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

Leonam Rezende Soares de Miranda

Aprendizado de Operadores de Agregação do Tipo Média Ponderada Ordenada em Redes Neurais Convolucionais

Belo Horizonte

2023

Leonam Rezende Soares de Miranda

Aprendizado de Operadores de Agregação do Tipo Média Ponderada Ordenada em Redes Neurais Convolucionais

Dissertação de Mestrado 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 Título de Mestre em Engenharia Elétrica.

Orientador: Frederico Gadelha Guimarães

Belo Horizonte
2023

Miranda, Leonam Rezende Soares de.
M672a Aprendizado de operadores de agregação do tipo média ponderada ordenada em redes neurais convolucionais [recurso eletrônico] / Leonam Rezende Soares de Miranda. - 2023.
1 recurso online (68 f. : il., color.) : pdf.

Orientador: Frederico Gadelha Guimarães.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Bibliografia: f. 63-68.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia elétrica - Teses. 2. Redes neurais convolucionais - Teses. 3. Aprendizado profundo - Teses. 4. Imagens - Teses. I. Guimarães, Frederico Gadelha. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 621.3(043)

Ficha catalográfica elaborada pela bibliotecária Ângela Cristina Silva CRB/6 2361
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

"APRENDIZADO DE OPERADORES DE AGREGAÇÃO DO TIPO MÉDIA PONDERADA ORDENADOS EM REDES NEURAIS CONVOLUCIONAIS"

LEONAM REZENDE SOARES DE MIRANDA

Dissertação de Mestrado 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 Mestre em Engenharia Elétrica. Aprovada em 15 de fevereiro de 2023. Por:

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

Prof. Dr. Janier Arias García
DELT (UFMG)

Prof. Dr. Cristiano Leite de Castro
DEE (UFMG)

Documento assinado eletronicamente por **Frederico Gadelha Guimaraes**, **Coordenador(a) de curso de pós-graduação**, em 16/02/2023, às 12:38, 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 **Janier Arias Garcia**, **Professor do Magistério Superior**, em 16/02/2023, às 15:00, 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 **Cristiano Leite de Castro**, **Professor do Magistério Superior**, em 23/02/2023, às 11:50, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A Deus, que me guiou nesta jornada e me deu forças para continuar. A meu pai e minha mãe, por serem as luzes da minha vida e por sempre acreditarem em mim.

Agradecimentos

Agradeço a Deus por me proporcionar as forças, sabedoria e coragem necessárias para completar este trabalho. A minha família foi fundamental nesta jornada, especialmente meu pai, por todo o seu amor, apoio e orientação, sem ele eu não teria chegado até aqui. A minha mãe, por todo o seu amor e carinho, por me ajudar a ultrapassar os obstáculos e por sempre acreditar em mim. Aos meus irmãos, por toda a diversão, apoio incondicional, por estarem sempre presentes em minha vida e serem as melhores pessoas com quem eu possa contar.

Agradeço ao meu orientador, Prof. Frederico Gadelha Guimarães, por toda orientação e suporte durante este processo, desde a definição da proposta até a defesa da dissertação. Também gostaria de agradecer a toda equipe do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais pela oportunidade de desenvolver o trabalho e a CAPES e CNPq pelo apoio financeiro. Este trabalho não teria sido possível sem a ajuda de todos vocês.

Quanto mais valorizamos as coisas fora do nosso controle, menos controle temos.

(Marco Aurélio)

Resumo

Nas redes neurais convolucionais são realizadas operações de agregação nas camadas de convolução, pooling e nas densas completamente conectadas. Resultados promissores foram obtidos nos últimos anos ao utilizar operadores de agregação do tipo media ponderada ordenada, mais conhecidos como operadores OWA, para agregar dados dentro das redes neurais convolucionais. Há trabalhos recentes demonstrando que há um ganho de performance significativo ao utilizar os operadores OWA, treinando os seus pesos, para realizar a operação de pooling, quando comparado com os operadores mais usuais (máximo e médio). Outros estudos demonstraram que os operadores OWA podem ser utilizados para aprender informações a partir do ordenamento dos canais de uma determinada camada, e as informações recém-geradas são usadas para complementar ou substituir os dados de entrada para a camada seguinte. O objetivo desta dissertação é analisar e combinar as duas ideias mencionadas. Vários testes foram feitos para avaliar a mudança de desempenho ao aplicar operadores OWA para classificar imagens, usando os modelos VGG13, Network in Network e AlexNet e os conjuntos de dados CIFAR-10 e CIFAR-100.

Palavras-chave: Redes Neurais Convolucionais, Operadores OWA, Aprendizado Profundo, Funções de Agregação, Classificação de Imagens.

Abstract

In convolutional neural networks, aggregation operations are performed in the convolution, pooling and fully connected dense layers. Promising results have been obtained in recent years when using ordered weighted averaging operators, better known as OWA operators, to aggregate data within convolutional neural networks. There are recent works demonstrating that there is a performance gain when using OWA operators, training their weights, to perform the pooling operation, when compared with the most usual operators (maximum and average). Other studies have shown that OWA operators can be used to learn additional order-based information from the feature maps of a certain layer, and the newly generated information is used to complement or replace the input data for the next layer. The purpose of this dissertation is to analyze and combine the two mentioned ideas. Several tests were done to evaluate the performance change when applying OWA operators to classify images, using the VGG13, Network in Network and AlexNet models and the CIFAR-10 and CIFAR-100 datasets.

Keywords: Convolutional Neural Networks, OWA operator, Deep Learning, Aggregation Functions, Image Classification.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

RNC	Rede Neural Convolucional
RNA	Redes Neurais Artificial
OWA	Ordered Weighted Averaging
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
RGB	Red, Green and Blue
VGG	Visual Geometry Group
NiN	Network in Network
CIFAR	Canadian Institute for Advanced Research
SVM	Support Vector Machine
k-NN	k-nearest neighbors
RNB	Rede Neural Biológica
MLP	Multi Layer Perceptron
ReLU	Rectified Linear Unit
ELU	Exponential Linear Unit
MNIST	Modified National Institute of Standards and Technology
ResNet	Residual Neural Network
BS	Base Model
OWAPL	OWA Pooling
OWACA	OWA Channel Aggregation

Sumário

Capítulo 1

Introdução

1.1 Contextualização

A classificação de imagens é um dos principais problemas na área de visão computacional e reconhecimento de padrões, sendo amplamente estudado pela comunidade científica nos últimos anos (Nath et al. [2014]). Usualmente, para resolver um problema de classificação de imagens, é treinado um modelo de aprendizado de máquina com dados rotulados para classificar corretamente os dados não rotulados. As principais dificuldades enfrentadas nesta área incluem identificar padrões em imagens, distinguir entre seres vivos e objetos, e rotular imagens coletadas. A maioria desses problemas envolve informações complexas que precisam ser identificadas, e métodos tradicionais como redes neurais convencionais, em que os neurônios de uma camada estão totalmente conectados aos neurônios da camada anterior, têm se mostrado ineficientes.

Apesar de Cybenko [1989a] ter demonstrado que é possível aproximar qualquer função com uma rede neural com uma camada escondida, o uso dos pixels de uma imagem, sem nenhum pré-processamento, como entrada em uma rede neural convencional não é uma abordagem eficiente. Isso ocorre porque um modelo de rede totalmente conectado trata todos os pixels de entrada de forma igual, ignorando a relação espacial entre eles. Isso resulta em uma alta conectividade na rede, tornando a dimensão do espaço de ajuste de parâmetros alta, o que é conhecido como a maldição da dimensionalidade. (Goodfellow et al. [2016]). Como resultado, torna-se difícil realizar o ajuste dos parâmetros do modelo durante o treinamento e facilmente se chega a soluções com baixa generalização (overfitting). Essa problemática tem sido mitigada atualmente através do uso de Redes Neurais Convolucionais (RNCs) que se valem do princípio da hierarquia de características e da existência de padrões locais e espaciais nas imagens.

Na década de 2010, ocorreu uma mudança significativa na comunidade científica quando Krizhevsky et al. [2012] venceu o desafio de classificação de imagens ILSVRC

2012[1] utilizando um modelo de Rede Neural Convolucional (RNC) profundo, chamado de AlexNet. Apesar de a técnica de RNC já existir há algum tempo, o uso dessa abordagem era inédito na competição, e os resultados obtidos foram consideravelmente superiores em comparação com as abordagens utilizadas anteriormente. As RNCs são redes neurais projetadas para trabalhar com dados espaciais, como imagens e vídeos, onde qualquer parte da informação de entrada é fortemente correlacionada com outras partes próximas, que formam a vizinhança. Essas redes utilizam operações convolucionais em camadas sobre as imagens, que normalmente se encontram organizadas em três canais em escala RGB, para extrair características. Quanto mais profunda a camada, mais complexa é a característica detectada. As operações convolucionais impõem restrições de conectividade local nos pesos da rede, resultando em camadas de processamento que exploram apenas informações locais, ignorando a perspectiva global dos dados. Isso só é considerado nas camadas finais da rede, diminuindo assim o impacto da maldição da dimensionalidade (Dominguez-Catena et al. [2021]).

Após esse episódio ocorreu um expressivo crescimento no uso das redes neurais profundas (Samek et al. [2021]), cujo bom desempenho possibilitou o avanço do estado-da-arte em áreas como classificação de imagens (Targ et al. [2016], Tan and Le [2019]), processamento de linguagem natural (Jin et al. [2020]) e geração de imagens sintéticas (Jabbar et al. [2021]).

1.2 Motivação

Nas RNCs, são realizadas operações de agregação nas camadas de convolução, pooling e totalmente conectadas ou densas. A função de agregação é responsável pelo processo de combinação de diferentes valores numéricos, retornando um único valor (Grabisch et al. [2009]). Um exemplo comum dessas funções de agregação são os operadores do tipo media ponderada ordenada, ou simplesmente operador OWA (Yager [1988, 1993]). Esses operadores OWA são um conjunto de classes parametrizadas de operadores de agregação, que têm sido comumente aplicados em diversos campos, como tomada de decisão multicritério e lógica fuzzy (Zhou et al. [2008], Herrera and Martínez [2001]). Eles permitem controlar a importância relativa dos diferentes valores numéricos, o que é útil para ajustar o modelo de acordo com a natureza do problema e os objetivos desejados.

Geralmente, na arquitetura de redes neurais convolucionais (RNCs) é utilizada uma camada de pooling logo após as camadas de convolução. Esta camada tem como objetivo realizar uma agregação dos mapas de características, reduzindo o tamanho dos dados de entrada, reduzindo a quantidade de parâmetros e de memória necessária para treinar as RNCs. A camada de pooling também ajuda a controlar o overfitting, tornando

[1] 1 *ImageNet Large Scale Visual Recognition Challenge* - www.image-net.org (Russakovsky et al. [2015])

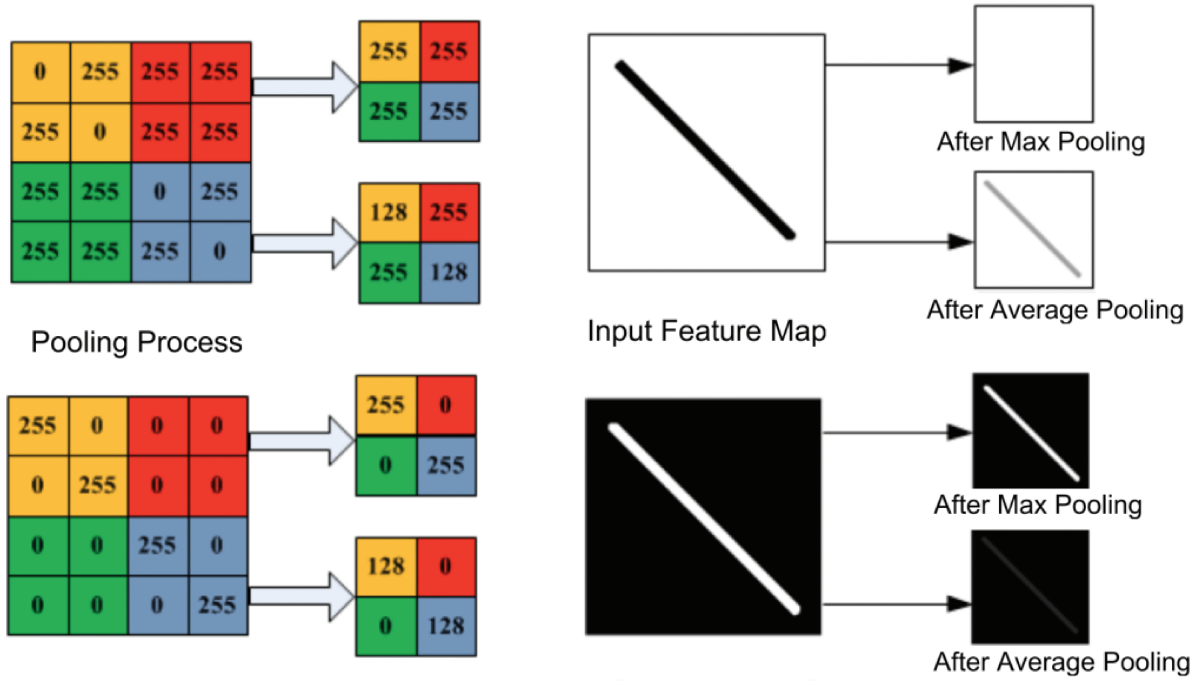

Figura 1 – Desvantagens do pooling máximo e médio (Jie and Wanda [2020].)

a rede mais robusta e menos sensível a pequenas variações nos dados de entrada (Géron [2022]).

Porém, é importante destacar que a redução do tamanho dos dados de entrada pode ser bastante destrutiva, levando a uma perda considerável de informação relevante. Por isso, é importante ajustar adequadamente os parâmetros da camada de pooling (tamanho da janela, quantidade de *stride*, a quantidade de *padding* e o operador utilizados) dependendo do tamanho e tipo dos dados de entrada e o objetivo da rede neural (Géron [2022]).

Os operadores de pooling mais comumente utilizados são o máximo, aplicado nas arquiteturas de rede AlexNet (Krizhevsky et al. [2012]) e VGG (Simonyan and Zisserman [2014]), e a média aritmética aplicada no NiN (Lin et al. [2013]) e GoogleNet (Szegedy et al. [2015]). No entanto, é importante notar que os operadores de máxima e média aritmética são apenas casos especiais dos operadores OWA. Alguns trabalhos têm mostrado que a escolha dos operadores de pooling mais convencionais (máximo e médio) pode não ser a melhor opção, devido as suas deficiências em alguns problemas de visão computacional (Zeiler and Fergus [2013]). A Figura 1 ilustra as desvantagens do agrupamento máximo e agrupamento médio na detecção de uma linha diagonal. Dependendo da imagem, pode ocorrer perdas significativas de informações ao usar o pooling máximo. Por outro lado, o agrupamento médio pode piorar se houver muitos elementos nulos na entrada, reduzindo significativamente os recursos significativos após o agrupamento (Yu et al. [2014]).

Buscando resolver o problema de generalização das camadas pooling, a primeira grande inspiração para esta dissertação foi o trabalho de Forcen et al. [2020], onde foi

criada uma nova camada de pooling utilizando operadores OWA, chamada pelos autores de **OWA-pooling**. Essa camada realiza uma média ponderada ordenada dos elementos, onde os pesos são aprendidos durante o treinamento para que a função de pooling propague as ativações mais significativas para o problema de classificação. A aplicação dos operadores OWA é feita ordenando cada região de agrupamento dos dados de entrada, de forma que os pesos sejam associados às suas magnitudes e não às entradas específicas. Além disso, a utilização de operadores OWA treinados reduz o número de hiperparâmetros das CNNs, já que não é necessário escolher entre pooling máximo ou médio, o que é uma desvantagem comum nas CNNs.

