notada ao comparar com os dados dispostos na Tabela 22, referentes ao dataset-1. Em que percebe-se que a Faster R-CNN, quando submetida ao método de partição de imagens, não apenas superou o desempenho do SSD, como também promoveu um aumento substancial nos índices de sensibilidade.

Essa abordagem além de aprimorar a detecção dos bacilos, também forneceu um melhor entendimento sobre como a heterogeneidade das imagens pode impactar métodos baseados em DL, e de que maneira estratégias de pré-processamento podem ser implementadas para mitigar tais efeitos e aprimorar o desempenho dos modelos em cenários aplicados. A melhora do desempenho provida pelo método de partição de imagens, principalmente o dataset-1, destaca o potencial dessa estratégia em otimizar o processo de detecção, minimizando os desafios impostos pela variabilidade de tamanho das imagens e, por consequência, contribuindo para obtenção de resultados mais precisos e confiáveis. Se mostrando, assim, a melhor estratégia implementada neste trabalho.

Tabela 26 – Resultados obtidos ao realizar o Experimento 5 (Exp. 5) e resultados sumarizados do Experimento 4 (Exp. 4) utilizando a Faster R-CNN.

	Dataset	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
Exp. 4	dataset-1	0,9467 (±0,0296)	0,4851 (±0,0380)	0,6410 (±0,0388)	0,4678 (±0,0422)
	dataset-3	0,8289 (±0,0374)	0,9356 (±0,0143)	0,8784 (±0,0206)	0,8278 (±0,0278)
	dataset-4	0,8060 (±0,0162)	0,8419 (±0,0279)	0,8232 (±0,0157)	0,7604 (±0,0258)
Exp. 5	dataset-1	0,9494 (±0,0155)	0,7978 (±0,0260)	0,8669 (±0,0196)	0,7677 (±0,0340)
	dataset-3	0,8328 (±0,0352)	0,9417 (±0,0100)	0,8835 (±0,0212)	0,8401 (±0,0301)
	dataset-4	0,8061 (±0,0258)	0,8681 (±0,0101)	0,8358 (±0,0157)	0,7820 (±0,0191)

6.6 Experimento 6: Avaliação Transversal dos Modelos em Diferentes Conjuntos de Dados

Até o momento, todos os experimentos realizados empregaram a técnica de 5-*fold cross-validation*, adotando uma abordagem na qual os dados de treinamento, validação e teste provêm do mesmo conjunto de dados. Contudo, com o intuito de aprofundar a avaliação e entender melhor as capacidades e limitações dos métodos de DL em cenários mais heterogêneos e que se assemelhem mais à realidade prática, propõe-se a realização do Experimento 6. O Experimento 6 tem como foco principal avaliar a capacidade de generalização dos modelos gerados por um determinado conjunto de dados, investigando se estes modelos são capazes de identificar os bacilos em imagens oriundas de outros conjuntos de dados, que foram criados de maneiras distintas, e que podem até mesmo

ser provenientes de laboratórios ou países diferentes. Detalhes adicionais e a metodologia específica adotada para este experimento podem ser encontrados na Seção 5.5.

Neste experimento, cada um dos quatro conjuntos de dados foi segmentado em conjuntos distintos para Treinamento, Validação e Teste, seguindo as proporções de 70%, 10% e 20%, respectivamente. Esse procedimento facilita a realização de treinamentos com um conjunto de dados e, posteriormente, a realização de testes com outro, promovendo uma avaliação mais confiável e justa. Para a execução das tarefas de treinamento e detecção dos bacilos de TB, optou-se pelo emprego do Faster R-CNN, incorporando a estratégia de partição de imagens, técnica que se destacou e apresentou os melhores resultados ao longo deste trabalho. O tempo total para a realização de todos os treinamentos e testes foi de aproximadamente 16 horas. Para simplificar, o método Faster R-CNN treinado com os conjuntos de treinamento de cada um dos dataset-1, dataset-2, dataset-3 e dataset-4 foram nomeados como M1, M2, M3 e M4, respectivamente.

Os resultados deste experimento oferecem duas perspectivas distintas para análise. Inicialmente, pode-se avaliar o desempenho individual que os métodos de detecção obtiveram em cada conjunto de dados, conforme Tabela 27. Como esperado, os melhores desempenhos em cada conjunto de teste foram alcançados pelos métodos que foram treinados com o respectivo conjunto de treinamento, resultando em valores superiores para sensibilidade, F-*measure* e AP (linhas destacadas de cinza na Tabela 27). Esta observação está alinhada com a lógica de que um modelo tende a apresentar melhor performance quando testado em dados que seguem as mesmas características daqueles utilizados durante o treinamento.

Analisando os valores do F-*meansure* na Tabela 27, é visto que os métodos enfrentaram maiores desafios no dataset-2, sendo que ao calcular as médias do F-*meansure* e AP, obtém-se os valores de 0,6888 e 0,5441, respectivamente. Por outro lado, o dataset-3 se destacou positivamente, evidenciando maior facilidade na identificação dos bacilos da TB, com médias de F-*meansure* e AP alcançando 0,7792 e 0,6940, respectivamente. Isso pode estar relacionado a fatores como resolução da imagem, o conjunto de cores do bacilo e do fundo da imagem, e/ou a forma como a anotação foi realizada. Além disso, nota-se que novamente os métodos com maiores índices de sensibilidade não necessariamente apresentam a maior precisão. Esse comportamento destaca a complexidade intrínseca da tarefa de detecção de bacilos, evidenciando que um equilíbrio entre sensibilidade e precisão não é simples de ser alcançado, uma vez que uma alta sensibilidade, embora desejável, pode vir acompanhada de um número elevado de falsos positivos, reduzindo a precisão do modelo.

A segunda perspectiva para análise dos resultados deste experimento envolve a observação do desempenho médio de cada método, calculando-se a média das métricas calculadas nos quatro conjuntos de dados. Essa abordagem proporciona uma visão global

sobre a eficácia de cada técnica, independente do conjunto de dados específico ao qual foi aplicada, como apresentando na Tabela 28. Nesta tabela, analisando os valores médios de desempenho (sensibilidade e AP), nota-se que o método treinado com o dataset-1 (M1) se sobressaiu, apresentando um desempenho superior em comparação aos demais. Por outro lado, o método treinado com o dataset-2 (M2) revelou-se o menos eficaz, conforme evidenciado pelos baixos valores de sensibilidade, F-*meansure* e AP. O dataset-2, em particular, mostrou-se significativamente distinto dos demais conjuntos de dados, impactando na capacidade de identificação de bacilos por métodos treinados em outros conjuntos de dados. Além disso, observou-se que o método M2, treinado com o dataset-2, não conseguiu generalizar adequadamente, apresentando dificuldades para detectar bacilos em imagens pertencentes a outros conjuntos de dados. Entretanto, apesar dessas dificuldades, os resultados gerais apresentam aspectos positivos. Mesmo que os desempenhos alcançados não sejam ideais para a implementação de um diagnóstico totalmente automatizado, é notável a capacidade dos métodos de DL em lidar com dados tão heterogêneos, provenientes de diferentes equipamentos e locais. Essa resiliência dos métodos de DL diante da variabilidade dos dados sugere um potencial significativo para aplicações práticas.

Tabela 27 – Resultados obtidos em cada conjunto de teste ao utilizando o método Faster R-CNN com partição das imagens.

Dataset Teste	**Método**	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
dataset-1	M1	0,9230	**0,8380**	**0,8784**	**0,7909**
dataset-1	M2	0,9076	0,1967	0,3234	0,1801
dataset-1	M3	0,9367	0,6596	0,7741	0,6288
dataset-1	M4	**0,9372**	0,7102	0,8081	0,6843
dataset-2	M1	0,5488	0,7708	0,6411	0,4792
dataset-2	M2	**0,7506**	**0,9128**	**0,8238**	**0,7906**
dataset-2	M3	0,6409	0,7071	0,6724	0,4775
dataset-2	M4	0,5366	0,7276	0,6177	0,4289
dataset-3	M1	0,6972	0,8750	0,7760	0,7394
dataset-3	M2	**0,8864**	0,4978	0,6376	0,4682
dataset-3	M3	0,8615	**0,9397**	**0,8989**	**0,8614**
dataset-3	M4	0,8148	0,7941	0,8043	0,7071
dataset-4	M1	0,6954	0,8915	0,7813	0,7454
dataset-4	M2	**0,7870**	0,2580	0,3886	0,2128
dataset-4	M3	0,7200	0,8325	0,7722	0,6671
dataset-4	M4	0,7830	**0,8942**	**0,8349**	**0,7790**

Tabela 28 – Média obtida por cada método treinado ao processar os quatro conjuntos de testes, utilizando o método Faster com partição das imagens.

Método	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
M1	0,7161 (±0,1338)	**0,8438** (±0,0464)	0,7692 (±0,0844)	**0,6887** (±0,1226)
M2	**0,8329** (±0,0658)	0,4663 (±0,2813)	0,5434 (±0,1999)	0,4129 (±0,2449)
M3	0,7898 (±0,1159)	0,7847 (±0,1095)	**0,7794** (±0,0803)	0,6587 (±0,1368)
M4	0,7679 (±0,1454)	0,7815 (±0,0722)	0,7663 (±0,0866)	0,6498 (±0,1323)

As conclusões derivadas do Experimento 6 ressaltam a importância de uma adequada correspondência entre os conjuntos de dados de treinamento e teste para assegurar a eficácia do modelo de DL, destacando a preferência por dados originados do mesmo contexto ou local. No cenário prático do uso de DL para diagnósticos médicos, esses resultados sugerem a vantagem de cada laboratório proceder com o treinamento utilizando seus próprios dados, e idealmente, realizar um treinamento específico para cada equipamento utilizado. Embora esta seja a situação ótima para alcançar o máximo desempenho dos métodos de DL, reconhece-se que implementar tal prática pode ser complexo e desafiador. No entanto, apesar destas complexidades, os resultados obtidos indicam que o emprego de modelos treinados com dados de outras fontes ainda pode ser benéfico. Esses métodos poderiam servir como ferramentas auxiliares para os especialistas, auxiliando na identificação dos bacilos da TB e contribuindo para uma análise mais acurada e eficiente. Essa abordagem colaborativa entre os métodos de DL e os profissionais de saúde pode, portanto, ser uma estratégia promissora, fomentando um ambiente de trabalho mais integrado e otimizado.

6.7 Experimento 7: Sistema de Votação para Combinação de Modelos na Detecção de Bacilos

Conforme discutido anteriormente no Capítulo 4 e na Seção 5.6, a tarefa de anotar imagens para treinar métodos de aprendizado profundo é uma atividade árdua e que demanda tempo, mesmo quando se faz uso de ferramentas que visam facilitar esse processo. Diante desse cenário, o Experimento 7 foi concebido com o objetivo de explorar uma nova estratégia, fundamentada em um sistema de votação, para atuar como um facilitador no processo de identificação e marcação dos bacilos em imagens do campo de visão, aproveitando da diversidade de diferentes conjuntos de dados. A estratégia proposta, explicada na Seção 5.6, busca integrar as capacidades distintas dos diversos métodos treinados (M1, M2, M3 e M4), visando melhorar precisão na identificação dos bacilos presentes nas imagens. Através dessa abordagem colaborativa, almeja-se agilizar significativamente o processo de anotação das imagens, contribuindo para a eficiência e precisão na preparação dos dados necessários para o treinamento de métodos de DL.

Como ilustrado na Figura 40, o objetivo é testar o sistema de votação usando M1, M2 M3, e M4, entretanto, foi realizados os seguintes combinações do sistema de votação: (i) M1, M2, M3 e M4; (ii) M1, M3 e M4; (iii) M1 e M3. Cada combinação dessa foi testada em cada conjunto de dados.

Conforme ilustrado nas Tabelas 29 e 30, a análise dos resultados da detecção de bacilos nos conjuntos de dados revela que a estratégia de combinar todos os métodos (M1 M2 M3 M4) resultou no desempenho menos satisfatório. Isso pode ser atribuído à

rigidez do critério adotado, que exige unanimidade entre todos os métodos para validar a presença de um bacilo em uma região específica da imagem. À medida que elimina-se os métodos de menor eficácia (ou seja, M2 e M4), observa-se uma melhora nos resultados. Sendo alcançado valores acima de 0,65 em todas as métricas avaliadas nos datasets-1, 3 e 4. No entanto, ao enfrentar o desafio de detectar bacilos de TB no dataset-2, essa configuração não atingiu marcas superiores a 0,80, reiterando a dificuldade já observada em experimentos anteriores para este conjunto específico de dados.

Tabela 29 – Resultados obtidos em cada conjunto de teste ao utilizando o método Faster R-CNN com partição das imagens em um sistema de votação.

Dataset Teste	**Métodos**	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
dataset-1	M1 M2 M3 M4	**0,9888**	0,2468	0,3951	0,2436
dataset-1	M1 M3 M4	0,9731	0,6747	0,7969	0,6571
dataset-1	M1 M3	0,9616	**0,6842**	**0,7996**	**0,6581**
dataset-2	M1 M2 M3 M4	**0,7843**	0,6518	0,7119	**0,5152**
dataset-2	M1 M3 M4	0,6258	0,6644	0,6445	0,4141
dataset-2	M1 M3	0,6217	**0,6772**	**0,6483**	0,42025
dataset-3	M1 M2 M3 M4	**0,9045**	0,4254	0,5786	0,3846
dataset-3	M1 M3 M4	0,8774	0,8271	0,8515	0,7326
dataset-3	M1 M3	0,8721	**0,8322**	**0,8517**	**0,7341**
dataset-4	M1 M2 M3 M4	**0,9031**	0,2976	0,4477	0,2711
dataset-4	M1 M3 M4	0,8206	0,8325	0,8265	0,7101
dataset-4	M1 M3	0,8093	**0,8458**	**0,8272**	**0,7165**

Tabela 30 – Média obtida ao processar os quatro conjuntos de testes, utilizando o método Faster com partição das imagens em um sistema de votação.

Métodos	**Precisão**	**Sensibilidade**	**F-*measure***	**AP**
M1 M2 M3 M4	0,8952 (±0,0728)	0,4054 (±0,1564)	0,5333 (±0,1229)	0,3536 (±0,1072)
M1 M3 M4	0,8242 (±0,1269)	0,7497 (±0,0802)	0,7799 (±0,0805)	0,6285 (±0,1268)
M1 M3	0,8162 (±0,1246)	0,7598 (±0,0793)	0,7817 (±0,0792)	0,6322 (±0,1256)

Para avaliar a contribuição da abordagem proposta em um contexto prático e em menor escala, foi concebido um segundo experimento, o Experimento 7.2. Para tal, o Especialista 1 foi convidado para participar da anotação de um conjunto de 50 imagens, distribuídas em dois cenários distintos, conforme detalhado na Seção 5.6. No Cenário A, o especialista se dedicou à anotação manual de 25 imagens, sem qualquer suporte da abordagem proposta. Já no Cenário B, o especialista realizou a anotação de outras 25 imagens, contando desta vez com a assistência da abordagem colaborativa, ou seja, as imagens já estavam previamente anotadas pelos métodos M1 e M3.

Ao analisar o tempo gasto em cada cenário, observou-se que, no Cenário A, o especialista levou 28 minutos e 45 segundos para anotar as imagens pertencentes a duas lâminas diferentes. Por outro lado, no Cenário B, o tempo necessário foi de apenas 7 minutos e 39 segundos para anotar 25 imagens diferentes das mesmas duas lâminas. Ao

comparar os tempos gastos nos dois cenários, fica evidente que a abordagem proposta foi capaz de proporcionar uma redução significativa, próxima de 73%, no tempo necessário para a realização das anotações.

Em um segundo momento de avaliação, foi focado na quantidade de bacilos identificados em cada cenário, realizando uma comparação detalhada. No Cenário A, o especialista detectou um total de 548 bacilos, enquanto a abordagem proposta identificou 523 bacilos. Ao comparar os bacilos identificados pela abordagem com aqueles identificados pelo especialista, observou-se uma taxa de concordância da abordagem foi de 96,75%. Já no Cenário B, o especialista identificou 418 bacilos, dos quais 97,85% foram identificados pela abordagem proposta e sugerida ao especialista.

Por fim, a abordagem proposta não buscou a perfeição imediata nas anotações, mas sim, a criação de um ponto de partida eficaz, que possa ser posteriormente aprimorado pela intervenção humana. A análise desses resultados, considerando o Experimento 7 e ambos os cenários do Experimento 7.2, ressalta o potencial significativo deste método para auxiliar o trabalho de anotação de imagens, contribuindo, assim, para a simplificação do processo. Ao facilitar substancialmente a criação e ampliação de novos conjuntos de dados, busca-se incentivar a criação de novos conjunto de dados, e consequentemente, a criação de modelos mais precisos e eficientes para a detecção do bacilo da TB.

6.8 Experimento 8: Diagnóstico de Lâminas de Baciloscopia

Embora o foco principal deste trabalho seja promover o uso de DL como uma ferramenta auxiliar no diagnóstico da TB, trabalhando em conjunto com um especialista no momento da contagem dos bacilos, é importante avaliar como uma IA poderia se comportar realizando o diagnóstico de maneira autônoma. Isso inclui analisar as imagens, contar os bacilos e, finalmente, apresentar um diagnóstico. Nessa perspectiva, este estudo estabeleceu uma parceria com o projeto "Contagem Automática de Bacilos Álcool Ácido Resistentes para o Diagnóstico da Tuberculose"(detalhado na Seção 5.8) para conduzir o Experimento 8.

Neste experimento é proposto avaliar um processo de diagnóstico ajustado às peculiaridades da análise de imagens via DL. Este processo considera adaptações nas regras convencionais de diagnóstico da TB por baciloscopia, de modo a contornar desafios inerentes ao uso de DL, como a presença de falsos negativos e/ou contagem densa de bacilos (como apresentado na Seção 5.8.3). Além disso, foram feitos ajustes específicos para adequar o diagnóstico ao formato em que o conjunto de dados foi construído.

Este experimento pode ser segmentado em duas fases. No Experimento 8.1, é avaliado o desempenho da Faster R-CNN, aplicando a estratégia de divisão de imagens, ao ser treinado com conjunto de dados (anotado) do projeto citado, a partir da abordagem

convencional de separação dos dados em conjuntos de treinamento, validação e teste. Já no Experimento 8.2, é estabelecido um cenário mais próximo da realidade, onde o processo de diagnóstico proposto é responsável por analisar 100 lâminas diferentes (com aproximadamente 400 imagens cada), com objetivo de estabelecer o diagnóstico para cada uma delas.

Ao realizar o Experimento 8.1, utilizando o conjunto de imagens selecionadas e anotadas apresentadas Seção 5.8.2 para treinar e testar a Faster R-CNN, foram alcançados valores de 0,8769, 0,9341, 0,9046, 0,8664, para precisão, sensibilidade, F-*measure* e AP, respectivamente. O desempenho alcançado se compara aos obtidos no Experimento 5, mostrando que a estratégia de particionamento de imagens juntamente com a Faster R-CNN realmente consegue alcançar uma generalização interessante dos dados, independentemente do conjunto de dados. O tempo total de treinamento e teste, usando o parâmetro de 25 épocas, foi de aproximadamente 5 horas.

Empregando o modelo treinado no experimento anterior como o método de detecção do processo de diagnóstico proposto (ilustrado na Figura 42), realizou-se o Experimento 8.2. Nesta fase, este processo foi executado em um conjunto composto por 100 lâminas, distribuídas da seguinte forma: 25 lâminas negativas, 25 lâminas positivas +, 25 lâminas positivas ++, e 25 lâminas positivas +++. Posteriormente, foram comparados os diagnósticos preditos pelo processo proposto com os diagnósticos realizados por especialistas. O tempo total de execução foi de aproximadamente 38 horas, sendo que em média foram necessários 23 minutos para processar as 400 imagens de cada lâmina.

Essa comparação é apresentada na matriz de confusão ilustrada na Figura 43. A partir da matriz de confusão, é possível calcular os indicadores de desempenho do método, alcançando valores de 0,5800 para acurácia, 0,4640 para sensibilidade, 0,7462 para precisão, 0,4640 para especificidade, e 0,5140 para F-*measure* (considerando as médias não ponderadas).

Em um primeiro momento, analisando o desempenho do processo de diagnóstico proposto com base apenas nas métricas apresentadas, o resultado pode parecer insatisfatório. Contudo, ao considerar a matriz de confusão sob uma perspectiva diferente, pode-se obter observações mais otimistas:

- Se não tratar a categoria "*To Check*"como um erro, mas como um indicativo para revisão humana, assumindo que um especialista seria capaz de diagnosticar corretamente esses casos, o processo proposto em conjunto com o especialista alcançaria aproximadamente 0,93 de acurácia, sensibilidade e especificidade.

- Sob a ótica da classe "*To Check*", seria necessário intervenção humana em apenas 35% dos diagnósticos para manter a alta acurácia mencionada anteriormente.

- Considerando que os diagnósticos equivocados estão relativamente próximos dos corretos (por exemplo, + classificado como ++, ++ como +++, e +++ como ++), e que estudos como Selvakumar et al. (2005), Trusov et al. (2009), Mosissa et al. (2016) já identificaram esse tipo de variação em alguns laboratórios ou centros de treinamento, pode-se considerar que o desempenho do processo proposto é parcialmente aceitável.

- Importante ressaltar que o processo proposto não classificou nenhuma lâmina negativa como positiva, ou vice-versa, um erro que poderia ter consequências sérias em um diagnóstico real.

- O tempo necessário para análise poderia ser significativamente reduzido com a implementação de soluções de processamento paralelo, possibilitando análises e diagnósticos mais rápidos que o especialista.

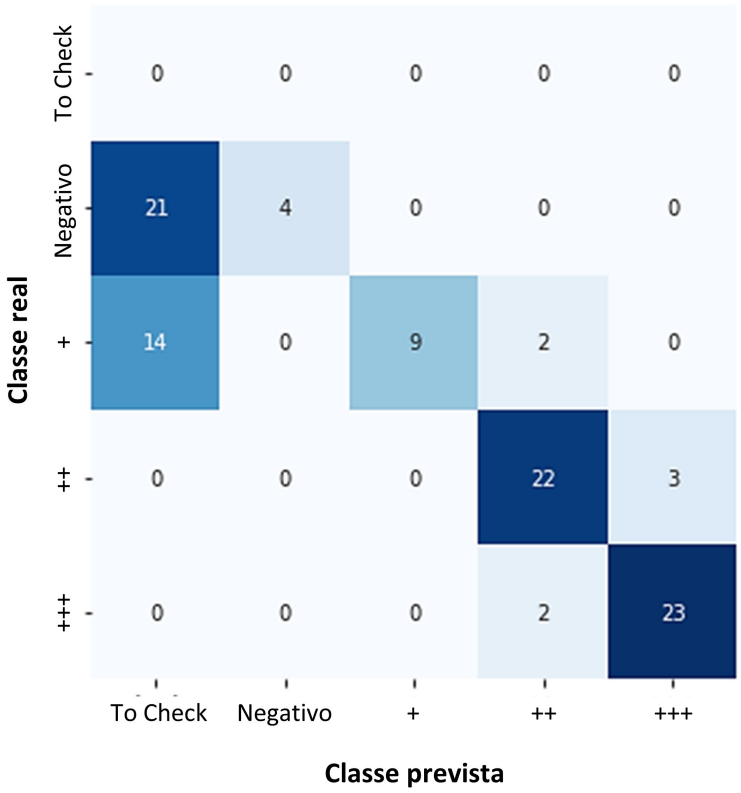


Figura 43 – Matriz de confusão gerada a partir do diagnóstico realizado pelo processo de diagnóstico proposto.

Além dos resultados positivos já mencionados, é interessante destacar o potencial dos métodos de DL para realizar a contagem extensiva de bacilos, uma atividade que pode ser cansativa em lâminas com alta carga bacteriana. Ao analisar uma lâmina positivo +++, um especialista pode não visualizar a lâmina por completo, e talvez não consiga quantificar com exatidão o total de bacilos presentes. Em contraste, um método de DL tem a capacidade de realizar uma análise detalhada de toda a lâmina. Isso é ilustrado na Figura

44, onde é apresentado um histograma da quantidade de bacilos de uma determinada lâmina, com as imagens agrupadas em conjuntos de 10. É possível observar que nas primeiras 80 imagens (equivalente a 20 campos de visão), já se pode estabelecer um diagnóstico de +++, e parar a leitura da lâmina (indicado pelas barras vermelhas). Contudo, o método de DL foi além, analisando todas as imagens disponíveis e contabilizando um total de 3817 bacilos. Apesar dessa precisão na contagem não ser estritamente necessária em um cenário de positivo +++, esse tipo de análise poderia ser extremamente útil para avaliar a gravidade e monitorar a evolução da doença em pacientes com TB. Portanto, ao incorporar tais técnicas de DL, talvez exista a oportunidade de enriquecer o processo diagnóstico.

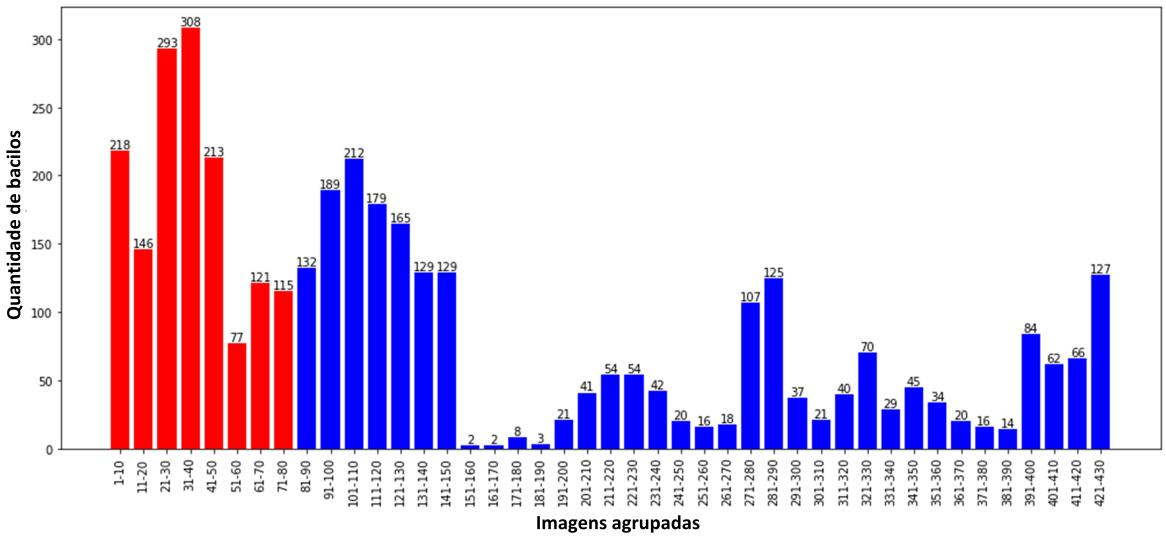


Figura 44 – Histograma com a distribuição de bacilos identificados pelo método de DL em um determinada lâmina durante o Experimento 8.2.

As análises realizadas evidenciam o enorme potencial dos métodos de DL na realização de diagnósticos da TB por meio da baciloscopia, podendo acelerar e aprimorar as análises realizadas com mínima intervenção de especialistas. No entanto, na ausência da categoria "*To Check*", o método teria mais dificuldades em classificar corretamente lâminas negativas e positivas +. A existência de falsos positivos associados às regras estabelecidas para diagnósticos (Tabela 1) poderia levar a um diagnóstico incorreto. Portanto, apesar do desempenho e potencial significativos alcançados pelos métodos de DL, para um diagnóstico 100% automatizado e sem intervenção humana, ainda se faz necessário um estudo mais aprofundado focado nesse objetivo.

6.9 Software de auxílio ao diagnóstico da TB

Nessa seção, são abordadas, em detalhes, as funcionalidades identificadas e implementadas durante a construção do protótipo de um sistema CADx destinado a apoiar o processo de diagnóstico da TB, um dos principais objetivos deste trabalho. Este protótipo

foi projetado para possuir uma interface intuitiva e melhorar a precisão e a eficiência na identificação de bacilos de TB, integrando a IA (por um método de DL) com o especialista. As funcionalidades foram identificadas a partir de observações, demonstrações e conversas com uma especialista responsável por realizar a baciloscopia.

Dado que o tempo de desenvolvimento deste trabalho ultrapassou o inicialmente previsto, o protótipo ainda não foi submetido a testes em um ambiente real de uso. Contudo, nos testes realizados em um ambiente controlado, ele demonstrou um desempenho promissor. Nesta seção, são apresentadas as características e a interface do protótipo desenvolvido para auxiliar no diagnóstico da TB.

O protótipo foi concebido para possibilitar a conexão a um smartphone via conexão sem fio[1](Wi-Fi) ou conexão via cabo[2]. Além disso, o sistema é flexível, permitindo a conexão a uma webcam (que poderia ser a câmera integrada ao microscópio) ou a leitura de imagens das lâminas armazenadas diretamente em um diretório do computador. Essa versatilidade assegura uma maior praticidade e adaptabilidade às diferentes necessidades operacionais, expandido, assim, as possibilidades de carregar imagens para serem analisadas. A Figura 45 apresenta um dos testes realizado do conjunto, protótipo / smartphone / microscópio.

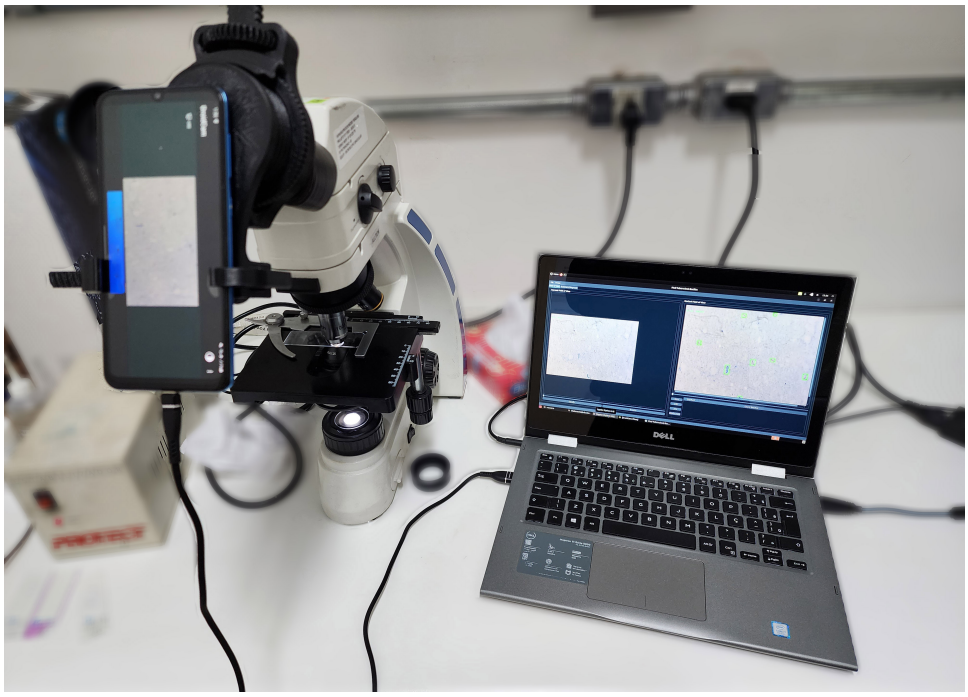


Figura 45 – Teste realizado durante integração do conjunto, protótipo / smartphone / microscópio.