O trabalho de Dominguez-Catena et al. [2020], que também visava integrar operadoras OWA em CNNs, também foi uma das principais fontes de inspiração para esta dissertação. Neste trabalho foi proposta outra camada, que nesta dissertação será chamada de **OWA-channel-aggregation**. Essa camada será colocada dentro da região convolucional da rede e modificará os dados de entrada, uma matriz 3D de ativações, adicionando novos mapas de recursos gerados usando operadores OWA. Usar operadores OWA para agregar informações baseadas em canais é uma ideia interessante, pois essas informações seriam mais difíceis de serem extraídas utilizando as camadas de convolução regulares. Por exemplo, uma visão global da ativação de um mapa de recursos não é fácil de ser computada e utilizada em uma CNN, pois as convoluções tendem a fornecer informações locais, e apenas o empilhamento de camadas permite que a rede capture informações mais globais. Portanto, tentamos aproveitar os operadores OWAs e usar esse tipo de informação global para gerar informações adicionais a partir do ordenamento dos canais. Esses dois fatores significam que a rede pode se beneficiar de informações que de outra forma seriam difíceis de obter.

A saída da camada **OWA-channel-aggregation** é obtida a partir da convolução de diversos filtros (cada kernel representa um operador OWA) sobre os canais dos dados de entrada, ordenados em ordem decrescente. O resultado dessa convolução é passado por uma camada de ativação com a função *ReLu* (Rectified Linear Unit) e, por fim, concatenado aos dados de entrada. A função de ativação é utilizada para encontrar um mapeamento não linear dos dados de entrada, garantindo assim que não sejam adicionados dados correlacionados ao modelo, podendo provocar uma perda de generalização. Durante o treinamento do modelo, os pesos de cada operador OWA utilizado são aprendidos, juntamente com os outros parâmetros treináveis do modelo.

A utilização da camada **OWA-channel-aggregation** no trabalho de Dominguez-Catena et al. [2020] foi realizada apenas na rede VGG13, demonstrando ser uma abordagem que que provoca melhoria de desempenho consistente com baixo custo computacional. Sendo interessante assim utilizar em diferentes modelos, visando validar o seu uso.

A arquitetura geral da camada **OWA-channel-aggregation** é representada na

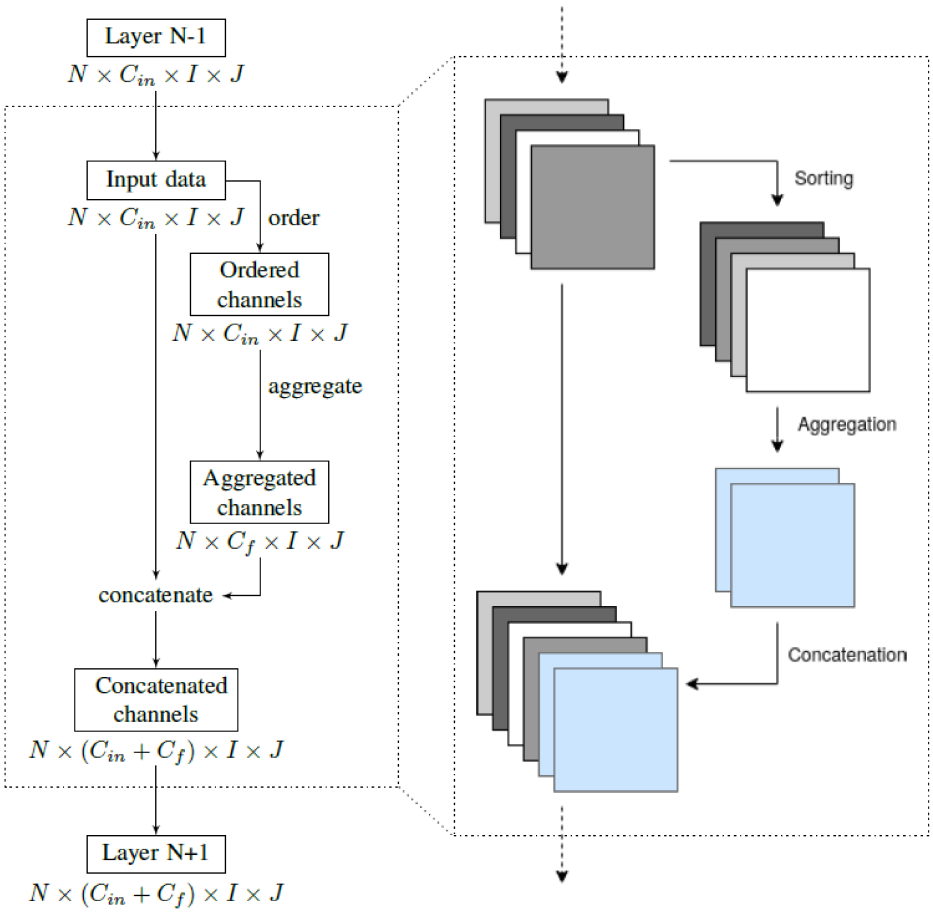

Figura 2 – Arquitetura da camada **OWA-channel-aggregation** (Dominguez-Catena et al. [2020]).

Figura 2. De acordo com a figura C_f operadores OWA distintos foram utilizados para agregar os C_{in} canais dos dados de entrada ordenados em ordem decrescente, gerando C_f novos canais que são concatenados aos dados de entrada. Assim quando os dados de entrada com dimensões $N \times C_{in} \times I \times J$ são processados pela camada **OWA-channel-aggregation** é gerado uma saída com dimensões $N \times (C_{in} + C_f) \times I \times J$, sendo N o número de imagens e $I \times J$ a resolução das imagens.

1.3 Objetivos

O objetivo desse trabalho é comparar e avaliar os resultados obtidos ao treinar diferentes modelos de de redes neurais convolucionais: AlexNet, VGG13 e NiN, aplicando as técnicas **OWA-pooling** e **OWA-channel-aggregation**. Além do mais, será utilizada a camada **OWA-channel-aggregation** para substituir os dados de entrada pelo resultado da agregação, além da concatenação proposta por Dominguez-Catena et al. [2020].

Todos os experimentos serão realizados sobre os conjuntos de dados CIFAR10 e

CIFAR100 (Krizhevsky et al. [2009]).

1.4 Organização do Trabalho

O restante do trabalho está organizado da seguinte forma. O capítulo 2 contém um levantamento bibliográfico, apresentando alguns trabalhos relacionados. No capítulo 3 é apresentado o referencial teórico desta dissertação, abordando alguns dos principais conceitos e definições relacionados ao aprendizado de máquina, redes neurais artificiais e redes neurais convolucionais. No capítulo 4 são apresentadas as camadas OWA operator, OWA-pooling e OWAchannel-aggregation, como cada uma realiza suas agregações, como elas podem ser integradas em uma RNC e como seus pesos são aprendidos. O capítulo 5 apresenta os experimentos realizados e os resultados obtidos. O trabalho é concluído no capítulo 6.

Capítulo 2

Trabalhos Relacionados

Os operadores OWA têm sido amplamente utilizados desde que foram propostos pela primeira vez por Yager (Yager [1988]). Consistem numa família muito conhecida de operadores de agregação, aplicados comumente em diversos campos , como tomada de decisão multicritério e lógica fuzzy (Zhou et al. [2008]). Os operadores OWA podem generalizar uma grande variedade de agregações, como mínimo, máximo ou mediano, entre outros. Esta capacidade de representar agregações muito diferentes tem uma desvantagem, que consiste em determinar os pesos para um operador OWA num determinado contexto. Essa desvantagem levou ao desenvolvimento de de vários métos para estabelecer os pesos dos operadores OWA.

A maior parte dos métodos para determinar os vetores de pesos dos operadores OWA desenvolvidos na literatura consistem em abordagens supervisionadas, onde o vetor de pesos é determinado a partir de uma regra (Xu [2005]). Nessas abordagens os vetores de pesos peso são frequentemente relacionados a famílias específicas de OWAs paramétricos, onde a expressividade do operador OWA é limitada (Kishor et al. [2019]).

Em outros problemas, é necessário uma maior expressividade dos operadores OWA. Assim, surgiram alguns trabalhos que exploram formas de determinar os pesos dos operadores a partir dos dados (Beliakov [2003]). Nesta dissertação, o foco será na segunda abordagem, onde se busca maior expressividade dos operadores OWA, integrando-os aos modelo, de forma que seus pesos sejam parâmetros treináveis do modelo.

Uma das formas mais comuns de usar os operadores OWA são os métodos de aprendizagem ensemble (Scott et al. [2017], Anderson et al. [2018]). A ideia é treinar vários modelos de forma independente e agregar os resultados usando um operador de agregação em uma única saída. É aqui que entram os operadores OWA para executar essa agregação ou outro operador de agregação baseado em logica fuzzy. Observando que os operadores OWA são um caso especial de operadores de agregação baseados em medidas fuzzy (Keller et al. [2016]).

Outra aplicação comum é o uso de operadores OWA nas camadas de pooling das RNCs. Geralmente essas camadas diminuem a resolução da entrada ao longo de suas dimensões espaciais (altura e largura) tomando o valor máximo ou valor médio. No entanto, existem vários trabalhos que exploraram as desvantagens de usar as camadas de agrupamento usuais. Em Boureau et al. [2010] é fornecida uma análise teórica detalhada de pooling máximo e pooling médio para tarefas de reconhecimento de objetos. Foi demonstrado que o tipo de agrupamento ideal para um determinado problema de classificação pode não ser pooling máximo e pooling médio, mas algo intermediário. Com essas desvantagens, aparenta-se ser uma ideia interessante a utilização operadores OWA para realizar a operação de pooling nas RNCs.

Para exemplificar o uso de operadores OWA na camada de pooling, em Dias et al. [2018] os autores aplicaram um operador baseado em logica Fuzzy em uma camada de pooling de uma RNC usando Integral do tipo Choquet, apresentando uma melhora nos resultados em relação ao pooling máximo e médio. Em Pagola et al. [2017] foi demonstrado que bons resultados são alcançado ao usar operadores OWA com pesos fixos não treinados em problemas de classificação de imagens.

Conforme mencionado no primeiro capítulo desta dissertação, uma das principais referências é o trabalho de Forcen et al. [2020], que propuseram uma nova camada de pooling chamada OWA-pooling, a qual utiliza operadores OWA para realizar uma média ponderada ordenada dos elementos. Durante o treinamento, os pesos são aprendidos para permitir que a camada propague as ativações mais importantes para a tarefa de classificação. A aplicação dos operadores OWA envolve a ordenação de cada região dos dados de entrada, permitindo que os pesos sejam associados às magnitudes em vez de entradas específicas. Além disso, o uso de operadores OWA treinados ajuda a reduzir o número de hiperparâmetros nas CNNs, evitando a necessidade de escolher entre pooling máximo ou médio, o que é uma desvantagem comum.

Uma das propostas mais ambiciosas de métodos de integração de operadores OWA em redes neurais é o Linear Order Statistic Neuron desenvolvido por Veal et al. [2019]. Neste trabalho, os autores propõem um neurônio artificial totalmente baseado em um operador OWA, modificando a ideia original do perceptron simples. Embora seja um trabalho preliminar, apresentou ser uma forma eficiente para compartilhar pesos entre N perceptrons, sendo que é capaz de atingir maior generalização com redes mais rasas.

Vale destacar o trabalho de Price et al. [2019], no qual foi proposta uma nova camada chamada de *"Camada Fuzzy"*, que serviu de base para o desenvolvimento da camada **OWA-channel-aggregation**. Essa camada foi projetada para obter informações em um ponto específico da rede e substituí-las pelo resultado da aplicação de seis operadores OWA predefinidos (máximo, mínimo, soft-max, soft-min, média e um operador aleatório) por canal, que são classificados por entropia. No entanto, ao invés de aumentar os dados,

adicionando novos mapas de recursos, gerados usando operadores OWA, a *"Camada Fuzzy"* é utilizada para substituir os dados enviados para a próxima camada.

Os trabalhos de Dominguez-Catena et al. [2020, 2021], que foram também uma das principais referências para essa dissertação, exploram uma nova maneira de usar agregações OWA no contexto de aprendizado profundo. A ideia central desses trabalhos foi a implementação de uma nova camada OWA dentro de uma rede neural convolucional, que nesta dissertação será chamada de **OWA-channel-aggregation**, a qual tem como objetivo aprender informações adicionais com base na ordem dos mapas de recursos de uma determinada camada. As informações geradas por essa nova camada são então utilizadas como entrada complementar para as camadas seguintes da rede. Vários testes foram realizados com a nova camada em uma rede modelo baseada em VGG13, os quais demonstraram melhorias significativas na rede sem aumentar substancialmente os tempos de treinamento. Os autores idicam que os resultados são preliminares e mais pesquisas são necessárias sobre como as redes estão aproveitando essa nova adição, e esperamos que um maior entendimento possa ajudar a estabilizar e aumentar os resultados relatados neste trabalho.

No trabalho de Miranda and Guimaraes [2022], dos mesmo autores dessa dissertação, foram realizados experimentos combinando as camadas **OWA-pooling** e **OWA-channel-aggregation**, resultando em resultados promissores. Os testes foram conduzidos exclusivamente na rede VGG13, com a camada **OWA-channel-aggregation** sendo utilizada para substituir os mapas de recursos de entrada, alem da utilização para agregar novas informações, possuindo um número fixo de filtros. Na avaliação da camada **OWA-pooling**, foram testados pesos distintos para cada canal, bem como o mesmo peso para todos os canais. Os resultados indicaram que a combinação das camadas OWA foi uma abordagem interessante para agregar informações em CNNs, com a obtenção de melhores resultados na camada **OWA-pooling** utilizando os mesmos pesos para todos os canais. Os resultados são considerados preliminares, pois não foi feita uma análise estatística aprofundada, pois o tempo de treinamaneto aumentou consideravelmente ao utilizar a camada **OWA-pooling**.

Capítulo 3

Referencial Teórico

Neste capítulo é apresentado o referencial teórico desta dissertação, abordando alguns dos principais conceitos e definições relacionados ao aprendizado de máquina, redes neurais artificiais e redes neurais convolucionais. A análise será baseada em estudos e trabalhos de outros autores, e visa fornecer uma compreensão sólida dos princípios fundamentais que embasam o tema desenvolvido na dissertação.

Aprendizado de máquina é uma subárea da inteligência artificial que se concentra na construção de sistemas capazes de aprender a partir de dados. As redes neurais artificiais são uma das técnicas de aprendizado de máquina mais populares e são amplamente utilizadas para tarefas de classificação e predição. As redes neurais convolucionais são uma variação das redes neurais artificiais, especialmente projetadas para lidar com dados de imagem e são amplamente utilizadas em tarefas de visão computacional. Nas próximas seções serão discutidos mais sobre esses conceitos e como eles se relacionam com o tema da dissertação.

3.1 Aprendizado de Máquina

> "Diz-se que um programa de computador aprende com a experiência E em relação a alguma tarefa T e alguma medida de desempenho P, se seu desempenho em T, medido por P, melhora com a experiência E."[Mitchell and Mitchell, 1997].

O aprendizado de máquina é um subcampo da inteligência artificial (IA). O objetivo do aprendizado de máquina é generalizar a partir de sua experiência (Bishop and Nasrabadi [2006]). A generalização neste contexto é a capacidade de uma máquina de aprendizado de performar com precisão em novos exemplos/tarefas não vistos depois de ter experimentado um conjunto de dados de aprendizado ou treinamento.

As abordagens de aprendizado de máquina são tradicionalmente divididas em três grandes categorias, que correspondem a paradigmas de aprendizado, dependendo da

natureza do "sinal" ou "feedback" disponível para o sistema de aprendizado (Stuart and Norvig [2010]):

- Aprendizagem supervisionada: O computador recebe exemplos de entradas e suas saídas desejadas, e o objetivo é aprender uma regra geral que mapeia as entradas para as saídas.
- Aprendizado não supervisionado: nenhum rótulo é dado ao algoritmo de aprendizado, deixando-o sozinho para encontrar a estrutura em sua entrada. O aprendizado não supervisionado pode ser um objetivo em si (descobrir padrões ocultos nos dados) ou um meio para um fim (aprendizado de novas features).
- Aprendizagem por reforço: um programa de computador interage com um ambiente dinâmico no qual deve realizar um determinado objetivo (como dirigir um veículo ou jogar um jogo contra um adversário). À medida que navega pelo espaço do problema, o programa recebe feedback análogo a recompensas, que ele tenta maximizar (Bishop and Nasrabadi [2006]).