Tendo como base o diagrama de caso de uso apresentado na Figura 46, as funcionalidades do protótipo (Figura 47) são descritas e ilustradas a seguir:

[1] Com o uso de um aplicativo apropriado, como por exemplo IP Webcam
[2] Com o uso de um aplicativo apropriado, como por exemplo o DroidCam

- **Carregar Imagem:** Esta funcionalidade permite ao usuário indicar a origem das imagens que contêm o campo de visão capturado ao realizar a baciloscopia, as quais serão processadas pelo método de DL. Essa funcionalidade é acessível pelo botão "*Image*", conforme ilustrado na Figura 48. Dependendo da opção escolhida, o sistema irá solicitar:
  - **Selecionar Imagem:** Utilizando a opção "*Load image*", o usuário deverá escolher uma imagem armazenada no computador.
  - **Informar IP da Câmera:** Selecionando "*Load IPCAMM*", será necessário fornecer a URL da câmera IP, a qual deve estar acessível através da mesma rede Wi-Fi.
  - **Selecionar Webcam:** A opção "*Load Webcam*"permite ao usuário escolher uma webcam conectada ao computador a partir de uma lista de dispositivos disponíveis. Esta opção também é válida para acessar a câmera de um smartphone, quando configurado adequadamente.
  - **Selecionar Pasta:** Por meio de "*Load Folder*", o usuário pode indicar o diretório no computador onde as imagens da lâmina estão salvas. Ao optar por este método, o sistema processará automaticamente todas as imagens presentes na pasta, iniciando a funcionalidade "Executar método DL".
- **Visualizar Imagem:** Uma vez carregada a fonte de imagem, esta é instantaneamente exibida na seção *"Current Field of View"* da interface, localizada à esquerda da tela, conforme pode ser visualizado na Figura 49. Esta área oferece funcionalidades de zoom para ampliar ou diminuir a imagem, facilitando assim, a observação detalhada dos bacilos.
- **Ajustar Cor:** A funcionalidade de ajuste de cor pode ser acessada ativando a opção *"Color Adjustment"* no menu "*Options*". Esta ferramenta disponibiliza controles para modificar contraste, brilho e saturação da imagem, permitindo ao usuário realçar a coloração dos bacilos em contraste com o fundo (Figura 50). É importante destacar que estes ajustes são aplicados apenas à visualização da imagem, não influenciando o processo de detecção realizado pelo método de DL.
- **Processar Imagem:** Ao acionar o botão "*Process*", o usuário dá início ao procedimento de identificação e contagem dos bacilos, tarefa esta conduzida pela funcionalidade "Executar Método DL".
- **Executar Método DL:** Esta função encaminha a imagem ao módulo de detecção, no qual o método de DL será aplicado para processar a imagem, identificando a localização dos bacilos presentes. Neste protótipo, optou-se pela implementação do método Faster R-CNN, aliado a uma técnica de particionamento de imagem, e o treinamento foi realizado utilizando o conjunto de dados do dataset-4.

- **Visualizar Resultado:** Concluído o processamento da imagem, o sistema exibe os resultados da detecção, indicando tanto a localização dos bacilos quanto o número total encontrado. Os resultados são apresentados na parte direita da interface, na seção denominada "*Marked Field of View*", conforme ilustrado na Figura 51a. Nessa janela, também é possível realizar zoom na imagem.

- **Alterar Quantidade de Bacilos:** Uma vez que o resultado é exibido, a contagem de bacilos identificada na imagem é apresentada. Caso o especialista identifique uma discrepância nesta contagem, ele tem a possibilidade de ajustar manualmente o valor exibido. Esta funcionalidade promove a colaboração entre o humano e a inteligência artificial, possibilitando a correção de eventuais erros cometidos pelo método (Figura 51b).

- **Editar Visualização:** Além da funcionalidade para ajustar numericamente a contagem de bacilos, o sistema oferece uma ferramenta que permite ao especialista realizar alterações diretamente na visualização da imagem. Ao acionar o botão "*Edit*", uma nova janela é aberta, proporcionando ao usuário a capacidade de interagir diretamente com a imagem (Figura 52), realizando as seguintes modificações:

  - **Remover Bacilo:** Para excluir a marcação de um bacilo, o especialista deve clicar uma vez com o botão direito do mouse sobre a caixa delimitadora correspondente.
  - **Marcar Bacilo:** Para adicionar a marcação de um bacilo, o especialista pode optar por desenhar manualmente uma caixa delimitadora, mantendo pressionado o botão esquerdo do mouse, ou simplesmente, clicar uma vez sobre o bacilo, permitindo que o sistema desenhe automaticamente a caixa ao redor do mesmo.

- **Adicionar Subcampo:** Após validar a detecção ou ajustar a contagem de bacilos, o especialista deve pressionar o botão "*Add*"para salvar a contagem realizada. Esta funcionalidade é crucial, pois, permite o armazenamento de múltiplas imagens referentes a um mesmo campo de visão, possibilitando a análise de diferentes focos e garantindo uma avaliação mais precisa. Além disso, esta ferramenta também é útil para a metodologia de captura detalhada na Seção 5.8.1, na qual cada campo de visão é composto por quatro imagens distintas, permitindo, assim, o agrupamento das imagens por campo de visão e a realização da contagem e proporção de bacilos na lâmina, de acordo com as diretrizes do Ministério da Saúde. Após adicionar uma imagem, o especialista pode processar a próxima, por exemplo, ajustando o foco no microscópio e acionando novamente o botão "*Process*". A cada vez que o botão "*Add*"é pressionado, as imagens processadas são compiladas em uma tabela para conferência, conforme ilustrado na Figura 53.

- **Finalizar campo:** Após processar as imagens de um campo de visão específico, é necessário concluir a análise no sistema. Para isso, o especialista deve clicar no botão "*Finalize*". Esta ação resulta na limpeza das informações na seção "*Marked Field of View*", permitindo ao especialista mover o microscópio (ou carregar uma nova imagem) para analisar o próximo campo de visão. Ao finalizar, as informações do campo de visão concluído são armazenadas e podem ser consultadas na aba "*Summary Diagnosis*", facilitando a conferência futura.

- **Visualizar resumo:** Durante ou no final da análise das imagens da lâmina, o especialista pode acessar um histórico dos campos de visão já processados. Na aba "*Summary Diagnosis*", é possível visualizar informações detalhadas sobre a análise em andamento, incluindo a data do diagnóstico, o número de campos analisados, o total de bacilos identificados, a média de bacilos por campo, o diagnóstico sugerido pela IA e o diagnóstico finalizado pelo especialista (conforme ilustrado na Figura 54.

- **Visualizar campo salvo:** Uma vez que um campo de visão é concluído e as informações são armazenadas no sistema, o especialista tem a possibilidade de acessar e revisar essa análise a qualquer momento. Para isso, é preciso apenas realizar um duplo clique com o botão esquerdo do mouse sobre o nome do campo desejado, localizado na tabela "*Fields*"na aba "*Summary Diagnosis*"(Figura 54). Isso possibilita uma revisão detalhada e uma validação adicional das análises realizadas.

- **Definir diagnóstico:** Quando o especialista conclui a análise do número de campos de visão que considera adequado para uma avaliação precisa, ele é direcionado para uma seção específica do sistema onde pode registrar o diagnóstico final. É importante ressaltar que, a IA desempenha um papel de auxílio na contagem dos bacilos, ou seja, a responsabilidade e a decisão final sobre o diagnóstico recaem sobre o especialista. A localização deste campo para registro do diagnóstico está destacada na Figura 55a. Para auxiliar o especialista, o sistema também realiza uma recomendação de diagnóstico, utilizando as regras baseadas na Tabela 1 (Figura 55b) .

- **Salvar diagnóstico:** Esta funcionalidade, botão "*Save*", assegura que todas as informações pertinentes ao diagnóstico, desde as imagens analisadas até o diagnóstico estabelecido pelo especialista, sejam devidamente armazenadas no disco rígido do computador, para possíveis consultas futuras e para manter um registro do exame (Figura 56a).

- **Abrir diagnóstico:** Com esta opção, botão "*Open*", o usuário tem a capacidade de carregar e revisar diagnósticos previamente salvos, garantindo que as análises possam ser reavaliadas ou utilizadas como referência para casos futuros (Figura 56b).

Para proporcionar uma compreensão mais clara e detalhada da interface e suas respectivas funcionalidades, as figuras a seguir (Figura 47 até Figura 56) foram elaboradas utilizando imagens de lâminas extraídas do conjunto de dados descrito na Seção 5.8.1. Com essas imagens ampliadas, a partir das capturas das telas, é possível tornar mais visíveis os bacilos, facilitando, assim, a compreensão do funcionamento do sistema. Em um cenário convencional, com observação direta por meio do microscópio usando um smartphone, o sistema exibiria a imagem integral do campo de visão, como ilustrado na Figura57a. Essa visualização completa permitiria ao especialista aplicar zoom para examinar os bacilos com maior detalhe, conforme demonstrado na Figura 57b.

Como visto nas imagens, a interface foi concebida para ser intuitiva e eficiente, visando potencializar a rapidez e facilitar o processo de contagem e registro dos bacilos de TB. A partir do uso de um método de DL, é possível agilizar o processo de localização dos bacilos na imagens, reduzindo, assim, o tempo necessário para realizar o diagnóstico. Além disso, é interessante observar que a integração entre o protótipo, o smartphone e o microscópio pode oferecer benefícios adicionais. Entre eles, destaca-se a melhoria na ergonomia e redução da fadiga ocular do especialista, que, ao invés de se debruçar sobre o ocular do microscópio, pode manter uma postura mais adequada enquanto analisa o campo de observação diretamente no computador. Além disso, o sistema proporciona a capacidade de auditoria. Em situações reais, as lâminas tendem a perder coloração ao longo do tempo, o que dificulta revisões posteriores. Contudo, com as imagens devidamente armazenadas no sistema, é possível realizar novas análises sempre que necessário, garantindo, portanto, uma maior confiabilidade e rastreabilidade do diagnóstico.

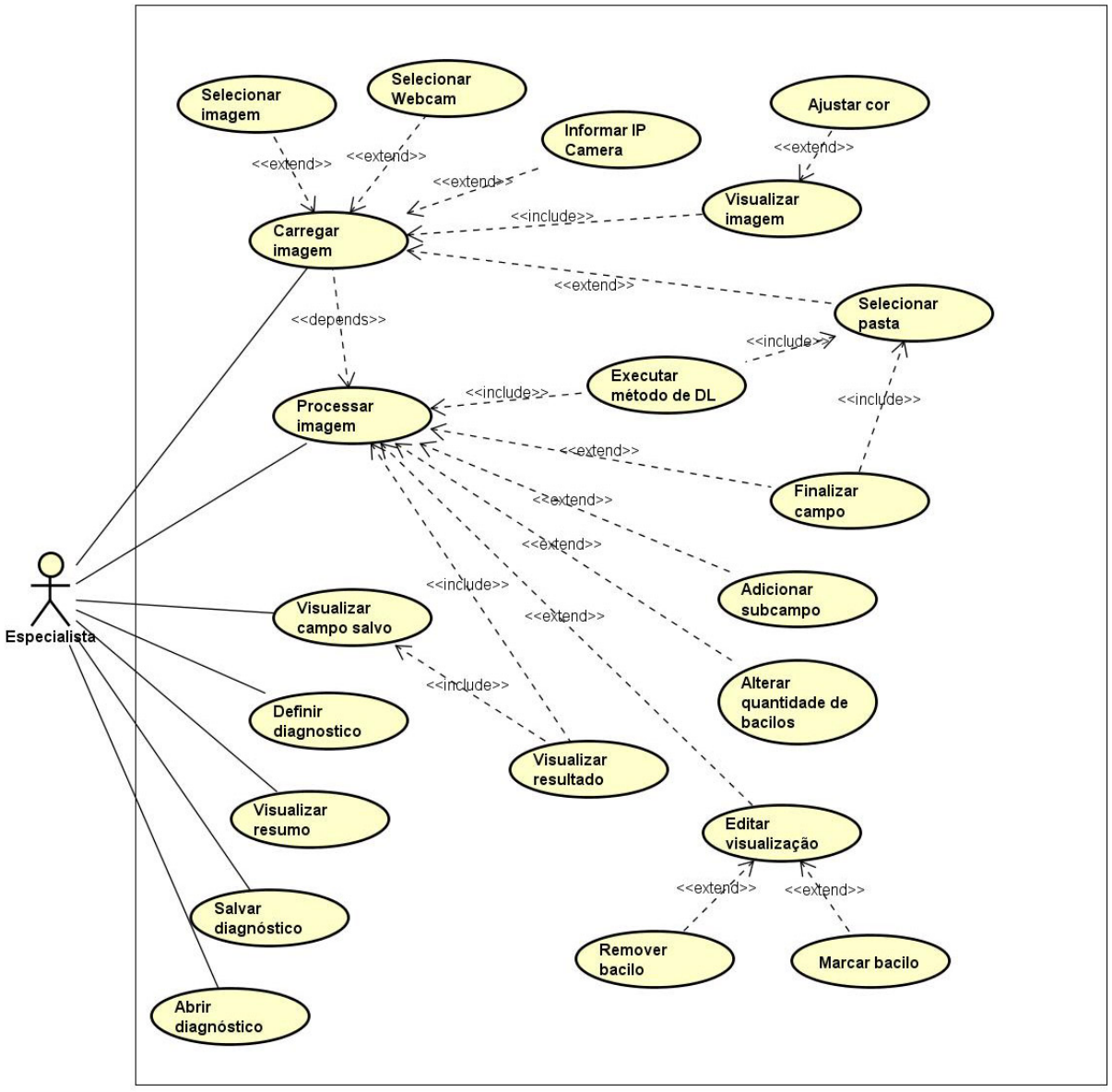


Figura 46 – Diagrama de caso de uso com as funcionalidades do protótipo construído.

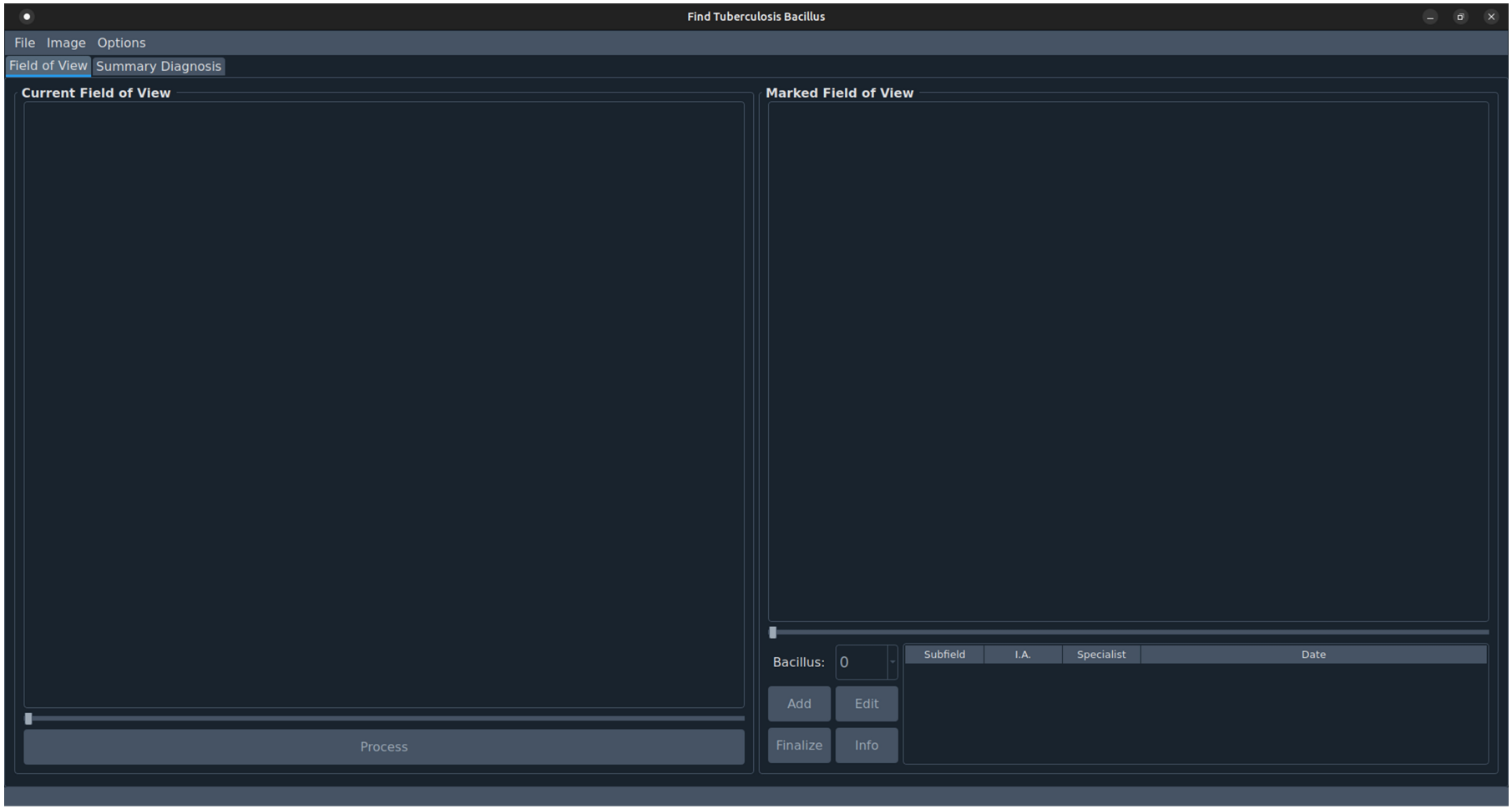


Figura 47 – Protótipo do sistema CADx desenvolvido: Tela Inicial.

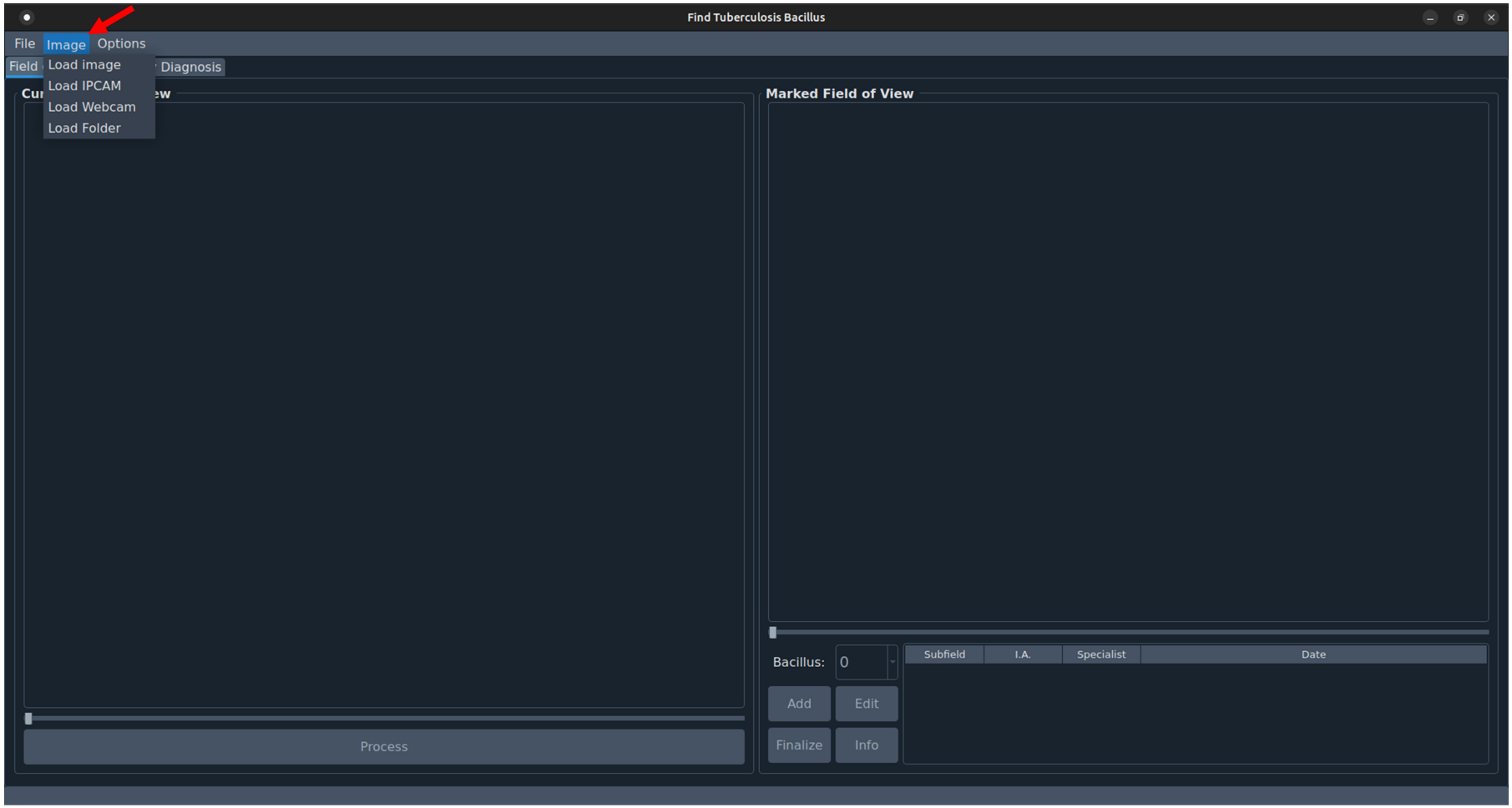


Figura 48 – Protótipo do sistema CADx desenvolvido: Menu de acesso às funcionalidades de carregar uma imagem ou uma fonte de imagem em tempo real.

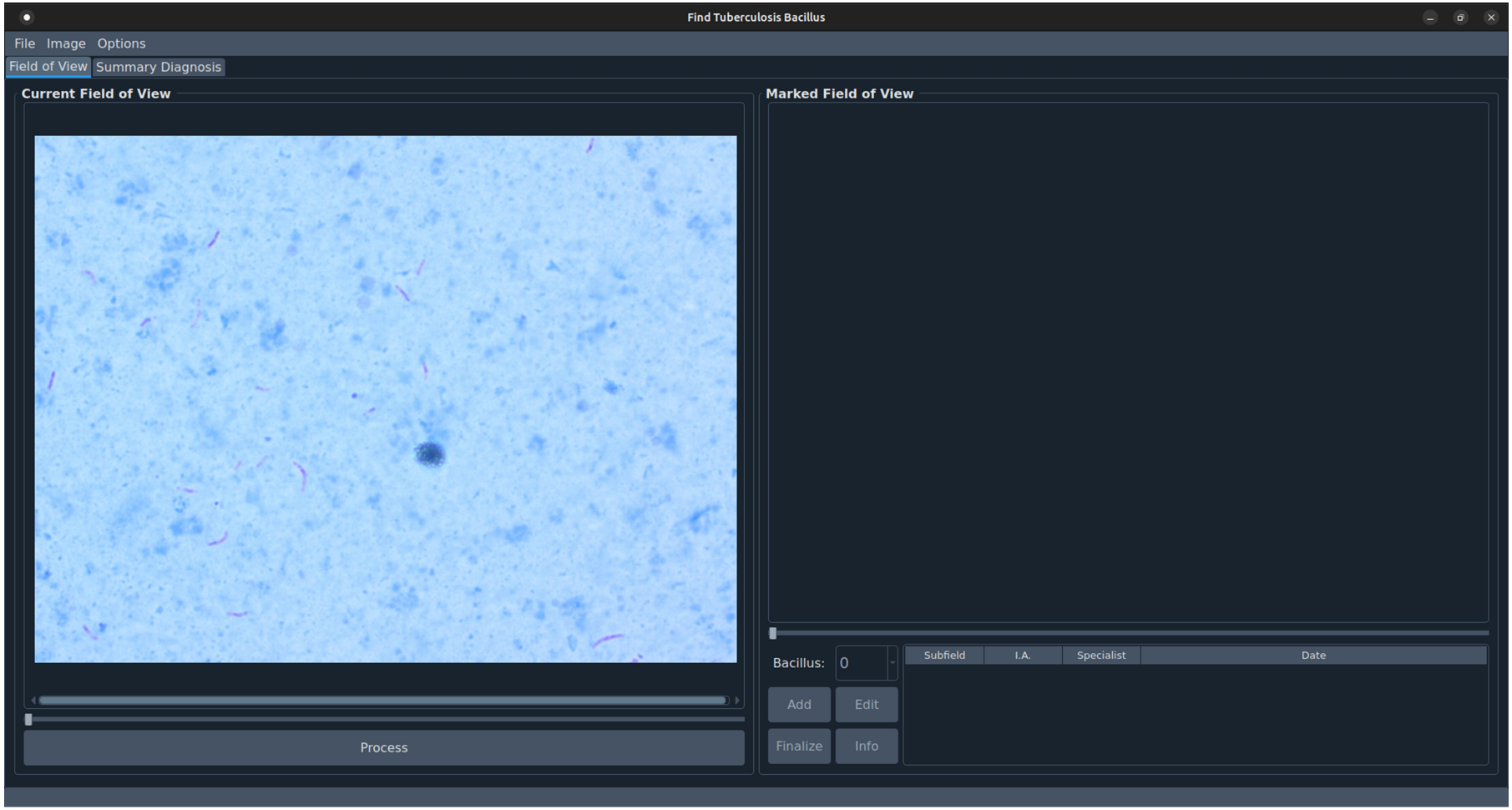


Figura 49 – Protótipo do sistema CADx desenvolvido: Visualização da imagem obtida em um determinada fonte.

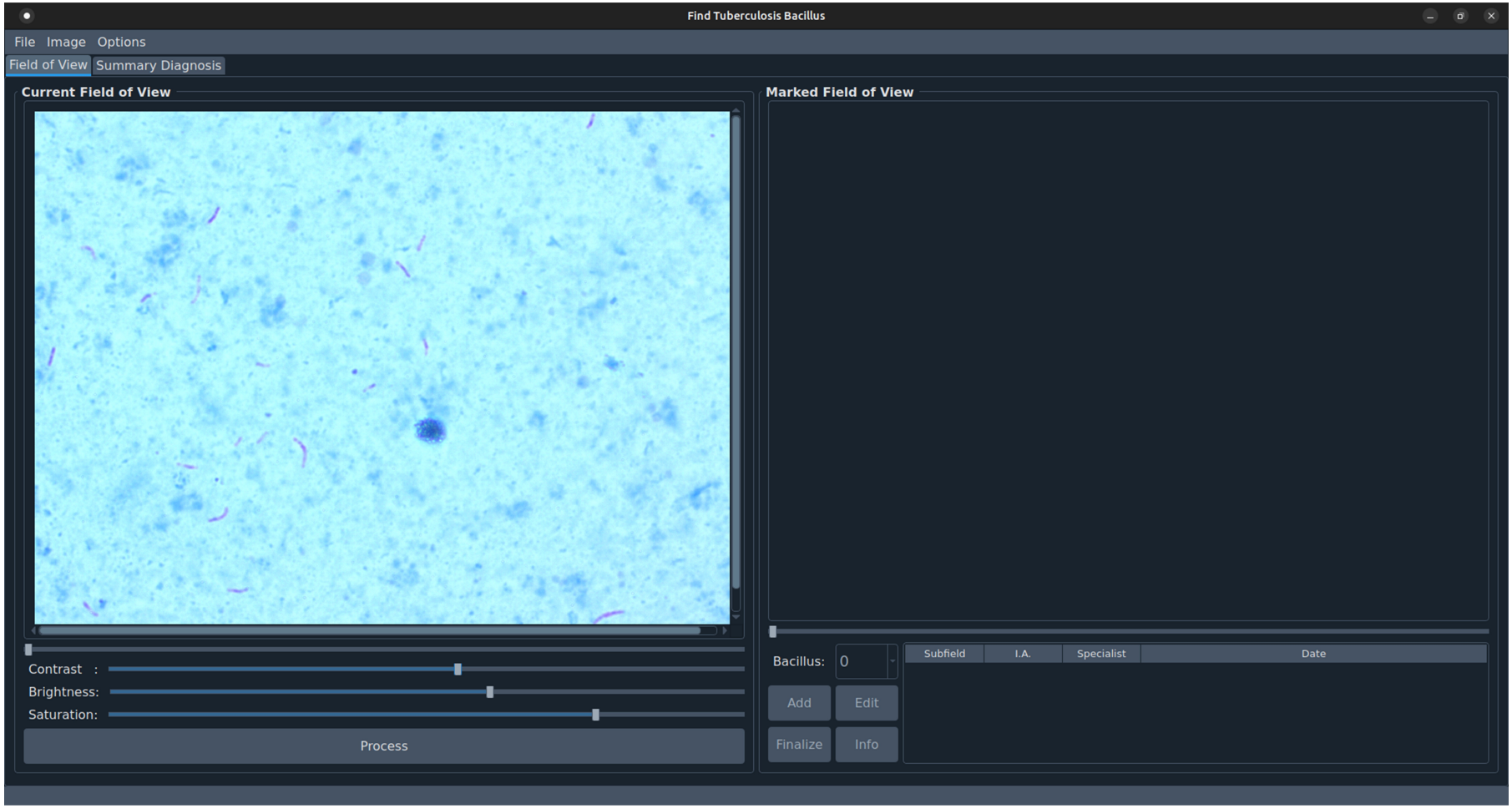


Figura 50 – Protótipo do sistema CADx desenvolvido: Opções de ajuste de cores da funcionalidade de visualização da imagem.