O foco deste trabalho sera no aprendizado supervisionado, por trabalhar com modelos de classificação de imagens previamente rotuladas.

3.1.1 Aprendizado Supervisionado

O aprendizado supervisionado ocorre quando um sistema recebe variáveis de entrada e saída com a intenção de aprender como tais variáveis são mapeadas. O objetivo é descobrir uma função de mapeamento para que, quando novas entradas desconhecidas forem fornecidas, o modelo possa prever a saída. Esse é um processo iterativo e, toda vez que o algoritmo faz uma previsão, ele é corrigido ou recebe um feedback até atingir um nível aceitável de desempenho (Rosebrock [2017], Zhou [2021]). A configuração do problema de classificação supervisionado é apresentada nos tópicos a seguir:

- Conjunto de treinamento: um conjunto de m pares (X, y), onde cada entrada $X \in R^d$, cada saída $y \in \{0, 1, 2, ..., n-1\}$, sendo d o número de dimensões dos dados de entrada e n o número de classes.
- Objetivo: Aprender uma função/modelo $f : X \implies y$ para prever corretamente novas entradas X.
  - Etapa 1: escolha um algoritmo de aprendizado
    - regressão logística, SVMs, k-NN, RNAs, etc.
  - Etapa 2: Otimize os parâmetros/pesos W usando o conjunto de treinamento

∗ Minimize a função de perda (3.1)

$$\min_{W} \sum_{i=1}^{m} (f_W(x^{(i)}) - y^{(i)})^2 \tag{3.1}$$

As aplicações do aprendizado supervisionado geralmente são divididas em duas categorias: classificação e regressão. A classificação ocorre quando o valor de saída é uma categoria, como identificar se uma imagem é de um carro ou de um caminhão. Já um problema de regressão acontece quando a saída é um valor real, como prever o preço de um imóvel, o peso de uma pessoa, a temperatura ou a umidade. A principal diferença entre os dois problemas é que, enquanto na classificação a saída é um valor discreto, na regressão ela é contínua (Murphy [2012]).

3.2 Redes Neurais Artificiais

3.2.1 Perceptron

O cérebro humano é composto por cerca de 100 bilhões de neurônios. Um neurônio biológico é uma célula nervosa responsável pelo processamento de informações (Jain et al. [1996]). Cada neurônio é delimitado por uma membrana e contém um núcleo que abriga os genes. O neurônio também apresenta projeções especializadas que permitem a entrada e saída de informações na célula nervosa, as quais são conhecidas como dendritos e axônios.

Os modelos de redes neurais artificiais (RNAs) são baseados na maneira como como as redes neurais biológicas (RNBs) operam. É essencial notar, porém, que esses modelos diferem significativamente em termos de funcionamento, escala e complexidade em comparação com o cérebro humano. No entanto, vários dos termos usados para descrever redes neurais artificiais são emprestados da literatura de neurociência (Khan et al. [2018]). Em outras palavras, as redes neurais artificiais (RNAs) consistem em unidades básicas de processamento interconectadas que trabalham juntas para processar informações de entrada e produzir resultados desejados, assim como as RNBs. A Figura 3 apresenta as semelhanças entre a simplificação de um neurônio biológico e um Perceptron.

Um neurônio, é um classificador linear simples em que a sua saída é dada pela Equação (3.2)

$$y = \sigma(W^T X + b) \tag{3.2}$$

sendo σ a função de ativação, W o vetor de pesos e X os dados de entrada e b o viés.

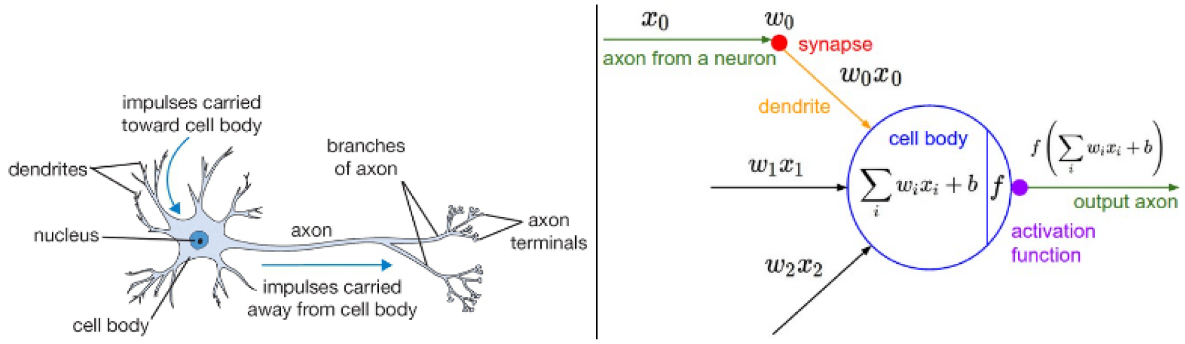

Figura 3 – Uma ilustração de um neurônio biológico (esquerda) e seu modelo matemático (direita). O neurônio biológico coleta o sinal de entrada (dentritos), processa a informação no núcleo e gera a saída pelo axônio.

Fonte: Karpathy et al. [2017]

O Perceptron (Rosenblatt [1958]) é indiscutivelmente o mais antigo e mais simples dos algoritmos de RNA. No Perceptron a função de ativação σ é dada pela função degrau (Equação (3.3)).

$$\sigma(net) = \begin{cases} +1 & \text{if } net > 0 \\ -1 & \text{if } net \leq 0 \end{cases} \tag{3.3}$$

sendo net igual a ao resultado da propagação direta da entrada pelo neuronio, iso é, $net = W^T X + b$.

Nos tópicos a seguir se encontra o algoritmo Perceptron, utilizado num problema de classificação binária:

1. Inicialize o vetor de pesos W com valores pequenos aleatório
2. Até a convergência ou atingir o número máximo de iterações, avalia cada amostra de $x_i \in X$:
   (a) Se a saída estiver correta, faça nada
   (b) Se a saída for -1 e a correta for $+1$: adicione o vetor de entrada ao vetor de peso
   (c) Se a saída for $+1$ e a correta for -1: subtraia o vetor de entrada ao vetor de peso

3.2.2 Backpropagation e Redes Multicamadas

O *backpropagation* é indiscutivelmente o algoritmo mais importante na história das redes neurais – sem o *backpropagation* (eficiente), seria impossível treinar redes profundas. O algoritmo de *backpropagation* pode ser resumido em duas etapas:

1. *The forward pass* onde nossas entradas são passadas pela rede e as previsões de saída obtidas (também conhecida como fase de propagação).

2. *The backward pass* onde calculamos o gradiente da função de perda na camada final (ou seja, camada de previsões) da rede e usamos esse gradiente para aplicar recursivamente a regra da cadeia para atualizar os pesos em nossa rede (também conhecido como peso fase de atualização).

O algoritmo de *backpropagation* é utilizado para treinar um perceptron de múltiplas camadas (MLP). Ao contrário do Perceptron com apenas 1 camada o MLP pode ser aplicado em problemas de aprendizado de máquina não lineares. As Redes Neurais com pelo menos uma camada oculta são aproximadores universais. Em Cybenko [1989b] é mostrado que, dado um número suficiente de neurônios na camada oculta, a MLP pode aproximar qualquer função contínua em um espaço de entrada finito e contínuo para um espaço de saída correspondente. Na prática, muitas vezes, as redes neurais de 3 camadas superam as redes de 2 camadas, mas ir ainda mais fundo (4,5,6 camadas) raramente ajuda muito mais no desempenho (Karpathy et al. [2017]). A Figura 4 apresenta uma MLP de três camadas.

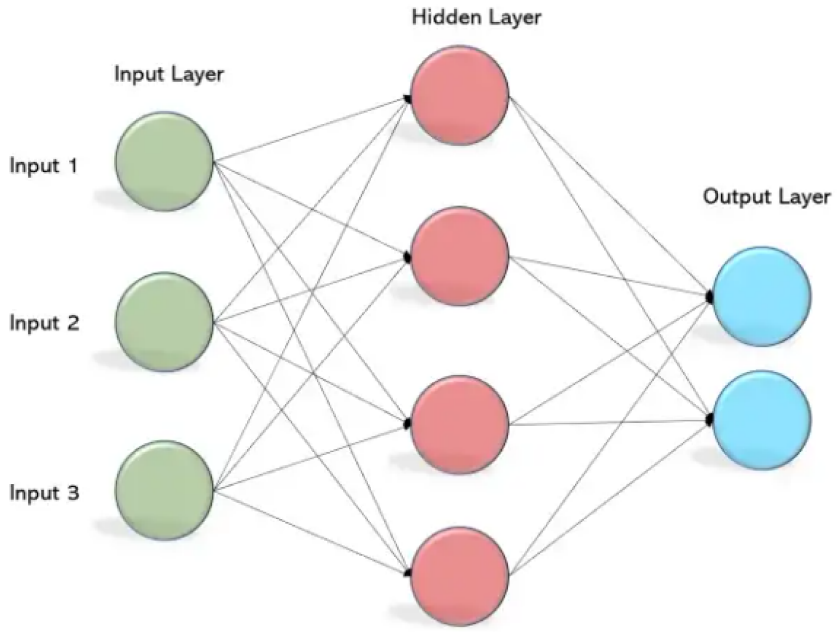

Figura 4 – Perceptron de Múltiplas Camadas

3.3 Funções de Ativação

Uma função de ativação é uma operação matemática que recebe uma única entrada e produz uma única saída. Ela é chamada de "não linear"porque sua saída não é uma simples combinação linear dos dados de entrada. Através de transformações não lineares, padrões mais complexos dos dados podem ser aprendidos, sendo possível representar

mapeamentos não lineares entre a entrada e a saída. Uma função não linear também pode ser entendida como um mecanismo de comutação ou seleção, que decide se um neurônio irá disparar ou não dado todas as suas entradas. As funções de ativação comumente usadas em redes profundas são diferenciáveis para permitir a retropropagação do erro. A Figura 5 ilustra algumas das principais funções de ativação.

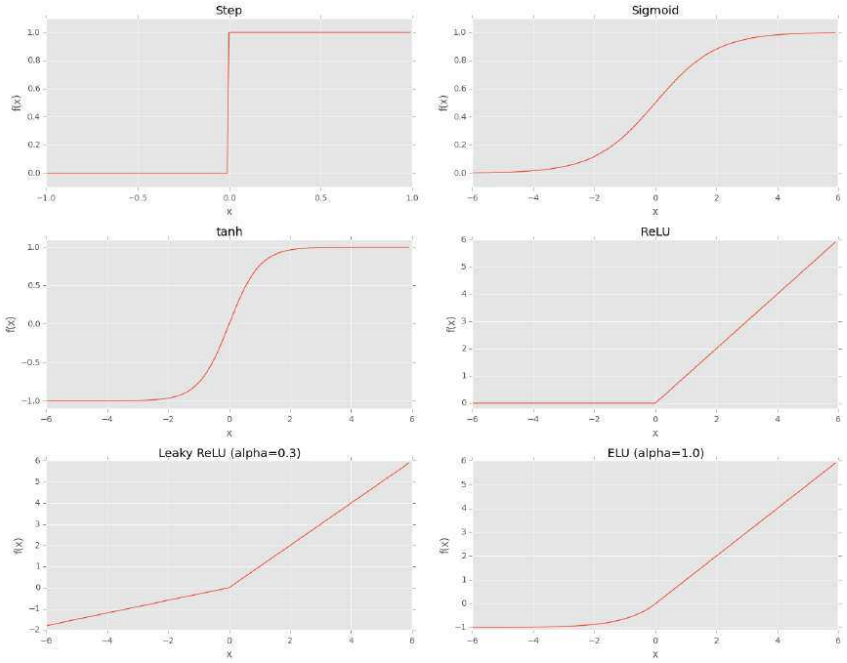

Figura 5 – Gráfico das principais funções de ativação.

Fonte: Rosebrock [2017]

Nesse trabalho foram utilizados nos modelos as funções de ativação *ReLU*, **Sigmoide** e **Softmax**. A função de ativação *degrau* já foi muito utilizada, não é mais utilizada hoje em dia, pois não é diferenciável e possui derivada igual a zero para todos os números diferentes de 0, impossibilitando a convergência do algoritmo de *backpropagation*. As funções de ativação *Sigmoide* e *tanh* são comumente utilizadas na saída dos modelos, mas não no interior dos modelos, pois caso ocorra saturação de algum neurônio, o gradiente tenderá a zero, dificultando o reajuste dos pesos após o backpropagation.

A função de ativação *ReLU* (Rectified Linear Unit) é a mais comumente utilizada. Na função de ativação *ReLU*, os valores positivos da entrada são mantidos e os valores negativos substituídos por zero Goodfellow et al. [2016]. A Equação 3.4 apresenta a função de ativação ReLU. As variações da função de ativação *ReLU*, a *Leaky ReLU* e *ELU* (Exponential Linear Unit), visam evitar que alguns neurônios permaneçam inativos para $x < 0$. A Figura 5 ilustra todas as funções de ativação que foram citadas.

$$f_{ReLU}(x) = \max(0, x) \tag{3.4}$$

A função de ativação **Sigmoide** é comumente utilizada em problemas de classificação binária, que envolvem apenas duas classes. Ela recebe uma entrada e gera um número entre 0 e 1 que representa a probabilidade de pertencer à classe 1 ($P(C_1)$). A probabilidade de pertencer à classe 2 é, portanto, $1 - P(C_1)$. è definida como:

$$f_{sigm}(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$

Para problemas de classificação multi-classe, costuma-se utilizar a função de ativação **Softmax** na saída do modelo, que é uma extensão da função **Sigmoide**. A **Softmax** é vetorizada, o que significa que recebe de entrada um vetor com o mesmo número de classes a serem classificadas e gera na saída um vetor onde cada componente representa a probabilidade de pertencer a uma das classes. Dessa forma, a **Softmax** permite a classificação de múltiplas classes simultaneamente (Dunne and Campbell [1997]). A equação da função de ativação softmax é dada por:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.6}$$

onde $\sigma(z_i)$ é a saída da função de ativação softmax para a i-ésima unidade, z_i é o valor de entrada para a i-ésima unidade, e é a constante matemática de Euler ($e \approx 2.718$) e K é o número total de unidades de saída.

A função softmax é aplicada a um vetor de valores de entrada z_i, geralmente correspondentes às saídas das camadas anteriores da rede neural. A função de ativação softmax transforma esses valores de entrada em um conjunto de valores normalizados que representam a probabilidade de cada classe de saída. A função softmax costuma ser aplicada como a camada de saída da rede neural, sendo que seus resultados representam distribuição de probabilidade sobre todas as classes possíveis. A soma das probabilidades para todas as classes é sempre igual a 1, garantindo que a rede neural sempre produzirá uma resposta válida (Khan et al. [2018]).

3.4 Redes Neurais Convolucionais

As redes neurais convolucionais (RNCs) são semelhantes as redes de multicamadas (MLPs), pois são compostas por neurônios que ajustam os seus pesos através do processo de aprendizado. Entretanto, ao contrário das MLPs, são particularmente adequadas para lidar com dados não estruturados, como imagens, devido à sua estrutura hierárquica de camadas de filtros. Essa estrutura hierárquica das RNCs, permite que sejam extraídas características com diferentes níveis de abstração a partir da imagem de entrada, permitindo que a rede reconheça padrões complexos e realize tarefas como classificação de imagens e detecção de objetos (Krizhevsky et al. [2012]).

Redes Neurais Convolucionais são redes neurais que utilizam a operação matemática conhecida como convolução em pelo menos uma de suas camadas. A convolução é uma operação linear especializada que é utilizada para extrair características dos dados de entrada aplicando diferentes filtros. Esses filtros são treinados para reconhecer padrões específicos, como linhas, curvas ou formas geométricas. Por exemplo, na classificação de imagens, as bordas podem ser detectadas na primeira camada e, nas camadas posteriores, formas mais complexas são detectadas (Goodfellow et al. [2016]). A Figura 6 exemplifica os padrões aprendidos para um problema de classificação de reconhecimento facial.