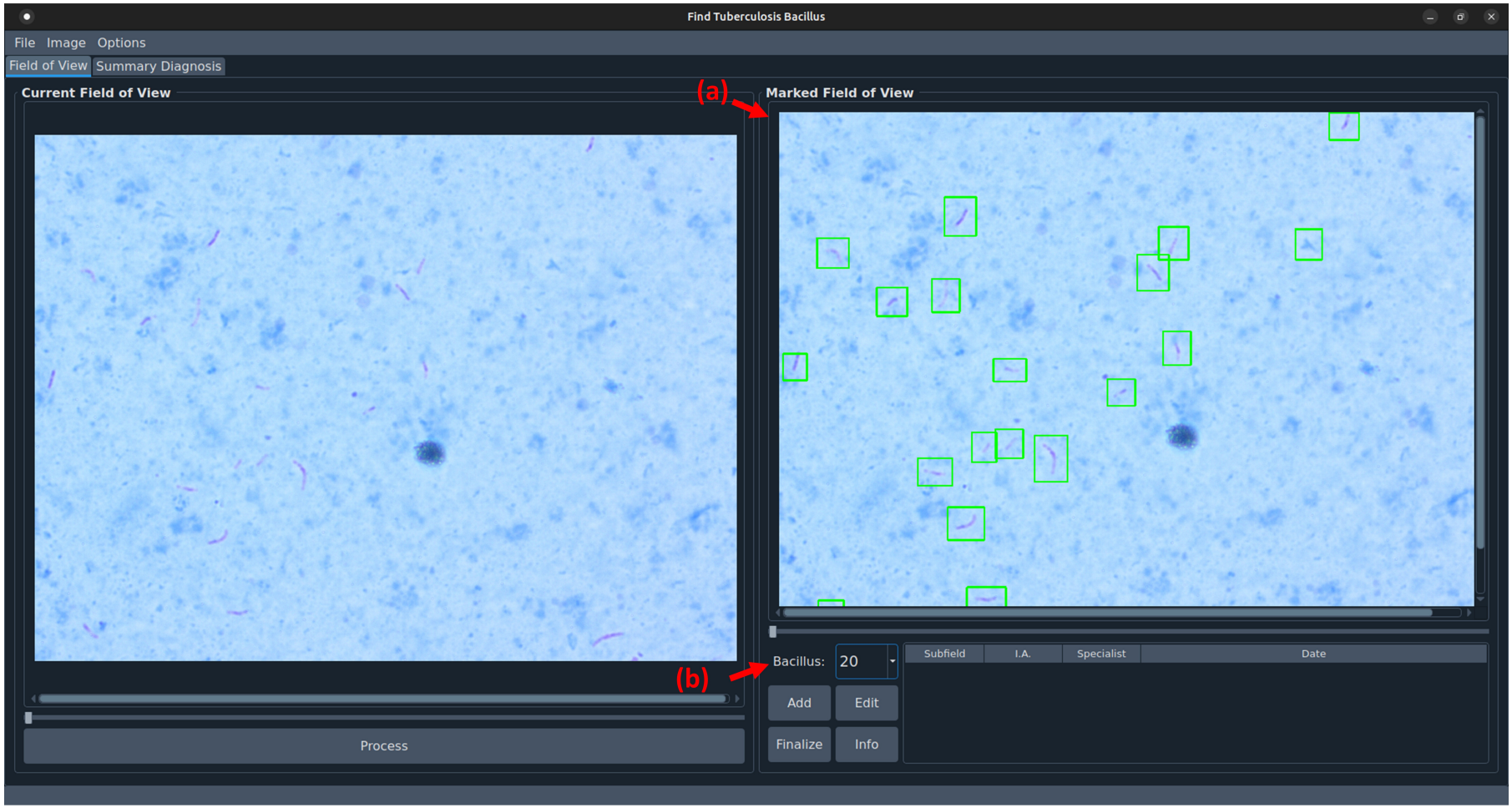


Figura 51 – Protótipo do sistema CADx desenvolvido: Visualização da imagem após análise do método de DL, (a) realçando os bacilos da TB identificados com um box verde, e (b) permitindo que o especialista altere a quantidade de bacilos contados.

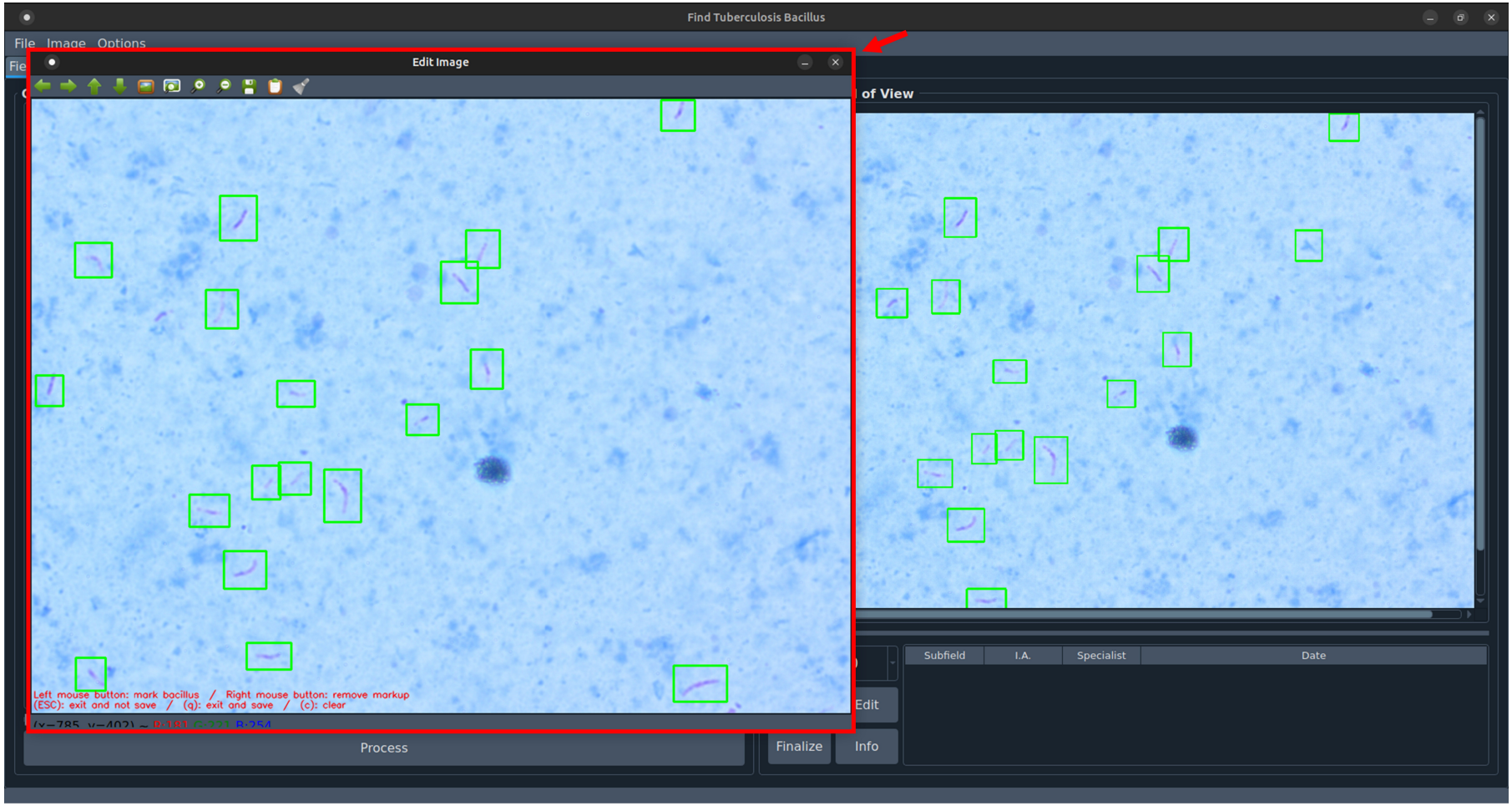


Figura 52 – Protótipo do sistema CADx desenvolvido: Interface de edição das marcações realizadas na imagem.

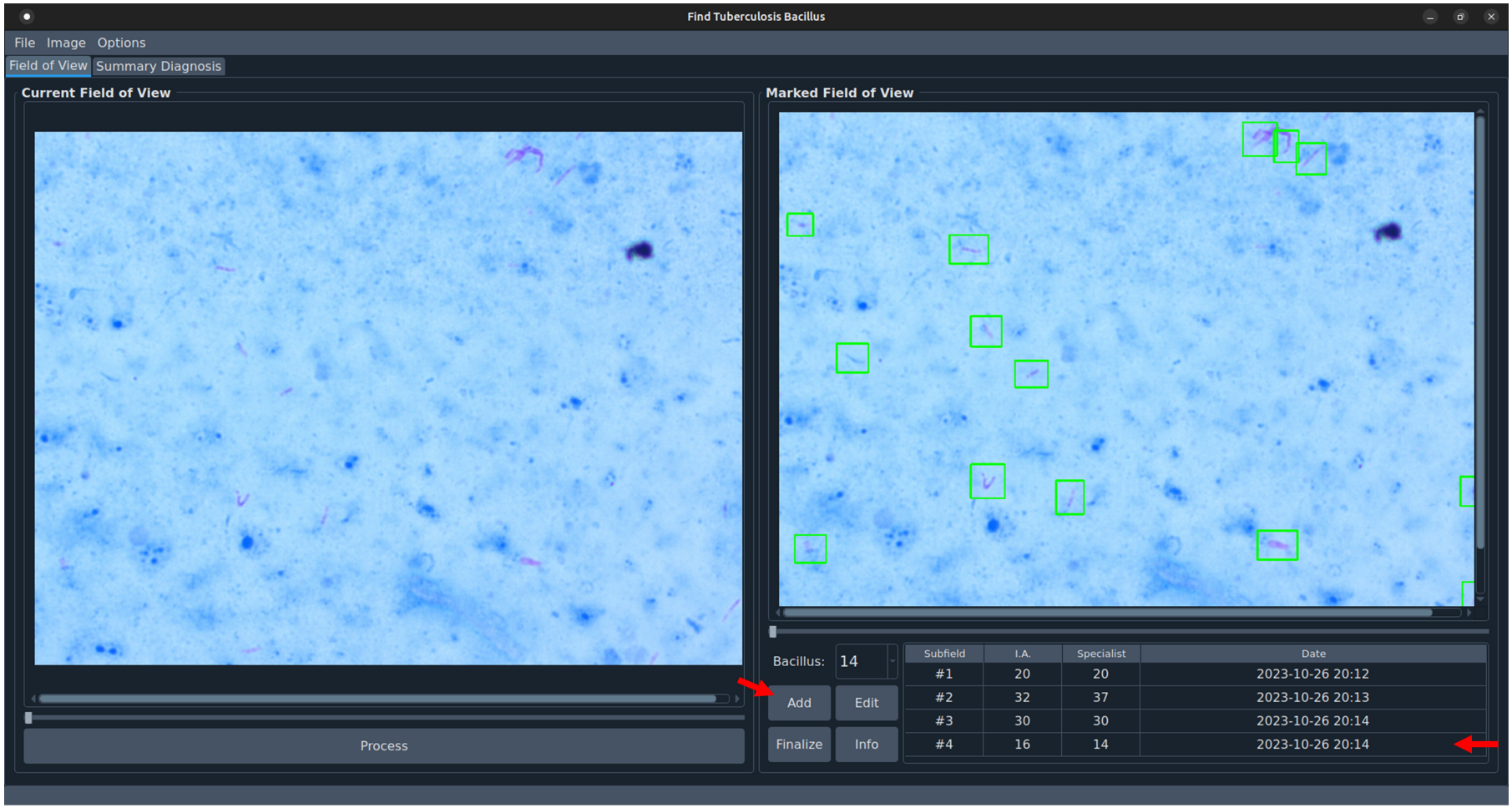
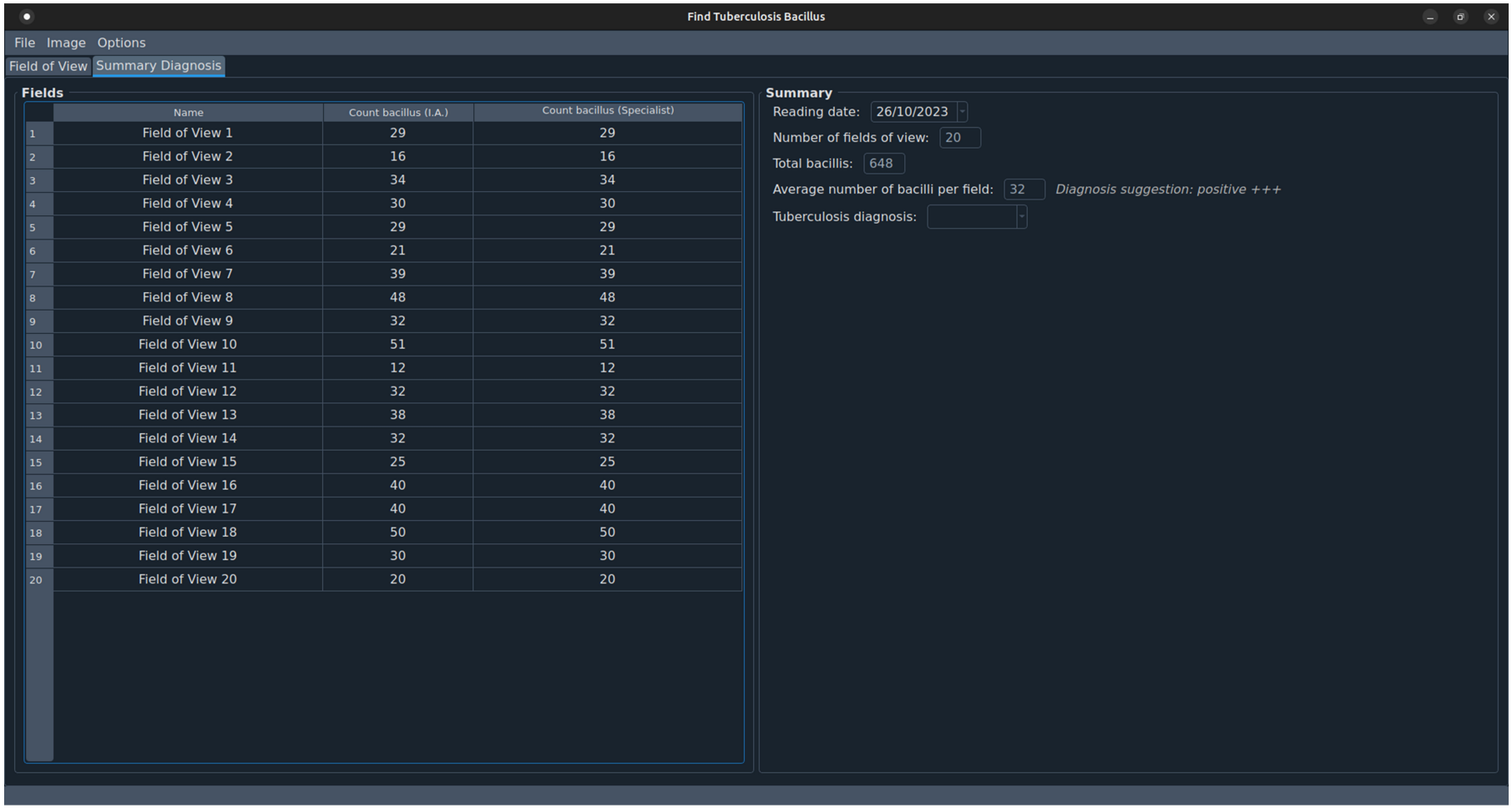


Figura 53 – Protótipo do sistema CADx desenvolvido: Funcionalidade de adicionar imagens que representam um mesmo campo de visualização (subcampos).



	Name	Count bacillus (I.A.)	Count bacillus (Specialist)
1	Field of View 1	29	29
2	Field of View 2	16	16
3	Field of View 3	34	34
4	Field of View 4	30	30
5	Field of View 5	29	29
6	Field of View 6	21	21
7	Field of View 7	39	39
8	Field of View 8	48	48
9	Field of View 9	32	32
10	Field of View 10	51	51
11	Field of View 11	12	12
12	Field of View 12	32	32
13	Field of View 13	38	38
14	Field of View 14	32	32
15	Field of View 15	25	25
16	Field of View 16	40	40
17	Field of View 17	40	40
18	Field of View 18	50	50
19	Field of View 19	30	30
20	Field of View 20	20	20

Figura 54 – Protótipo do sistema CADx desenvolvido: Interface que permite acessar informações que são geradas durante análise das imagens.

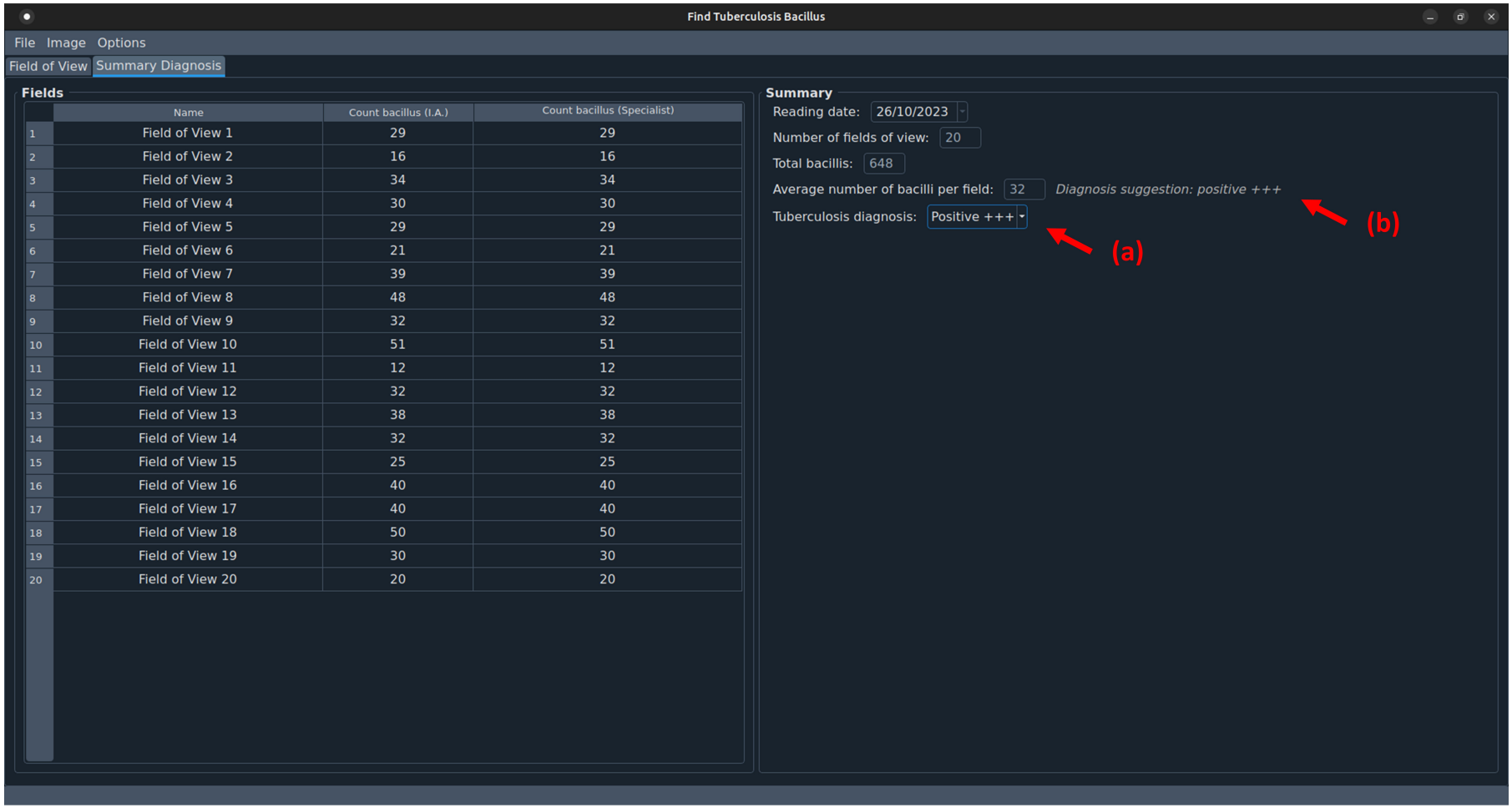


Figura 55 – Protótipo do sistema CADx desenvolvido: Funcionalidade para o especialista definir o diagnóstico da lâmina que está sendo analisada.

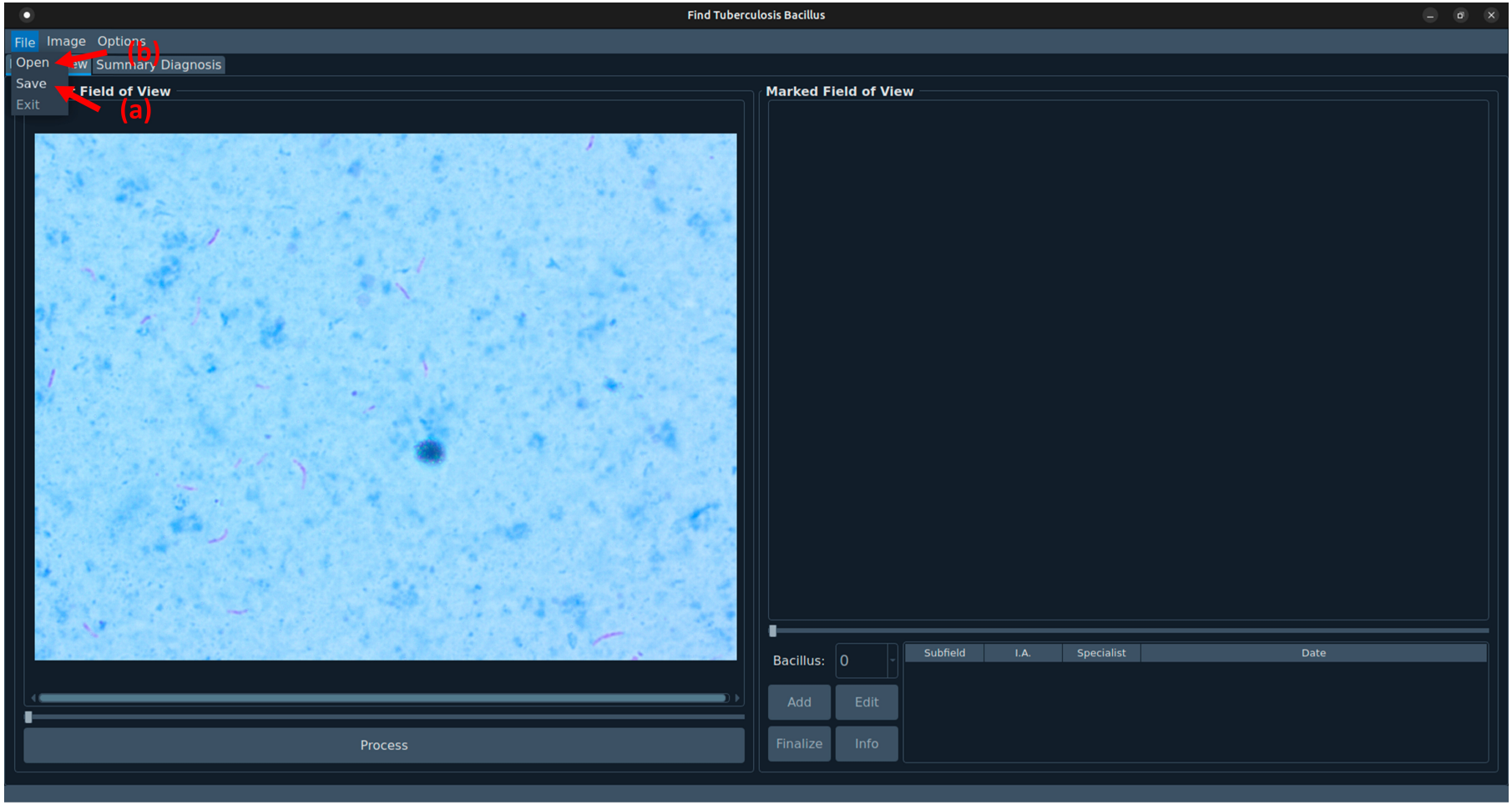


Figura 56 – Protótipo do sistema CADx desenvolvido: Menu com opções de (a) salvar e (b) abrir um diagnóstico realizado.

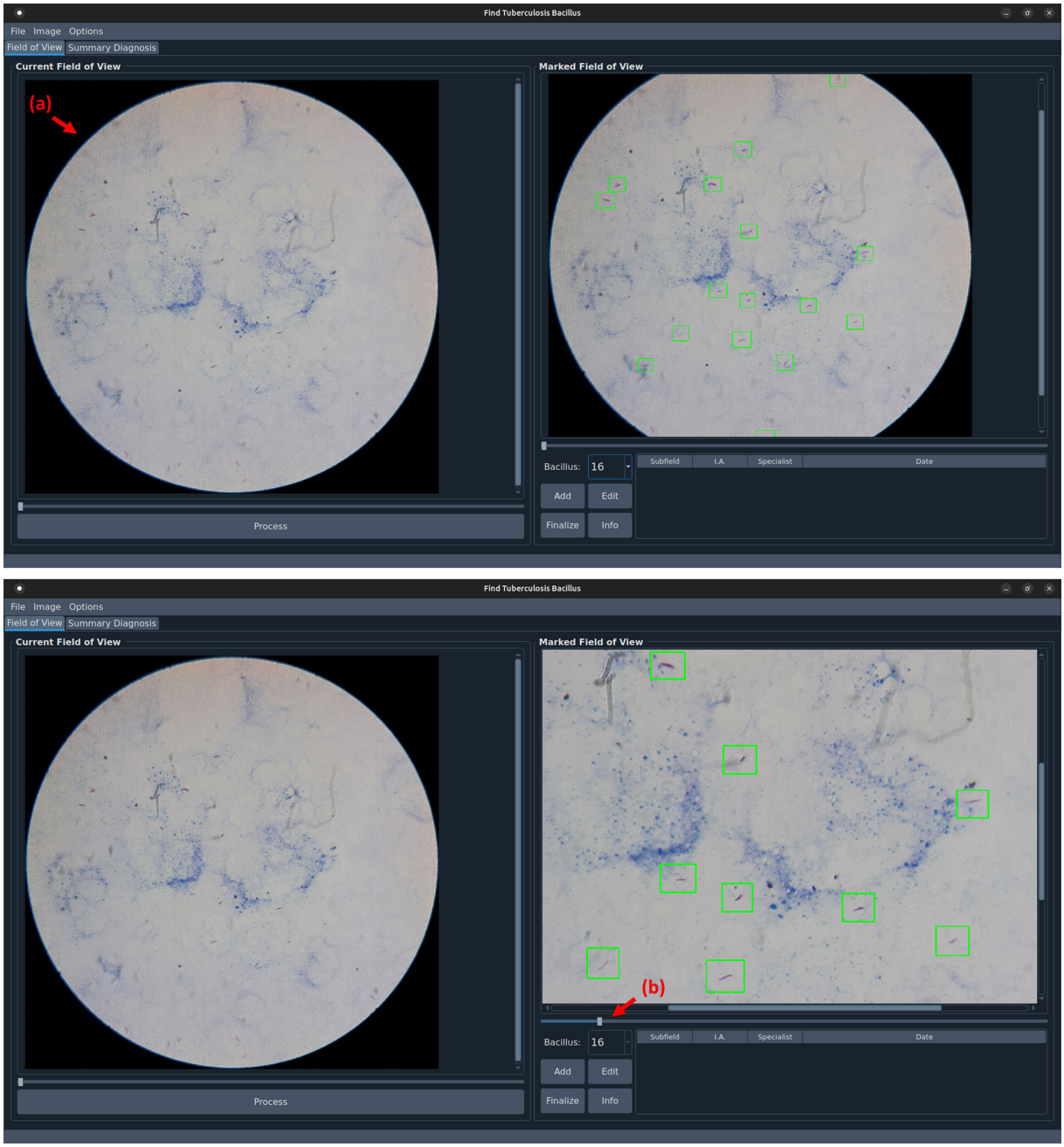


Figura 57 – Protótipo do sistema CADx desenvolvido: (a) simulação da visualização de um campo de visão obtido pelo smartphone e enviado para o sistema, e (b) opção de zoom sento utilizado na visualização da imagem com os bacilos da TB detectados. Imagem obtida do conjunto de dados do Capítulo 4, sem recorte.

7 Conclusões e Trabalhos Futuros

7.1 Considerações Finais

A TB, doença causada pelo *Mycobacterium tuberculosis*, é uma das doenças infecciosas mais letais do mundo, matando mais de um milhão de pessoas a cada ano e afetando outros milhões. Uma das principais técnicas de diagnóstico é a baciloscopia pelo método de *Ziehl-Neelsen*. Resumidamente, essa técnica consiste no preparo da amostra de escarro, descoloração e coloração da amostra, e contagem dos bacilos através da observação no microscópio. Dentre alguns fatores que podem comprometer a eficiência da baciloscopia, está a falta de experiência ou cansaço do profissional. Neste cenário, o presente trabalho tem como objetivo explorar o uso de métodos de DL para auxiliar no processo de identificação/contagem do bacilo, e consequentemente, no diagnostico da TB. Além disso, tem-se como objetivo desenvolver um protótipo de software que possa ser usado na etapa de análise realizada no microscópio, oferecendo um apoio automatizado para tornar o diagnóstico mais rápido e preciso.

Ao longo deste trabalho, foi explorado de forma extensa o uso de métodos baseados em DL para detectar o bacilo da TB através de imagens de baciloscopia. A partir da revisão sistemática realizada, foi possível realizar uma compilação abrangente e estruturada da literatura existente, identificando as tendências atuais, as lacunas de pesquisa e as melhores práticas na área. Com isso, foi possível obter uma visão clara do estado da arte, orientar futuras pesquisas de forma mais eficaz, e contribuir para o direcionamento e aprimoramento dos métodos de DL aplicados ao diagnóstico da TB.

Em sequência, o conhecimento adquirido foi utilizado para propor estratégias e métodos com o intuito de avaliar e aprimorar o uso de DL neste contexto. Onde, foi proposto um novo conjunto de dados público de imagens da baciloscopia, foram replicados estudos da literatura, testou-se uma variedade de métodos de DL em distintos conjuntos de dados, desenvolveu-se uma estratégia para otimizar a detecção em diferentes resoluções, e foi proposto um processo de diagnóstico para viabilizar a avaliação destas técnicas em um cenário autônomo de análise de lâminas de baciloscopia. Além disso, foi desenvolvido um protótipo com possível potencial para auxiliar na tarefa de contagem dos bacilos.