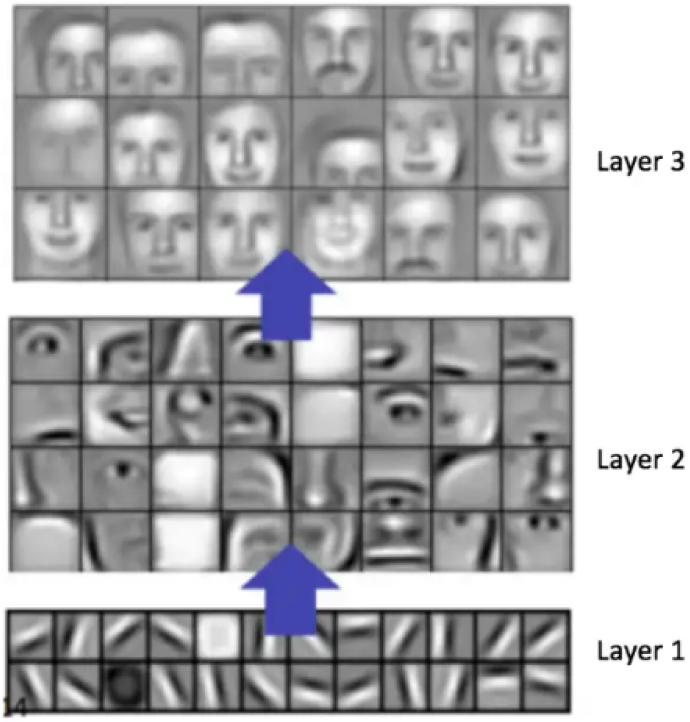

Figura 6 – Padrões aprendidos em um problema de reconhecimento facial. Quanto mais profunda a camada, mais complexo os padrões aprendidos.

Fonte: Lee et al. [2009]

Em 1998, LeCun et al. [1998] propôs as primeiras Redes Neurais Convolucionais (RNCs) que utilizaram o algoritmo de retropropagação (backpropagation) para reconhecimento de dígitos, por meio da arquitetura denominada "LeNet". No entanto, a notoriedade das RNCs cresceu exponencialmente em 2012, quando Krizhevsky et al. [2012] venceu o desafio ILSVRC (ImageNet Large Scale Visual Recognition Challenge), um concurso anual de classificação de imagens organizado pelo laboratório de pesquisa da Universidade de Stanford (Russakovsky et al. [2015]). A solução desenvolvida por Krizhevsky et al. [2012] foi a primeira a superar o desempenho humano nessa tarefa, consagrando as redes neurais convolucionais como tecnologia de ponta no campo de aprendizado de máquina. O desafio ILSVRC ocorreu entre 2010 e 2017.

As RNCs são compostas principalmente por quatro tipos de camadas. Estas são

camadas convolucionais (CONV), camadas de pooling (POOL), camadas de ativação (ACT ou RELU) e camadas totalmente conectadas (FC). O empilhamento de uma série dessas camadas de uma maneira específica produz uma RNC. Assim uma CNN simples pode ser representada da seguinte maneira: INPUT => CONV => RELU => POOL => FC => SOFTMAX (Rosebrock [2017]).

As camadas de ativação são frequentemente omitidas durante a definição das arquiteturas de Redes Neurais Convolucionais (RNCs), supondo-se que uma camada de ativação - normalmente ReLU - seguirá após uma camada de convolução ou totalmente conectada, e que a última camada de ativação usará a função softmax em problemas de classificação multi-classe. Além dessas camadas básicas, outras camadas como batch normalization e dropout podem ser adicionadas para melhorar a performance e evitar o overfitting. No entanto, essas camadas geralmente são consideradas como secundárias na definição da arquitetura da rede, mas não devem ser subestimadas pois podem afetar consideravelmente a performance do modelo (Rosebrock [2017]).

Uma arquitetura CNN simplificada para classificação da base de dados MNIST está ilustrada na Figura 7. o MNIST é uma grande base de dados de dígitos manuscritos (Deng [2012]).

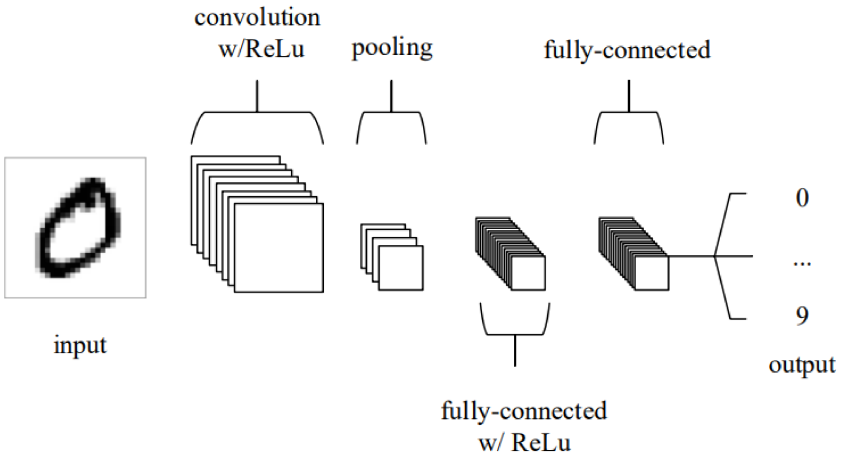

Figura 7 – Arquitetura simples de uma RNC, composta apenas por cinco camadas

Fonte: O'Shea and Nash [2015]

3.4.1 Camada de Convolução

As camadas convolucionais são utilizadas para a extração automática de características, sendo composta por um conjunto de filtros (também chamados de kernels), onde é feita a convolução de cada com os dados de entrada.

O que é um Filtro? Cada filtro em uma camada convolucional é uma matriz de números discretos. Os pesos de cada filtro são aprendidos durante o treinamento da RNC.

Em relações as suas dimensões, cada filtro tem uma largura, uma altura, e se estendem por toda a profundidade do volume, isto é, possuem a mesma quantidade de canais dos dados de entrada. Caso os dados entrada sejam uma imagem RGB com 3 canais, cada *kernel* também deve possuir 3 canais.

A Convolução definida na Equação 3.7 (denotada pelo operador $\star$) realizada sobre uma imagem de entrada bidimensional I de tamanho (m, n) e o kernel bidimensional K, é equivalente à saída gerada ao se deslocar kernel invertido sobre a imagem, realizando a multiplicação elemento a elemento, seguida de uma soma (Goodfellow et al. [2016]).

$$S(i,j) = (I \star K)(i,j) = \sum_m \sum_n K(i-m, j-n) I(m,n) \tag{3.7}$$

No entanto, quase todas as bibliotecas de aprendizado de máquina e aprendizado profundo usam a função de correlação cruzada, nas camadas de convolução, onde o kernel não é invertido antes de ser deslizado pela imagem (Equação 3.8).

$$S(i,j) = (I \star K)(i,j) = \sum_m \sum_n K(i+m, j+n) I(m,n) \tag{3.8}$$

Através da operação de convolução é possível identificar padrões presentes nos pixels da imagem que sejam semelhantes aos padrões presentes no filtro utilizado. Ao deslizar o filtro totalmente pela imagem, aplicando a operação de convolução, é obtida uma matriz como resultado, chamada de mapa de características. Esse mapa de características é representação das características extraídas da imagem. Após a aplicação de uma função de ativação sobre o mapa de características, é gerada uma nova matriz, chamada de mapa de ativação. Conforme apresentado na subseção 3.3, a função de ativação é aplicada para introduzir não linearidade na saída da camada, permitindo que a rede aprenda funções mais complexas. O mapa de ativação então pode ser utilizado como entrada para uma próxima camada convolucional, permitindo a extração de padrões cada vez mais complexos. Ao utilizar diferentes filtros durante o treinamento, diferentes padrões são aprendidos. Em problemas de classificação, o objetivo é encontrar os padrões nos dados que sejam mais discriminativos possível. A Figura 8 ilustra como é realizada a operação de convolução numa imagem 4×4, utilizando um filtro 2×2 Khan et al. [2018].

Para determinar as dimensões do mapa de características de saída, além das dimensões do kernel e da matriz de entrada, devemos considerar os valores de *stride* e de *padding*.

Stride é o valor do passo que o filtro realiza ao deslizar pela matriz de entrada. Na Figura 8, para calcular cada valor do mapa de características de saída, o filtro dá um passo de 1 ao longo da posição horizontal ou vertical (ou seja, ao longo da coluna ou linha da entrada). Quanto maior o valor do *stride* menor a quantidade de cálculos e, portanto,

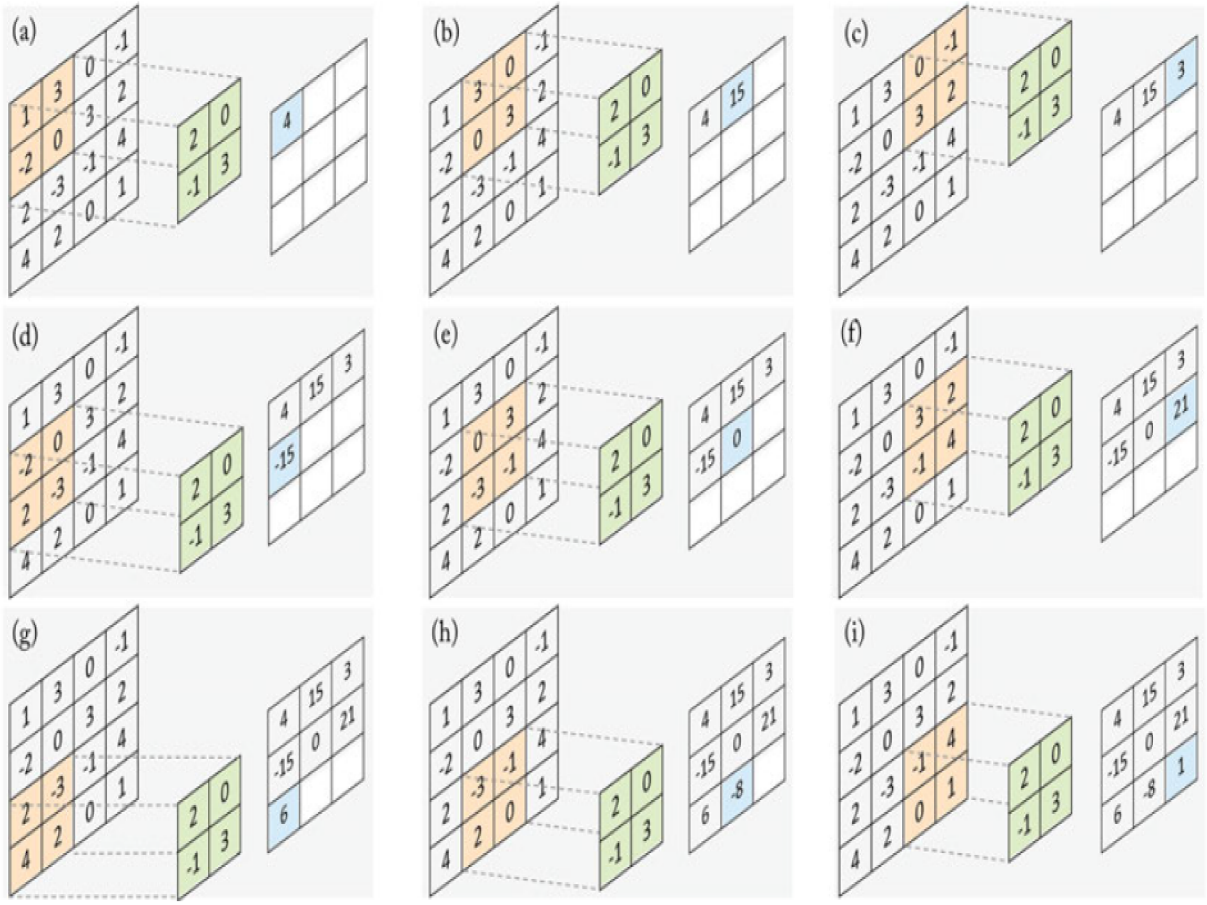

Figura 8 – A operação de uma camada de convolução é ilustrada na figura acima. (a)–(i) mostram os cálculos realizados em cada etapa, conforme o filtro é deslizado pelos dados de entrada para calcular o valor correspondente no mapa de características de saída. O filtro 2×2 (mostrado em verde) é multiplicado com a mesma região de tamanho (mostrado em laranja) dentro dos dados de entrada 4×4 e os valores resultantes são somados para obter um mapa de características de saída(mostrada em azul)

Fonte: Khan et al. [2018]

menos recursos computacionais são utilizados. Por outro lado, aumentar o valor do *stride* também resulta em uma menor quantidade de detalhes sendo preservados no mapa de características de saída. Portanto há um *trade-off* entre performe e custo computacional ao ao ajustar o valor do *stride*. O ideal é encontrar um equilíbrio entre esses dois aspectos (Khan et al. [2018]).

Padding é uma técnica utilizada para ter maior controle sobre as dimensões do mapa de características de saída. É feito adicionando uma borda de pixels "vazios"(geralmente preenchidos com zero) ao redor dos dados de entrada antes de aplicar a operação de convolução. Além disso, o padding também é utilizado para lidar com uma das desvantagens da etapa de convolução, que é a perda de informações que possam existir nas bordas da imagens. Ao utilizar padding, a informação presente nas bordas da imagem é preservada e não há perda de dados, resultando em mapas de características mais precisos

e detalhados (Albawi et al. [2017]). A Figura 9 mostra como é realizado o *zero-padding* sobre uma matriz de dados.

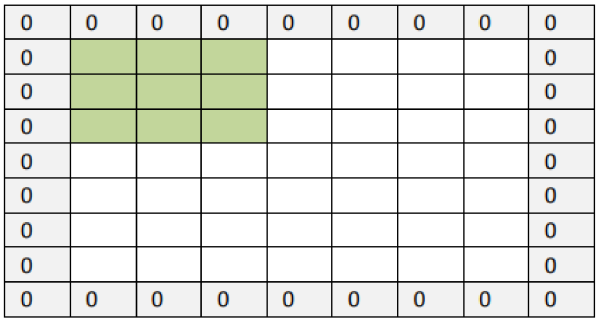

Figura 9 – Padding realizado adicionando uma borda de de 1 pixel de zeros

É importante entender que, ao usar essas técnicas, alteraremos a dimensionalidade espacial da saída das camadas convolucionais. Para um filtro de tamanho $f \times f$, um mapa de recursos de entrada com tamanho $h \times w$, um *stride* de tamanho s e um *padding* de largura p, as dimensões (h', w') do mapa de características de saída são definidas nas equações a seguir (Khan et al. [2018]):

$$h' = \left\lfloor \frac{h - f + 2p}{s} + 1 \right\rfloor, w' = \left\lfloor \frac{w - f + 2p}{s} + 1 \right\rfloor \tag{3.9}$$

$\lfloor . \rfloor$ denota a operação piso.

3.4.2 Camada de Pooling

As camadas de pooling visam reduzir gradualmente a dimensionalidade da representação e, assim, reduzir o número de parâmetros e a complexidade computacional do modelo (O'Shea and Nash [2015]). Elas são geralmente aplicadas após as camadas de convolução e existem diferentes tipos de métodos de pooling, como o max-pooling e o average-pooling que são os mais comuns. O max-pooling divide a imagem em sub-regiões e retorna apenas o valor máximo dentro dessa sub-região, conforme ilustrado na Figura 10.