Através de uma sequência criteriosa de experimentos exaustivos, foi possível compreender, passo a passo, as vantagens e limitações dos métodos implementados. Cada experimento proporcionou conclusões valiosas, destacando-se:

1. As CNNs referenciadas na literatura de fato apresentam desempenhos impressionantes.

2. Contudo, sua aplicação em cenários de detecção, com o uso do K-*means* para extração de ROIs, revelou uma potencial superestimação de seu desempenho.

3. A fim de testar e minimizar essa superestimação, foi necessário um aumento na quantidade de padrões negativos de forma estratégica, visando uma representatividade mais abrangente.

4. De forma geral, métodos consagrados para detecção de objetos demostraram bons resultados, com destaque para a Faster R-CNN, que superou os demais.

5. A adoção de uma estratégia de particionamento de imagens se mostrou eficaz para lidar com variações de tamanho nas imagens e potencializar o desempenho da Faster R-CNN, mantendo a padronização dos parâmetros.

6. A avaliação da Faster R-CNN usando diferentes conjuntos de dados (laboratórios/países) para treinamento e teste, evidenciou uma boa capacidade de generalização, embora para aplicações práticas, o ideal seria o uso de conjuntos de treinamento personalizados para cada laboratório/equipamento, visando máxima eficiência.

7. A construção destes conjuntos personalizados pode ser agilizada através de um sistema de votação, que integre diferentes modelos treinados com variados conjuntos de dados.

8. Concluiu-se que os métodos de DL têm o potencial de contribuir significativamente para a automação da contagem de bacilos, embora a presença de falsos positivos reforce a necessidade de intervenção humana em casos de incerteza por parte da rede.

Todos os experimentos conduzidos forneceram uma perspectiva detalhada sobre o comportamento dos métodos de DL, tanto pela diversidade de métodos avaliados quanto pela variedade de conjunto de dados utilizados. Este estudo se destaca na literatura, não tendo sido identificado nenhum outro trabalho tão abrangente em sua abordagem. Os experimentos realizados, os métodos sugeridos e os recursos disponibilizados têm o potencial de contribuir substancialmente para futuras pesquisas, direcionadas tanto ao desenvolvimento de novas estratégias para detecção do bacilo da TB quanto à avaliação prática da aplicabilidade desses métodos em laboratórios.

Adicionalmente, este estudo conduziu uma análise sobre as características necessárias e a mecânica operacional necessária para desenvolver um sistema CADx que possa auxiliar no diagnóstico da TB, por meio da contagem de bacilos utilizando técnicas de DL. O protótipo desenvolvido demonstra uma abordagem inovadora para a realização do exame, proporcionando ao especialista o suporte de uma Inteligência Artificial para auxiliar na contagem dos bacilos, bem como um sistema informatizado que facilita o

registro de bacilos por campo visual. Além disso, o sistema propõe uma solução mais ergonômica, sugerindo a conexão ao microscópio por meio da câmera de um smartphone ou câmera acoplada, permitindo, assim, que as análises dos campos visuais sejam realizadas diretamente no computador, ao invés da ocular do microscópio. Mesmo que o protótipo ainda não tenha sido testado em um ambiente clínico real, simulações e testes de prova de conceito realizados indicam um significativo potencial de aplicação e eficácia.

Visando fornecer uma compreensão prática e diretamente aplicável das estratégias e métodos discutidos ao longo desta tese, o Apêndice B é dedicado à apresentação dos códigos-fonte mais relevantes implementados, permitindo a reprodução de boa parte dos experimentos realizados e promovendo a integridade, transparência e confiabilidade dos resultados obtidos. Além disso, códigos-fonte adicionais, o conjunto de dados disponibilizado no Capítulo 4 e outros materiais relevantes são acessíveis através do site http://tmserver.com.br/ufmg/TuberculosisDataSet, constituindo um repositório abrangente de recursos para futuras pesquisas e aplicações práticas.

Por fim, compreendendo o cenário global da TB e a importância crítica de um diagnóstico preciso, este trabalho representa um avanço na integração da DL na baciloscopia, oferecendo uma visão abrangente, estratégias inovadoras e um protótipo promissor. Foi avaliado rigorosamente diversas técnicas de DL, identificando desafios e delineando um caminho para futuras pesquisas e aplicações práticas. O protótipo proposto, ainda em fase de testes conceituais, juntamente com os experimentos realizados, mostram a viabilidade de uma abordagem automatizada, destacando o potencial da tecnologia para transformar o diagnóstico da TB, aumentar a acurácia e reduzir a carga sobre os profissionais da saúde. Este trabalho não só contribui para o corpo de conhecimento existente, mas também estabelece um fundamento sólido para futuras inovações e aplicações práticas na luta contra a TB.

7.2 Trabalhos Futuros

Como trabalhos futuros são propostas as seguintes atividades:

- Avaliar o desempenho do protótipo do sistema CADx em um ambiente clínico real, focando na contagem de bacilos da TB em exames de baciloscopia. Além de identificar e propor melhorias para otimizar o desempenho e a usabilidade do sistema;
- Conduzir uma análise de otimização dos parâmetros dos algoritmos empregados, visando maximizar a eficácia do modelo, garantindo assim resultados mais precisos e confiáveis;
- Adicionar um classe para os cluster ao conjunto de dados e/ou integrar diferentes conjuntos de dados disponíveis em um banco de dados unificado, com o intuito de

explorar a possibilidade de alcançar uma generalização mais abrangente e robusta dos métodos de DL;

- Desenvolver estratégias para adaptar e padronizar os conjuntos de dados, de modo que seja viável a aplicação e avaliação de outros métodos de DL, tais como Mask R-CNN, U-Net, DeepLab, entre outros, proporcionando assim um escopo mais amplo de análise.

- Aperfeiçoar o processo de aquisição das imagens das lâminas de baciloscopia, seja através de sistemas automáticos de digitalização de lâminas, ou aprimoramento do método de captura de imagens utilizando smartphones ou câmeras acopladas. Essa melhoria no processo de aquisição das imagens pode melhorar a qualidade e a confiabilidade dos dados inseridos no sistema, além de agilizar a realização do diagnóstico.

REFERÊNCIAS

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: ⟨https://www.tensorflow.org/⟩.

ACADEMY, D. S. *Deep Learning Book*. 2021. Https://www.deeplearningbook.com.br/o-neuronio-biologico-e-matematico/. Acesso em: 22 Jun. 2021.

ANACONDA. [S.l.]: Anaconda Inc., 2020. ⟨https://docs.anaconda.com/⟩. Accessed: 2022-10-01.

ANGAYARKANNI, S. P. et al. A novel architecture for improving tuberculosis detection from microscopic sputum smear images. In: SPRINGER NATURE. *Third International Conference on Image Processing and Capsule Networks: ICIPCN 2022*. [S.l.], 2022. v. 514, p. 51–62.

ARLOT, S.; CELISSE, A. et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, The author, under a Creative Commons Attribution License, v. 4, p. 40–79, 2010.

AWAD, M.; KHANNA, R. *Efficient learning machines: theories, concepts, and applications for engineers and system designers*. [S.l.]: Springer nature, 2015.

BRADLEY, A. P. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, Elsevier, v. 30, n. 7, p. 1145–1159, 1997.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

BRASIL. Ministério da saúde. Manual nacional de vigilância laboratorial da tuberculose e outras micobactérias. *Ministério da Saúde*, 2008.

BRASIL. Manual de recomendações para o controle da tuberculose no brasil. *Ministério da Saúde. Secretaria de Vigilância em Saúde. Departamento de Vigilância das Doenças Transmissíveis, Brasilia, DF, Brasil.*, 2019.

BRASIL. Manual de recomendações para o controle da tuberculose no brasil. *Ministério da Saúde. Secretaria de Vigilância em Saúde. Departamento de Vigilância das Doenças Transmissíveis, Brasilia, DF, Brasil.*, 2022.

BRASIL. Boletim epidemiológico tuberculose 2023. *Ministério da Saúde. Secretaria de Vigilância em Saúde. Departamento de Doenças de Condições Crônicas e Infecções Sexualmente Transmissíveis, Brasilia, DF, Brasil.*, Mar 2023. ISSN 9352–7864.

CAMPELO, C. et al. Tuberculose–diagnóstico laboratorial–baciloscopia. *Série Telelab. Brasília: Ministério da Saúde*, 2001.

CARVALHO, T. F. M. et al. A systematic review and repeatability study on the use of deep learning for classifying and detecting tuberculosis bacilli in microscopic images. *Progress in Biophysics and Molecular Biology*, Elsevier, 2023.

CHEN, Q. et al. You only look one-level feature. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. [S.l.: s.n.], 2021. p. 13039–13048.

CHITHRA, R.; JAGATHEESWARI, P. Severity detection and infection level identification of tuberculosis using deep learning. *International Journal of Imaging Systems and Technology*, Wiley Online Library, v. 30, n. 4, p. 994–1011, 2020.

CHOLLET, F. et al. *Keras*. 2015. ⟨https://keras.io⟩.

CLARK, A. *Pillow (PIL Fork) Documentation*. 2022. Https://pillow.readthedocs.io/en/stable/releasenotes/9.0.1.html. Accessed: 2023-10-10.

COSTA, M. G. et al. A sputum smear microscopy image database for automatic bacilli detection in conventional microscopy. In: IEEE. *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. [S.l.], 2014. p. 2841–2844.

DANIEL, T. M.; BATES, J. H.; DOWNES, K. A. History of tuberculosis. *Tuberculosis: pathogenesis, protection, and control*, Wiley Online Library, p. 13–24, 1994.

DENG, L.; YU, D. Deep learning: methods and applications. *Foundations and trends in signal processing*, Now Publishers Inc. Hanover, MA, USA, v. 7, n. 3–4, p. 197–387, 2014.

EL-MELEGY, M.; MOHAMED, D.; ELMELEGY, T. Automatic detection of tuberculosis bacilli from microscopic sputum smear images using faster r-cnn, transfer learning and augmentation. In: SPRINGER. *Iberian Conference on Pattern Recognition and Image Analysis*. [S.l.], 2019. p. 270–278.

EL-MELEGY, M. et al. Identification of tuberculosis bacilli in zn-stained sputum smear images: A deep learning approach. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. [S.l.: s.n.], 2019. p. 1131–1137.

EVERINGHAM, M. et al. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, Springer, v. 111, n. 1, p. 98–136, 2015.

EZEMBRO, E.; AZAM, K.; CADIR, N. *PROGRAMA NACIONAL DE CONTROLO DA TUBERCULOSE: Manual de Baciloscopia da Tuberculose*. [S.l.]: Maputo, 2012.

FELIX, J. H. da S. et al. Avaliação computacional de enfisema pulmonar em tc: comparação entre um sistema desenvolvido localmente e um sistema de uso livre. *J Bras Pneumol*, SciELO Brasil, v. 35, n. 9, p. 868–876, 2009.

FERREIRA, A. A. d. A. et al. Os fatores associados à tuberculose pulmonar e a baciloscopia: uma contribuição ao diagnóstico nos serviços de saúde pública. *Revista Brasileira de Epidemiologia*, SciELO Public Health, v. 8, p. 142–149, 2005.

FILHO, A. C. et al. Ii consenso brasileiro de tuberculose: diretrizes brasileiras para tuberculose 2004. *Jornal Brasileiro de Pneumologia*, SciELO Brasil, v. 30, p. S57–S86, 2004.

FILHO, C. B. *História social da tuberculose e do tuberculoso: 1900-1950*. [S.l.]: Editora Fiocruz, 2001.

FORRELLAD, M. A. et al. Virulence factors of the mycobacterium tuberculosis complex. *Virulence*, Taylor & Francis, v. 4, n. 1, p. 3–66, 2013.

FORSYTH, D. A.; PONCE, J. *Computer vision: a modern approach*. [S.l.]: prentice hall professional technical reference, 2002.

FRäNTI, P.; SIERANOJA, S. How much can k-means be improved by using better initialization and repeats? *Pattern Recognition*, v. 93, p. 95–112, 2019. ISSN 0031-3203. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0031320319301608⟩.

FUKUSHIMA, K. Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, v. 36, n. 4, p. 193–202, 1980.

GIRSHICK, R. Fast r-cnn. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2015. p. 1440–1448.

GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2014. p. 580–587.

GONÇALVES, C. B. et al. Detecção de câncer de mama utilizando imagens termográficas. Universidade Federal de Uberlândia, 2017.

HALEVY, A.; NORVIG, P.; PEREIRA, F. The unreasonable effectiveness of data. *IEEE intelligent systems*, IEEE, v. 24, n. 2, p. 8–12, 2009.

HARTIGAN, J. *Clustering Algorithms*. Wiley, 1975. (Out-of-print Books on demand). ISBN 9780471356455. Disponível em: ⟨https://books.google.com.br/books?id=cDnvAAAAMAAJ⟩.

HARTIGAN, J. A.; WONG, M. A. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, JSTOR, v. 28, n. 1, p. 100–108, 1979.

HE, K. et al. Mask r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2017. p. 2961–2969.

HU, M. et al. Automatic detection of tuberculosis bacilli in sputum smear scans based on subgraph classification. In: IEEE. *2019 International Conference on Medical Imaging Physics and Engineering (ICMIPE)*. [S.l.], 2019. p. 1–7.

HUANG, J.; LING, C. X. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, IEEE, v. 17, n. 3, p. 299–310, 2005.

IBRAHIM, A. U. et al. Automated detection of mycobacterium tuberculosis using transfer learning. *The Journal of Infection in Developing Countries*, v. 15, n. 05, p. 678–686, 2021.

JOCHER, G. *Ultralytics YOLOv5*. 2020. Disponível em: ⟨https://github.com/ultralytics/yolov5⟩.

KANT, S.; SRIVASTAVA, M. M. Towards automated tuberculosis detection using deep learning. In: IEEE. *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. [S.l.], 2018. p. 1250–1253.

KHUTLANG, R. et al. Classification of mycobacterium tuberculosis in images of zn-stained sputum smears. *IEEE transactions on information technology in biomedicine*, IEEE, v. 14, n. 4, p. 949–957, 2009.

KOCH, R. et al. The aetiology of tuberculosis. *American Review of Tuberculosis*, National tuberculosis association, v. 25, n. 3, p. 285–323, 1932.

KONEMAN, E. et al. Diagnóstico microbiológico: texto e atlas colorido. In: *Diagnóstico microbiológico: texto e atlas colorido*. [S.l.: s.n.], 2012.

KOTEI, E.; THIRUNAVUKARASU, R. Computational techniques for the automated detection of mycobacterium tuberculosis from digitized sputum smear microscopic images: A systematic review. *Progress in Biophysics and Molecular Biology*, Elsevier, 2022.

KUOK, C.-P. et al. An effective and accurate identification system of mycobacterium tuberculosis using convolution neural networks. *Microscopy research and technique*, Wiley Online Library, v. 82, n. 6, p. 709–719, 2019.

LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Ieee, v. 86, n. 11, p. 2278–2324, 1998.

LIKAS, A.; VLASSIS, N.; J. Verbeek, J. The global k-means clustering algorithm. *Pattern Recognition*, v. 36, n. 2, p. 451–461, 2003. ISSN 0031-3203. Biometrics. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0031320302000602⟩.

LIMA, A. C. d. M. et al. Aprendizagem profunda aplicada ao diagnóstico do glaucoma. Universidade Federal do Maranhão, 2019.

LIN, T.-Y. et al. Feature pyramid networks for object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 2117–2125.

LIN, T.-Y. et al. Focal loss for dense object detection. In: *Proceedings of the IEEE international conference on computer vision (ICCV)*. [S.l.: s.n.], 2017. p. 2980–2988.

LING, C. X.; HUANG, J.; ZHANG, H. Auc: a better measure than accuracy in comparing learning algorithms. In: SPRINGER. *Conference of the canadian society for computational studies of intelligence*. [S.l.], 2003. p. 329–341.

LITJENS, G. et al. A survey on deep learning in medical image analysis. *Medical image analysis*, Elsevier, v. 42, p. 60–88, 2017.

LIU, W. et al. Ssd: Single shot multibox detector. In: SPRINGER. *European conference on computer vision*. [S.l.], 2016. p. 21–37.

LIU, Y. et al. Privacy-preserving object detection for medical images with faster r-cnn. *IEEE Transactions on Information Forensics and Security*, IEEE, v. 17, p. 69–84, 2019.

LIU, Y. et al. A survey and performance evaluation of deep learning methods for small object detection. *Expert Systems with Applications*, Elsevier, v. 172, p. 114602, 2021.

LOGAN, N. A. *Bacterial systematics*. [S.l.]: John Wiley & Sons, 2009.

LÓPEZ, Y. P. et al. Automatic classification of light field smear microscopy patches using convolutional neural networks for identifying mycobacterium tuberculosis. In: IEEE. *2017 CHILEAN conference on electrical, electronics engineering, information and communication technologies (CHILECON)*. [S.l.], 2017. p. 1–5.

MA, J. et al. Multi-scale retinex with color restoration image enhancement based on gaussian filtering and guided filtering. *International Journal of Modern Physics B*, World Scientific, v. 31, n. 16-19, p. 1744077, 2017.

MACEDO, S. O. d. et al. Desenvolvimento de um sistema de auxílio ao diagnóstico de pneumonia na infância utilizando visão computacional. Universidade Federal de Goiás, 2012.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

MIKOŁAJCZYK, A.; GROCHOWSKI, M. Data augmentation for improving deep learning in image classification problem. In: IEEE. *2018 international interdisciplinary PhD workshop (IIPhDW)*. [S.l.], 2018. p. 117–122.

MITHRA, K.; EMMANUEL, W. S. Automated identification of mycobacterium bacillus from sputum images for tuberculosis diagnosis. *Signal, Image and Video Processing*, Springer, v. 13, n. 8, p. 1585–1592, 2019.

MOHER, D. et al. Preferred reporting items for systematic reviews and meta-analyses: the prisma statement. *Annals of internal medicine*, American College of Physicians, v. 151, n. 4, p. 264–269, 2009.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. *Foundations of machine learning*. [S.l.]: MIT press, 2018.

MOMENY, M. et al. Greedy autoaugment for classification of mycobacterium tuberculosis image via generalized deep cnn using mixed pooling based on minimum square rough entropy. *Computers in Biology and Medicine*, Elsevier, v. 141, p. 105175, 2022. ISSN 0010-4825.

MOSISSA, L. et al. External quality assessment of afb smear microscopy performances and its associated factors in selected private health facilities in addis ababa, ethiopia. *The Pan African medical journal*, African Field Epidemiology Network, v. 24, 2016.

MS. *MANUAL DE RECOMENDAÇÕES PARA O DIAGNÓSTICO LABORATORIAL DE TUBERCULOSE E MICOBACTÉRIAS NÃO TUBERCULOSAS DE INTERESSE EM SAÚDE PÚBLICA NO BRASIL*. [S.l.]: Ministério da Saúde, 2022. ISBN 978-65-5993-146-0.

MUYAMA, L.; NAKATUMBA-NABENDE, J.; MUDALI, D. Automated detection of tuberculosis from sputum smear microscopic images using transfer learning techniques. In: ABRAHAM, A. et al. (Ed.). *Intelligent Systems Design and Applications*. Cham: Springer International Publishing, 2021. p. 59–68. ISBN 978-3-030-49342-4.

NETO, C.; HIDALGA, S. A. de L. Reconhecimento de tumores cerebrais utilizando redes neurais convolucionais. Universidade Federal do Pampa, 2017.

NGUYEN, N.-D. et al. An evaluation of deep learning methods for small object detection. *Journal of electrical and computer engineering*, Hindawi Limited, v. 2020, p. 1–18, 2020.

NTI, I. K.; NYARKO-BOATENG, O.; ANING, J. Performance of machine learning algorithms with different k values in k-fold cross-validation. *Inter. J. Info. Technol. Comp. Sci.*, v. 13, p. 61–71, 2021.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

O'MAHONY, N. et al. Deep learning vs. traditional computer vision. In: SPRINGER. *Science and information conference*. [S.l.], 2019. p. 128–144.

PADILLA, R.; NETTO, S. L.; SILVA, E. A. D. A survey on performance metrics for object-detection algorithms. In: IEEE. *2020 international conference on systems, signals and image processing (IWSSIP)*. [S.l.], 2020. p. 237–242.

PANICKER, R. O. et al. Automatic detection of tuberculosis bacilli from microscopic sputum smear images using deep learning methods. *Biocybernetics and Biomedical Engineering*, Elsevier, v. 38, n. 3, p. 691–699, 2018.

PARK, I.; KIM, S. Performance indicator survey for object detection. In: IEEE. *2020 20th International Conference on Control, Automation and Systems (ICCAS)*. [S.l.], 2020. p. 284–288.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

PINTO, V. J. *Detecção do bacilo da tuberculose através do processamento e análise de imagem microscópica*. Tese (Doutorado) — Mestrado em Sistemas de Informação e Gestão do Conhecimento, 2018.

QUINN, J. A. et al. Deep convolutional neural networks for microscopy-based point of care diagnostics. In: PMLR. *Machine Learning for Healthcare Conference*. [S.l.], 2016. p. 271–281.

RACHMAD, A.; CHAMIDAH, N.; RULANINGTYAS, R. Mycobacterium tuberculosis images classification based on combining of convolutional neural network and support vector machine. *Commun. Math. Biol. Neurosci.*, v. 2020, p. Article–ID, 2020.

RAHMAT, T.; ISMAIL, A.; ALIMAN, S. Chest x-ray image classification using faster r-cnn. *Malaysian Journal of Computing (MJoC)*, Universiti Teknologi MARA Press (Penerbit UiTM), v. 4, n. 1, p. 225–236, 2019.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 779–788.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. p. 779–788.

REFAEILZADEH, P.; TANG, L.; LIU, H. Cross-validation. *Encyclopedia of database systems*, Springer, v. 5, p. 532–538, 2009.

REIST, F. J. D. S. et al. Bacillusnet: An automated approach using retinanet for segmentation of pulmonary tuberculosis bacillus. In: IEEE. *2021 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.], 2021. p. 1–4.

REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. In: CORTES, C. et al. (Ed.). *Advances in Neural Information Processing Systems*. [S.l.]: Curran Associates, Inc., 2015. v. 28, p. 91—99.

SACHAN, A. *Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD*. 2017. Disponível em: ⟨https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/⟩.

SAMUEL, R. D. J.; KANNA, B. R. Tuberculosis (tb) detection system using deep neural networks. *Neural Computing and Applications*, Springer, v. 31, n. 5, p. 1533–1545, 2019.

SELVAKUMAR, N. et al. 843. selvakumar n, et al. proficiency to read sputum afb smears by senior tuberculosis laboratory supervisors under training at reference laboratory in india. *Indian Journal of Tuberculosis*, v. 52, n. 1, p. 11–14, 2005.

SERRÃO, M. et al. Automatic bacillus detection in light field microscopy images using convolutional neural networks and mosaic imaging approach. In: IEEE. *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. [S.l.], 2020. p. 1903–1906.

SERRÃO, M. K. M. et al. Detecção automática de bacilos em baciloscopia de campo claro usando aprendizado profundo e técnica de imagem mosaico. Universidade Federal do Amazonas, 2020.

SETHI, K.; PARMAR, V.; SURI, M. Low-power hardware-based deep-learning diagnostics support case study. In: IEEE. *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. [S.l.], 2018. p. 1–4.

SHAH, M. I. et al. Ziehl–neelsen sputum smear microscopy image database: a resource to facilitate automated bacilli detection for tuberculosis diagnosis. *Journal of Medical Imaging*, International Society for Optics and Photonics, v. 4, n. 2, p. 027503, 2017.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. (Understanding Machine Learning: From Theory to Algorithms). ISBN 9781107057135. Disponível em: ⟨https://books.google.com.br/books?id=ttJkAwAAQBAJ⟩.

SHORTEN, C.; KHOSHGOFTAAR, T. M. A survey on image data augmentation for deep learning. *Journal of Big Data*, Springer, v. 6, n. 1, p. 1–48, 2019.

SILVA, J. M. S. C. F. et al. *Deteção de convulsões epiléticas em eletroencefalogramas usando deep learning*. Tese (Doutorado), 2017.

SIMON, A. et al. Shallow cnn with lstm layer for tuberculosis detection in microscopic images. *Machine learning for Biomedical Applications*, 2019.

SOKOLOVA, M.; JAPKOWICZ, N.; SZPAKOWICZ, S. Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In: SPRINGER. *Australasian joint conference on artificial intelligence*. [S.l.], 2006. p. 1015–1021.

SOMMERVILLE, I. Software engineering 9th edition. *ISBN-10*, v. 137035152, p. 18, 2011.

STEELE, J. H.; RANNEY, A. F. Animal tuberculosis. *American Review of Tuberculosis and Pulmonary Diseases*, National Tuberculosis Association, v. 77, n. 6, p. 908–922, 1958.

SUN, C. et al. Revisiting unreasonable effectiveness of data in deep learning era. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2017. p. 843–852.

SWETHA, K.; SANKARAGOMATHI, B.; THANGAMALAR, J. B. Convolutional neural network based automated detection of mycobacterium bacillus from sputum images. In: IEEE. *2020 International Conference on Inventive Computation Technologies (ICICT)*. [S.l.], 2020. p. 293–300.

TAN, M.; LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In: PMLR. *International conference on machine learning*. [S.l.], 2019. p. 6105–6114.

TAN, M.; PANG, R.; LE, Q. V. Efficientdet: Scalable and efficient object detection. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. [S.l.: s.n.], 2020. p. 10781–10790.

TIAN, Z. et al. Fcos: Fully convolutional one-stage object detection. In: *Proceedings of the IEEE/CVF international conference on computer vision*. [S.l.: s.n.], 2019. p. 9627–9636.

TRILAKSANA, H. et al. Sputum smear images database: A resource for deep learning study based to detect bacilli for tb diagnose. *AIP Conference Proceedings*, v. 2314, n. 1, p. 040013, 2020.

TRUSOV, A. et al. Comparison of lumin™ led fluorescent attachment, fluorescent microscopy and ziehl-neelsen for afb diagnosis. *The International journal of tuberculosis and lung disease*, International Union Against Tuberculosis and Lung Disease, v. 13, n. 7, p. 836–841, 2009.

UDEGOVA, E.; SHELOMENTSEVA, I.; CHENTSOV, S. Optimizing convolutional neural network architecture for microscopy image recognition for tuberculosis diagnosis. In: SPRINGER. *International Conference on Neuroinformatics*. [S.l.], 2021. p. 204–209.

VALUEVA, M. V. et al. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, Elsevier, v. 177, p. 232–243, 2020.

VANITHA, V.; ANGAYARKANNI, S. P.; SATHISHKUMAR, M. Segmentation of tuberculosis bacteria: Evaluation and comparison. In: IEEE. *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*. [S.l.], 2021. p. 1690–1696.

VÁZQUEZ, L.; HASSAINIA, F. *IceVision: An agnostic object detection framework*. [S.l.]: Github, 2020.

VERGARA, R. F. Detecção de alterações cerebrais anatômicas associadas à esquizofrenia com base em redes convolucionais aplicadas a imagens de ressonância magnética. Universidade de Brasília, 2018.

WAIBEL, A. et al. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, IEEE, v. 37, n. 3, p. 328–339, 1989.

WANG, A. Y.-T. et al. Machine learning for materials scientists: an introductory guide toward best practices. *Chemistry of Materials*, ACS Publications, v. 32, n. 12, p. 4954–4965, 2020.

WHO. *Global tuberculosis report 2020*. Licence: CC BY-NC-SA 3.0 IGO: Geneva: World Health Organization, 2020. ISBN 978-92-4-001313-1.

WHO. *WHO consolidated guidelines on tuberculosis. Module 3: diagnosis-rapid diagnostics for tuberculosis detection*. [S.l.]: World Health Organization, 2020.

WHO. *WHO consolidated guidelines on tuberculosis. Module 2: screening-systematic screening for tuberculosis disease*. [S.l.]: World Health Organization, 2021.

WHO. *Global tuberculosis report 2022*. Licence: CC BY-NC-SA 3.0 IGO: Geneva: World Health Organization, 2022. ISBN 978-92-4-006172-9.

WINARNO; BILHAQ, F. G.; AGOES, A. S. Learning from the scratch for tuberculosis (tb) bacilli detection using dsod. In: *Proceedings of the 2nd International Seminar of Science and Applied Technology (ISSAT 2021)*. [S.l.]: Atlantis Press, 2021. p. 216–219. ISBN 978-94-6239-451-3. ISSN 2352-5401.

XIONG, Y. et al. Automatic detection of mycobacterium tuberculosis using artificial intelligence. *Journal of thoracic disease*, AME Publications, v. 10, n. 3, p. 1936, 2018.

YANG, M. et al. A cnn-based active learning framework to identify mycobacteria in digitized ziehl-neelsen stained human tissues. *Computerized Medical Imaging and Graphics*, Elsevier, v. 84, p. 101752, 2020.

ZHU, C.; HE, Y.; SAVVIDES, M. Feature selective anchor-free module for single-shot object detection. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. [S.l.: s.n.], 2019. p. 840–849.

ZURAC, S. et al. A new artificial intelligence-based method for identifying mycobacterium tuberculosis in ziehl–neelsen stain on tissue. *Diagnostics*, Multidisciplinary Digital Publishing Institute, v. 12, n. 6, p. 1484, 2022.

A Find TB Bacillus, um Ambiente Virtual de Aprendizagem

O uso de Tecnologias da Informação e Comunicação (TICs) na educação tornou-se imprescindível no processo de ensino e aprendizagem para a sociedade contemporânea. Estas ferramentas transformaram as diferentes formas de relacionamentos entre os indivíduos, além de oferecer a opção pedagógica pelo Ambiente Virtual de Aprendizagem (AVA).

Dentre os diversos conteúdos de ensino em cursos da área de saúde, pode-se destacar todo o procedimento de ensino-aprendizagem da baciloscopia, onde percebe-se que a metodologia adotada são geralmente aulas teóricas. E quando há laboratórios que atenda esta finalidade são realizados exercícios práticos, porém é indispensável uma supervisão direta dos alunos neste laboratório. Dessa forma, entende-se que pode existir uma limitação no formato de ensino-aprendizagem da baciloscopia adotado nos cursos da área de saúde e que o uso de um AVA contribuiria no aprendizado do diagnóstico da TB por meio de uma plataforma que permite uma fácil interação entre o professor e o aluno, e vice-versa.

Tendo em vista o impacto da tuberculose em escala mundial e nacional, os procedimentos totalmente manuais de diagnóstico, e a complexidade de realizar aulas práticas de identificação do bacilo da TB, faz-se necessário propor novas abordagens que contribuam de forma significativa para ensinar, capacitar e avaliar os alunos na realização do diagnóstico da TB, tornando-o mais rápido e preciso.

Diante deste cenário e do desenvolvimento da ferramenta colaborativa de marcação de bacilo (Find TB Bacillus) apresentada na Seção 4, percebeu-se a possibilidade de desenvolver um AVA. Dessa forma, o Find TB Bacillus está sendo desenvolvido para: (i) apoiar na construção do conjunto de dados, (ii) auxiliar na capacitação dos alunos e profissionais da área de saúde no diagnóstico da TB e (iii) auxiliar os docentes no processo de ensino-aprendizagem. Destaca-se ainda que o AVA teria como objetivo disponibilizar um ambiente digital que permita realizar o processo de análise das imagens obtidas da baciloscopia fora de um ambiente de laboratório e sem uso do microscópio.

O AVA proposto distingue-se dos demais AVAs conhecidos pelo sua navegação e interação com imagens, ao invés do tradicional enfoque linear das plataformas de ensino tradicionais que se baseiam em perguntas fechadas e abertas no formato de quiz. Assim, este AVA apresenta propriedades mais evoluídas com foco em exercícios de identificação do bacilo, por meio de uma interação dinâmica com imagens.

Portanto, entende-se que o AVA proposto pode viabilizar a adoção de uma orientação pedagógica integrativa e interativa, buscando um modelo mais orgânico e

dinâmico de aprendizagem e atuação. Suas funcionalidades são apresentadas a seguir e ilustram como ele pode contribuir positivamente nesse processo de ensino.

A.1 Funcionalidades do Find TB Bacillus

A Figura 58 mostra as funcionalidades do Find TB Bacillus como ambiente virtual, representadas no diagrama de Casos de Uso da UML (um acrônimo para Unified Modeling Language, em português, Linguagem de Modelagem Unificada). Neste diagrama, são demonstrados os requisitos funcionais do software, além da interação dos atores (usuários do tipo professores e alunos) com os casos de uso (funcionalidades do sistema).

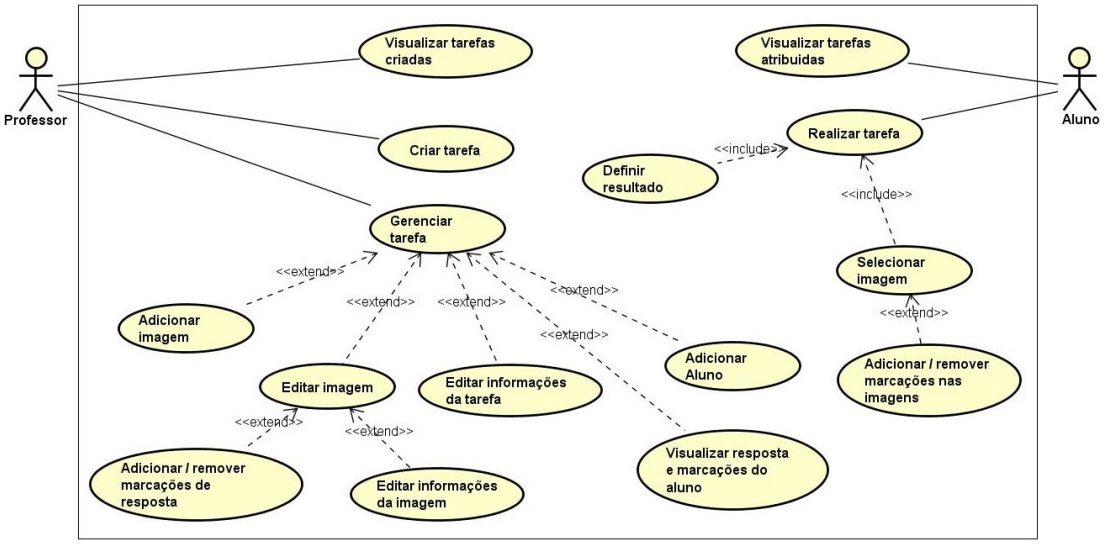


Figura 58 – Diagrama de caso de uso do Find TB Bacillus.

A seguir é realizada a descrição textual de cada caso de uso e seus relacionamentos apresentados na Figura 58. Além disso, são exibidas algumas ilustrações da interface do sistema a partir da prototipação. Vale ressaltar que todas os protótipos de interfaces apresentadas foram criados utilizando o idioma inglês, visando a disponibilização dessa ferramenta em escala mundial.

- **Criar tarefa:** permite o professor criar tarefas para os alunos. O protótipo da interface é apresentado na Figura 59.
- **Visualizar tarefas criadas:** apresenta ao professor a listagem das tarefas criadas por ele. A Figura 60 ilustra como o professor visualizará essas tarefas.
- **Gerenciar tarefa:** o professor usa o sistema para gerenciar (incluir, remover, alterar, consultar) o andamento das tarefas criadas por ele. Neste caso de uso (Figura 58), percebe-se que existem outros casos de uso relacionando com ele. Nessa interface o professor tem acesso a todas as operações de gerência da tarefa, como ilustrado na Figura 61.

- **Adicionar imagem:** o professor pode adicionar uma ou mais imagens em a tarefa. Para realizar essa operação, basta utilizar a opção "Add a new image" apresentada na Figura 61.
- **Editar imagem:** o professor pode realizar a edição da imagem de uma tarefa. Neste caso de uso, há uma extensão para realizar marcações (identificar) o bacilo da TB na imagem, e outra extensão para editar informações da imagem. A Figura 62 ilustra a interface de marcação dos bacilos em uma imagem de microscópio.
- **Editar informações da tarefa:** o professor pode editar informações já cadastradas na tarefa. Para isso, basta acessar a opção "Edit information" na Figura 61, onde será redirecionado para uma interface semelhante a Figura 59.
- **Visualizar resposta e marcações do aluno:** o professor visualiza respostas que recebeu da tarefa de um aluno e compara com a marcação realizada por ele (Figura 63).
- **Adicionar Aluno:** o professor adiciona o aluno para ter acesso a tarefa. O professor pode permitir que um aluno realize uma tarefa informando o e-mail do aluno na opção "Grant access to this task for the user" na Figura 61.
- **Visualizar tarefas atribuídas:** o aluno visualiza as tarefas que foram atribuídas a ele em uma interface semelhante a Figura 60.
- **Realizar tarefa:** o aluno usa o sistema para realizar tarefas, realizando marcações na imagem e definir o resultado. A Figura 64 ilustra a interface que permite o aluno acessar as imagens e definir a resposta da atividade.
- **Definir resultado:** o aluno deve definir o resultado ao realizar a tarefa. Para realizar essa operação, basta selecionar a opção "You sputu, snear microscopy answer" ilustrado na Figura 64. Nessa opção, o aluno define a resposta da tarefa em "negative", "+", "++", "+++" (nomenclatura utilizada na baciloscopia para indicar o resultado do exame) a partir das análises das imagens.
- **Selecionar imagem:** o aluno deve selecionar uma imagem para realizar a tarefa. Neste caso de uso, o aluno pode ainda editar (adicionar ou remover) marcações na imagem. A Figura 65 ilustra essas operações.

Desta forma, apresentou-se as funcionalidades e as principais ilustrações da interface do sistema Find TB Bacillus.

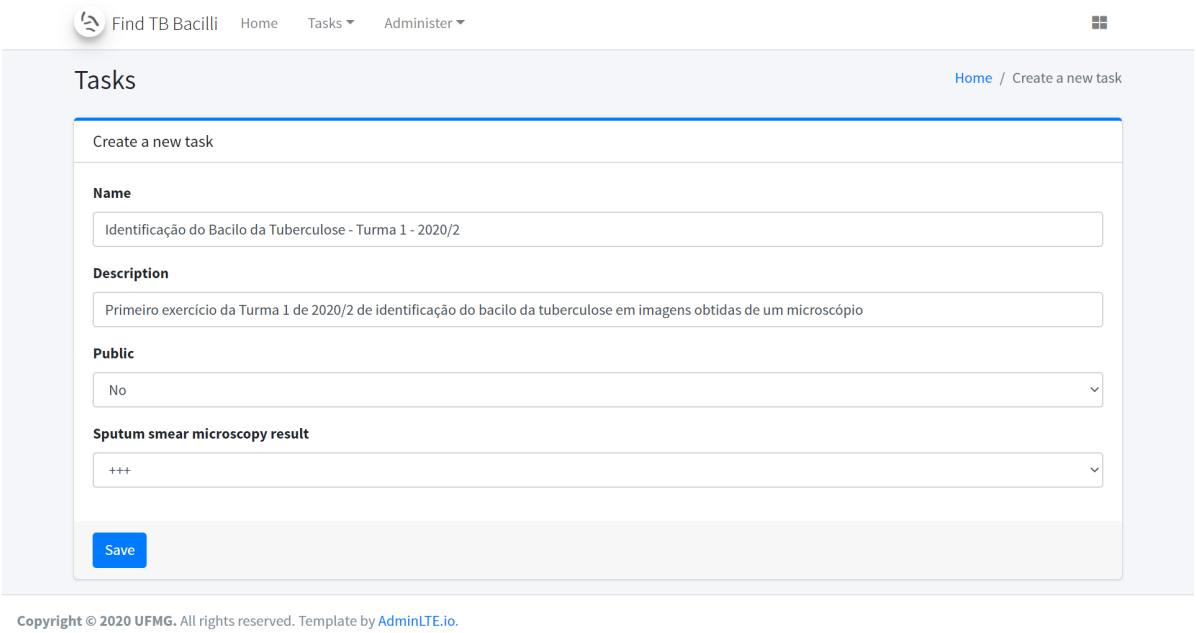


Figura 59 – Protótipo da interface para criar tarefa.

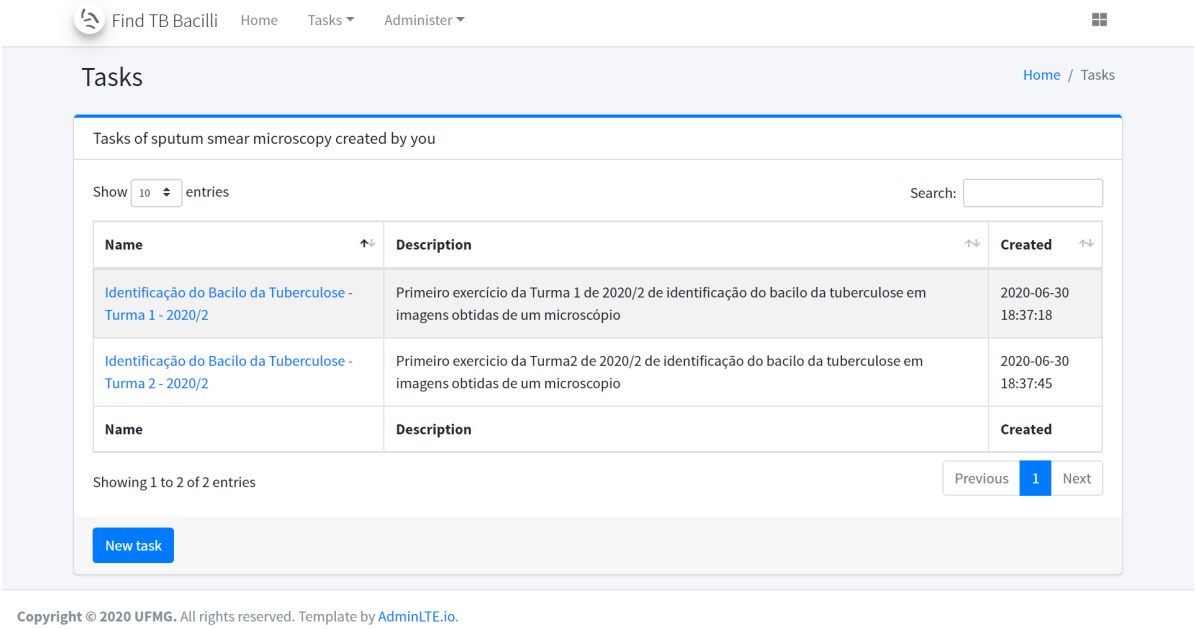


Figura 60 – Protótipo da interface para visualizar as tarefas criadas.

A.2 Desenvolvimento

Para o desenvolvimento do AVA apresentado, está sendo utilizado apenas softwares e linguagens de programação de código aberto, seguindo uma metodologia de desenvolvimento incremental. Para isso, a plataforma está sendo codificada utilizando as linguagens de programação e marcação: PHP, JavaScript, HTML e CSS. Juntamente com o framework CodeIgniter (https://codeigniter.com), o design do template AdminLTE Bootstrap (https://adminlte.io), e o banco de dados MySQL.

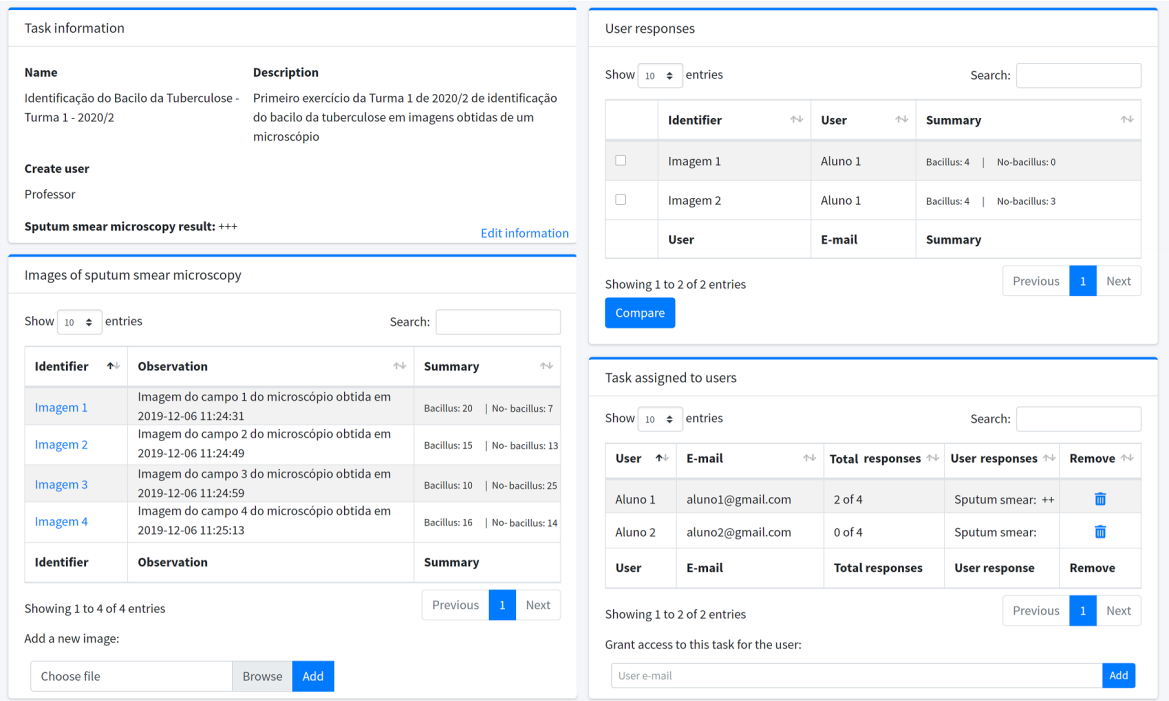


Figura 61 – Protótipo da interface para gerenciar a tarefa.

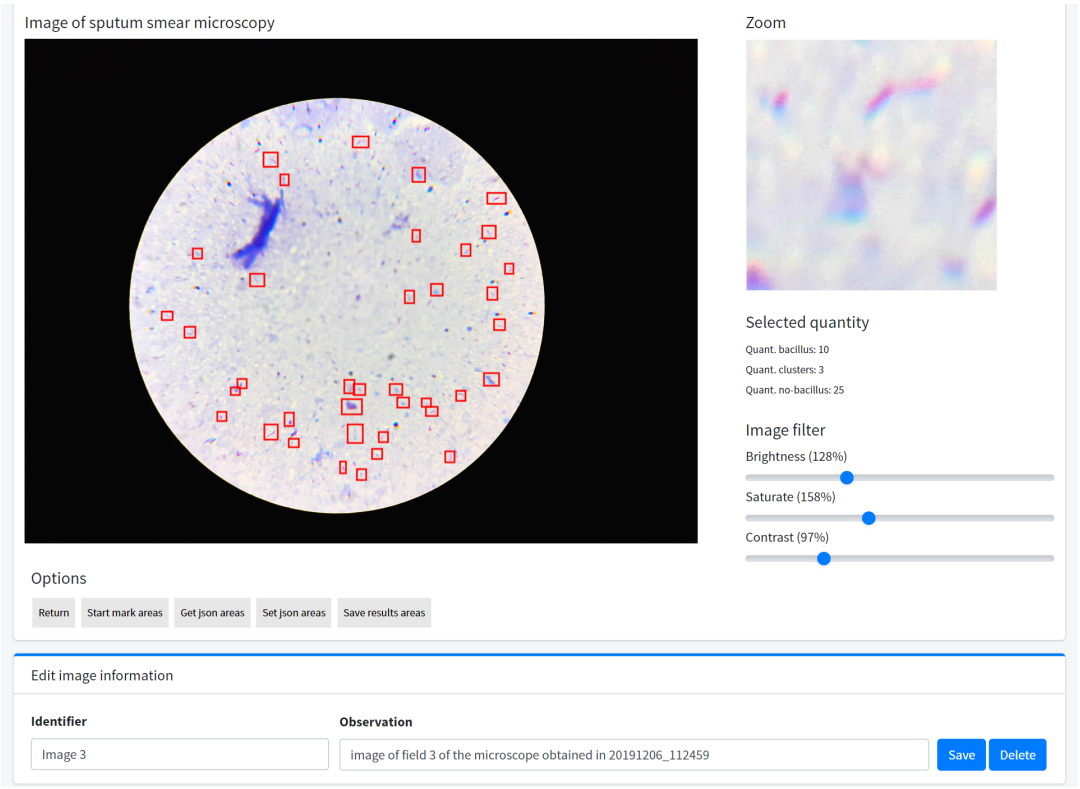


Figura 62 – Protótipo da interface para editar uma imagem e marcar os bacilos.

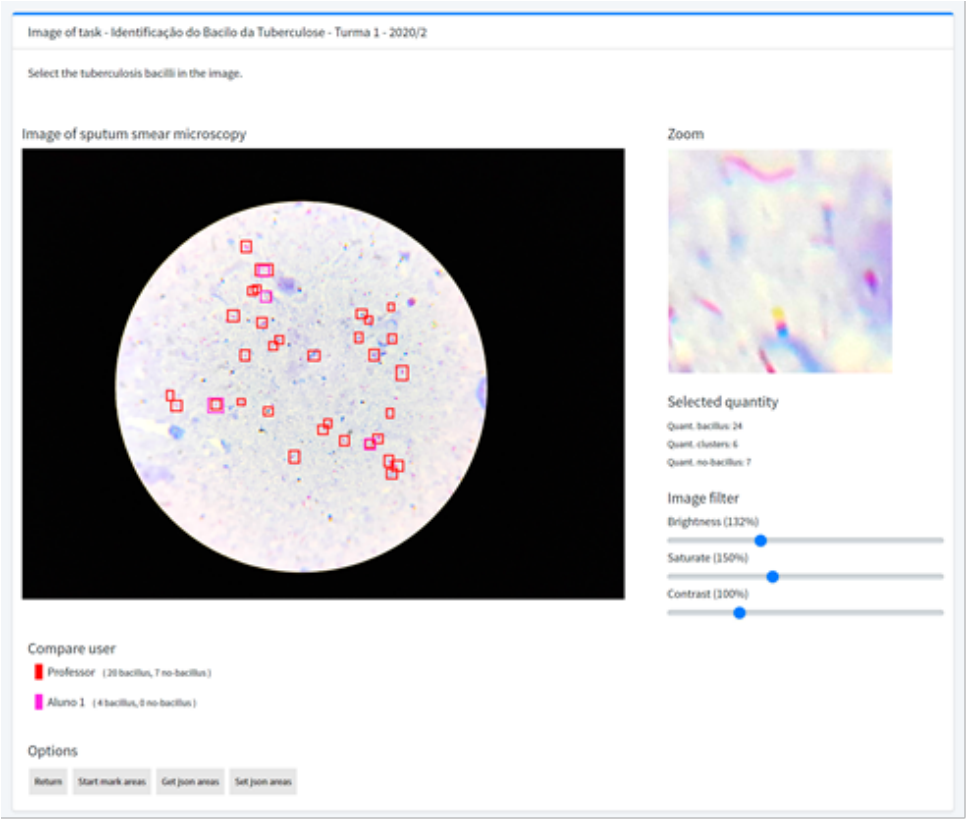


Figura 63 – Protótipo da interface para comparar as marcações dos bacilos.

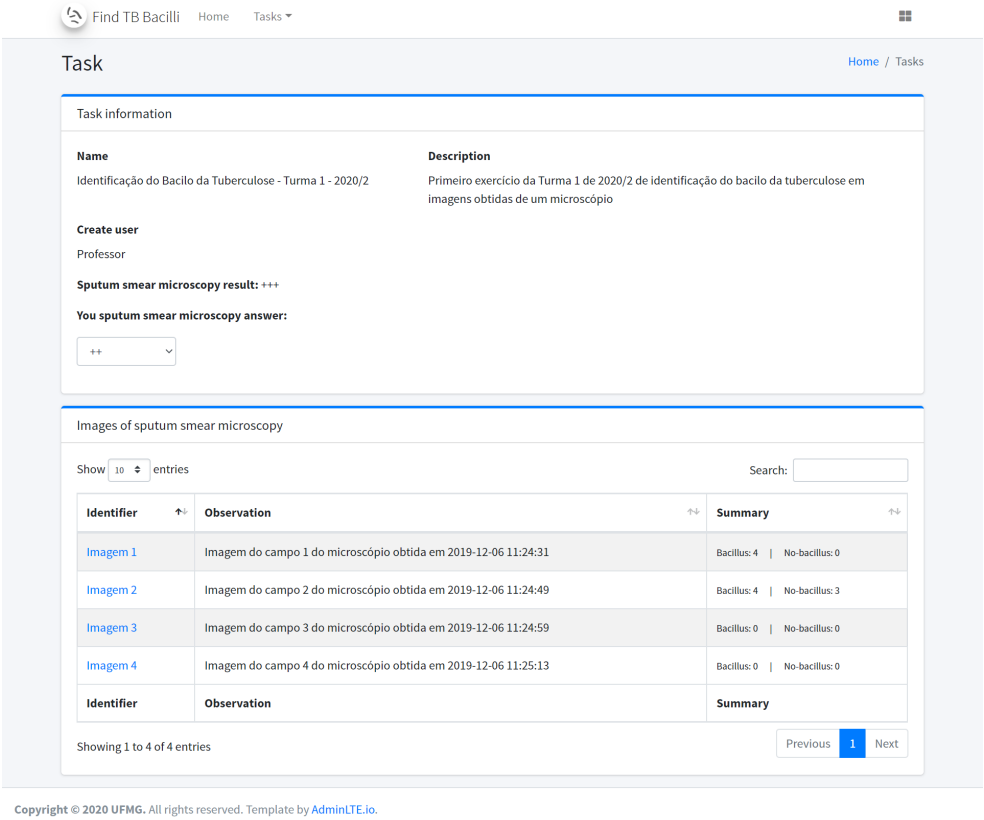


Figura 64 – Protótipo da interface para o aluno realizar a tarefa.

Figura 65 – Protótipo da interface para realizar a marcação de bacilos em uma imagem.

B Códigos-Fontes

Visando fornecer um entendimento prático e diretamente aplicável das estratégias e metodologias discutidas ao longo desta tese, este apêndice se dedica a apresentar os códigos-fonte e implementações mais significativas realizadas durante o trabalho. A intenção é não apenas ilustrar os conceitos teóricos abordados nos capítulos, mas também assegurar que outros pesquisadores e interessados possam facilmente reproduzir e validar os experimentos conduzidos, promovendo assim a integridade e a confiabilidade dos resultados obtidos.

Para garantir a acessibilidade e a facilidade de uso, os códigos foram implementados utilizando ferramentas e bibliotecas amplamente reconhecidas na comunidade científica e de aprendizado de máquina, como TensorFlow, Keras, IceVision, OpenCV, Scikit-learn, Numpy, dentre outras, disponibilizadas para o Python. Além disso, optou-se pelo uso da plataforma Jupyter Notebook, conhecida por sua interface intuitiva e pela capacidade de integrar código, visualizações e texto em um único ambiente.

Este apêndice, portanto, não apenas reforça os princípios de transparência e reprodutibilidade inerentes a uma pesquisa científica rigorosa, mas também se estabelece como um recurso didático e prático de grande valor. Ele serve tanto para aqueles que buscam validar os resultados apresentados nesta tese, quanto para aqueles que desejam se aprofundar na aplicação prática de técnicas avançadas de aprendizado de máquina para o diagnóstico de tuberculose através da baciloscopia.

B.1 Treinamento e teste das CNNs implementadas

Este apêndice disponibiliza o código-fonte implementado para treinar e testar as arquiteturas de redes neurais convolucionais (CNNs) abordadas na Seção 5.2.

Train CNN

1. Load libraries, functions and settings

In [ ]:
```
print("\n* Loading libraries... ", end="")

import sys
import os
import numpy as np
import time
import json
import psutil
import cv2
import cpuinfo
import gc
import psutil
import pandas as pd

from PIL import Image, ImageDraw, ImageOps
from io import StringIO
from datetime import datetime
from matplotlib import pyplot

import tensorflow as tf
import tensorflow.keras.backend as tfback
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, AveragePooling2D, Input
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Activation
from tensorflow.keras import activations
from tensorflow.keras.utils import plot_model
from tensorflow.python.client import device_lib
from tensorflow.keras.layers import Reshape

from sklearn.metrics import *
from sklearn.model_selection import train_test_split
from sklearn.utils.multiclass import unique_labels
from sklearn import metrics

import matplotlib.pyplot as plt

tf.config.threading.set_inter_op_parallelism_threads(12)
tf.config.threading.set_intra_op_parallelism_threads(24)


import warnings
warnings.filterwarnings('ignore')

###------------------------------------------------
type_exec = "GPU"
#type_exec = "CPU"
### To use CPU only device
if type_exec == "CPU":
    tf.config.set_visible_devices([], 'GPU')
###------------------------------------------------
### To use GPU
if type_exec == "GPU":
    print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

    from tensorflow.compat.v1 import ConfigProto
    from tensorflow.compat.v1 import InteractiveSession
    config = ConfigProto()
    config.gpu_options.per_process_gpu_memory_fraction = 0.8
    config.gpu_options.allow_growth = True
    session = InteractiveSession(config=config)

    #https://medium.com/ibm-data-ai/memory-hygiene-with-tensorflow-during-model-training-and-deployment-for-inference-45cf49a15688
    gpus = tf.config.list_physical_devices('GPU')
    if gpus:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu,True)

###------------------------------------------------


all_begin = time.time()
print("Ok")
```

1.1. Execution parameters and data

In [ ]:
```
# ================================================================================================================
# ================================================================================================================
# Execution parameters and data
# ================================================================================================================
# ================================================================================================================
kfold = 5

PATH_DATASET    = "/media/thales_mota/Novo volume/data/dataset-4/kfold/"+str(kfold)

OUT_PATH        = "/media/thales_mota/Novo volume/Google Drive/UFMG/thesis_source_codes/thesis/"
```

```
BATCH_SIZE      = 25
EPOCHS_SIZE     = 50
architecture    = "Udegova"
#architecture   = "Panicker"
#architecture   = "Quinn"
#architecture   = "Simon"
# ==================================================================================================
# ==================================================================================================
```

In [ ]:
```
if(architecture == "Udegova"):
    TARGET_SIZE_IMG = 148
elif(architecture == "Panicker"):
    TARGET_SIZE_IMG = 224
elif(architecture == "Quinn"):
    TARGET_SIZE_IMG = 20
elif(architecture == "Simon"):
    TARGET_SIZE_IMG = 20

print("\n* Parameters")
print("  - PATH_DATASET:",PATH_DATASET)
print("  - OUT_PATH    :",OUT_PATH)
print("  - BATCH_SIZE  :",BATCH_SIZE)
print("  - EPOCHS_SIZE :",EPOCHS_SIZE)
print("  - RUN IN      :",type_exec)
print("  - SIZE_IMG    :",TARGET_SIZE_IMG)

date_time    = datetime.now().strftime("%d-%m-%Y_%H%M%S")
FILE_ID_RUN  = "CNN-"+architecture+"__dataset-4-k"+str(kfold)+"__"+date_time+"__"
```

1.2. System information

In [ ]:
```
# ==================================================================================================
# ==================================================================================================
# Recover system information
# ==================================================================================================
# ==================================================================================================
print("\n* System information")

print("  - sys.version:", sys.version)
print("  - tf version:", tf.__version__)
print("  - tf keras version:", tf.keras.__version__)

print('  - CPU')
print("    -",cpuinfo.get_cpu_info()['brand_raw'])
print("    - Physical cores:", psutil.cpu_count(logical=False))
print("    - Total cores:", psutil.cpu_count(logical=True))
cpufreq = psutil.cpu_freq()
#print(f"   - Max Frequency: {cpufreq.max:.2f}Mhz")
#print(f"   - Min Frequency: {cpufreq.min:.2f}Mhz")

def _get_available_gpus():
    #global _LOCAL_DEVICES
    if tfback._LOCAL_DEVICES is None:
        devices = tf.config.list_logical_devices()
        tfback._LOCAL_DEVICES = [x.name for x in devices]
    return tfback._LOCAL_DEVICES

tfback._get_available_gpus = _get_available_gpus

print("  - Devices:",device_lib.list_local_devices())
print()

device_name = tf.test.gpu_device_name()
#if device_name != '/device:GPU:0':
#  raise SystemError('GPU device not found')
print('  - Found GPU at: {}'.format(device_name))

print('  - tf threading')
print("    - inter_op_parallelism_threads:", tf.config.threading.get_inter_op_parallelism_threads() )
print("    - intra_op_parallelism_threads:", tf.config.threading.get_intra_op_parallelism_threads() )
# ==================================================================================================
# ==================================================================================================
```