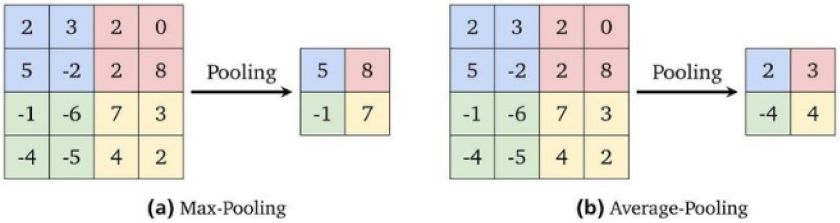

Figura 10 – Exemplo de max-pooling e average pooling com um tamanho de filtro de 2×2 e stride 2×2. Adaptado de Karpathy et al. [2017]

Assim como as camadas de convolução, define-se valores *stride* e de *padding* para alterar a dimensionalidade da saída. Normalmente, o *stride* e os filtros são definidos com

tamanho 2×2, o que permitirá que a camada se estenda por toda a dimensionalidade espacial da entrada. Além disso, é possível usar pooling com sobreposição, onde o *stride* é definido como 2 com um tamanho de filtro definido como 3. O *padding* pode ser necessário para processar entradas com uma forma que não se ajusta perfeitamente ao tamanho do *stride* e do filtro, mas normalmente é nulo. Devido à natureza destrutiva do pooling, um tamanho de filtro maior que 3 geralmente diminui significativamente o desempenho do modelo (O'Shea and Nash [2015]).

Para um filtro de tamanho $f \times f$, dados de entrada de entrada com tamanho $h \times w$, um *stride* de tamanho s, um *padding* vertical dado por p_h e um *padding* horizontal dado por p_w, as dimensões da saída da camada de pooling ($h'ew'$) são definidas nas equações a seguir (Khan et al. [2018]):

$$h' = \left\lfloor \frac{h - f + p_h}{s} + 1 \right\rfloor, w' = \left\lfloor \frac{w - f + p_w}{s} + 1 \right\rfloor \tag{3.10}$$

3.4.3 Camadas Totalmente Conectadas

As camadas totalmente conectadas são utilizadas no final do processo de uma RNC para conectar todos os neurônios de uma camada anterior a todos os neurônios de uma camada seguinte. Isso permite que a rede possa aprender relações não lineares entre os dados de entrada e saída. Cada camada totalmente conectada são como uma camada escondida de uma MLP (Khan et al. [2018]).

Cada neurônio da camada totalmente conectada realiza uma operação matemática simples, como a multiplicação de peso e soma de bias, para produzir uma saída (Equação 3.2). A saída de cada um dos neurônios são então passadas para a próxima camada. É comum usar uma ou duas camadas totalmente conectadas no final da RNC (Rosebrock [2017]).

3.5 Aprendizado da RNC

Nesta seção, serão apresentados os diferentes mecanismos e técnicas utilizadas nos experimentos drealizados. Começaremos discutindo sobre a função de perda (subseção 3.5.1) e o processo de treinamento de uma RNC (subseção 3.5.2), apresentando como os pesos são atualizados. Em seguida serão discutidos conceitos fundamentais como a inicialização de pesos e a regularização de redes nas subseções 3.5.3 e 3.5.4, respectivamente, os quais são fundamentais para o sucesso na otimização de redes neurais.

3.5.1 Função de Perda

Uma função de perda quantifica a diferença entre a saída estimada do modelo (a previsão) e a saída correta. O objetivo do treinamento das RNCs é minimizar a função de perda (Khan et al. [2018]).

O tipo de função de perda a ser utilizada depende do objetivo do problema de aprendizado de máquina. Para problemas de classificação multi-classe geralmente é utilizada a função *cross-entropy loss*, também chamada de perda *softmax*, dada pela Equação 3.11.

$$L(p, y) = -\sum_{n} y_n \log(p_n), \qquad n \in [1, N] \tag{3.11}$$

onde y denota a saída desejada e p é a probabilidade para cada uma das possíveis categorias. Há um total de N neurônios na camada de saída, portanto, $p, y \in R^N$. A probabilidade para cada uma das classes pode ser calculada utilizando a função soft-max: $p_n = \frac{exp(\hat{p}_n)}{\sum_k exp(\hat{p}_k)}$, sendo $\hat{p}_n$ a pontuação de saída não normalizada proveniente da camada anterior na rede.

3.5.2 Aprendizado Baseado em Gradientes

O processo de treinamento de uma RNC é essencial para garantir que o modelo possua boa capacidade de generalização. Isso é alcançado através da otimização dos pesos da RNC, minimizando a função de custo. A função de custo calcula a média da função de perda para todas as amostras no conjunto de treinamento (Khan et al. [2018]).

Uma maneira intuitiva, mas simples, de abordar esse problema de otimização é atualizando repetidamente os pesos da RNC, de modo que a função de custo reduza progressivamente a um valor mínimo. Aqui, os métodos baseados em gradiente são uma escolha natural, pois precisamos atualizar os parâmetros na direção em que a descida é mais ingrime. A velocidade de atualização dos parâmetros é chamada de "**taxa de aprendizado**". Cada iteração que atualiza os parâmetros usando o conjunto de treinamento completo é chamada de "**época de treinamento**". As Equações 3.12 e 3.13 apresentam como é realizada a atualização dos parâmetros dado cada época de treinamento t.

$$\theta_t = \theta_{t-1} - \eta\delta_t \tag{3.12}$$

$$\delta_t = \Delta_\theta \mathcal{F}(\delta_t) \tag{3.13}$$

δ_t representa o gradiente da função de perda na iteração t, η a taxa de aprendizagem, $\mathcal{F}(\cdot)$ denota a função a função de custo da rede neural com parâmetros θ e $\Delta_\theta \mathcal{F}(\delta_t)$ representa a direção do gradiente em relação aos parâmetros do modelo na iteração t. Essa direção é usada para atualizar os parâmetros na direção de minimização da função de perda.

Existem três formas mais usuais de se aplicar o algoritmo de gradiente descendente: Gradiente Descendente em Lotes, Gradiente Descendente Estocástico (SGD), Gradiente Descendente em Minilotes. Em todos os experimentos neste trabalho foi utilizado o **Gradiente Descendente em Minilotes** (Khan et al. [2018]).

O **Gradiente Descendente em Minilotes** é uma abordagem aprimorada de gradiente estocástico, fornecendo um bom *trade-off* entre Gradiente Descendente Estocástico (SGD), que calcula o gradiente da função de perda usando apenas um exemplo de treinamento selecionado aleatoriamente a cada iteração, e o Gradiente Descendente em Lotes, que calcula o gradiente da função de perda usando todos os exemplos de treinamento de uma só vez. O Gradiente Descendente em Minilotes calcula o gradiente da função de perda usando um pequeno conjunto aleatório de exemplos de treinamento, chamado de "minilote"em cada iteração. A atualização dos parâmetros é então realizada depois de computar os gradientes em cada minilote (Rosebrock [2017]).

A utilização do gradiente descendente em minilotes ajuda a reduzir a variância na atualização dos parâmetros do modelo, levando a uma convergência mais estável, mais eficiente em termos computacionais e menos suscetível a estagnar num mínimo locais (Bottou [2010]).

3.5.2.1 Momento

Uma versão melhorada do gradiente descendente com melhores propriedades de convergência pode ser obtida usando momento. Ao incluir um termo de "momento"é obtido maior estabilidade na atualização dos pesos. O objetivo é evitar que o modelo fique preso em um mínimo local, aumentando a capacidade de alcançar o mínimo global (Aggarwal et al. [2018]). As equações 3.14 e 3.15 apresentam como é realizada a atualização dos parâmetros dado cada época de treinamento t

$$\theta_t = \theta_{t-1} - \eta a_t \tag{3.14}$$

$$a_t = \Delta_\theta \mathcal{F}(\delta_t) - \gamma a_{t-1} \tag{3.15}$$

δ_t representa o gradiente da função de custo na iteração t, η a taxa de aprendizagem, $\mathcal{F}(\cdot)$ denota a função de custo do modelo com parâmetros θ, $\Delta_\theta \mathcal{F}(\delta_t)$ representa a direção do gradiente em relação aos parâmetros do modelo na iteração t e γ representa o momento, normalmente definido como $0,9$.

3.5.3 Inicialização de Pesos

As redes neurais geralmente exibem problemas de estabilidade no sentido de que as ativações de cada camada se tornam sucessivamente mais fracas ou sucessivamente mais

fortes. O efeito está exponencialmente relacionado com a profundidade da rede e, portanto, é particularmente grave em redes profundas. Uma maneira de evitar esses problemas é escolher bons pontos de inicialização dos pesos, de forma que os gradientes sejam estáveis nas diferentes camadas (Aggarwal et al. [2018]). Ao inicializar os pesos utilizando uma distribuição normal ou uniforme, a variância de sua saída de torna diretamente proporcional ao número de conexões de entrada. Os métodos de inicialização de pesos Xavier (Glorot and Bengio [2010]) e He (He et al. [2015]) amostram os pesos a partir de uma distribuição que leva em consideração o número de conexões de cada neurônio. Nos experimentos deste trabalho foi utilizada a inicialização He.

3.5.3.1 Inicialização He

A inicialização **He** (He et al. [2015]) leva em consideração que a função de ativação ReLU reduz quase metade das entradas a zero. Assim, os pesos são inicializados a partir de uma distribuição normal com média 0 e desvio padrão igual a $\sqrt{2/n}$, sendo n o número de conexões de entrada.

3.5.4 Regularização dos Pesos

Um problema central no aprendizado de máquina é como treinar um modelo para que tenha um bom desempenho não apenas nos dados de treinamento, mas também em novas entradas não vistas durante o treinamento. Muitas estratégias usadas no aprendizado de máquina são explicitamente projetadas para reduzir o erro sobre os dados de teste, possivelmente às custas do aumento do erro sobre os dados treinamento. Essas estratégias são conhecidas como métodos de regularização (Goodfellow et al. [2016]). Nas próximas subseções serão apresentados oe métos de regularização utilizados nesse trabalho.

3.5.4.1 Data Augmentation

Uma forma de aprimorar um modelo de aprendizado de máquina é fornecer-lhe uma grande quantidade de dados para treinamento. No entanto, é comum que não haja dados suficientes disponíveis. Uma solução para isso é gerar dados adicionais artificialmente e incluí-los no conjunto de treinamento, abordagem conhecida como *data augmentation* (Goodfellow et al. [2016]).

Data augmentation é realizada através de operações simples como rotação, recorte, preenchimento, alteração de contraste, alteração de brilho, dentre outras transformações (Figura 11). O objetivo é aumentar a variedade de dados disponíveis, permitindo que o modelo aprenda a reconhecer diferentes variações de uma mesma imagem e assim melhorando sua capacidade de generalizar e se adaptar a novos dados. Essas transformações podem ser aplicadas a cada lote de imagens, fornecendo um conjunto de dados diferente a cada época de treinamento do modelo.

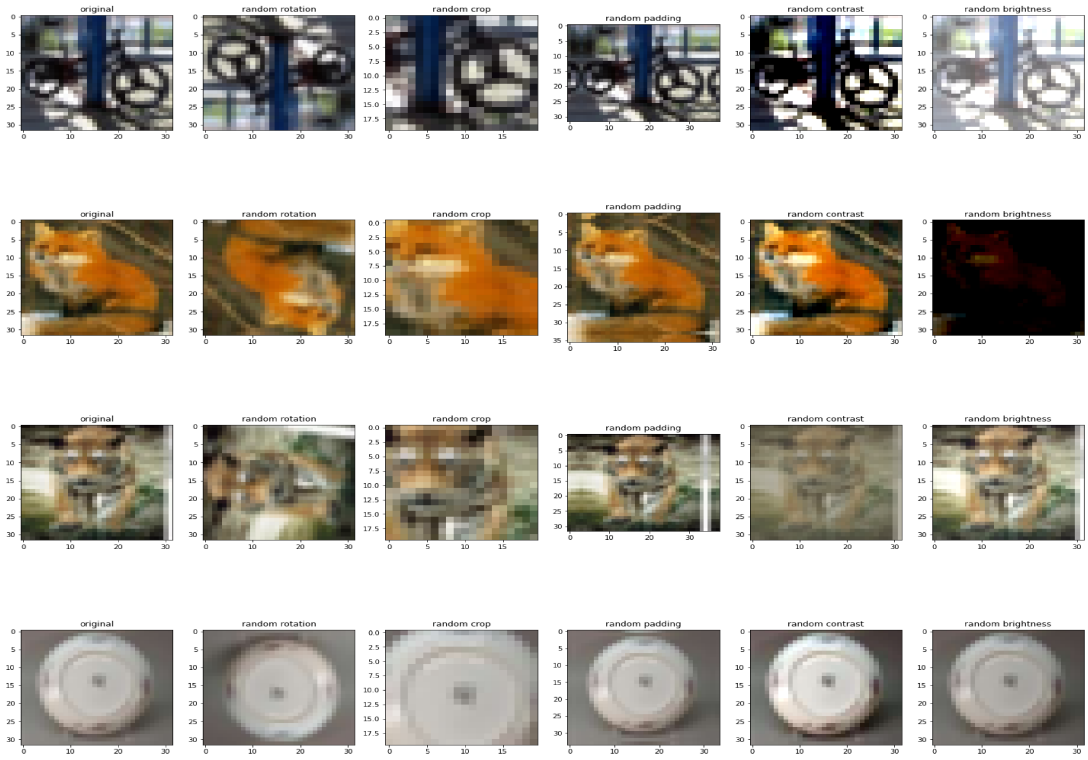

Figura 11 – Data Augmentation sobre amostras da base dados CIFAR100

3.5.4.2 Dropout

Uma das abordagens mais comuns de regularização é o *Dropout* (Srivastava et al. [2014]). A camada de dropout normalmente é inserida entre as camadas totalmente conectadas. Para cada minilote no conjunto de treinamento, as camadas de dropout desconectam aleatoriamente, com probabilidade p, os neurônios do modelo. Ao remover de forma aleatória os neurônio garante-se que nenhum nó no modelo seja o único responsável por "ativar" quando apresentado a um determinado padrão. Em vez disso, o *dropout* garante que haja vários nós redundantes que serão ativados quando apresentados com entradas semelhantes, tornando o modelo menos sensível a pequenas variações da entrada. A Figura 12 apresenta aplicação do dropout onde alguns neurônios foram desconectados com probabilidade $p = 0,5$.

3.5.4.3 Batch Normalization

A normalização em lote, ou Batch Normalization, é uma técnica que ajusta a distribuição das ativações de saída de uma camada da CNN, fazendo-a seguir uma distribuição gaussiana normal (Ioffe and Szegedy [2015]). Essa técnica é muito útil para treinar redes neurais profundas, pois ajuda a reduzir o problema de "mudança de covariância interna"das ativações de cada camada. Isso se refere ao fato de que a distribuição das ativações de uma camada pode variar enquanto os parâmetros da rede são atualizados durante o treinamento. Quando a distribuição varia muito, o treinamento pode ser lento

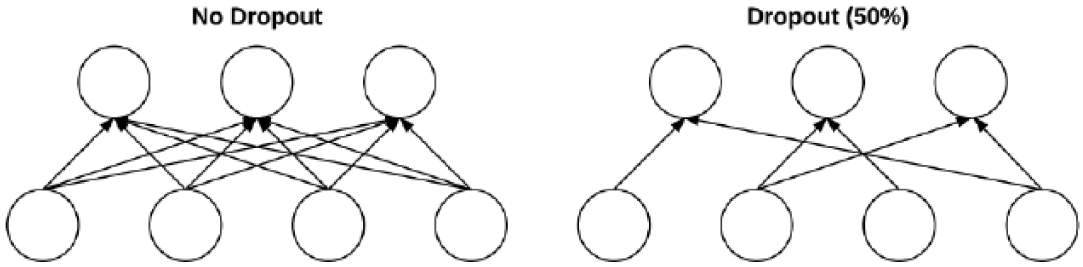

Figura 12 – **Esquerda**: Duas camadas de uma rede neural totalmente conectadas sem *dropout*. **Direita**: As mesmas duas camadas depois de descartar 50% das conexões

Fonte: Rosebrock [2017]

e a rede pode levar muito tempo para convergir. A normalização das ativações também ajuda a evitar problemas de instabilidade na rede, como a explosão ou desaparecimento dos gradientes e saturação das ativações (Khan et al. [2018]).