1.3. Useful and diverse functions

In [ ]:
```
# ==================================================================================================
# ==================================================================================================
# Useful and diverse functions
# ==================================================================================================
# ==================================================================================================

# ----------------------------------------------------------------------------
# plot diagnostic learning curves
def summarize_diagnostics(history, out_filename):
  # plot loss
  pyplot.subplot(211)
  pyplot.title('Cross Entropy Loss')
  pyplot.plot(history.history['loss'], color='blue', label='train')
  pyplot.plot(history.history['val_loss'], color='orange', label='test')
  pyplot.suptitle('_', fontsize=16)
  # plot accuracy
  pyplot.subplot(212)
  pyplot.title('Classification Accuracy')
  pyplot.plot(history.history['accuracy'], color='blue', label='train')
  pyplot.plot(history.history['val_accuracy'], color='orange', label='test')
  pyplot.suptitle('_', fontsize=16)
  pyplot.subplots_adjust(left=0.125, bottom=0.1,  right=0.9,  top=0.9,  wspace=0.2,  hspace=0.35)
```

```
  # save plot to file
  pyplot.savefig( out_filename)
  #pyplot.show()
  pyplot.close()
# --------------------------------------------------------------------------

# --------------------------------------------------------------------------
def my_imshow(img,scale_percent):
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dsize = (width, height)
    frame2 = cv2.resize(img, dsize)

    #frame2 = cv2.resize(img, (800, 600))
    cv2.imshow('frame1',frame2)
    k = cv2.waitKey(0) & 0xFF
    if k == 27:         # wait for ESC key to exit
        cv2.destroyAllWindows()
# --------------------------------------------------------------------------

# --------------------------------------------------------------------------
def show(img, label):
    pyplot.figure()
    pyplot.imshow(img)
    pyplot.title(label)
    #plt.axis('off')
# --------------------------------------------------------------------------

# --------------------------------------------------------------------------
class MyCustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        print("-----------------")
        tf.keras.backend.clear_session()
        tf.compat.v1.reset_default_graph()
        tf.compat.v1.get_default_graph().finalize()
        gc.collect()
        print("Memory Ram: ",round(psutil.virtual_memory().used/1024**3,2), " (Free: ",round(psutil.virtual_memory().available * 100 / psutil.vir
        print("-----------------")
# --------------------------------------------------------------------------

# --------------------------------------------------------------------------
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=pyplot.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    #    print("Normalized confusion matrix")
    #else:
    #    print('Confusion matrix, without normalization')

    #print(cm)

    fig, ax = pyplot.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    pyplot.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    #plt.xlim(-0.5, len(np.unique(y))-0.5)
    #plt.ylim(len(np.unique(y))-0.5, -0.5)
    return ax
# --------------------------------------------------------------------------
# ==========================================================================================================
# ==========================================================================================================
```

2. Load the data

In [ ]:
```
# ================================================================================================================
# ================================================================================================================
# Load the data
# ================================================================================================================
# ================================================================================================================
print("\n* Loading dataset")


#create ImageDataGenerator
train_datagen       = ImageDataGenerator(rescale = 1./255)
test_datagen        = ImageDataGenerator(rescale = 1./255)
validation_datagen  = ImageDataGenerator(rescale = 1./255)


print("    - Reading files folders" )
train_datagen      = train_datagen.flow_from_directory(     PATH_DATASET+'/train/',      target_size = (TARGET_SIZE_IMG, TARGET_SIZE_IMG), batch_
validation_datagen = validation_datagen.flow_from_directory(PATH_DATASET+'/validation/', target_size = (TARGET_SIZE_IMG, TARGET_SIZE_IMG), batch_
test_datagen       = test_datagen.flow_from_directory(      PATH_DATASET+'/test/',       target_size = (TARGET_SIZE_IMG, TARGET_SIZE_IMG), batch_

validation_datagen_labels = validation_datagen.classes
test_datagen_labels       = test_datagen.classes
class_indices             = train_datagen.class_indices

print("   - train (",round((len(train_datagen.classes) * 100) / (len(train_datagen.classes) + len(validation_datagen.classes) + len(test_datagen.
print("      -", train_datagen.class_indices)
print("      - 0:", np.count_nonzero(train_datagen.classes == 0))
print("      - 1:", np.count_nonzero(train_datagen.classes == 1))

print("   - Validation (",round((len(validation_datagen.classes) * 100) / (len(train_datagen.classes) + len(validation_datagen.classes) + len(tes
print("      -", validation_datagen.class_indices)
print("      - 0:", np.count_nonzero(validation_datagen.classes == 0))
print("      - 1:", np.count_nonzero(validation_datagen.classes == 1))

print("   - test (",round((len(test_datagen.classes) * 100) / (len(train_datagen.classes) + len(validation_datagen.classes) + len(test_datagen.cl
print("      -", test_datagen.class_indices)
print("      - 0:", np.count_nonzero(test_datagen.classes == 0))
print("      - 1:", np.count_nonzero(test_datagen.classes == 1))


plt.figure(figsize=(16,5))
plt.suptitle('Some images from the training dataset', fontsize=16)
for i in range(24):
    plt.subplot(2,12,i+1)
    for x,y in train_datagen:
        plt.imshow((x[0]))
        plt.title('{} ({})'.format( list(class_indices.keys())[list(class_indices.values()).index(y[0])] , int(y[0]) )     )
        plt.axis('off')
        break
plt.tight_layout()
plt.show()


train_datagen.reset()
gc.collect()
# ================================================================================================================
# ================================================================================================================
```

3. Layer architecture structure to generate the CNN models

In [ ]:
```
# ================================================================================================================
# ================================================================================================================
# layer architecture structure to generate the CNN models
# ================================================================================================================
# ================================================================================================================
model = Sequential()


#-------------------------------------------------------------------------------------------------
# Quinn et al. (2016)
if(architecture == "Quinn"):
    model.add(Conv2D(7, (3,3), input_shape = (20, 20, 3), activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))
    model.add(Conv2D(12, (3,3), activation = 'relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Flatten())
    model.add(Dense(units = 500, activation = 'relu'))
    model.add(Dense(units = 1, activation = 'sigmoid'))
#-------------------------------------------------------------------------------------------------

#-------------------------------------------------------------------------------------------------
# Panicker et al. (2018)
if(architecture == "Panicker"):
    model.add(Conv2D(32, (3,3), input_shape = (224, 224, 3), activation = 'relu'))
    model.add(BatchNormalization())
    model.add(Activation(activations.relu))
    model.add(MaxPooling2D(pool_size = (2,2)))

    model.add(Conv2D(32, (3,3), activation = 'relu' ))
    model.add(BatchNormalization())
    model.add(Activation(activations.relu))
    model.add(MaxPooling2D(pool_size = (2,2)))

    model.add(Conv2D(64, (3,3), activation = 'relu' ))
    model.add(BatchNormalization())
    model.add(Activation(activations.relu))
    model.add(MaxPooling2D(pool_size = (2,2)))

    model.add(Conv2D(64, (3,3), activation = 'relu' ))
```

```
    model.add(BatchNormalization())
    model.add(Activation(activations.relu))
    model.add(MaxPooling2D(pool_size = (2,2)))

    model.add(Conv2D(64, (3,3), activation = 'relu' ))
    model.add(BatchNormalization())
    model.add(Activation(activations.relu))
    model.add(MaxPooling2D(pool_size = (2,2)))

    model.add(Conv2D(128, (3,3), activation = 'relu' ))
    model.add(BatchNormalization())
    model.add(Activation(activations.relu))
    model.add(MaxPooling2D(pool_size = (2,2)))
    model.add(Flatten())
    model.add(Dense(units = 128, activation = 'relu'))
    model.add(Dropout(0.05))

    model.add(Dense(units = 1, activation = 'sigmoid'))
#-----------------------------------------------------------------------------------------------



#-----------------------------------------------------------------------------------------------
# Udegova et al. (2021)
if(architecture == "Udegova"):
    model.add( Input([148, 148, 3]) )
    model.add(BatchNormalization())
    #model.add( tf.keras.layers.experimental.preprocessing.Resizing(148,148) )

    model.add(Conv2D(32, (1,1), activation = 'relu' ))
    model.add(MaxPooling2D(pool_size = (2,2)))

    model.add(Conv2D(32, (1,1), activation = 'relu' ))
    model.add(MaxPooling2D(pool_size = (11,11), strides=(2, 2)))

    model.add(Conv2D(32, (1,1), activation = 'relu' ))
    model.add(AveragePooling2D())

    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(units = 512, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(units = 254, activation = 'relu'))
    model.add(Dropout(0.4))
    model.add(Dense(units = 64, activation = 'relu'))
    model.add(Dropout(0.3))
    model.add(Dense(units = 2, activation = 'relu'))
    model.add(Dense(units = 1, activation = 'sigmoid'))
#-----------------------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------------
#  Simon et al (2019)
if(architecture == "Simon"):
    model = Sequential()
    model.add( Input([20, 20, 3]) )
    model.add(Conv2D(7, (3,3),   activation = 'relu'))
    model.add(MaxPooling2D(pool_size = (2,2)))
    model.add(tf.keras.layers.Reshape((1,9,9,7)))
    model.add( tf.keras.layers.ConvLSTM2D(20, (3,3) ) )
    model.add(Flatten())
    model.add(Dense(units = 1, activation = 'sigmoid'))
#-----------------------------------------------------------------------------------------------


opt = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer = opt, loss = 'binary_crossentropy',
              metrics = ['accuracy',
                         tf.keras.metrics.Precision(),
                         tf.keras.metrics.Recall(),
                         tf.keras.metrics.MeanSquaredError(),
                         tf.keras.metrics.AUC() ] ,run_eagerly=True)


print("* Summarize Model: ")
print(model.summary())
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
# ================================================================================================
# ================================================================================================
```

4. Training

In [ ]:

```
# ================================================================================================
# ================================================================================================
# Perform the training and generate the model
# ================================================================================================
# ================================================================================================

print("\n* Conducting the training and generating the model...")
time1 = time.time()

print('  - Starting  (', datetime.now().strftime("%d-%m-%Y_%H%M%S") ,')' )
# Conduct the training
history = model.fit_generator(train_datagen, steps_per_epoch=len(train_datagen),
                                                        validation_data = validation_datagen, validation_steps= len(validation_datagen),
                                                        epochs=EPOCHS_SIZE, verbose=1,callbacks=[MyCustomCallback()] )

print('  - Finished' )
time2 = time.time()
```

```
time_model_train = round( (((time2-time1)*1000.0)/1000), 2)
time_model_train = str(time_model_train)+'s  ('+str(time_model_train/60)+'m)('+str(time_model_train/60/60)+'h)'
print('  - Time: ',time_model_train )


plt.title('History Train')
plt.plot(history.history['accuracy'], label='accuracy ('+ "{:.4f}".format( history.history['accuracy'][-1])+')' )
plt.plot(history.history['precision'], label='precision ('+ "{:.4f}".format( history.history['precision'][-1])+')' )
plt.plot(history.history['recall'], label='recall ('+ "{:.4f}".format( history.history['recall'][-1])+')' )
plt.plot(history.history['mean_squared_error'], label='mean_squared_error ('+ "{:.4f}".format( history.history['mean_squared_error'][-1])+')' )
plt.plot(history.history['auc'], label='auc ('+ "{:.4f}".format( history.history['auc'][-1])+')' )
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Value')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.savefig((OUT_PATH + FILE_ID_RUN +'train-plot1.png'))
display(Image.open((OUT_PATH + FILE_ID_RUN +'train-plot1.png')))

summarize_diagnostics(history, (OUT_PATH + FILE_ID_RUN +'train-plot2.png') )

# Save the model
print("  - Saving model...")
modelNameSaved = OUT_PATH + FILE_ID_RUN +'MODEL.h5'
model.save(modelNameSaved)
print("      - Model architecture saved in:", modelNameSaved, '\n')

dot_img_file = OUT_PATH + FILE_ID_RUN +'MODEL-architecture.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
print("      - Image architecture saved in:", dot_img_file, '\n')
```

In [ ]:
```
# Evaluate model in validation data
print('  - Evaluate model in validation data')
out_loss_, out_acc, out_precision, out_recall, out_mse, out_auc = model.evaluate_generator(validation_datagen, steps=len(validation_datagen), ver
print('  - validation Accuracy: %.3f' % (out_acc * 100))

# ================================
# Save train/validation results and settings
print("  - Saving train/validation results and settings...")
modelConfigNameSaved = OUT_PATH + FILE_ID_RUN +'_result-training-validation.txt'
f = open(modelConfigNameSaved, "a")
f.write("-------------------------------------------------------------------------------\n")
f.write("Evaluate model in validation data\n")
f.write("* Accuracy train/validation  : "+"{:.4f}".format(out_acc)+"\n")
f.write("* Precision train/validation : "+"{:.4f}".format(out_precision)+"\n")
f.write("* Recall train/validation    : "+"{:.4f}".format(out_recall)+"\n")
f.write("* MSE train/validation       : "+"{:.4f}".format(out_mse)+"\n")
f.write("* AUC train/validation       : "+"{:.4f}".format(out_auc)+"\n")
f.write("\n\n")
f.write("-------------------------------------------------------------------------------\n")
f.write("* Time training: "+time_model_train+"\n")
f.write("* Path_dataset : "+PATH_DATASET+"\n")
f.write("* Batch_size   : "+str(BATCH_SIZE)+"\n")
f.write("* Epochs       : "+str(EPOCHS_SIZE)+"\n")
f.write("* Run in       : "+type_exec+"\n")
f.write("\n\n")
f.write("-------------------------------------------------------------------------------\n")
f.write("* Model architecture:\n" )
f.write( json.dumps(model.get_config(), indent=3, sort_keys=True) )

tmp_smry = StringIO()
model.summary(print_fn=lambda x: tmp_smry.write(x + '\n'))
f.write("\nSummary:\n")
f.write(tmp_smry.getvalue()+"\n")

f.close()

print("Saved file content:")
a_file = open(modelConfigNameSaved)
file_contents = a_file.read()
print(file_contents)
a_file.close()
# ================================================================================================================
# ================================================================================================================
```

5. Evaluate the model on the test dataset in detail

In [ ]:
```
# ================================================================================================================
# ================================================================================================================
# Evaluate the model on the test dataset in detail
# ================================================================================================================
# ================================================================================================================
# Copy datagen
data_run_datagen        = test_datagen
data_run_datagen_labels = data_run_datagen.classes
data_run_datagen.reset()

modelTestFileSaved = OUT_PATH + FILE_ID_RUN +'test-result.txt'
f = open(modelTestFileSaved, "a")
f.write("Result in test set\n\n")


print('\n* Evaluating test set...')

data_run_datagen.reset()
Y_pred = model.predict(data_run_datagen, steps=len(data_run_datagen), verbose=1)
data_run_datagen.reset()
```

```
Y_pred = np.around(Y_pred)
Y_pred = Y_pred.astype(int)
y_pred = np.squeeze(np.asarray(Y_pred))
print('  - Ok')


# ----------------------
# Confusion matrix
confusion = confusion_matrix(data_run_datagen_labels, y_pred)
f.write('Confusion Matrix\n')
f.write(str(confusion))
f.write('\n\n')

np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
ax = plot_confusion_matrix(data_run_datagen_labels, y_pred, classes=np.array(['bacillus', 'no-bacillus']),
                      title='Confusion matrix, without normalization (test dataset)')
pyplot.savefig(  OUT_PATH + FILE_ID_RUN +'CM-test.png' )


# Plot normalized confusion matrix
plot_confusion_matrix(data_run_datagen_labels, y_pred, classes= np.array(['bacillus', 'no-bacillus']), normalize=True,
                      title='Normalized confusion matrix (test dataset)')
pyplot.savefig(  OUT_PATH + FILE_ID_RUN +'CM-test-normalized.png' )


target_names = ['bacillus', 'no-bacillus']
f.write( classification_report(data_run_datagen_labels, y_pred, target_names=target_names) +"\n")
# ----------------------


# ----------------------
# AUC
fpr_keras, tpr_keras, thresholds_keras = roc_curve(data_run_datagen_labels, y_pred)
auc_keras = auc(fpr_keras, tpr_keras)
# ----------------------


# ----------------------
# Details
report_tp          = confusion[1][1]
report_tn          = confusion[0][0]
report_fp          = confusion[0][1]
report_fn          = confusion[1][0]
report_accuracy    = metrics.accuracy_score(data_run_datagen_labels, y_pred)
report_recall      = metrics.recall_score(data_run_datagen_labels, y_pred)
report_specificity = (report_tn / float(report_tn + report_fp))
report_precision = metrics.precision_score(data_run_datagen_labels, y_pred)
report_f1        = metrics.f1_score(data_run_datagen_labels, y_pred)
report_mse = metrics.mean_squared_error(data_run_datagen_labels, y_pred)
report_auc = auc_keras

f.write("TP         : "+'{:7.0f}'.format(report_tp) +" # true positive (TP) "+"\n")
f.write("TN         : "+'{:7.0f}'.format(report_tn) +" # true negative (TN)"+"\n")
f.write("FP         : "+'{:7.0f}'.format(report_fp) +" # false positive (FP)"+"\n")
f.write("FN         : "+'{:7.0f}'.format(report_fn) +" # false negative (FN)"+"\n")
f.write("Accuracy   : "+'{:7.4f}'.format(report_accuracy)  +" # accuracy"+"\n")
f.write("Recall     : "+'{:7.4f}'.format(report_recall)    +" # sensitivity, recall, hit rate, or true positive rate (TPR)"+"\n")
f.write("Specificity: "+'{:7.4f}'.format(report_specificity) +" # specificity, selectivity or true negative rate (TNR) "+"\n")
f.write("Precision  : "+'{:7.4f}'.format(report_precision) +" # precision or positive predictive value (PPV)"+"\n")
f.write("F1 score   : "+'{:7.4f}'.format(report_f1) +" # F1 score"+"\n")
f.write("MSE        : "+'{:7.4f}'.format(report_mse) +" # mean squared error"+"\n")
f.write("AUC        : "+'{:7.4f}'.format(report_auc) +" # Area Under the Receiver Operating Characteristic Curve (ROC AUC)"+"\n")
f.write("Time (s)   : "+'{:7.0f}'.format((time.time() - all_begin)) +"\n")



# ----------------------

f.close()

print("  - Saved file content:",modelTestFileSaved,"\n")
a_file = open(modelTestFileSaved)
file_contents = a_file.read()
print(file_contents)
a_file.close()
# ================================================================================================================
# ================================================================================================================
```

In [ ]:

```
file_log_csv_name = OUT_PATH + 'exec_result_cnn.csv'

header_csv = ""
if( os.path.isfile(file_log_csv_name) == False):
    header_csv = "ID;TP;TN;FP;FN;Accuracy;Recall;Specificity;Precision;F1 score;MSE;AUC;Time (s)\n"

file_log_csv = open( file_log_csv_name , 'a')

file_log_csv.write(header_csv)
file_log_csv.write(str( FILE_ID_RUN ) +";")
file_log_csv.write(str( report_tp ) +";")
file_log_csv.write(str( report_tn ) +";")
file_log_csv.write(str( report_fp ) +";")
file_log_csv.write(str( report_fn ) +";")
file_log_csv.write(str( report_accuracy ) +";")
file_log_csv.write(str( report_recall ) +";")
file_log_csv.write(str( report_specificity  ) +";")
file_log_csv.write(str( report_precision ) +";")
file_log_csv.write(str( report_f1 ) +";")
file_log_csv.write(str( report_mse ) +";")
```

```
file_log_csv.write(str( report_auc ) +";")
file_log_csv.write(str(time.time() - all_begin) +";")


file_log_csv.write("\n")

file_log_csv.close()
```

Website references and acknowledgments

https://www.tensorflow.org/tutorials/images/cnn

https://medium.com/ibm-data-ai/memory-hygiene-with-tensorflow-during-model-training-and-deployment-for-inference-45cf49a15688

https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/

https://github.com/tensorflow/tensorflow/issues/31312

https://www.dlology.com/blog/simple-guide-on-how-to-generate-roc-plot-for-keras-classifier/

https://stackoverflow.com/questions/71958700/valueerror-input-0-of-layer-lstm-14-is-incompatible-with-the-layer-expected-nd

And many other websites...

B.2 Teste K-means + CNN

Este apêndice disponibiliza o código-fonte implementado para realizar o teste com o método K-means para identificação das regiões de interesse e a classificação usando a CNNs abordadas na Seção 5.3 .

Detection tuberculosis bacilli from microscopic sputum smear images - K-means + CNN

1. Load libraries, functions and settings

1.1 Load libraries

In [ ]:
```
print("\n* Loading libraries... ", end="")

import cv2
import numpy as np
import matplotlib.pyplot as plt
from copy import deepcopy
import time
import os
from IPython.display import display
from IPython.display import Image
from PIL import Image, ImageDraw , ExifTags, ImageOps
import PIL
from tensorflow.keras.preprocessing.image import img_to_array, array_to_img
from tensorflow.keras.models import load_model
import csv
import sklearn.metrics as skm
from sklearn.metrics import average_precision_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
from datetime import datetime
from mean_average_precision import MetricBuilder
import numpy as np
import pandas as pd
from sklearn.metrics import precision_score, recall_score
import matplotlib.pyplot as plt

import tensorflow as tf
###------------------------------------------------
type_exec = "GPU"
#type_exec = "CPU"
### To use CPU only device
if type_exec == "CPU":
    tf.config.set_visible_devices([], 'GPU')
###------------------------------------------------
### To use GPU
if type_exec == "GPU":
    print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

    from tensorflow.compat.v1 import ConfigProto
    from tensorflow.compat.v1 import InteractiveSession
    config = ConfigProto()
    config.gpu_options.per_process_gpu_memory_fraction = 0.8
    config.gpu_options.allow_growth = True
    session = InteractiveSession(config=config)

    #https://medium.com/ibm-data-ai/memory-hygiene-with-tensorflow-during-model-training-and-deployment-for-inference-45cf49a15688
    gpus = tf.config.list_physical_devices('GPU')
    if gpus:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu,True)

###------------------------------------------------


print("Ok")
```

1.2. Useful and diverse functions

In [ ]:
```
#------------------------------------------------------------------------------------------
def non_max_suppression_fast(pred, overlapThresh):
    #https://pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/
    pred_ = deepcopy(pred)
    boxes=[]
    for i in range(len(pred_)):
        #print(pred['detection']['bboxes'][i])
        if(len(pred_[i]) == 5):
            boxes.append([ pred_[i][0] , pred_[i][1] , pred_[i][2] , pred_[i][3], pred_[i][4]])
        else:
            boxes.append([ pred_[i][0] , pred_[i][1] , pred_[i][2] , pred_[i][3]])

    boxes = np.array(boxes)

    # if there are no boxes, return an empty list
    if len(boxes) == 0:
        return []
    # if the bounding boxes integers, convert them to floats --
    # this is important since we'll be doing a bunch of divisions
    if boxes.dtype.kind == "i":
        boxes = boxes.astype("float")
    # initialize the list of picked indexes
    pick = []
    # grab the coordinates of the bounding boxes
    x1 = boxes[:,0]
    y1 = boxes[:,1]
    x2 = boxes[:,2]
```

```
    y2 = boxes[:,3]
    # compute the area of the bounding boxes and sort the bounding
    # boxes by the bottom-right y-coordinate of the bounding box
    area = (x2 - x1 + 1) * (y2 - y1 + 1)
    idxs = np.argsort(y2)
    # keep looping while some indexes still remain in the indexes
    # list
    while len(idxs) > 0:
        # grab the last index in the indexes list and add the
        # index value to the list of picked indexes
        last = len(idxs) - 1
        i = idxs[last]
        pick.append(i)
        # find the largest (x, y) coordinates for the start of
        # the bounding box and the smallest (x, y) coordinates
        # for the end of the bounding box
        xx1 = np.maximum(x1[i], x1[idxs[:last]])
        yy1 = np.maximum(y1[i], y1[idxs[:last]])
        xx2 = np.minimum(x2[i], x2[idxs[:last]])
        yy2 = np.minimum(y2[i], y2[idxs[:last]])
        # compute the width and height of the bounding box
        w = np.maximum(0, xx2 - xx1 + 1)
        h = np.maximum(0, yy2 - yy1 + 1)
        # compute the ratio of overlap
        overlap = (w * h) / area[idxs[:last]]
        # delete all indexes from the index list that have
        idxs = np.delete(idxs, np.concatenate(([last], np.where(overlap > overlapThresh)[0])))

    #return pred_
    return boxes[pick].astype("int")
#------------------------------------------------------------------------------------------------

#------------------------------------------------------------------------------------------------
def bb_intersection_over_union(boxA, boxB):
    #https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])
    # compute the area of intersection rectangle
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the interesection area
    iou = interArea / float(boxAArea + boxBArea - interArea)
    # return the intersection over union value
    return iou
#------------------------------------------------------------------------------------------------

#------------------------------------------------------------------------------------------------
# Print iterations progress
def printProgressBar (iteration, total, prefix = '', suffix = '', decimals = 1, length = 100, fill = '', printEnd = "\r"):
    #https://splunktool.com/python-command-line-loading-bar
    """
    Call in a loop to create terminal progress bar
    @params:
        iteration   - Required  : current iteration (Int)
        total       - Required  : total iterations (Int)
        prefix      - Optional  : prefix string (Str)
        suffix      - Optional  : suffix string (Str)
        decimals    - Optional  : positive number of decimals in percent complete (Int)
        length      - Optional  : character length of bar (Int)
        fill        - Optional  : bar fill character (Str)
        printEnd    - Optional  : end character (e.g. "\r", "\r\n") (Str)
    """
    percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration / float(total)))
    filledLength = int(length * iteration // total)
    bar = fill * filledLength + '-' * (length - filledLength)
    print(f'\r{prefix} |{bar}| {percent}% {suffix}', end = printEnd)
    # Print New Line on Complete
    if iteration == total:
        print()
#------------------------------------------------------------------------------------------------

#------------------------------------------------------------------------------------------------
def VOC_PASCAL_mAP(pred, gt, iou_thresh):
    #https://pypi.org/project/mean-average-precision/

    pred_ = []
    gt_ = []
    for i in range(len(pred)):
            pred_xmin   = pred[i][0]
            pred_ymin   = pred[i][1]
            pred_xmax   = pred[i][2]
            pred_ymax   = pred[i][3]
            pred_class_ = pred[i][5]
            pred_score  = pred[i][4]

            pred_.append( [pred_xmin, pred_ymin, pred_xmax, pred_ymax, pred_class_ , pred_score] )

    pred_ = np.array(pred_)


    for j in range(len(gt)):
            gt_xmin   = gt[i][0]
```

```
            gt_ymin   = gt[i][1]
            gt_xmax   = gt[i][2]
            gt_ymax   = gt[i][3]
            gt_class_ = gt[i][4]

            gt_.append(  [gt_xmin, gt_ymin, gt_xmax, gt_ymax, gt_class_, 0])

    gt_ = np.array(gt_)

    # create metric_fn
    metric_fn = MetricBuilder.build_evaluation_metric("map_2d", async_mode=True, num_classes=1)

    # add some samples to evaluation
    for i in range(10):
        metric_fn.add(pred_, gt_)

    return metric_fn.value(iou_thresholds=iou_thresh)['mAP']
#-----------------------------------------------------------------------------------------


#-----------------------------------------------------------------------------------------
def AveragePrecisionScore(y_true_class, pred_scores_class, plot=False, thresholds = np.arange(start=0.2, stop=0.9, step=0.05)  ):
  # Based on: https://analyticsindiamag.com/how-does-mean-average-precision-map-work-in-object-detection/

    def precision_recall_curve(y_true, pred_scores, thresholds):
        precisions = []
        recalls = []

        for threshold in thresholds:
            y_pred = [1 if score >= threshold else 0 for score in pred_scores]

            precision = skm.precision_score(y_true=y_true, y_pred=y_pred, pos_label=1)
            recall = skm.recall_score(y_true=y_true, y_pred=y_pred, pos_label=1)

            precisions.append(precision)
            recalls.append(recall)

        return precisions, recalls


    precisions, recalls = precision_recall_curve(y_true=y_true_class, pred_scores=pred_scores_class, thresholds=thresholds)
    precisions.append(1)
    recalls.append(0)

    precisions = np.array(precisions)
    recalls = np.array(recalls)

    avg_precision_class = np.sum((recalls[:-1] - recalls[1:]) * precisions[:-1])

    if(plot):
        precisions_, recalls_ = precision_recall_curve(y_true=y_true_class, pred_scores=pred_scores_class, thresholds=thresholds)
        plt.plot(recalls_, precisions_, label=('Average precision score: '+str(np.round(avg_precision_class,4))), linewidth=2, color="red", zorde
        plt.xlabel("Recall", fontsize=12, fontweight='bold')
        plt.ylabel("Precision", fontsize=12, fontweight='bold')
        plt.title("Precision-Recall Curve", fontsize=15, fontweight="bold")
        plt.legend(loc="best")
        plt.show()

    return avg_precision_class
#-----------------------------------------------------------------------------------------
```

In [ ]:

```
#-----------------------------------------------------------------------------------------
def get_map(pred, gt, iou_thresh):
    T = []
    P = []
    P_index = []
    iou = []
    scores = []
    iou_class = []


    if( len(pred) == 0 and len(gt) > 0):
      for i in range(len(gt)):
            T.append(1)
            P.append(0)
            P_index.append(-1)
            scores.append(0)
            iou.append(0)
      return T, P, iou, scores, P_index


    gt_xmin_matched    = np.zeros(len(gt), dtype=bool)
    pred_matched_quant = np.zeros(len(pred), dtype=int)

    for i in range(len(pred)):
            pred_xmin    = pred[i][0]
            pred_ymin    = pred[i][1]
            pred_xmax    = pred[i][2]
            pred_ymax    = pred[i][3]
            pred_score   = pred[i][4]
            pred_class_  = pred[i][5]

            bbox_matched = False

            iou.append(0.0);
            iou_class.append(0);

            for j in range(len(gt)):
                gt_xmin   = gt[j][0]
                gt_ymin   = gt[j][1]
```

```
                gt_xmax   = gt[j][2]
                gt_ymax   = gt[j][3]
                gt_class_ = gt[j][4]

                iou_ = bb_intersection_over_union([gt_xmin,gt_ymin,gt_xmax,gt_ymax], [pred_xmin,pred_ymin,pred_xmax,pred_ymax])

                if( iou_ >= iou_thresh    ):
                    if( iou_ >= iou[i] ):
                        iou[i] = iou_

                    iou_class[i]       = gt_class_
                    bbox_matched       = True
                    #gt_xmin_matched[j] = True
                    pred_matched_quant[i] = pred_matched_quant[i] + 1

                    if(bbox_matched and iou[i] >= iou_thresh and pred_class_ == iou_class[i]):
                        T.append(1)
                        P.append(1)
                        P_index.append(i)
                        gt_xmin_matched[j] = True


                    scores.append(pred_score)

            if(bbox_matched == False):
                T.append(0)
                P.append(1)
                P_index.append(i)
                scores.append(pred_score)

    for j in range(len(gt)):
        if( gt_xmin_matched[j] == False ):
            T.append(1)
            P.append(0)
            P_index.append(-1)
            scores.append(0)
            iou.append(0)


    return T, P, iou, scores, P_index
#-----------------------------------------------------------------------------------------
```