3.6 Resumo do Capítulo

Neste capítulo foi apresentado o referencial teórico desta dissertação, abordando alguns dos principais conceitos e definições relacionados ao aprendizado de máquina, redes neurais artificiais e redes neurais convolucionais. Inicialmente, foi discutido sobre o aprendizado de máquina, com ênfase no aprendizado supervisionado. Em seguida, foram abordadas as redes neurais artificiais, destacando os conceitos de perceptron e backpropagation, além das redes multicamadas. Também foi discutido sobre as funções de ativação. Em seguida, foi apresentado o tema das redes neurais convolucionais, incluindo as camadas de convolução, pooling e totalmente conectadas. Por fim, foram discutidos aspectos importantes do aprendizado da RNC, como a função de perda, aprendizado baseado em gradientes, inicialização e regularização de pesos. Com estes tópicos, espera-se que se tenha uma compreensão sólida dos princípios fundamentais que embasam o tema desenvolvido na dissertação, que será mais detalhado no próximo capítulo.

Capítulo 4

Operadores OWA

Neste capítulo será apresentado o operador OWA, como foi implementado as camadas *OWA-pooling* e *OWA-channel-aggregation*, e finalmente, após inserir essas camadas numa RNC, como seus pesos podem ser treinados.

Antes de aprofundar explicando o que são os operadores OWA, serão apresentados alguns conceitos preliminares, explicado a definição de função de agregação e funções de ponderação, pois os operadores OWA são operadores de agregação do tipo média ponderada ordenada.

4.1 Conceitos Preliminares

4.1.1 Função de Agregação

Uma função de agregação (Beliakov et al. [2007]) é uma função que realiza o mapeamento $f : [0,1]^n \to [0,1]$, isto é, combinam vários valores em único resultado. As funções de agregação possuem as seguintes propriedades:

1. $f(\underbrace{0,0,...,0}_{n-vezes}) = 0$ e $f(\underbrace{1,1,...,1}_{n-vezes}) = 1$
2. $x \leq y \implies f(x) \leq f(y)\ \forall\ x, y \in [0,1]^n$

A segunda propriedade indica que a função de agregação é monotônica. Isto é, a partir de duas listas X e Y de tamanho n, se para todos os valores x_i e y_i ($i \in [0, n-1]$), das lista X e Y respectivamente, $x_i \leq y_i$, então os resultado da agregação de X será menor ou igual ao resultado da agregação de Y: $f(x) \leq f(y)$.

4.1.2 Função de ponderação

Ponderar é a maneira mais comum de agregar dados. Um função de ponderação $f : [0,1]^n \to [0,1]$ possui as seguintes propriedade:

$$\min(x) \leq f(x) \leq \max(x) \forall x \in [0,1]^n \tag{4.1}$$

Casos especiais de funções de ponderação são a média aritmética, a mediana, a média geométrica, e os operadores OWA, que são a base para o desenvolvimento deste trabalho.

4.1.3 Operadores OWA

Operadores de agregação do tipo media ponderada ordenada (operadores OWA), propostos Ronald R. Yager (Yager [1988]), pertencem a uma classe de funções de agregação ponderadas. Diferentemente das médias aritméticas ponderadas, os seus pesos não estão associados com alguma entrada em particular, mas com as magnitudes das entradas. Formalmente, um operador OWA de dimensão $\mathbf{n}$ é um mapeamento $f : [0,1]^n \to [0,1]$ tendo um vetor de pesos $\mathbf{w} = [w_1, ..., w_n]$, com as condições $w_i \in [0,1]$ e $\sum_{i=1}^{n} w_i = 1$. Especificado as suas condições, a Equação 4.2 apresenta como é definido um operador OWA.

$$OWA(x_{\searrow}) = \sum_{i=1}^{n} w_i x_i \tag{4.2}$$

sendo que $x_{\searrow}$ denota o vetor obtido de $\mathbf{x}$ ordenando seus componentes em ordem decrescente $x_{(1)} \geq x_{(2)} \geq ... \geq x_{(n)}$.

A partir da definição de um operador OWA (Equação 4.2) fica evidente que o cálculo do valor de uma função OWA envolve a ordenação dos valores a serem agregados. Alguns exemplos notáveis de operadores OWA são *max* (correspondendo a $\mathbf{w} = [1, 0, ..., 0]$), *min* ($\mathbf{w} = [0, ..., 0, 1]$), e a *média aritmética* (para o qual $\mathbf{w} = 1/n, ..., 1/n$).

4.2 OWA-pooling

A camada de *OWA-pooling*, como as camadas de pooling usuais, é responsável por reduzir a dimensionalidade dos dados, conforme apresentado na seção 3.4.2. Ao aplicar a operação de pooling, espera-se manter informações relevantes da entrada enquanto remove detalhes irrelevantes e confusos. Assim, reduz o número de parâmetros a serem aprendidos pelo modelo e a quantidade de computação realizada.

Nessa camada, a primeira operação realizada é a extração dos patches do mapa de recursos de ativações de entrada. Tipicamente o tamanho do filtro, ou da janela, é 2×2 ou 3×3, com strides com as mesmas dimensões do filtro, ou uma unidade menor. A Figura 13 ilustra como é realizada a extração de patches a partir de uma mapa de ativações. Após a extração dos patches, seus valores são ordenados em ordem decrescente, para que então possam ser agregados utilizando um operador OWA. A Figura 14 exemplifica a realização da operação de pooling utilizando um operador OWA, onde cada uma das cores representa um patch e o resultado das suas respectivas agregações.

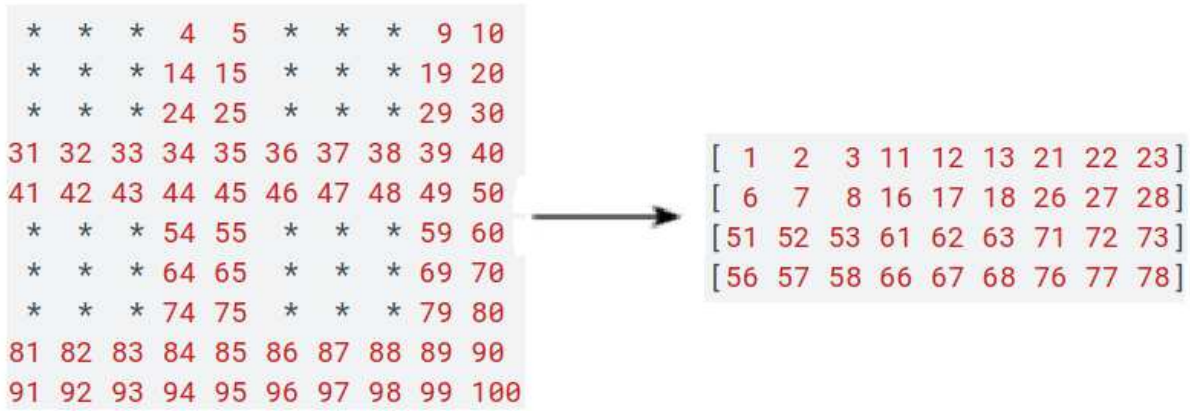

Figura 13 – Extração de patches com uma janela de tamanho 3 x 3 e stride igual a 5.

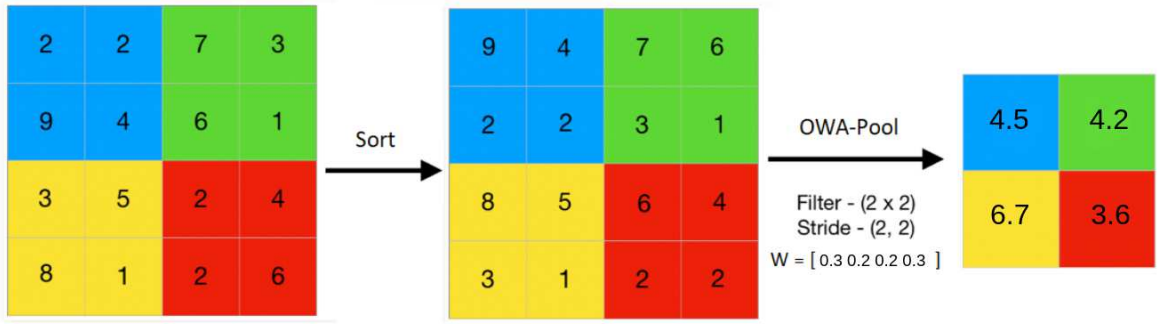

Figura 14 – Exemplo de pooling usando um oprador OWA..

A abordagem adotada nos experimento é que será aprendido um único operador OWA para todos os canais dos dados de entrada. Os pesos dos operadores OWA serão aprendidos durante o treinamento do modelo a partir de um processo iterativo, com o objetivo de gerar agregações mais discriminativas do que as obtidas nas camadas pooling usuais. Em Forcen et al. [2020] foram consideras outras abordagens ao treinar a camada *OWA-pooling*, como treinar um operador OWA distinto para cada canal ou treinar sem as restrições que definem um operador OWA (Subseção 4.1.3). Retirar as restrições fazem com que os pesos deixem de representar um operador OWA, e o foco dessa dissertação é trabalhar com operadores OWA, e em Miranda and Guimaraes [2022] foi verificado que os

melhores resultados são obtidos ao treinar um único operador OWA para todos os canais. Assim, nos experimentos dessa dissertação, será treinado apenas um operador OWA para todos os canais dos dados de entrada.

4.3 OWA-channel-aggregation

Esta camada de agregação OWA (Dominguez-Catena et al. [2020, 2021]) será inserida antes de uma camada de convolução da rede modificando os dados de entrada para essa camada de convolução. Em primeiro lugar, esta camada reordena os canais de dados de entrada de forma descendente, com base em uma métrica de ordenação global dos mapas de recursos. Depois de ordenar os os mapas de recursos (canais) dos dados de entrada, cada entrada, com resolução de de I linhas, J colunas e C_{in} canais, é convoluída com C_f filtros formados por operadores OWA, cada um definido por uma matriz de ponderação de altura e largura unitária e profundidade igual a C_{in}. Como resultado, C_f novos mapas de recursos são gerados. Esses novos mapas de recursos passam por um camada de ativação com função de ativação ReLu, obtendo assim um mapa de ativações. A camada de ativação é utilizada para garantir que seja encontrado uma mapeamento não linear dos dados de entrada e não sejam adicionados dados correlacionados ao modelo (a adição de dados correlacionados pode provocar uma perda de generalização do modelo). Os mapas de ativações então podem ser concatenados aos dados de entrada ou substituir os dados de entrada, conforme mostrado na Fig. 15. O objetivo aprender novas informações com base na ordem dos mapas de recursos de uma determinada camada e verificar se essas informações adicionais aumentam a generalização do modelo quando substituem ou são concatenadas aos dados de entrada.

Deve-se notar que a agregação não considera a altura e a largura dos mapas de recursos, como é feito na camada de pooling OWA. A agregação é baseada nos dados de cada um dos canais, para cada valor de altura $i \in I$ e largura $j \in J$.

Em Dominguez-Catena et al. [2020] foram realizados testes com diferentes funções de ordenação para os canais dos dados a serem agregados com a camada *OWA-channel-aggregation*. Os testes foram realizados na rede VGG13, utilizando os conjuntos de dados CIFAR10 e CIFAR100. Várias funções de ordenação foram consideradas, como entropia de Shannon (Shannon [1948]), soma dos valores dos canais, variação total dos valores dos canais (Rudin et al. [1992]), mediana dos valores dos canais e máximo dos valores dos canais. No final do estudo, concluiu-se que a soma das ativações supera claramente as restantes medidas. Como consequência, neste trabalho, a soma dos valores dos canais também será utilizada para ordenar os mapas de recursos, conforme a Eq. 4.3.

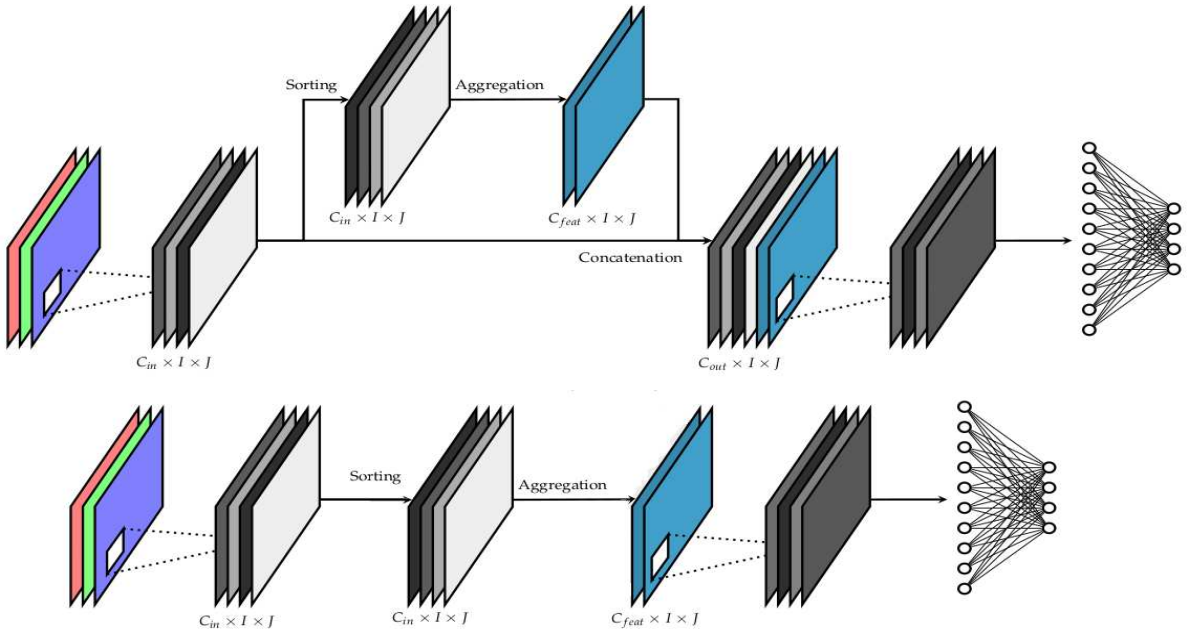

Figura 15 – Estrutura da camada OWA. Na CNN superior, os mapas de recursos recém-gerados são adicionados aos dados de entrada e, na CNN inferior, a substituição é realizada. Adaptado de Dominguez-Catena et al. [2021]..

$$S(X) = \sum_{i=1}^{I} \sum_{j=1}^{J} x_{ij} \tag{4.3}$$

dado um canal X de dimensões I × J.

4.4 Pesos OWA

Os valores dos pesos dos operadores OWA foram inicializados através de amostras de distribuição uniforme $U(0, 1)$. Esses pesos são tratados como parâmetros treináveis da RNC e aprendidos pelo processo de *backpropagation*, ao mesmo tempo que os demais parâmetros treináveis do modelo. Para garantir que os pesos OWA das camadas **OWA-pooling** e **OWA-channel-aggregation** atendam às condições que definem um operador OWA, $\mathbf{w} = [w_1, ..., w_n]$, com as condições $w_i \in [0, 1]$ e $\sum_{i=1}^{n} w_i = 1$, de acordo com a subseção 4.1.3, foram adicionadas restrições o modelo. Essas restrições são funções de projeção por variável aplicadas à variável alvo após cada atualização de gradiente (Chollet et al. [2015]). A Equação 4.4 mostra a restrição que foi aplicada aos pesos de cada operador OWA.

$$w_i = \frac{\max(w_i, 0)}{\sum_{i=1}^{f} \max(w_f, 0)}, \forall i \in f \tag{4.4}$$

sendo f o número total de pesos do operador OWA.

Capítulo 5

Experimentos e Resultados

Neste capítulo serão apresentados as bases de dados utilizadas, os modelos utilizados, alguns detalhes de implementação, os experimentos realizados e os resultados obtidos.

5.1 Base de Dados

Os experimentos foram realizados nos conjuntos de dados CIFAR-10 e CIFAR-100 (Krizhevsky et al. [2009]). CIFAR10 é um conhecido conjunto de dados composto por 60.000 imagens coloridas em uma resolução de 32x32 pixels, classificadas em 10 classes diferentes com 6.000 exemplos cada. O CIFAR100 é um conjunto de dados semelhante composto por outras 60.000 imagens coloridas de resolução de 32x32 pixels, com 100 classes diferentes, cada uma com 600 exemplos. Ambos os conjuntos de dados já estão divididos em partições de treinamento e teste, com 50.000 exemplos de treinamento e 10.000 exemplos de teste cada, com uma distribuição de classe uniforme. Para ambos os conjuntos de dados foi separado 20% do conjunto de treinamento para realizar a validação durante o treinamento. A Figura 16, a seguir, apresenta algumas amostra da base de dados CIFAR10.