1.3. Execution parameters and data

In [ ]:
```
#-----------------------------------------------------------------------------------------
#Parameters

modelNameSaved =  "/media/thales_mota/Novo volume/Google Drive/UFMG/thesis_source_codes/thesis/CNN-Udegova__dataset-4+error-k1__25-08-2023_062935

type_data           = "test" # train validation test
kfold               = "1"

path_csv_annotation = "/media/thales_mota/Novo volume/data/dataset-4/kfold/"+kfold+"/"+type_data+"-positive--augmentation.csv"
path_images         = "/media/thales_mota/Novo volume/data/dataset-4/original_images_D.A/"
OUT_PATH            = "/media/thales_mota/Novo volume/Google Drive/UFMG/thesis_source_codes/thesis/"

K         = 5
IoU_map   = 0.1
Threshold = 0.5

save_erro_image_path = ""
if type_data == "train" or type_data == "validation" :
    IoU_map   = 0.005
    save_erro_image_path = "D:/data_error/dataset-2+error/kfold/"+kfold+"/"+type_data+"/"


date_time    = datetime.now().strftime("%d-%m-%Y_%H%M%S")
FILE_ID_RUN  = "KMeans-CNN__dataset-4-test-k"+kfold+"__"+date_time+"__"
datetime     = time.strftime("%Y%m%d-%H%M%S")
```

In [ ]:
```
print("modelNameSaved       :",modelNameSaved)
print("type_data            :",type_data)
print("kfold                :",kfold)
print("path_csv_annotation  :",path_csv_annotation)
print("path_images          :",path_images)
print("OUT_PATH             :",OUT_PATH)
print("save_erro_image_path :",save_erro_image_path)
print("FILE_ID_RUN          :",FILE_ID_RUN)
```

2. K-Means

In [ ]:
```
#-----------------------------------------------------------------------------------------
def increase_saturation(imagem, aumento):
    # Convert image to HSV color space
    hsv = cv2.cvtColor(imagem, cv2.COLOR_BGR2HSV)

    # Increase saturation
    hsv[:,:,1] = np.clip(hsv[:,:,1] * aumento, 0, 255)

    # Convert image back to BGR color space
    bgr = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

    return bgr
#-----------------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------
def filter_map_color(imagem):
```

```
    hsv = cv2.cvtColor(imagem, cv2.COLOR_BGR2HSV)

    # Set the color ranges to pink, purple, and red in HSV
    rosa_min = np.array([150, 50, 50])
    rosa_max = np.array([170, 255, 255])
    roxo_min = np.array([120, 50, 50])
    roxo_max = np.array([150, 255, 255])
    vermelho_min1 = np.array([0, 50, 50])
    vermelho_max1 = np.array([10, 255, 255])
    vermelho_min2 = np.array([170, 50, 50])
    vermelho_max2 = np.array([180, 255, 255])

    # Create masks for pink, purple and red colors
    mask_rosa = cv2.inRange(hsv, rosa_min, rosa_max)
    mask_roxo = cv2.inRange(hsv, roxo_min, roxo_max)
    mask_vermelho1 = cv2.inRange(hsv, vermelho_min1, vermelho_max1)
    mask_vermelho2 = cv2.inRange(hsv, vermelho_min2, vermelho_max2)
    mask_vermelho = mask_vermelho1 + mask_vermelho2

    # Match all masks
    mask = mask_rosa + mask_roxo + mask_vermelho

    imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)

    # Apply the mask to preserve the colors of interest
    imagem_preservada = cv2.bitwise_and(imagem, imagem, mask=mask)

    # Convert grayscale part to original color
    imagem_preservada = cv2.bitwise_and(imagem, imagem, mask=mask)
    return imagem_preservada
#-------------------------------------------------------------------------------------

#-------------------------------------------------------------------------------------
def classification(img_file_in, k, model_load, threshold, degub_view_op1, degub_view_op2):
    #=============================================================================
    # kmeans segmentation
    #=============================================================================
    begin1 = time.time()
    if(degub_view_op1):
        print("\n* Starting kmeans segmentation (k=",k,")...",sep="")

    size_figsize1 = (14,14)
    size_figsize2 = (9,9)

    image = cv2.imread(img_file_in)
    if(degub_view_op2):
        plt.figure(figsize = size_figsize1)
        print("Original Image:")
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.show()


    if(degub_view_op2):
        image = increase_saturation(image, 1.5)
        plt.figure(figsize = size_figsize1)
        print("Image [aumentar_saturacao]:")
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.show()

    image = filter_map_color(image)
    image = increase_saturation(image, 1.5)
    if(degub_view_op2):
        plt.figure(figsize = size_figsize1)
        print("Image [filtro_mapa_cor]:")
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.show()


    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    pixel_values = image.reshape((-1, 3))
    pixel_values = np.float32(pixel_values)


    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

    _, labels, (centers) = cv2.kmeans(pixel_values, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

    centers  = np.uint8(centers)

    labels   = labels.flatten()

    segmented_image = centers[labels.flatten()]
    segmented_image = segmented_image.reshape(image.shape)


    #if(degub_view_op2):
    #    plt.figure(figsize = size_figsize1)
    #    print("Segmented_image:")
    #    plt.imshow(segmented_image)
    #    plt.show()


    if(degub_view_op1):
        print('    - Elapsed time = {}s '.format( round(time.time() - begin1,2) ))
    #=============================================================================


    #=============================================================================
```

```
# binary segmentation
#================================================================================
begin1 = time.time()
if(degub_view_op1):
    print("\n* Starting binary segmentation...")


segmented_image = centers[labels.flatten()]
segmented_image.shape

all_boxs = []

i=0
while i < len(centers):
    begin = time.time()
    t = np.array(centers[i])

    if(degub_view_op2):
        print("\n=============================================================")
        print("    - K:",i,"-> rgb center (",t,")")


    segmented_image_ = deepcopy(segmented_image.copy())
    segmented_image_[segmented_image_ != t] = 0
    segmented_image_[segmented_image_ == t] = 255
    segmented_image_ = segmented_image_.reshape(image.shape)


    img = cv2.cvtColor(segmented_image_, cv2.COLOR_BGR2GRAY)

    #Binarization resulting in white and black pixels
    T = 0
    bin = img.copy()
    bin[bin > T] = 255
    bin[bin < 255] = 0
    bin = cv2.bitwise_not(bin)

    #Edge detection with Canny
    bordas = cv2.Canny(bin, 70, 150)

    kernel = np.ones((5,5),np.uint8)
    bordas = cv2.dilate(bordas,kernel,iterations = 1)

    #Identification and counting of image contours
    (objetos,lx) = cv2.findContours(bordas.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    #(objetos,lx) = cv2.findContours(bordas.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    #print("      - Objetos =",len(objetos) )


    boxs = []
    idx = 0
    for objeto in objetos:
        #retrieves the extreme points of the detected object
        a = objeto.max(axis=0)[0][0]
        b = objeto.max(axis=0)[0][1]
        c = objeto.min(axis=0)[0][0]
        d  = objeto.min(axis=0)[0][1]

        #Cut the fragment taking a larger area
        start_point = (c-5, d-5)
        end_point = (a+5, b+5)
        boxs.append([start_point[0],start_point[1], end_point[0],end_point[1]])

        idx += 1

    if(degub_view_op2):
        imgC2 = image.copy()
        for box in boxs:
            start_point = (box[0], box[1])
            end_point = (box[2], box[3])

            #cv2.drawContours(imgC2, [objeto], -1, (0, 0, 0), 1)
            cv2.rectangle(imgC2, start_point , end_point , (255, 0,0), 2)


    for box in boxs:
        all_boxs.append(box.copy())


    if(degub_view_op2):
        imgC3 = image.copy()
        for box in boxs:
            start_point = (box[0], box[1])
            end_point = (box[2], box[3])

            #cv2.drawContours(imgC2, [objeto], -1, (0, 0, 0), 1)
            cv2.rectangle(imgC3, start_point , end_point , (255, 0,0 ), 2)


        f = plt.figure(figsize = size_figsize1)
        f.add_subplot(2,2, 1)
        plt.imshow((image))
        f.add_subplot(2,2, 2)
        plt.imshow((segmented_image_))
        f.add_subplot(2,2, 3)
        plt.imshow((imgC2))
        f.add_subplot(2,2, 4)
        plt.imshow((imgC3))
        plt.show(block=True)
```

```
            print('    - Elapsed time = {}s '.format( round(time.time() - begin,2) ))

        i = i + 1

    if(degub_view_op1):
        print('    - Elapsed time = {}s '.format( round(time.time() - begin1,2) ))
    #================================================================================


    #================================================================================
    # Size filter...
    #================================================================================
    if(degub_view_op1):
        print("\n* Starting filter...")
        print("    - All boxs (",len(all_boxs),"):")

    imgC4 = image.copy()

    if(degub_view_op2):
        imgC3 = image.copy()
        for box in all_boxs:
            start_point = (box[0], box[1])
            end_point = (box[2], box[3])

            cv2.rectangle(imgC3, start_point , end_point , (255, 0,0), 2)

        plt.figure(figsize = size_figsize1)
        print("    - All boxs (",len(all_boxs),"):")
        plt.imshow(imgC3)
        plt.show()


    area_total = imgC4.shape[0] * imgC4.shape[1]

    boxs_filter_area = []
    for box in all_boxs:
        l = (box[0] - box[2] ) * -1
        h = (box[1] - box[3] ) * -1

        if( l*h < (area_total*0.005)):
            boxs_filter_area.append(box)

    all_boxs = boxs_filter_area
    if(degub_view_op1):
        print("    - After the filter (",len(all_boxs),"):")


    if(degub_view_op2):
        imgC3 = image.copy()
        for box in all_boxs:
            start_point = (box[0], box[1])
            end_point = (box[2], box[3])

            cv2.rectangle(imgC3, start_point , end_point , (255, 0,0), 2)

        plt.figure(figsize = size_figsize1)
        plt.imshow(imgC3)
        plt.show()

    if(degub_view_op1):
        print('    - Ok')
    #================================================================================


    #================================================================================
    # Starting classification...
    #================================================================================
    begin1 = time.time()
    if(degub_view_op1):
        print("\n* Starting classification...")


    img2 = PIL.Image.open(img_file_in).convert("RGB")

    imgC4 = image.copy()
    boxs_pred       = []
    boxs_pred_value = []


    all_imgs_boxs = []
    for box in all_boxs:
        start_point = (box[0], box[1])
        end_point = (box[2], box[3])
        #cropped_img = img2.crop( (box[0], box[1], box[2], box[3]))
        cropped_img = img2.crop( (box[0]-15, box[1]-15, box[2]+15, box[3]+15))
        cropped_img2 = cropped_img.resize((148, 148))
        cropped_img2 = img_to_array(cropped_img2)
        cropped_img2 /= 255
        cropped_img2 = cropped_img2.reshape( 148, 148, 3)
        all_imgs_boxs.append( cropped_img2 )
    all_imgs_boxs = np.array( all_imgs_boxs)


    if(len(all_imgs_boxs) == 0):
        return all_imgs_boxs

    predict22 = model_load.predict(all_imgs_boxs , batch_size=8, verbose=0)

    for box_i in range(0,len(all_boxs)):
        box = all_boxs[box_i]
```

```
        start_point = (box[0], box[1])
        end_point = (box[2], box[3])
        cv2.rectangle(imgC4, start_point , end_point , (255, 0, 0), 2)

        predict2 = predict22[box_i][0]
        if( round(1-predict2, 2) >= threshold):
            boxs_pred.append( [  box[0]-15, box[1]-15, box[2]+15, box[3]+15, 10000*round(1-predict2, 4)  ])

            boxs_pred_value.append(round(1-predict2, 2))

            cv2.rectangle(imgC4, start_point , end_point , (0,255, 0), 2)
            cv2.putText(imgC4, str(round(1-predict2, 2)), start_point, cv2.FONT_HERSHEY_SIMPLEX, 1,  ( 0,255, 0), 2, cv2.LINE_AA, False)


    if(degub_view_op1):
        print("    - Result boxs (",len(boxs_pred),"):")

    if(degub_view_op2):
        plt.figure(figsize = size_figsize1)
        plt.imshow(imgC4 )
        plt.show()

    # Non-Maximum Suppression
    boxs_pred = non_max_suppression_fast(boxs_pred, 0.5)

    if(degub_view_op2):
        imgC4 = image.copy()

        key_ = img_file_in.split('/')[-1]
        for child in annotation_in[key]:
            if( child[4] == 'no-bacillus'):
                continue
            start_point = (int(child[0]), int(child[1]))
            end_point   = (int(child[2]), int(child[3]))
            cv2.rectangle(imgC4, start_point , end_point , (255, 0, 0), 3)

        for box in boxs_pred:
            start_point = (box[0], box[1])
            end_point = (box[2], box[3])
            cv2.rectangle(imgC4, start_point , end_point , (0,255, 0), 2)
            cv2.putText(imgC4, str(box[4]), start_point, cv2.FONT_HERSHEY_SIMPLEX, 1,  ( 0,255, 0), 2, cv2.LINE_AA, False)


        print("    - Result boxs (Non-Maximum Suppression:",len(boxs_pred),")")
        plt.figure(figsize = size_figsize1)
        plt.imshow(imgC4 )
        plt.show()


    if(degub_view_op1):
        print('    - Elapsed time = {}s '.format( round(time.time() - begin1,2) ))

    return np.array(boxs_pred)
    #=============================================================================
#-----------------------------------------------------------------------------------------
```

2.1. Detect test set

In [ ]:

```
all_begin = time.time()


degub_view_op1  = True
degub_view_op2  = False

#=============================================================================
#  Read CSV
#=============================================================================
print("* Read CSV")
annotation_in = {}
i = 0
#print("        - ", end = "")
with open(path_csv_annotation, 'r') as file:
    csvreader = csv.reader(file)
    header = next(csvreader)
    for row in csvreader:

        key = row[0]
        if( key not in annotation_in):
            annotation_in[key] = []
            #print("[",i,"] ", key, sep="", end = " / ")
            i +=1

        annotation_in[key].append(row[1:])

print("    - len:", len(annotation_in))
print("    - OK")
#=============================================================================


print("\n* Starting classification...\n")

model_load = load_model(modelNameSaved)

all_T      = []
all_P      = []
all_scores = []

key_idx = 0
for key in annotation_in:
```

```
begin = time.time()

if(key_idx > 10):
    degub_view_op1  = False
    degub_view_op2  = False

printProgressBar(key_idx + 1, len(annotation_in), prefix = 'Progress:', suffix = 'Complete',length = 100)


img_file_in = path_images+key
#================================================================================
# classification
#================================================================================
#-----------------------------------------------------
if(degub_view_op1):
    print("\n* Run classification...")
    print("    - ",key_idx,":",img_file_in)

#print(img_file_in)
boxs_pred_result = classification(img_file_in, K, model_load, Threshold, degub_view_op1, degub_view_op2).tolist()

for j in range(0,len(boxs_pred_result)):
    boxs_pred_result[j][4] = float(boxs_pred_result[j][4]/10000.0)
    boxs_pred_result[j].append(1)
#-----------------------------------------------------




#-----------------------------------------------------
image     = cv2.imread(img_file_in)
image_org = image.copy()
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
imgC4 = image.copy()
key_ = img_file_in.split('/')[-1]

gt = []
for child in annotation_in[key]:
    if( child[4] == 'no-bacillus'):
        continue
    start_point = (int(child[0]), int(child[1]))
    end_point   = (int(child[2]), int(child[3]))
    cv2.rectangle(imgC4, start_point , end_point , (255, 0, 0), 3)

    if( child[4] == 'bacillus'):
        gt.append([int(child[0]), int(child[1]),int(child[2]), int(child[3]), 1])
    if( child[4] == 'no-bacillus'):
        gt.append([int(child[0]), int(child[1]),int(child[2]), int(child[3]), 0])

#-----------------------------------------------------



T, P, iou, scores, P_index = get_map(boxs_pred_result, gt, IoU_map)


#-----------------------------------------------------
box_i = 0;
for box in boxs_pred_result:
    start_point = (box[0], box[1])
    end_point = (box[2], box[3])
    if(iou[box_i] >= IoU_map):
        cv2.rectangle(imgC4, start_point , end_point , (0,255, 0), 2)
        cv2.putText(imgC4, "["+str(box_i)+"] "+str(box[4]), start_point, cv2.FONT_HERSHEY_SIMPLEX, 1,  ( 0,255, 0), 1, cv2.LINE_AA, False)
    else:
        cv2.rectangle(imgC4, start_point , end_point , (0,0,255), 2)
        cv2.putText(imgC4, "["+str(box_i)+"] "+str(box[4]), start_point, cv2.FONT_HERSHEY_SIMPLEX, 1,  ( 0,0, 255), 1, cv2.LINE_AA, False)
    box_i += 1

if(degub_view_op1):
    print("\n    - Result of classification (",len(boxs_pred_result),"):")
    plt.figure(figsize = (14,14))
    plt.imshow(imgC4 )
    plt.show()
#-----------------------------------------------------



#-----------------------------------------------------
if(save_erro_image_path != "" ):

    if(degub_view_op1):
        begin2 = time.time()
        print("    - saving error images... ")
    num_save = 0

    for boxs_i in range(0, len(P_index)):

        if(P_index[boxs_i] != -1):
            if(T[boxs_i] == 0 and P[boxs_i] == 1):
                boxs_pred_result_i =  P_index[boxs_i]
                start_point_box = (boxs_pred_result[boxs_pred_result_i][0], boxs_pred_result[boxs_pred_result_i][1])
                end_point_box = (boxs_pred_result[boxs_pred_result_i][2], boxs_pred_result[boxs_pred_result_i][3])

                crop = image_org[start_point_box[1]:end_point_box[1], start_point_box[0]:end_point_box[0]]

                if( crop.shape[0] > 0 and crop.shape[1] > 0):
                    path_save = save_erro_image_path+"no-bacillus/"+str(num_save)+"_"+key_
                    #print(path_save)
                    cv2.imwrite( path_save , crop)
```

```
                    num_save += 1

        if(degub_view_op1):
            print('        - num_save:',num_save )
            print('        - Elapsed time = {}s \n'.format( round(time.time() - begin2,2) ))
    #-----------------------------------------------------


    #-----------------------------------------------------
    if(degub_view_op1):
        print("ytrue    :",T)
        print("predi    :",P)
        print("P_index  :",P_index)
        print("iou      :",iou)
        print("scores   :",scores)
        print("Metrics  :",scores)
        print('   - Precision : %.4f' % skm.precision_score(T, P))
        print('   - Recall    : %.4f' % skm.recall_score(T, P))
        print('   - F1_score  : %.4f' % skm.f1_score(T, P) )

        print('total time = {}s ...'.format( round(time.time() - begin,2) ))
        print("\n-----------------------------------------------------------------------------------------------\n")

    #accumulate results
    all_T      = all_T + T
    all_P      = all_P + P
    all_scores = all_scores + scores
    #-----------------------------------------------------

    key_idx = key_idx + 1
    #================================================================================
```

In [ ]:
```
print('* Total elapsed time = {}s ...'.format( round(time.time() - all_begin,2) ))
```

In [ ]:
```
cf_matrix        = skm.confusion_matrix(all_T, all_P)
ap_              = average_precision_score(all_T, all_scores)
accuracy_score_  = skm.accuracy_score(all_T, all_P)
precision_score_ = skm.precision_score(all_T, all_P)
recall_score_    = skm.recall_score(all_T, all_P)
f1_score_        = skm.f1_score(all_T, all_P)


FileSavedLog = OUT_PATH + FILE_ID_RUN +'-result.txt'
f = open(FileSavedLog, "a")
f.write("Result\n")


f.write('\n* Metrics:\n')

f.write('   - annotation: {} ...\n'.format( path_csv_annotation ))
f.write('   - model used: {} ...\n'.format( modelNameSaved ))
f.write('   - IoU_map   : '+'{:7.4f}'.format(IoU_map) +"\n")
f.write('   - Threshold : '+'{:7.4f}'.format(Threshold) +"\n")
f.write('   - k-means   : '+'{:7d}'.format(K) +"\n")
f.write('   - Precision : '+'{:7.4f}'.format(precision_score_) +"\n")
f.write('   - Recall    : '+'{:7.4f}'.format(recall_score_) +"\n")
f.write('   - F1_score  : '+'{:7.4f}'.format(f1_score_) +"\n")
f.write('   - AP        : '+'{:7.4f}'.format(AveragePrecisionScore(all_T, all_scores, False) ) +"\n")
f.write('   - Total time: {}s \n'.format( time.time() - all_begin ))
f.write('   - cf_matrix: \n')
f.write(str(cf_matrix))
f.write('')

f.close()

print("Saved file content:",FileSavedLog,"\n")
a_file = open(FileSavedLog)
file_contents = a_file.read()
print(file_contents)
a_file.close()



#Plot Confusion Matrix.
#https://www.stackvidhya.com/plot-confusion-matrix-in-python-and-why/
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts      = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}\n" for v1, v2 , v3 in zip(group_counts,group_percentages,group_names)]

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Confusion Matrix\n\n');
ax.set_xlabel('\nPredicted')
ax.set_ylabel('Actual');

ax.xaxis.set_ticklabels(['no-bacillus','bacillus'])
ax.yaxis.set_ticklabels(['no-bacillus','bacillus'])

plt.show()

AveragePrecisionScore(all_T, all_scores, True)
```

B.3 Treinamento e teste com diferentes métodos de detecção

Este apêndice disponibiliza o código-fonte implementado para treinar e testar os métodos de detecção (usando ou não a estrategia de partição), conforme descrito na Seção 5.4.

Detection tuberculosis bacilli from microscopic sputum smear images

This Jupyter Notebook was coded based on:

- IceVision (https://airctic.com/)
- https://medium.com/@yrodriguezmd/artificial-intelligence-can-help-feed-humans-9233f1c941f6
- https://medium.com/@yrodriguezmd/a-close-look-on-modelling-blood-cells-4625e832311f
- https://discord.com/channels/735877944085446747/780951884683935744
- and several other websites

1. Load libraries, functions and settings

In [ ]:
```
# ==========================================================================================================
# ==========================================================================================================
# Load libraries, functions and settings
# ==========================================================================================================
# ==========================================================================================================
print("\n* Loading libraries... ", end="")


import os
#os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "max_split_size_mb:25"
#os.environ['TF_FORCE_GPU_ALLOW_GROWTH']='true'
os.environ['CUDA_LAUNCH_BLOCKING']='1'


import cv2
import time
import gc
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import datetime
from copy import deepcopy

import icevision
from icevision.all import *
from icevision.models import *

from fastai.callback.tracker import SaveModelCallback

import sklearn.metrics as skm
from sklearn.metrics import average_precision_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

from IPython import display
%matplotlib inline
#%matplotlib notebook
from IPython.display import display # to display images
from IPython.display import Image

from mean_average_precision import MetricBuilder


import warnings
warnings.filterwarnings('ignore')

all_begin = time.time()
print("Ok")
# ==========================================================================================================
# ==========================================================================================================
```

1.1. Execution parameters and data

In [ ]:
```
# ==========================================================================================================
# ==========================================================================================================
# Execution parameters and data
# ==========================================================================================================
# ==========================================================================================================

learn_type  = "learn_frcnn"
#learn_type  = "learn_yolov5"
#learn_type  = "learn_fsaf"
#learn_type  = "learn_ssd"
#learn_type  = "learn_efficientdet"
#learn_type  = "learn_retinanet"
#learn_type  = "learn_yolof"

size_epochs  = 50
batch_size   = 5

path_root    = '/media/thales_mota/Novo volume/Google Drive/UFMG/thesis_source_codes/thesis'

k = "1"
image_folder_name        = '/media/thales_mota/Novo volume/data/dataset-4/splited_images/'
annotate_file_name_train = '/media/thales_mota/Novo volume/data/dataset-4/kfold_splited_images/'+k+'/train-positive--augmentation_splited.csv'
annotate_file_name_valid = '/media/thales_mota/Novo volume/data/dataset-4/kfold_splited_images/'+k+'/validation-positive--augmentation_splited.cs
annotate_file_name_test  = '/media/thales_mota/Novo volume/data/dataset-4/kfold_splited_images/'+k+'/test-positive--augmentation.csv'
dataset_name             = 'dataset4-splited-k'+k
```

In [ ]:
```
datetime_var         = time.strftime("%Y%m%d-%H%M%S")
id_model_name_save   = learn_type+"_"+dataset_name+"model_"+datetime_var
out_log_name         = learn_type+"_"+dataset_name+"_log_"+datetime_var

# start log file
file_log = open( path_root+'/'+out_log_name+".txt" , 'a')
file_log.write("# Teste com o "+dataset_name+"\n")

file_log.write("# "+time.strftime("%Y/%m/%d - %H:%M:%S")+"\n\n")
file_log.write("* path_root               : "+path_root+"\n")
file_log.write("* image_folder_name       : "+image_folder_name+"\n")
file_log.write("* annotate_file_name_train: "+annotate_file_name_train+"\n")
file_log.write("* annotate_file_name_test : "+annotate_file_name_test+"\n")
file_log.write("* id_model_name_save      : "+id_model_name_save+"\n")
file_log.write("* out_log_name            : "+out_log_name+"\n\n")

file_log.write("* batch_size : "+str(batch_size)+"\n")
file_log.write("* size_epochs: "+str(size_epochs)+"\n")
file_log.write("* learn_type : "+learn_type+"\n")
```

1.3. Useful and diverse functions

In [ ]:
```
# ----------------------------------------------------------------------------------------
def plot_metrics(learn, title):
  plt.plot(L(learn.recorder.values).itemgot())
  plt.xlabel('epoch')
  plt.ylabel('mAP (green), Loss (blue, orange)')
  plt.title(title)
  plt.text(0,-0.25, 'Legend: mAP(green), train_loss(blue), valid_loss(orange')
  plt.ylim(0,1.2);
  #display(plt)
  plt.show()

#-----------------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------
def non_max_suppression_fast(pred, overlapThresh):
    #https://pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/
    pred_ = deepcopy(pred)
    boxes=[]
    for i in range(len(pred_['detection']['bboxes'])):
        #print(pred['detection']['bboxes'][i])
        boxes.append([ pred_['detection']['bboxes'][i].xmin , pred_['detection']['bboxes'][i].ymin , pred_['detection']['bboxes'][i].xmax , pred_

    boxes = np.array(boxes)

    # if there are no boxes, return an empty list
    if len(boxes) == 0:
        return []
    # if the bounding boxes integers, convert them to floats --
    # this is important since we'll be doing a bunch of divisions
    if boxes.dtype.kind == "i":
        boxes = boxes.astype("float")
    # initialize the list of picked indexes
    pick = []
    # grab the coordinates of the bounding boxes
    x1 = boxes[:,0]
    y1 = boxes[:,1]
    x2 = boxes[:,2]
    y2 = boxes[:,3]
    # compute the area of the bounding boxes and sort the bounding
    # boxes by the bottom-right y-coordinate of the bounding box
    area = (x2 - x1 + 1) * (y2 - y1 + 1)
    idxs = np.argsort(y2)
    # keep looping while some indexes still remain in the indexes
    # list
    while len(idxs) > 0:
        # grab the last index in the indexes list and add the
        # index value to the list of picked indexes
        last = len(idxs) - 1
        i = idxs[last]
        pick.append(i)
        # find the largest (x, y) coordinates for the start of
        # the bounding box and the smallest (x, y) coordinates
        # for the end of the bounding box
        xx1 = np.maximum(x1[i], x1[idxs[:last]])
        yy1 = np.maximum(y1[i], y1[idxs[:last]])
        xx2 = np.minimum(x2[i], x2[idxs[:last]])
        yy2 = np.minimum(y2[i], y2[idxs[:last]])
        # compute the width and height of the bounding box
        w = np.maximum(0, xx2 - xx1 + 1)
        h = np.maximum(0, yy2 - yy1 + 1)
        # compute the ratio of overlap
        overlap = (w * h) / area[idxs[:last]]
        # delete all indexes from the index list that have
        idxs = np.delete(idxs, np.concatenate(([last], np.where(overlap > overlapThresh)[0])))
    # return only the bounding boxes that were picked using the
    # integer data type
    #print(pick)
    #print(pred)

    pred_['detection']['bboxes'] = [i for j, i in enumerate(pred_['detection']['bboxes']) if j in pick]
    pred_['detection']['labels'] = [i for j, i in enumerate(pred_['detection']['labels']) if j in pick]
    pred_['detection']['label_ids'] = [i for j, i in enumerate(pred_['detection']['label_ids']) if j in pick]
    pred_['detection']['scores'] = [i for j, i in enumerate(pred_['detection']['scores']) if j in pick]

    return pred_
    #return boxes[pick].astype("int")
```

```
#-----------------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------
def bb_intersection_over_union(boxA, boxB):
    #https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])
    # compute the area of intersection rectangle
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
    # compute the area of both the prediction and ground-truth
    # rectangles
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
    # compute the intersection over union by taking the intersection
    # area and dividing it by the sum of prediction + ground-truth
    # areas - the interesection area
    iou = interArea / float(boxAArea + boxBArea - interArea)
    # return the intersection over union value
    return iou
#-----------------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------
def get_map(pred, gt, iou_thresh):
    T = []
    P = []
    P_index = []
    iou = []
    scores = []
    iou_class = []
    gt_xmin_matched    = np.zeros(len(gt.detection.bboxes), dtype=bool)

    if( pred != []):
        pred_matched_quant = np.zeros(len(pred['detection']['bboxes']), dtype=int)

        for i in range(len(pred['detection']['bboxes'])):
                #BBox   = pred['detection']['bboxes'][i]
                pred_xmin   = pred['detection']['bboxes'][i].xmin
                pred_ymin   = pred['detection']['bboxes'][i].ymin
                pred_xmax   = pred['detection']['bboxes'][i].xmax
                pred_ymax   = pred['detection']['bboxes'][i].ymax
                pred_score  = pred['detection']['scores'][i]
                pred_class_ = pred['detection']['labels'][i]

                bbox_matched = False

                iou.append(0.0);
                iou_class.append(0);

                for j in range(len(gt.detection.bboxes)):
                    #BBox   = valid_records[index].detection.bboxes[i]
                    gt_xmin   = gt.detection.bboxes[j].xmin
                    gt_ymin   = gt.detection.bboxes[j].ymin
                    gt_xmax   = gt.detection.bboxes[j].xmax
                    gt_ymax   = gt.detection.bboxes[j].ymax
                    gt_score  = gt.detection.bboxes[j]
                    gt_class_ = gt.detection.labels[j]

                    iou_ = bb_intersection_over_union([gt_xmin,gt_ymin,gt_xmax,gt_ymax], [pred_xmin,pred_ymin,pred_xmax,pred_ymax])

                    if( iou_ >= iou_thresh    ):
                        if( iou_ >= iou[i] ):
                            iou[i] = iou_

                        iou_class[i]       = gt_class_
                        bbox_matched       = True
                        #gt_xmin_matched[j] = True
                        pred_matched_quant[i] = pred_matched_quant[i] + 1

                        if(bbox_matched and iou[i] >= iou_thresh and pred_class_ == iou_class[i]):
                            T.append(1)
                            P.append(1)
                            P_index.append(i)
                            gt_xmin_matched[j] = True

                        scores.append(pred_score)

                if(bbox_matched == False):
                    T.append(0)
                    P.append(1)
                    P_index.append(i)
                    scores.append(pred_score)

    for j in range(len(gt.detection.bboxes)):
        if( gt_xmin_matched[j] == False ):
            T.append(1)
            P.append(0)
            P_index.append(-1)
            scores.append(0.0)
            iou.append(0)


    return T, P, iou, scores, P_index
#-----------------------------------------------------------------------------------------

#-----------------------------------------------------------------------------------------
def AveragePrecisionScore(y_true_class, pred_scores_class, plot=False, thresholds = np.arange(start=0.2, stop=0.9, step=0.05)  ):
  # Based on: https://analyticsindiamag.com/how-does-mean-average-precision-map-work-in-object-detection/
```

```
    def precision_recall_curve(y_true, pred_scores, thresholds):
        precisions = []
        recalls = []

        for threshold in thresholds:
            y_pred = [1 if score >= threshold else 0 for score in pred_scores]

            precision = skm.precision_score(y_true=y_true, y_pred=y_pred, pos_label=1)
            recall = skm.recall_score(y_true=y_true, y_pred=y_pred, pos_label=1)

            precisions.append(precision)
            recalls.append(recall)

        return precisions, recalls


    precisions, recalls = precision_recall_curve(y_true=y_true_class, pred_scores=pred_scores_class, thresholds=thresholds)
    precisions.append(1)
    recalls.append(0)

    precisions = np.array(precisions)
    recalls = np.array(recalls)

    avg_precision_class = np.sum((recalls[:-1] - recalls[1:]) * precisions[:-1])

    if(plot):
        precisions_, recalls_ = precision_recall_curve(y_true=y_true_class, pred_scores=pred_scores_class, thresholds=thresholds)
        plt.plot(recalls_, precisions_, label=('Average precision score: '+str(np.round(avg_precision_class,4))), linewidth=2, color="red", zorde
        plt.xlabel("Recall", fontsize=12, fontweight='bold')
        plt.ylabel("Precision", fontsize=12, fontweight='bold')
        plt.title("Precision-Recall Curve", fontsize=15, fontweight="bold")
        plt.legend(loc="best")
        plt.show()

    return avg_precision_class
#-------------------------------------------------------------------------------------------


#-------------------------------------------------------------------------------------------
# Print iterations progress
def printProgressBar (iteration, total, prefix = '', suffix = '', decimals = 1, length = 100, fill = '', printEnd = "\r"):
    #https://splunktool.com/python-command-line-loading-bar
    """
    Call in a loop to create terminal progress bar
    @params:
        iteration   - Required  : current iteration (Int)
        total       - Required  : total iterations (Int)
        prefix      - Optional  : prefix string (Str)
        suffix      - Optional  : suffix string (Str)
        decimals    - Optional  : positive number of decimals in percent complete (Int)
        length      - Optional  : character length of bar (Int)
        fill        - Optional  : bar fill character (Str)
        printEnd    - Optional  : end character (e.g. "\r", "\r\n") (Str)
    """
    percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration / float(total)))
    filledLength = int(length * iteration // total)
    bar = fill * filledLength + '-' * (length - filledLength)
    print(f'\r{prefix} |{bar}| {percent}% {suffix}', end = printEnd)
    # Print New Line on Complete
    if iteration == total:
        print()
#-------------------------------------------------------------------------------------------
```