5.1.1 Pré-Processamento

A seguir serão apresentadas as transformações realizadas sobre as bases de dados CIFAR10 e CIFAR100 antes de serem utilizadas para treinar os modelos.

Para os experimentos realizados na AlexNet foi adicionada mais uma etapa de pré-processamento das dados. Foi realizado um redimensionando dos dados de dimensões 32×32 para 224×224. A rede AlexNet foi projetada para trabalhar com imagens de tamanho 224×224.

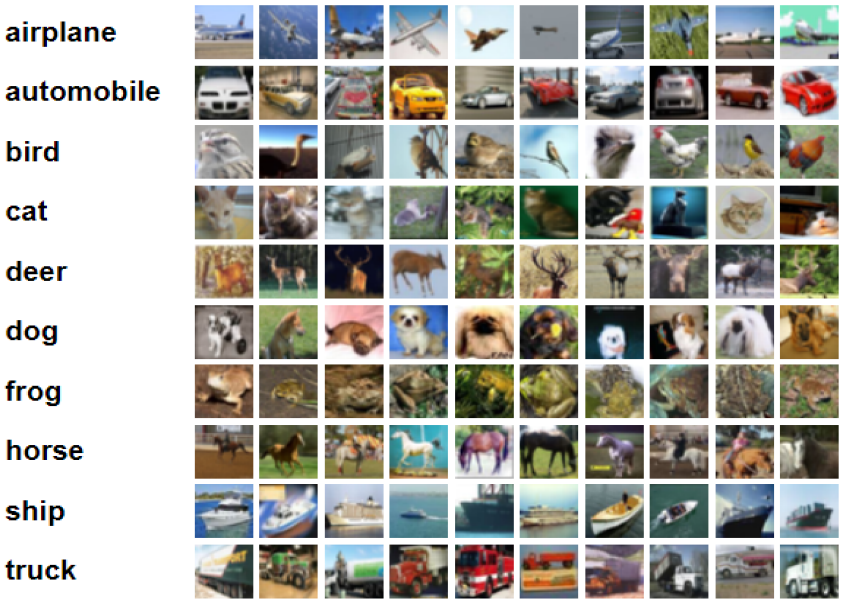

Figura 16 – Amostras da base de dados CIFAR10.

Fonte: Krizhevsky et al. [2009]

5.1.1.1 Normalização dos Dados

A normalização dos dados é uma etapa importante que garante que cada parâmetro de entrada (pixel, neste caso) tenha uma distribuição de dados semelhante. Isso torna a convergência mais rápida durante o treinamento da rede.

Uma forma comum de normalizar os dados (X), que foi adotada nos experimentos, é realizada subtraindo a média dos dados e dividindo pelo desvio padrão, conforme a Equação 5.1 a seguir:

$$X_{normalized} = \frac{(X - mean(X)}{std(X)} \tag{5.1}$$

A normalização dos dados foi feita considerando a média e desvio padrão dos valores dos pixels dos dados de treinamento.

5.1.1.2 Data Augmentation

Operações simples de *data augmentation* foram empregadas nos conjuntos de dados CIFAR10 e CIFAR100, após serem normalizados. As operações realizadas foram:

- *flip* horizontal aleatório com uma probabilidade de 0,5;
- *padding* aleatório de 1 a 4 pixels para os modelos VGG13 e NiN, e de 1 a 28 pixels para o modelo ALexNet, usando espelhamento para realizar o preenchimento;

- corte aleatório para o tamanho inicial de 32×32 para os modelos VGG13 e NiN, 224×224 para AlexNet.

Alguns exemplos de aplicação dessas operações podem ser visualizadas na Figura 11. Essas operações de *data augmentation* foram descritas no trabalho de He et al. [2016], onde foram treinadas redes residuais profundas (ResNet) sobre as bases de dados CIFAR-10 e CIFAR-100.

5.2 Arquiteturas Utilizadas

Todos os testes para avaliar o desempenho ao usar as camadas OWA-pooling e OWA-channel-aggregation foram realizados nos modelos Alexnet (Krizhevsky et al. [2012]), Network in Network (Lin et al. [2013]) e na rede VGG13 (Simonyan and Zisserman [2014]).

5.2.1 VGG13

Esta é uma arquitetura de RNC bem conhecida com 10 blocos convolucionais, cada um consistindo em uma camada convolucional, uma camada de batch normalization e uma camada de ativação ReLU. No entanto, uma pequena alteração foi feita nas camadas totalmente conectadas da rede, substituindo as últimas 3 camadas densas por uma única camada densa composta por 512 neurônios, da mesma forma que em Liu and Deng [2015]. Essa alteração reduz bastante o número de parâmetros treinados, sem impactar muito no desempenho. Entre as camadas totalmente conectadas (densas) há uma camada de dropout.

Ao invés do VGG16 ou VGG19, que são os modelos VGG mais citados na literatura, optou-se pelo VGG13, no trabalho de Dominguez-Catena et al. [2020] foi verificado que ao utilizar o VGG13 não houve perda significativa de desempenho, mas diminuiu consideravelmente o tempo de treinamento. Assim. nos experimentos desse trabalho, se optou pela VGG13.

A Tabela 1 mostra a arquitetura modificada do modelo VGG13, utilizada neste trabalho. A camada **OWA-ca** representa uma camada *OWA-channel-aggregation*, onde a saída tem dimensões $16 \times 16 \times 128 + Cf$ se usada para adicionar mais C_f mapas de recursos, ou $16 \times 16 \times Cf$ se usada para substituir os mapas de recursos de entrada. A saída de rede tem dimensão 10 ou 100 para CIFAR-10 ou CIFAR-100 respectivamente. A camada chamada de **pooling** apenas representa a camada que será substituída nos experimentos pela camada *OWA-pooling*.

Em Dominguez-Catena et al. [2020] foram considerados vários pontos de inserção da camada *OWA-channel-aggregation* na VGG13, cada uma imediatamente antes de cada uma das camadas de convolução, com exceção da primeira camada de convolução. A partir

Tabela 1 – Arquitetura da VGG13

Nome	Tamanho Kernel	Stride	Tamanho Saída
input_data	-	-	$32 \times 32 \times 3$
block1_conv1	3×3	1	$32 \times 32 \times 64$
block1_conv2	3×3	1	$32 \times 32 \times 64$
max_pooling2d_1	2×2	2	$16 \times 16 \times 64$
block2_conv1	3×3	1	$16 \times 16 \times 128$
OWA-ca	1×1	1	$16 \times 16 \times 128 + C_f$ or $16 \times 16 \times C_f$
block2_conv2	3×3	1	$16 \times 16 \times 128$
pooling	2×2	2	$8 \times 8 \times 128$
block3_conv1	3×3	1	$8 \times 8 \times 256$
block3_conv2	3×3	1	$8 \times 8 \times 256$
max_pooling2d_2	2×2	2	$4 \times 4 \times 256$
block4_conv1	3×3	1	$4 \times 4 \times 512$
block4_conv2	3×3	1	$4 \times 4 \times 512$
max_pooling2d_3	2×2	2	$2 \times 2 \times 512$
block5_conv1	3×3	1	$2 \times 2 \times 512$
block5_conv2	3×3	1	$2 \times 2 \times 512$
max_pooling2d_4	2×2	2	$1 \times 1 \times 512$
dense	-	-	512
dense	-	-	10 or 100

dos experimentos realizados, concluiu-se que a adição de novos mapas de características pouco antes da 2ª camada de convolução do 2º bloco VGG apresentou melhor desempenho. Os autores suspeitam que essa tendência de performar melhor nas primeiras camadas da rede se deve ao fato das imagens das bases dados CIFAR10 e CIFAR100 possuírem tamanho 32×32 e que a *OWA-channel-aggregation* deva performar melhor ao trabalhar com imagens de maior dimensão. Devido às camadas de pooling, nas camadas inferiores da rede, os mapas de características tendem a possuir dimensões menores.

Essa ideia será aplicada em todos os experimentos avaliando a inserção de apenas uma *OWA-channel-aggregation* e uma *OWA-pooling*. Mais detalhes sobre os experimentos se encontram nas seções 5.3 e 5.4.

5.2.2 AlexNet

O modelo AlexNet, proposto por Krizhevsky et al. [2012], foi o primeiro modelo RNC que levou ao ressurgimento de redes neurais profundas na visão computacional. Esta arquitetura ganhou o ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) em 2012 por uma grande margem. Consiste em um total de oito camadas de parâmetros, entre as quais as cinco camadas iniciais são camadas convolucionais, enquanto as três últimas camadas são camadas totalmente conectadas. A camada final totalmente conectada realiza a classificação. Entre as camadas totalmente conectadas (densas) há uma camada de dropout. Outro aspecto da AlexNet é a presença da não linearidade ReLU após cada camada convolucional e totalmente conectada (densa).

Tabela 2 – Arquitetura da AlexNet

Nome	Tamanho Kernel	Stride	Tamanho Saída
input_data	-	-	$224 \times 224 \times 3$
block1_conv1	11×11	4	$54 \times 54 \times 96$
max_pooling2d_1	3×3	2	$26 \times 26 \times 96$
OWA-ca	5×5	1	$26 \times 26 \times 96 + C_f$ or $26 \times 26 \times C_f$
block2_conv1	5×5	1	$26 \times 26 \times 256$
max_pooling2d_2	3×3	2	$12 \times 12 \times 256$
block3_conv1	3×3	1	$12 \times 12 \times 384$
block4_conv1	3×3	1	$12 \times 12 \times 384$
block5_conv1	3×3	1	$12 \times 12 \times 256$
pooling	3×3	2	$5 \times 5 \times 256$
dense	-	-	4096
dense	-	-	4096
dense	-	-	10 or 100

A Tabela 2 apresenta a arquitetura do modelo AlexNet, utilizada neste trabalho. A camada **OWA-ca** representa uma camada *OWA-channel-aggregation*, onde a saída tem dimensões $26 \times 26 \times 96 + Cf$ se usada para adicionar mais C_f mapas de recursos, ou $26 \times 26 \times Cf$ se usada para substituir os mapas de recursos de entrada. A saída de rede tem dimensão 10 ou 100 para CIFAR10 ou CIFAR100 respectivamente. A camada chamada de **pooling** apenas representa a camada que será substituída nos experimentos pela camada *OWA-pooling*.

5.2.3 NiN

O modelo Network in Network (NiN) (Lin et al. [2013]) é um modelo de RNC simples e leve que geralmente funciona muito bem em conjuntos de dados de pequena escala, como o CIFAR. Na NiN foram introduzidas duas novas ideias de design de RNCs. **Primeiro**, demonstra que adicionar camadas totalmente conectas entre as camadas de convolução é útil no treinamento da rede. A arquitetura da rede consiste em três blocos de camadas onde cada possui uma camada de convolução, seguida por um par de camadas totalmente conectadas (ou camadas convolucionais com tamanhos de filtro 1×1) e uma camada de pooling máximo. Nos dois primeiros blocos é inserida uma cada de *dropout* após a camada de *max pooling*. **Segundo**, essa arquitetura utiliza um uma camada de *average pooling* global no final do modelo como um regularizador. Este esquema de agrupamento combina todas as ativações dentro de cada mapa de recursos (através do calculo da média), obtendo uma pontuação de classificação única que é encaminhada para ultima camada densa que realiza a classificação.

Tabela 3 – Arquitetura da NiN

Nome	Tamanho Kernel	Stride	Tamanho Saída
input_data	-	-	$32 \times 32 \times 3$
block1_conv1	5×5	1	$32 \times 32 \times 192$
block1_conv2	1×1	1	$32 \times 32 \times 160$
block1_conv3	1×1	1	$32 \times 32 \times 96$
max_pooling2d_1	3×3	2	$16 \times 16 \times 96$
block2_conv1	5×5	1	$16 \times 16 \times 192$
OWA-ca	1×1	1	$16 \times 16 \times 192 + C_f$ or $16 \times 16 \times C_f$
block2_conv2	1×1	1	$16 \times 16 \times 192$
block2_conv3	1×1	1	$16 \times 16 \times 192$
max_pooling2d_1	3×3	2	$8 \times 8 \times 192$
block3_conv1	5×5	1	$8 \times 8 \times 192$
block3_conv2	1×1	1	$8 \times 8 \times 192$
block3_conv3	1×1	1	$8 \times 8 \times 10$
pooling	8×8	1	$1 \times 1 \times 10$
dense	-	-	10

A Tabela 3 apresenta a arquitetura do modelo NiN, utilizada neste trabalho. A camada **OWA-ca** representa uma camada *OWA-channel-aggregation*, onde a saída tem dimensões $16 \times 16 \times 192 + Cf$ se usada para adicionar mais C_f mapas de recursos, ou

$16 \times 16 \times Cf$ se usada para substituir os mapas de recursos de entrada. A saída de rede tem dimensão 10 ou 100 para CIFAR10 ou CIFAR100 respectivamente. A camada chamada de **pooling** apenas representa a camada que será substituída nos experimentos pela camada *OWA-pooling*.

5.3 Detalhes de Implementação

Quanto ao custo computacional necessário para treinar um modelo de rede neural convolucional, é comum encontrar trabalhos que realizam apenas uma execução do mesmo modelo no mesmo conjunto de dados. Os conjuntos de dados CIFAR10 e CIFAR100 possuem divisão de conjunto de treinamento e conjunto de teste, isso permite uma comparação entre os modelos, porém o desempenho do modelo pode variar de acordo com os valores iniciais dos pesos e a ordem de apresentação das imagens à CNN.

Ao substituir as camadas de pooling usuais pelas camadas de *OWA-pooling*, observou-se um aumento significativo no tempo de treinamento dos modelos, uma vez que a operação de extração de *patches* se mostrou muito custosa computacionalmente. Em decorrência disso, neste trabalho, foi realizada apenas uma execução para cada um dos experimentos realizados, utilizando o método de avaliação *hold-out*. Esse método consiste em separar uma parte dos dados para testar o modelo e outra parte para treiná-lo. Para garantir uma comparação justa com o modelo de referência, fixou-se um número de semente ("*seed*") para gerar números pseudo-aleatórios, de modo que os mesmos pesos iniciais do modelo, a mesma divisão em lotes e outros valores gerados aleatoriamente durante o treinamento fossem utilizados.

Em todos os experimentos o modelo foi treinado do zero por 30 épocas (conforme [Dominguez-Catena et al. [2020]], onde os modelos também foram treinado por 30 épocas), com a taxa de aprendizado inicializada com o valor de 1×10^{-3}, sendo reduzida pela metade ao estabilizar por 3 épocas seguidas. O tamanho do lote foi definido como 32. O otimizador escolhido foi o gradiente descendente estocástico, com momento igual a 0,9. Os pesos das camadas convolução e das camadas densas foram inicializados usando a distribuição uniforme de He (He et al. [2015]).

A linguagem Python foi utilizada em conjunto com o framework Keras [1] para o desenvolvimento das camadas OWA-pooling e OWA-channel-aggregation e realização dos testes. Os modelos foram treinados usando o Google Collaboratory [2]. Devido a utilização do Google Collaboratory não foi monitorado o tempo de treinamento dos modelos, pois a disponibilidade das GPUs de maior desempenho é dinâmica.

Visando fomentar a colaboração e permitir que outros pesquisadores possam

[1] https://keras.io/
[2] colab.research.google.com

reproduzir os experimentos e aprimorar o trabalho desenvolvido, os principais códigos implementados neste projeto estão disponíveis em um repositório do Github. Para acessá-los, basta acessar o seguinte link: https://github.com/Leonamrsm/Application_of_OWA_Operator_on_CNNs.

5.4 Configuração dos Experimentos

Dez diferentes configurações de experimentos foram realizados, em cada uma das 3 arquiteturas avalidadas (VGG13, AlexNet e Nin), que representam diferentes combinações das camadas **OWA-pooling** e **OWA-channel-aggregation**. Foram definidas as seguintes siglas para identificar cada um dos experimentos realizados:

- **BS**
- **OWAPL**
- **OWACA$_{\mathbf{C}}$ - HALF**
- **OWACA$_{\mathbf{C}}$ - HALF - OWAPL**
- **OWACA$_{\mathbf{C}}$ - SAME**
- **OWACA$_{\mathbf{C}}$ - SAME - OWAPL**
- **OWACA$_{\mathbf{R}}$ - HALF**
- **OWACA$_{\mathbf{R}}$ - HALF - OWAPL**
- **OWACA$_{\mathbf{R}}$ - SAME**
- **OWACA$_{\mathbf{R}}$ - SAME - OWAPL**

A sigla **BS** (Base Model) indica os experimentos realizados sem qualquer modificação nas arquiteturas apresentadas nas tabelas da seção 5.2. Já a sigla OWAPL (OWA-Pooling Layer) indica os experimentos que foram realizados substituindo a camada **pooling** em destaque nas tabelas pela camada *OWA-pooling*. A sigla OWACA$_{\mathrm{C}}$ indica que foi utilizada a camada **OWA-channel-aggregation** para **concatenar** novos canais, com opções HALF e SAME que indicam, respectivamente, que o número de canais finais é metade ou igual ao número de canais da camada anterior. A sigla OWACA$_{\mathrm{R}}$ indica que foi utilizada a camada **OWA-channel-aggregation** para **substituir** os canais de entrada, com opções HALF e SAME que indicam, respectivamente, que o número de canais finais é metade ou igual ao número de canais da camada anterior.

5.5 Resultados dos Experimentos

Os resultados dos experimentos sobre as arquiteturas VGG13, AlexNet e NiN, se encontram nas Tabelas 4, 5 e 6, respectivamente.

5.6 Análise de Resultados

A partir dos resultados obtidos nos experimentos observa-se que no geral os melhores resultados foram obtidos na configuração OWAPL, onde foi inserida uma camada

Tabela 4 – VGG13 - Acurácias nas bases de dados CIFAR10 e CIFAR100 em diferentes configurações do modelo

Configuração	CIFAR10 (% acc)	CIFAR100 - acurácia top 5 (% acc)
BS	88.37	80.26
OWAPL	**89.13**	**81.51**
$OWACA_C$ - HALF	88.31	79.29
$OWACA_C$ - HALF - OWAPL	88.83	80.01
$OWACA_C$ - SAME	88.33	79.14
$OWACA_C$ - SAME - OWAPL	88.53	80.61
$OWACA_R$ - HALF	87.30	74.15
$OWACA_R$ - HALF - OWAPL	87.43	74.40
$OWACA_R$ - SAME	86.89	75.62
$OWACA_R$ - SAME - OWAPL	86.52	75.77

Tabela 5 – AlexNet - Acurácias nas bases de dados CIFAR10 e CIFAR100 em diferentes configurações do modelo

Configuração	CIFAR10 (% acc)	CIFAR100 - acurácia top 5 (% acc)
BS	87.31	81.06
OWAPL	**87.92**	**82.34**
$OWACA_C$ - HALF	85.74	79,92
$OWACA_C$ - HALF - OWAPL	86.83	81.34
$OWACA_C$ - SAME	87.27	79.26
$OWACA_C$ - SAME - OWAPL	87.88	80.83
$OWACA_R$ - HALF	84.50	78.10
$OWACA_R$ - HALF - OWAPL	85.14	80.76
$OWACA_R$ - SAME	84.32	77.56
$OWACA_R$ - SAME - OWAPL	84,52	79.86

OWA-pooling, para as bases de dados CIFAR10 e CIFAR100. A única exceção foram os experimentos realizados sobre o modelo NiN sobre a base de dados CIFAR100, onde os melhores resultados foram obtidos pelo modelo base, mas foram minimamente superiores comparando com a configuração OWAPL. Essa diferença talvez se deva ao fato de ter feito a inserção da camada *OWA-pooling* no final da arquitetura para realizar a operação de pooling global. Nas arquiteturas VGG13 e AlexNet o ponto onde foi feita a inserção da camada *OWA-pooling* recebe entradas com maiores dimensões.

Era esperado que ao utilizar a camada *OWA-pooling* juntamente com a camada

Tabela 6 – NiN - Acurácias nas bases de dados CIFAR10 e CIFAR100 em diferentes configurações do modelo

Configuração	CIFAR10 (% acc)	CIFAR100 - top 5 ACC (% acc)
BS	83.39	**84.99**
OWAPL	**84.85**	84.91
$OWACA_C$ - HALF	82.81	85.14
$OWACA_C$ - HALF - OWAPL	83.00	85.85
$OWACA_C$ - SAME	83.72	84.95
$OWACA_C$ - SAME - OWAPL	83.93	85.29
$OWACA_R$ - HALF	75.23	81.57
$OWACA_R$ - HALF - OWAPL	76.76	81.83
$OWACA_R$ - SAME	76.38	78.86
$OWACA_R$ - SAME - OWAPL	75.85	79.05

OWA channel-aggregation, melhores resultados seriam alcançados, pois a capacidade do modelo foi aumentada, já que foi aumentado a quantidade de parâmetros treináveis. No entanto, em alguns experimentos, a utilização dessas duas técnicas simultaneamente não resultou em melhores resultados do que os obtidos pelo modelo base, o que não permite afirmar com precisão se a utilização das duas técnicas juntas é positiva.

Nos casos em que apenas foi utilizada apenas a camada *OWA channel-aggregation* para concatenar novos mapas de recursos com os mapas de recursos de entrada, somente alguns experimentos retornaram resultados ligeiramente superiores ao do modelo base. Esses resultados são condizentes com o artigo que propos a camada *OWA channel-aggregation* (Dominguez-Catena et al. [2020]), onde somente alguns experimentos foram só um pouco superiores ao modelo base (menos de 1%).

Nos casos em que os mapas de recursos de entrada foram substituídos por mapas de recursos gerados pela camada *OWA channel-aggregation*, observou-se que houve uma perda significativa de desempenho, indicando que uma grande quantidade de informações significativas codificadas nos mapas de recursos de dados de entrada são perdidos ao agregá-los usando um operador OWA.

A inclusão da camada *OWA-pooling* foi altamente benéfica para aumentar a capacidade de generalização dos modelos. Essa afirmação foi comprovada em todos os experimentos em que a camada foi utilizada, apresentando resultados superiores em comparação aos experimentos que não contaram com sua utilização. Em outras palavras, a camada *OWA-pooling* se mostrou eficaz na melhoria da capacidade de generalização dos modelos, resultando em uma melhor performance nos dados de teste.

Por fim, devido ao fato de que a base de dados CIFAR-100 possuir 100 classes, em comparação com apenas 10 na base de dados CIFAR-10, observa-se que é mais desafiador para o modelo prever corretamente a classe correta. Isso se deve ao fato de que, apesar da quantidade de dados ser a mesma em ambas as bases, a CIFAR-100 possui uma complexidade adicional devido à quantidade maior de classes. Isso pode ser observado nos resultados obtidos, onde o desempenho foi inferior em todos os experimentos na base CIFAR-100 em comparação com o desempenho obtido na base CIFAR-10.

Deve-se notar que ao usar a camada *OWA-pooling*, o tempo de treinamento aumenta de 10 a 20 vezes, quando comparado como modelo base. Como as GPUs fornecidas pelo *google Colaboratory* possuem têm desempenho variável, os tempos de treinamento não foram salvos.

Capítulo 6

Conclusão

Em conclusão, este trabalho investigou a eficácia da utilização conjunta e separada das camadas OWA-pooling e OWA-channel-aggregation em diferentes modelos de redes neurais convolucionais, como VGG13, AlexNet e NiN, nas bases de dados CIFAR-10 e CIFAR-100. Os resultados obtidos foram promissores e sugerem que a utilização de operadores OWA para realizar agregações em redes neurais é uma abordagem que vale a pena ser considerada em trabalhos futuros.

Os resultados obtidos mostraram que a utilização conjunta das duas camadas foi positiva em alguns experimentos quando comparado com os modelos base, no entanto, os melhores resultados foram obtidos utilizando somente a camada OWA-pooling. Entretanto, o tempo de treinamento aumentou significativamente ao utilizar a camada *OWA-pooling*. Ao aumentar a capacidade dos modelos, utilizando as camadas *OWA-pooling* e *OWA-channel-aggregation*, provocou um perda de acurácia sobre o conjunto de testes nos experimentos, indicando que ocorreu sobreajuste ou que os modelos de maior capacidade não foram treinados por tempo suficiente.

Apesar de não terem proporcionado resultados capazes de impulsionar significativamente o estado da arte, os experimentos realizados com a camada *OWA-channel-aggregation* sugerem que ela pode ser uma abordagem valiosa para aumentar a quantidade de informações disponíveis para as redes neurais convolucionais, dependendo do modelo e da base de dados utilizados.

Como propostas para trabalhos futuros, é sugerido realizar análises estatísticas mais aprofundadas, repetindo os experimentos para obter maior certeza sobre a melhora no desempenho. É importante considerar que, ao comparar modelos com diferentes capacidades, treinar todos os modelos com a mesma quantidade de épocas pode não ser a melhor abordagem, uma vez que modelos com diferentes capacidades podem ter tempos de convergência distintos. Portanto, seria interessante treinar os modelos até que a performance se estabilize. Ademais, é recomendado testar essa abordagem em redes mais complexas e em diferentes bases de dados, para avaliar sua generalidade e robustez.

Por fim, é importante destacar que a camada OWA-pooling foi implementada utilizando a biblioteca Tensorflow. No entanto, verificou-se que a operação de extração de *patches* é bastante custosa. Como outra proposta de trabalho futuro, sugere-se o desenvolvimento da operação OWA-pooling em C++, como sugerido pela própria documentação da biblioteca Tensorflow, para criação de uma operação personalizada. Isso permitiria um ganho significativo de performance, possibilitando uma análise estatística aprofundada dos experimentos.

Referências

C. C. Aggarwal et al. Neural networks and deep learning. *Springer*, 10:978–3, 2018.

S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

D. T. Anderson, G. J. Scott, M. A. Islam, B. Murray, and R. Marcum. Fuzzy choquet integration of deep convolutional neural networks for remote sensing. In *Computational Intelligence for Pattern Recognition*, pages 1–28. Springer, 2018.

G. Beliakov. How to build aggregation operators from data. *International Journal of Intelligent Systems*, 18(8):903–923, 2003.

G. Beliakov, A. Pradera, T. Calvo, et al. *Aggregation functions: A guide for practitioners*, volume 221. Springer, 2007.

C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

Y.-L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

F. Chollet et al. Keras. GitHub, 2015. URL https://github.com/fchollet/keras.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989a.

G. Cybenko. Approximation by superpositions of a sigmoidal function - mathematics of control, signals, and systems, 1989b. URL https://link.springer.com/article/10.1007/BF02551274.

L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

C. A. Dias, J. Bueno, E. N. Borges, S. S. Botelho, G. P. Dimuro, G. Lucca, J. Fernandéz, H. Bustince, and P. L. J. Drews Junior. Using the choquet integral in the pooling layer in deep learning networks. In *North american fuzzy information processing society annual conference*, pages 144–154. Springer, 2018.

I. Dominguez-Catena, D. Paternain, and M. Galar. Additional feature layers from ordered aggregations for deep neural networks. In *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8. IEEE, 2020.

I. Dominguez-Catena, D. Paternain, and M. Galar. A study of owa operators learned in convolutional neural networks. *Applied Sciences*, 11(16):7195, 2021.

R. A. Dunne and N. A. Campbell. On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, volume 181, page 185. Citeseer, 1997.

J. I. Forcen, M. Pagola, E. Barrenechea, and H. Bustince. Learning ordered pooling weights in image classification. *Neurocomputing*, 411:45–53, 2020.

A. Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. "O'Reilly Media, Inc.", 2022.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

M. Grabisch, J.-L. Marichal, R. Mesiar, and E. Pap. *Aggregation functions*, volume 127. Cambridge University Press, 2009.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

F. Herrera and L. Martínez. A model based on linguistic 2-tuples for dealing with multigranular hierarchical linguistic contexts in multi-expert decision-making. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 31(2):227–234, 2001.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

A. Jabbar, X. Li, and B. Omar. A survey on generative adversarial networks: Variants, applications, and training. *ACM Computing Surveys (CSUR)*, 54(8):1–49, 2021.

A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

H. J. Jie and P. Wanda. Runpool: A dynamic pooling layer for convolution neural network. *Int. J. Comput. Intell. Syst.*, 13(1):66–76, 2020.

N. Jin, J. Wu, X. Ma, K. Yan, and Y. Mo. Multi-task learning model based on multi-scale cnn and lstm for sentiment classification. *IEEE Access*, 8:77060–77072, 2020.

A. Karpathy, F. Li, and J. Johnson. Cs231n convolutional neural networks for visual recognition (2016). http://cs231n.github.io, 2017. [Online; acessado em 10/03/2023].

J. M. Keller, D. Liu, and D. B. Fogel. *Fundamentals of computational intelligence: neural networks, fuzzy systems, and evolutionary computation*. John Wiley & Sons, 2016.

S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun. A guide to convolutional neural networks for computer vision. *Synthesis lectures on computer vision*, 8(1):1–207, 2018.

A. Kishor, A. K. Singh, S. Sonam, and N. R. Pal. A new family of owa operators featuring constant orness. *IEEE Transactions on Fuzzy Systems*, 28(9):2263–2269, 2019.

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. *cs.utoronto.ca*, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616, 2009.

M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*, pages 730–734. IEEE, 2015.

L. R. Miranda and F. G. Guimaraes. Application of learned owa operators in pooling and channel aggregation layers in convolutional neural networks. In *Anais do XIX Encontro Nacional de Inteligência Artificial e Computacional*, pages 567–578. SBC, 2022.

T. M. Mitchell and T. M. Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

S. S. Nath, G. Mishra, J. Kar, S. Chakraborty, and N. Dey. A survey of image classification methods and techniques. In *2014 International conference on control, instrumentation, communication and computational technologies (ICCICCT)*, pages 554–557. IEEE, 2014.

K. O'Shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

M. Pagola, J. I. Forcen, E. Barrenechea, C. Lopez-Molina, and H. Bustince. Use of owa operators for feature aggregation in image classification. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6. IEEE, 2017.

S. R. Price, S. R. Price, and D. T. Anderson. Introducing fuzzy layers for deep learning. In *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6. IEEE, 2019.

A. Rosebrock. *Deep Learning for Computer Vision with Python*. PyImageSearch.com, 2017.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3):247–278, 2021.

G. J. Scott, R. A. Marcum, C. H. Davis, and T. W. Nivin. Fusion of deep convolutional neural networks for land cover classification of high-resolution imagery. *IEEE Geoscience and Remote Sensing Letters*, 14(9):1638–1642, 2017.

C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

J. Stuart and P. Norvig. Artificial intelligence: a modern approach prentice-hall. *A Simon & Schuster Company Englewood Cliffs*, 2010.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

S. Targ, D. Almeida, and K. Lyman. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

C. Veal, A. Yang, A. Hurt, M. A. Islam, D. T. Anderson, G. Scott, J. M. Keller, T. C. Havens, and B. Tang. Linear order statistic neuron. In *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6. IEEE, 2019.

Z. Xu. An overview of methods for determining owa weights. *International journal of intelligent systems*, 20(8):843–865, 2005.

R. R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on systems, Man, and Cybernetics*, 18(1):183–190, 1988.

R. R. Yager. Families of owa operators. *Fuzzy sets and systems*, 59(2):125–148, 1993.

D. Yu, H. Wang, P. Chen, and Z. Wei. Mixed pooling for convolutional neural networks. In *International conference on rough sets and knowledge technology*, pages 364–375. Springer, 2014.

M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

S.-M. Zhou, F. Chiclana, R. I. John, and J. M. Garibaldi. Type-1 owa operators for aggregating uncertain information with uncertain weights induced by type-2 linguistic quantifiers. *Fuzzy Sets and Systems*, 159(24):3281–3296, 2008.

Z.-H. Zhou. *Machine learning*. Springer Nature, 2021.