2. Load the data

2.1 Data exploration

In [ ]:
```
print("Train file information:\n",(annotate_file_name_train),"\n", sep ='')
df = pd.read_csv( annotate_file_name_train)

print("Number of Images : ", len(df.filename.unique().tolist() ) )
print("Number of objects: ", df.shape[0]  )
print("Class: ",df.label.unique().tolist() )

df.tail(5)
#df.info()
```

In [ ]:
```
print("Validation file information:\n",(annotate_file_name_valid),"\n", sep ='')
df = pd.read_csv( annotate_file_name_valid)

print("Number of Images : ", len(df.filename.unique().tolist() ) )
print("Number of objects: ", df.shape[0]  )
print("Class: ",df.label.unique().tolist() )

df.tail(5)
#df.info()
```

In [ ]:
```
print("Test file information:\n",(annotate_file_name_test),"\n", sep ='')
df = pd.read_csv( annotate_file_name_test)

print("Number of Images : ", len(df.filename.unique().tolist() ) )
print("Number of objects: ", df.shape[0]  )
print("Class: ",df.label.unique().tolist() )

df.tail(5)
#df.info()
```

In [ ]:
```
#Establish Directories
data_dir = Path(path_root)
class_map = ClassMap( df.label.unique().tolist()  )
```

2.2 Custom parse file in IceVision

In [ ]:
```
class TuberculoseParser(Parser):
    def __init__(self, annotate_file, template_record, data_dir):
        super().__init__(template_record=template_record)

        #self.data_dir = path_data
        self.df = pd.read_csv(  annotate_file)
        self.class_map = ClassMap(list(self.df['label'].unique()))

    def __iter__(self) -> Any:
        for o in self.df.itertuples():
            yield o

    def __len__(self) -> int:
        return len(self.df)

    def record_id(self, o) -> Hashable:
        return o.filename

    def parse_fields(self, o, record, is_new):
        if is_new:
            #print( str(image_folder_name +"/"+ image_folder_name +"/"+ o.filename) )
            #img = cv2.imread( str(self.data_dir +"/"+ image_folder_name +"/"+ o.filename) , 0)
            #print( str(image_folder_name +"/"+ o.filename) )
            img = cv2.imread( str(image_folder_name +"/"+ o.filename) , 0)
            height = img.shape[0]
            width = img.shape[1]
            #print( str(self.data_dir +"/"+ image_folder_name +"/"+ o.filename) ,  height,  width)

            record.set_filepath(image_folder_name +"/"+ o.filename)
            #record.set_filepath(self.data_dir +"/"+ image_folder_name +"/"+ o.filename)
            record.set_img_size(ImgSize(width=width, height=height))
            record.detection.set_class_map(self.class_map)

        record.detection.add_bboxes([BBox.from_xyxy(o.xmin, o.ymin, o.xmax, o.ymax)])
        record.detection.add_labels([o.label])
```

2.3 Read datasets

In [ ]:
```
template_record = ObjectDetectionRecord()
annotate_file_name_train
template_record
data_dir
parser_train = TuberculoseParser(annotate_file_name_train, template_record, data_dir)
parser_valid = TuberculoseParser(annotate_file_name_valid, template_record, data_dir)
parser_test  = TuberculoseParser(annotate_file_name_test, template_record, data_dir)

train_records, tr_tmp = parser_train.parse(data_splitter=RandomSplitter([1, 0]))
valid_records, vr_tmp = parser_valid.parse(data_splitter=RandomSplitter([1, 0]))
test_records, ter_tmp  = parser_test.parse(data_splitter=RandomSplitter([1, 0]))
```

2.4 View images

In [ ]:
```
#print(train_records[0].detection.bboxes)

sum_train_records_bboxes=0;
for i in range(len(train_records)):
    sum_train_records_bboxes = sum_train_records_bboxes + len(train_records[i].detection.bboxes)

sum_valid_records_bboxes=0;
for i in range(len(valid_records)):
    sum_valid_records_bboxes = sum_valid_records_bboxes + len(valid_records[i].detection.bboxes)

sum_test_records_bboxes=0;
for i in range(len(test_records)):
    sum_test_records_bboxes = sum_test_records_bboxes + len(test_records[i].detection.bboxes)


print("train_records = ", len(train_records)," / bboxes = ",sum_train_records_bboxes)
print("valid_records = ", len(valid_records)," / bboxes = ",sum_valid_records_bboxes)
print("test_records  = ", len(test_records)," / bboxes = ",sum_test_records_bboxes)
print("total records = ", len(test_records)+ len(valid_records) +len(train_records)," / bboxes = ",sum_train_records_bboxes+sum_valid_records_bbo


file_log.write("\n")
file_log.write("* size_train_records: "+ str(len(train_records)) + " / bboxes = " + str(sum_train_records_bboxes)+"\n")
file_log.write("* size_valid_records: "+ str(len(valid_records)) + " / bboxes = " + str(sum_valid_records_bboxes)+"\n")
file_log.write("* size_test_records : "+ str(len(test_records))  + " / bboxes = " + str(sum_test_records_bboxes)+"\n")
file_log.write("* size_total_records: "+ str(len(test_records)+ len(valid_records) +len(train_records))  + " / bboxes = " + str(sum_train_records
```

```
show_records(train_records[:3], ncols=3, class_map=class_map, figsize=(15, 15))
show_records(valid_records[:3], ncols=3, class_map=class_map, figsize=(15, 15))
show_records(test_records[:3] , ncols=3, class_map=class_map, figsize=(15, 15))
```

2.5 Transforms data

In [ ]:
```
presize    = 512
image_size = 384

train_tfms = tfms.A.Adapter([*tfms.A.aug_tfms(size=image_size, presize=presize), tfms.A.Normalize()])
valid_tfms = tfms.A.Adapter([*tfms.A.resize_and_pad(size=image_size), tfms.A.Normalize()])

train_ds = Dataset(train_records, train_tfms)
valid_ds = Dataset(valid_records, valid_tfms)

#samples = [train_ds[6] for _ in range(3)]
#show_samples(samples, ncols=3, display_label=False)
```

4. Generate model

In [ ]:
```
# Model
#metrics = [COCOMetric(metric_type=COCOMetricType.bbox, print_summary=true)]
metrics = [COCOMetric(metric_type=COCOMetricType.bbox)]

method_chosen        = None
model_type_method    = None
backbone_method      = None
model_method         = None
learn_method         = None
model_name_method    = None
backbone_name_method = None

import gc
torch.cuda.empty_cache()
gc.collect()
```

4.1 Faster R-CNN

In [ ]:
```
if(learn_type == "learn_frcnn"):
    method_chosen = 'FRCNN'
    ########################################################################
    # Prepare
    ########################################################################
    model_type_method = models.torchvision.faster_rcnn
    backbone_method   = model_type_method.backbones.resnet50_fpn(pretrained=True)
    #backbone_method = model_type_method.backbones.resnet101_fpn_2x
    model_method      = model_type_method.model( backbone=backbone_method,  num_classes= len(parser_train.class_map))

    model_name_method    = 'models.torchvision.faster_rcnn'
    backbone_name_method = 'resnet50_fpn'

    train_dl = model_type_method.train_dl( train_ds, batch_size=batch_size, num_workers=4, shuffle=True)
    valid_dl = model_type_method.valid_dl( valid_ds, batch_size=batch_size, num_workers=4, shuffle=False)

    learn_method = model_type_method.fastai.learner( dls=[train_dl, valid_dl],  model=model_method, metrics=metrics)

    lr = learn_method.lr_find()
    print("suggestion valley:", lr.valley)
    ########################################################################
```

3.2 YOLOv5

In [ ]:
```
if(learn_type == "learn_yolov5"):
    method_chosen = 'YoloV5'
    ########################################################################
    # Prepare
    ########################################################################
    model_type_method = models.ultralytics.yolov5
    backbone_method = model_type_method.backbones.small
    model_method = model_type_method.model(backbone = backbone_method(pretrained=True), num_classes=len(parser_train.class_map), img_size = image

    model_name_method    = 'models.ultralytics.yolov5'
    backbone_name_method = 'small'

    train_dl = model_type_method.train_dl(train_ds, batch_size=batch_size, num_workers=4, shuffle=True)
    valid_dl = model_type_method.valid_dl(valid_ds, batch_size=batch_size, num_workers=4, shuffle=False)

    learn_method = model_type_method.fastai.learner( dls=[train_dl, valid_dl],  model=model_method, metrics=metrics)

    lr = learn_method.lr_find()
    print("suggestion valley:", lr.valley)
    ########################################################################
```

3.3 VarifocalNet

In [ ]:
```
if(learn_type == "learn_VFNet"):
    method_chosen = 'VarifocalNet'
    ########################################################################
    # Prepare
    ########################################################################
    model_type_method = models.mmdet.vfnet
```

```
    backbone_method = model_type_method.backbones.resnet50_fpn_1x
    #backbone_method = model_type_method.backbones.resnet50_fpn_mstrain_2x
    model_method = model_type_method.model(backbone = backbone_method(pretrained=True), num_classes=len(parser_train.class_map) )

    model_name_method    = 'models.mmdet.vfnet'
    backbone_name_method = 'resnet50_fpn_1x'

    train_dl = model_type_method.train_dl(train_ds, batch_size=batch_size, num_workers=4, shuffle=True)
    valid_dl = model_type_method.valid_dl(valid_ds, batch_size=batch_size, num_workers=4, shuffle=False)

    learn_method = model_type_method.fastai.learner( dls=[train_dl, valid_dl],  model=model_method, metrics=metrics)

    learn_method.lr_find()
    ########################################################################
```

3.4 SSD

In [ ]:
```
if(learn_type == "learn_ssd"):
    method_chosen = 'SSD'
    ########################################################################
    # Prepare
    ########################################################################
    model_type_method    = models.mmdet.ssd
    backbone_method      = model_type_method.backbones.ssd300
    model_name_method    = 'mmdet.ssd'
    backbone_name_method = 'ssd300'

    model_method = model_type_method.model( backbone=backbone_method(pretrained=True),  num_classes= len(parser_train.class_map))

    train_dl = model_type_method.train_dl( train_ds, batch_size=batch_size, num_workers=4, shuffle=True)
    valid_dl = model_type_method.valid_dl( valid_ds, batch_size=batch_size, num_workers=4, shuffle=False)

    learn_method = model_type_method.fastai.learner( dls=[train_dl, valid_dl],  model=model_method, metrics=metrics)

    lr = learn_method.lr_find()
    print("suggestion valley:", lr.valley)
    ########################################################################
```

3.5 EfficientDet

In [ ]:
```
if(learn_type == "learn_efficientdet"):
    method_chosen = 'EfficientDet'
    ########################################################################
    # Prepare
    ########################################################################
    model_type_method = models.ross.efficientdet
    backbone_method = model_type_method.backbones.tf_lite0
    model_method = model_type_method.model(backbone = backbone_method(pretrained=True), num_classes=len(parser_train.class_map), img_size = image

    model_name_method    = 'ross.efficientdet'
    backbone_name_method = 'tf_lite0'

    train_dl = model_type_method.train_dl(train_ds, batch_size=batch_size, num_workers=4, shuffle=True)
    valid_dl = model_type_method.valid_dl(valid_ds, batch_size=batch_size, num_workers=4, shuffle=False)

    learn_method = model_type_method.fastai.learner( dls=[train_dl, valid_dl],  model=model_method, metrics=metrics)

    lr = learn_method.lr_find()
    print("suggestion valley:", lr.valley)
    ########################################################################
```

3.6 RetinaNet

In [ ]:
```
if(learn_type == "learn_retinanet"):
    method_chosen = 'RetinaNet'
    ########################################################################
    # Prepare
    ########################################################################
    model_type_method = models.mmdet.retinanet
    backbone_method   = model_type_method.backbones.resnet50_fpn_1x
    model_method      = model_type_method.model( backbone=backbone_method(pretrained=True),  num_classes= len(parser_train.class_map))

    model_name_method    = 'mmdet.retinanet'
    backbone_name_method = 'backbones.resnet50_fpn_1x'

    train_dl = model_type_method.train_dl( train_ds, batch_size=batch_size, num_workers=4, shuffle=True)
    valid_dl = model_type_method.valid_dl( valid_ds, batch_size=batch_size, num_workers=4, shuffle=False)

    learn_method = model_type_method.fastai.learner( dls=[train_dl, valid_dl],  model=model_method, metrics=metrics)

    lr = learn_method.lr_find()
    print("suggestion valley:", lr.valley)
    ########################################################################
```

3.8 YoloF

In [ ]:
```
if(learn_type == "learn_yolof"):
    method_chosen = 'YoloF'
    ########################################################################
    # Prepare
    ########################################################################
    model_type_method = models.mmdet.yolof
    backbone_method = model_type_method.backbones.yolof_r50_c5_8x8_1x_coco
    model_method      = model_type_method.model( backbone=backbone_method(pretrained=True),  num_classes= len(parser_train.class_map))
```

```
    model_name_method    = 'mmdet.yolof'
    backbone_name_method = 'yolof_r50_c5_8x8_1x_coco'

    train_dl = model_type_method.train_dl( train_ds, batch_size=batch_size, num_workers=4, shuffle=True)
    valid_dl = model_type_method.valid_dl( valid_ds, batch_size=batch_size, num_workers=4, shuffle=False)

    learn_method = model_type_method.fastai.learner( dls=[train_dl, valid_dl],  model=model_method, metrics=metrics)

    lr = learn_method.lr_find()
    print("suggestion valley:", lr.valley)
    ######################################################################
```

FSAF

In [ ]:
```
if(learn_type == "learn_fsaf"):
    method_chosen = 'FSAF'
    ######################################################################
    # Prepare
    ######################################################################
    model_type_method = models.mmdet.fsaf
    backbone_method   = model_type_method.backbones.x101_64x4d_fpn_1x_coco
    model_method      = model_type_method.model( backbone=backbone_method(pretrained=True),  num_classes= len(parser_train.class_map))

    model_name_method    = 'mmdet.fsaf'
    backbone_name_method = 'x101_64x4d_fpn_1x_coco'

    train_dl = model_type_method.train_dl( train_ds, batch_size=batch_size, num_workers=4, shuffle=True)
    valid_dl = model_type_method.valid_dl( valid_ds, batch_size=batch_size, num_workers=4, shuffle=False)

    learn_method = model_type_method.fastai.learner( dls=[train_dl, valid_dl],  model=model_method, metrics=metrics)

    lr = learn_method.lr_find()
    print("suggestion valley:", lr.valley)
    ######################################################################
```

In [ ]:
```
torch.cuda.empty_cache()
gc.collect()
```

---

5. Training model

In [ ]:
```
begin = time.time()

# Train
fname = id_model_name_save
learn_method.fine_tune(size_epochs,  lr.valley, freeze_epochs=1, cbs=SaveModelCallback(monitor='COCOMetric', fname=fname))

plot_metrics(learn_method, 'Mean Average Precision and Losses for '+ fname )
print(fname)


# Save the last generated model
checkpoint_path = path_root+'/models/'+fname+'_end.pth'
save_icevision_checkpoint(learn_method,
                          model_name= model_name_method,
                          backbone_name=backbone_name_method,
                          classes =  parser_train.class_map.get_classes(),
                          img_size=image_size,
                          filename=checkpoint_path)
print("Save:",checkpoint_path)



model_type_generated = model_type_method
model_generated      = model_method
print('Completely Elapsed time = {}'.format(time.time() - begin))

file_log.write("\n")
file_log.write("* model_name: "+model_name_method+"\n")
file_log.write("* backbone_name: "+backbone_name_method+"\n")
file_log.write("* time_train (s): "+str(time.time() - begin)+" ("+str(datetime.timedelta(seconds=(time.time() - begin)))+")\n")
file_log.write("* learn.validate:\n"+str(learn_method.validate())+"\n")
```

In [ ]:
```
torch.cuda.empty_cache()
gc.collect()
```

---

6. Evaluating the test set

In [ ]:
```
test_tfms = tfms.A.Adapter([*tfms.A.resize_and_pad(image_size), tfms.A.Normalize()])

iou_thresh       = 0.1 #0.5
detection_thresh = 0.5
debug_level_1  = True
debug_level_2  = True

use_split        = True
factor           = 2000

all_mAP_VOC         = []
all_accuracy_score  = []
all_precision_score = []
all_recall_score    = []
```

```
all_f1_score        = []

all_T      = []
all_P      = []
all_scores = []


records_in = test_records

begin = time.time()
printProgressBar(0, len(records_in), prefix = 'Progress:', suffix = 'Complete', length = 100)

for index in range(len(records_in)):

    if(index > 20):
        debug_level_1  = False
        debug_level_2  = False

    printProgressBar(index + 1, len(records_in), prefix = 'Progress:', suffix = 'Complete',length = 100)

    gt  = records_in[index]

    if( len(gt.detection.bboxes) ==  0):
        print(len(gt.detection.bboxes),"bbox")
        print()
        continue

    img_path = gt.filepath

    img = PIL.Image.open(img_path).convert("RGB")
    draw = PIL.ImageDraw.Draw(img)


    #-----------------------------------------------------------------------------------------------
    #Split image am predict
    img_cv2 = cv2.imread( str(img_path) )
    height, width, channels = img_cv2.shape

    #rgb_img = cv2.cvtColor(img_cv2, cv2.COLOR_BGR2RGB)
    #plt.imshow(rgb_img)
    #plt.show()


    pred_all = {}
    pred_all['detection']={}
    pred_all['detection']['labels'] =  []
    pred_all['detection']['label_ids'] =  []
    pred_all['detection']['scores'] =  ()
    pred_all['detection']['bboxes'] =   []



    num_cut_col  = num_cut_line = int(  math.ceil( (width if width > height else height) / factor) )

    if(num_cut_col == 0 or use_split == False):
        pred = model_type_generated.end2end_detect(img_path, test_tfms, model_generated, class_map=parser_test.class_map, detection_threshold=det
        pred = non_max_suppression_fast(pred, 0.1)

    else:
        for i in range(0, num_cut_col):
            for j in range(0, num_cut_line):

                #Calculates the positions to crop the new image
                x1 = int( i * width/num_cut_col )
                y1 = int( j * height/num_cut_line )
                x2 = int( (i+1) * width/num_cut_col )
                y2 = int( (j+1) * height/num_cut_line )

                if( x1 > 0 ):
                    x1 = int( x1 - width * 0.05  )

                if(  y1 > 0):
                    y1 = int( y1 - height * 0.05 )

                if( x2 < width ):
                    x2 = int( x2 + width * 0.05  )

                if( y2 < height):
                    y2 = int( y2 + height * 0.05 )

                cropped_image = img_cv2.copy()[y1:y2, x1:x2]

                #rgb_img = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB)
                #plt.imshow(rgb_img)
                #plt.show()

                temp_filename = "/tmp/tmp_icevisionImage"+ os.path.splitext( img_path )[-1]
                cv2.imwrite( temp_filename, cropped_image)
                pred = model_type_generated.end2end_detect(temp_filename, test_tfms, model_generated, class_map=parser_test.class_map, detection_

                for d in range(len(pred['detection']['bboxes'])):
                    pred['detection']['bboxes'][d].xmin   = int(pred['detection']['bboxes'][d].xmin  + x1  )
                    pred['detection']['bboxes'][d].ymin   = int(pred['detection']['bboxes'][d].ymin + y1  )
                    pred['detection']['bboxes'][d].xmax   = int(pred['detection']['bboxes'][d].xmax  + x1 )
                    pred['detection']['bboxes'][d].ymax   = int(pred['detection']['bboxes'][d].ymax  + y1   )

                pred_all['detection']['labels'] = pred_all['detection']['labels']  + pred['detection']['labels']
                pred_all['detection']['label_ids'] = pred_all['detection']['label_ids']  + pred['detection']['label_ids']
                pred_all['detection']['scores'] = np.concatenate( (pred_all['detection']['scores'] , pred['detection']['scores']))
                pred_all['detection']['bboxes'] = pred_all['detection']['bboxes']  + pred['detection']['bboxes']
```

```
            draw.rectangle(((x1, y1), (x2, y2)), fill=None, outline="yellow", width=2)

        pred = pred_all
        pred = non_max_suppression_fast(pred, 0.1)
    #-------------------------------------------------------------------------------------------------



    T, P, iou, scores, P_index = get_map(pred, gt, iou_thresh)

    all_T      = all_T + T
    all_P      = all_P + P
    all_scores = all_scores + scores


    #font = PIL.ImageFont.truetype(r"./esparac.ttf", 20)
    font = PIL.ImageFont.truetype("/usr/share/fonts/truetype/freefont/FreeMono.ttf", 20)

    #draw original box
    for i in range(len(gt.detection.bboxes)):
            BBox    = gt.detection.bboxes[i]
            xmin    = gt.detection.bboxes[i].xmin
            ymin    = gt.detection.bboxes[i].ymin
            xmax    = gt.detection.bboxes[i].xmax
            ymax    = gt.detection.bboxes[i].ymax
            class_ = ''+str(i)

            draw.rectangle(((xmin, ymin), (xmax, ymax)), fill=None, outline="green", width=4)
            draw.text((xmin, ymax+5),f"{class_}",(0,0,0), font=font)

    #draw detect box
    if(pred != []):
        for i in range(len(pred['detection']['bboxes'])):
                BBox    = pred['detection']['bboxes'][i]
                xmin    = pred['detection']['bboxes'][i].xmin
                ymin    = pred['detection']['bboxes'][i].ymin
                xmax    = pred['detection']['bboxes'][i].xmax
                ymax    = pred['detection']['bboxes'][i].ymax
                score   = pred['detection']['scores'][i]
                class_ = pred['detection']['labels'][i]

                if(iou[i] >= iou_thresh):
                    draw.rectangle(((xmin, ymin), (xmax, ymax)), fill=None, outline="red", width=2)
                else:
                    draw.rectangle(((xmin, ymin), (xmax, ymax)), fill=None, outline="black", width=2)

                draw.text((xmin, ymin-20),f"{class_} | {score:.3f} | {iou[i]:.3f}",(0,0,0), font=font)

    if(debug_level_2):
        print('\n\n\n')
        print('File     :',img_path, " [",index,'/',len(records_in),"]")

    if(debug_level_1):
        print(len(gt.detection.bboxes),"bbox")
        print("ytrue    :",T)
        print("predi    :",P)
        print("iou      :",iou)
        print("scores   :",scores)

        basewidth = 800
        wpercent = (basewidth/float(img.size[0]))
        hsize = int((float(img.size[1])*float(wpercent)))
        img = img.resize((basewidth,hsize), PIL.Image.ANTIALIAS)
        display(img)

    accuracy_score  = skm.accuracy_score(T, np.around(P, decimals=0))
    precision_score = skm.precision_score(T, np.around(P, decimals=0), zero_division =0)
    recall_score    = skm.recall_score(T, np.around(P, decimals=0), zero_division =0)
    f1_score        =  skm.f1_score(T, np.around(P, decimals=0))

    if(debug_level_2):
        #print('Accuracy : %.4f' % accuracy_score)
        print('Precision      : %.4f' % precision_score)
        print('Recall         : %.4f' % recall_score)
        print('F1_score       : %.4f' % f1_score )
        print('AP             : %.4f' % np.round(AveragePrecisionScore(T, scores, False),4))
        print('')

    all_accuracy_score.append(accuracy_score)
    all_precision_score.append(precision_score)
    all_recall_score.append(recall_score)
    all_f1_score.append(f1_score)


print('\n\nCompletely Elapsed time = {}'.format(time.time() - begin))
```

In [ ]:

```
cf_matrix        = skm.confusion_matrix(all_T, all_P)
accuracy_score_  = skm.accuracy_score(all_T, all_P)
precision_score_ = skm.precision_score(all_T, all_P)
recall_score_    = skm.recall_score(all_T, all_P)
f1_score_        = skm.f1_score(all_T, all_P)

print('\n* Metrics calculated considering all hits and errors made in all images')
print('   - IoU@'+str(iou_thresh) )
print('   - Threshold   : %.4f' % (detection_thresh)  )
print('   - Precision   : %.4f' % precision_score_)
print('   - Recall      : %.4f' % recall_score_)
print('   - F1_score    : %.4f' % f1_score_ )
```

```
print('   - AP          : %.4f' % np.round(AveragePrecisionScore(all_T, all_scores,True),4) )
print('   - cf_matrix: ')
print(str(cf_matrix))
print('')


#Plot Confusion Matrix.
#https://www.stackvidhya.com/plot-confusion-matrix-in-python-and-why/
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts      = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}\n" for v1, v2 , v3 in zip(group_counts,group_percentages,group_names)]

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Confusion Matrix\n\n');
ax.set_xlabel('\nPredicted')
ax.set_ylabel('Actual');

ax.xaxis.set_ticklabels(['no-bacillus','bacillus'])
ax.yaxis.set_ticklabels(['no-bacillus','bacillus'])

plt.show()
figure = ax.get_figure()
figure.savefig( path_root+'/'+out_log_name+".png" , dpi=400)
```

In [ ]:
```
file_log.write("\n")


file_log.write("\n")
file_log.write("* Evaluating the test set:\n")
file_log.write("# (Metrics calculated considering all hits and errors made in all images)\n")
file_log.write("* IoU@: "+str(iou_thresh)+"\n")
file_log.write("* detection threshold: "+str(detection_thresh)+"\n")
if( use_split == True):
    file_log.write("* factor split: "+str(factor)+"\n")
file_log.write("* AP: "+str( np.round(AveragePrecisionScore(all_T, all_scores,False),4) )+"\n")
file_log.write("* precision: "+str( skm.precision_score(all_T, all_P) )+"\n")
file_log.write("* recall: "+str( skm.recall_score(all_T, all_P) )+"\n")
file_log.write("* f1 score: "+str( skm.f1_score(all_T, all_P) )+"\n")
file_log.write("* cf_matrix \n")
file_log.write(str(cf_matrix)+"\n")
file_log.write("\n")

file_log.write("# End: "+time.strftime("%Y/%m/%d - %H:%M:%S")+"\n\n")

file_log.close()

print("Log file:")
print(path_root+'/'+out_log_name+".txt","\n")
with open(path_root+'/'+out_log_name+".txt", 'r') as f:
    print(f.read())
```

Installed packages

In [ ]:
```
!pip freeze
```

In [ ]:

In [ ]: