UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Marcos Felipe Vendramini Carvalho

Generative Models for Open Set Image Recognition

Belo Horizonte
2021

Marcos Felipe Vendramini Carvalho

Generative Models for Open Set Image Recognition

Final Version

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Jefersson Alex Dos Santos
Co-Advisor: Alexei Manso Correa Machado

Belo Horizonte
2021

Carvalho, Marcos Felipe Vendramini

C331g Generative models for open set image recognition [manuscrito] / Marcos Felipe Vendramini Carvalho — 2021.
94 f. il.

Orientador: Jefersson Alex dos Santos.
Coorientador: Alexei Manso Correa Machado.
Dissertação (mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação
Referências: f. 91-94.
1. Computação – Teses. 2. Visão computacional –Teses. 3 Redes neurais (Computação) – Teses. 4. Reconhecimento de conjunto aberto –Teses. I. Santos, Jefersson Alex dos. II. Machado, Alexei Manso Correa. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. IV.Título.

CDU 519.6*82.10 (043)

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa
CRB 6/1510 – Insituto de Ciências Exatas da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Generative Models for Open Set Image Recognition

MARCOS FELIPE VENDRAMINI CARVALHO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]

Prof. Jefersson Alex dos Santos - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]

Prof. Alexei Manso Corrêa Machado - Coorientador
Departamento de Anatomia e Imagem - UFMG

Hugo Neves de Oliveira

Doutor Hugo Neves de Oliveira
Instituto de Matemática e Estatísitca - USP

Matheus Pinheiro Ferreira

Prof. Matheus Pinheiro Ferreira
Departamento de Ciência e Tecnologia - Instituto Militar de Engenharia

[illegible]

Prof. Fabrício Murai Ferreira
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 10 de Dezembro de 2021.

Dedico esse trabalho aos meus pais.

Acknowledgments

Agradeço à minha família e amigos por todo suporte e apoio nesta e em todas as etapas da minha vida. Obrigado por estarem sempre do meu lado.

Também agradeço aos meus orientadores e ao Hugo por toda a confiança, apoio e ensinamento ao longo do desenvolvimento deste trabalho. Por fim, agradeço também à Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) pelo projeto "CADCOVID-19: Monitoramento e Auxílio ao Diagnóstico de Pacientes com COVID-19 Usando Radiografias Torácicas e Deep Learning" (APQ-00519-20).

“*Can’t be one-hundred if you’re only giving ninety-five.*”

(All Time Low)

Resumo

Os métodos de classificação de imagens geralmente são treinados para realizar previsões levando em consideração um grupo predefinido de classes conhecidas. Problemas do mundo real, no entanto, podem não permitir um conhecimento completo de todas as entrada e rótulos do espaço, fazendo com que as falhas no reconhecimento seja um problema para o aprendizado visual profundo. Os métodos de reconhecimento de conjunto aberto são caracterizados pela capacidade de identificar corretamente as entradas de classes conhecidas e desconhecidas. Neste contexto, propomos GeMOS: módulos de reconhecimento de conjunto aberto simples que podem ser anexados a Redes Neurais Profundas pré-treinadas para reconhecimento visual. O framework GeMOS emparelha redes neurais convolucionais pré-treinadas com modelos generativos para introduzir o reconhecimento de conjunto aberto através da extração de pontuações para cada amostra, permitindo o reconhecimento de falha em tarefas de reconhecimento de objeto. Conduzimos uma avaliação completa do método proposto em comparação com algoritmos do estado-da-arte de conjunto aberto. Nesses testes foram utilizados diferentes datasets como dentro e fora da distribuição, onde, com o MNIST dentro da distribuição, atingimos o F1-score de 0.91 enquanto o melhor baseline do teste referente atingiu 0.85, e, para o CIFAR10 dentro da distribuição, atingimos o F1-score de 0.93 enquanto o melhor baseline do teste referente atingiu 0.81. Também foram realizados teste utilizando um mesmo dataset como dentro e fora da distribuição, um caso mais complexo que mostrou a dependência do método a acurácia das redes pré treinadas. Os resultados mostraram que o GeMOS compete com modelos mais complexos e caros e em muitos casos os superam. Para os trabalhos futuros propomos inicialmente aplicar o método a outros domínios e a problemas do mundo real, e modificar o método para outras tarefas de visão computacional.

Palavras-chave: Visão Computacional, Redes Neurais, Reconhecimento de Conjunto Aberto.

Abstract

Image classification methods are usually trained to perform predictions taking into account a predefined group of known classes. Real-world problems, however, may not allow for a full knowledge of the input and label spaces, making failures in recognition a hazard to deep visual learning. Open set recognition methods are characterized by the ability to correctly identify inputs of known and unknown classes. In this context, we propose GeMOS: simple and plug-and-play open set recognition modules that can be attached to pre-trained Deep Neural Networks for visual recognition. The GeMOS framework pairs pre-trained Convolutional Neural Networks with generative models for open set recognition to extract open set scores for each sample, allowing for failure recognition in object recognition tasks. We conduct a thorough evaluation of the proposed method against state-of-the-art open set algorithms. In these tests, different datasets were used, such as in and out of distribution, with the MNIST as in distribution, we reached the F1-score of 0.91 while the best baseline of the referent test reached 0.85, and, for the CIFAR10 as in distribution, we reached the F1-score of 0.93 while the best baseline of the benchmark test reached 0.81. Tests were also performed using the same dataset as in and out of distribution, a more complex case that showed the dependence of the method on the accuracy of pre-trained networks. The results showed that GeMOS competes with more complex and expensive models and in many cases outperforms them. For future work, we initially propose to apply the method to other domains and real-world problems, and to modify the method for other computer vision tasks.

Keywords: Computer Vision, Neural Networks, Open Set Recognition.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Context and Motivation

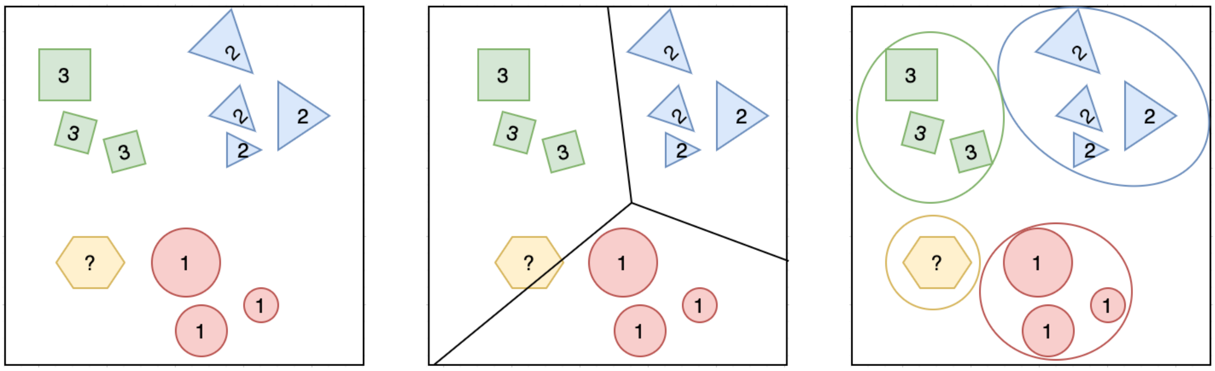
Nowadays images can be acquired by many gadgets, such as cameras, satellites, thermal cameras, x-rays, and tomography, each of which can capture a type of spectral information, the fact that contributes to the increase of datasets size, type, and the number of classes. This growing of image data jointly with the increasing of computing power creates a favorable scenario for the improvement of neural networks in the area of computer vision. The process of extraction and recognition of information from images is slow and costly, so its automation would contribute to several applications in medicine, agriculture, urbanization, autonomous driving, remote sensing, security, and many others [25].

In this context, image classification has become fundamental in computer vision. This task is typically modelled as a discriminative, supervised learning problem. An image classifier model is trained from a given set of images associated with a known number of classes, characterized as a closed set scenario [14]. The model is expected to be effective while assigning a new image to the correct class. However, it is not capable of correctly labelling an image that belongs to an unknown class as this image will be wrongly assigned to one of the known classes.

On the other hand, Open Set Recognition (OSR) is characterized as the ability to correctly classify inputs of known and unknown classes. The main challenges of OSR are:

- To group several unknown classes;
- To differentiate outliers from unknown classes;
- To divide similar unknown and known classes;
- To handle the imbalance between the number of unknown and known examples;
- To classify the dataset without considering a fixed number of classes.

According to [33], during the inference phase, an OSR system should be able to correctly classify the instances of the classes used during the training (Known Known Classes – KKCs) whereas recognizing the samples of classes that were not seen during training (Unknown Unknown Classes – UUCs). The differences between open and closed set scenarios are exemplified in Figure 1.1 were given a dataset with the known classes “1”, “2” and “3”, in a closed set scenario the space is divided completely between the three classes while in an open set scenario a limit is set for each class without taking up all the space. In this way, an unknown object in a closed scenario will be found inside the space of a class that it does not belong to.



(a) Dataset. (b) Closed set space. (c) Open set space.

Figure 1.1: Example of an OSR scenario. Given a dataset (*a*) with three known classes (“1”, “2” and “3”) and one unknown class (“?”), closed set methods partition the space considering only known classes (*b*) while in open set methods each known class is assigned a limited space.

Based on this definition, the main difference between closed and open set scenarios can be associated with the degree of knowledge of the world, for example, the awareness of all possible classes. Specifically, while in the closed set scenario the methods assume full knowledge of the world, open set approaches must assume that they do not know all the possible classes during the training. Different approaches may have distinct degrees of knowledge depending on the problem’s domain. As deep visual learning assumes full knowledge of class space, traditional implementation of Convolutional Neural Networks (CNNs) is inherently closed set, rendering them unsuitable for detecting recognition failures on their own.

[13] categorize types of classes in categories based on their existence and significance during the training and test steps. These categories are exemplified in Figure 1.2, and can be defined as:

- Known known classes (KKCs): Composed of classes with positive (useful for the problem as a target) and negative (non significant classes) samples and labels;

- Known unknown classes (KUCs): Composed of classes with negative labels that were not grouped into significant classes (usually background classes);

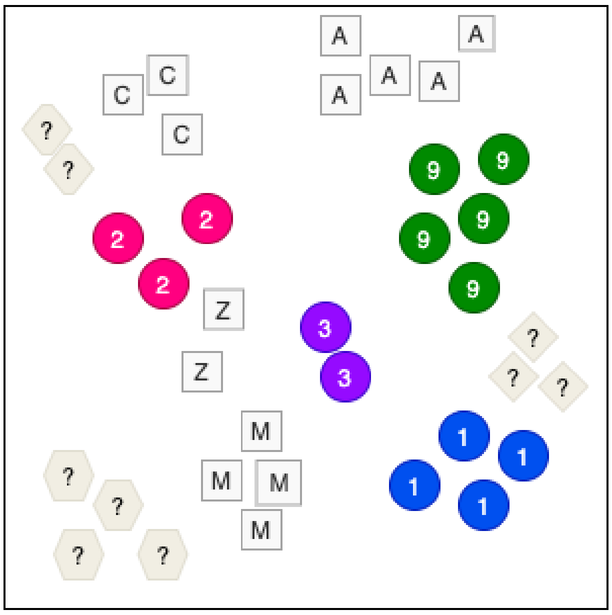


Figure 1.2: An example of KKCs, KUCs, and UUCs in the space. Considering a dataset composed of numbers and letters for a task of number classification, the circles "1", "2", "3" and "9" are KKCs, while the remaining letter classes in it are KUCs. The remaining classes "?" are UUCs.

- Unknown known classes (UKCs): Composed of classes without samples in training, only metadata;
- Unknown unknown classes (UUCs): Composed of classes without any information during training.

Aiming to adapt Deep Neural Networks (DNNs) for open set scenarios, recognizing UUCs, in this dissertation a novel plug-and-play method called **Ge**nerative **M**odels for **O**pen **S**et recognition (GeMOS) [39] is proposed to opening pre-trained closed set CNNs. To avoid the costly training of additional deep models, a simpler, shallower, generative modelling is used to approximate the likelihood of each sample pertaining or not to the data distribution.

The GeMOS method consists of a pre-trained network, to classify KKCs, and a generative model for each known class. The generative models are trained from activations extracted from the backbone network and are responsible for identifying UUCs. The exemplification of the method is shown in Figure 1.3. To evaluate the performance of generative models among themselves and in comparison with state-of-the-art baselines, thorough evaluations are conducted comparing different backbones, generative models, and datasets.

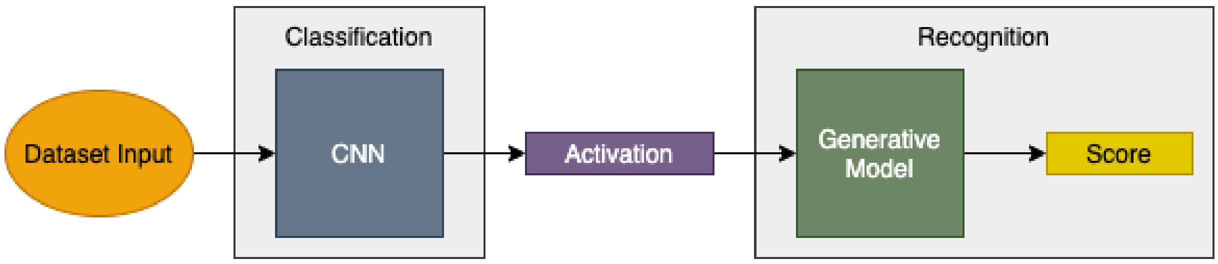


Figure 1.3: GeMOS method overview. Given a dataset, GeMOS uses a pre-trained CNN to extract the activation vectors, which is used to train the generative models. With this process, the CNN is responsible to classify KKCs inputs while the Generative Models Recognize UUCs based on the generated score.

1.2 Objectives

The main objective of this dissertation is to develop a new image recognition method for open set scenarios. Specific objectives include:

- Proposing a new, Open Set Recognition method named GeMOS [39];
- Comprehending the most important open set image recognition methods;
- Verifying the performance of this method for known and unknown classes in different domains;
- Verifying the performance of this method for known and unknown classes with different backbones and generative models;
- Analyzing the results of this method for KKCs and UUCs in comparison with other image recognition techniques.

1.3 Contribution

The contributions of this work include:

- A new method for OSR capable of opening closed set deep models without the need to train a new model;
- The evaluation and comparison of different generative models for class separation, considering unimodal and multimodal methods;

- The evaluation and comparison of the GeMOS in different CNN models as backbone;
- An article in 2021 IEEE International Conference on Image Processing (ICIP) [39] Qualis A1.

1.4 Outlines

The remainder of this work is organized as follows. Chapter 2 presents the background information necessary to understand this work, it includes an explanation about CNNs, presenting main architectures, feature extraction, generative models, and OSR methods. Chapter 3 presents related work written about this subject, reviewing open set image recognition methods. Chapter 4 presents the methodology with an explanation of the proposed method and implementation details. Chapter 5 presents the experimental setup with the following: dataset definition, metrics and, evaluation protocol. Chapter 6 presents the results of the experiments and a discussion about them. Finally, Chapter 7 presents the conclusions and future work.

Chapter 2

Theoretical Background

This chapter describes essential concepts to understand this work. Section 2.1 explains how image classification works and shows state-of-art convolutional neural network architectures. Section 2.2 defines generative models. Section 2.3 defines Open Set Recognition and briefly explains the methods used as the baseline.

2.1 Image Classification

Image classification is one of the most common computer vision tasks. It aims to assign an input image to the class it belongs to, based on a group of predefined labels. An Image classification task $\mathcal{T}$ is composed of a dataset D and a set of classes C. A dataset D is a set of pairs (x,y), where x $(H \times W \times D)$ is an image with dimensions $H \times W$ and D channels, and y is the label of the image. In this context, the objective of $\mathcal{T}$ is to classify correctly the entries of D in one of C classes. Figure 2.1 shows a possible pipeline used to solve this type of problem.

The first step in classification tasks is the pre-processing of the input image. Some common processes in this step are resizing and cropping, used to standardize the image size, and normalizing, used to bring features to the same scale. For normalizing, min-max or z-score methods are generally used. The first method normalizes values between zero and one, where the largest value becomes one, the smallest zero, and the others are

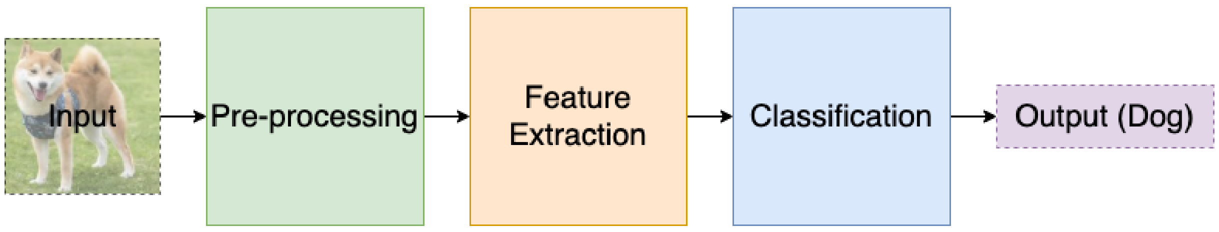


Figure 2.1: Image classification pipeline.

proportionally distributed in the interval following the formula:

$$\frac{value - min}{max - min}. \tag{2.1}$$

In the second method, values are redistributed around the mean in proportion to the standard deviation following the formula:

$$\frac{value - \mu}{\sigma}. \tag{2.2}$$

The second step is feature extraction followed by classification. Feature extraction is important to reduce the amount of information that will be used by classification models, facilitating pattern detection. Some options for feature extraction are: texture descriptors like Haralick [16], feature detectors like Scale Invariant Feature Transform (SIFT) [29]; and dimensionality reducers such as Principal Component Analysis (PCA) [38]. As classification methods, some examples are: K-Nearest Neighbour (KNN) [8]; Support Vector Machine (SVM) [7]; Decision Trees [32]; and Convolutional Neural Networks (CNN) [23]. In deep models such as CNN, both steps are performed by the neural network as they use the entire image as input. This process will be detailed in the next section.

2.1.1 Convolutional Neural Networks

Nowadays CNN is the state-of-the-art for image classification tasks. This model is composed of a group of convolutional blocks (convolutional and pooling layers) followed by fully connected layers. The convolutional blocks are responsible for extracting the spatial features of the image through the convolutions while the fully connected layers are responsible for the classification. In this way, it performs the two activities of the Pipeline 2.1 in the same process.

The convolutional layer is the main CNN layer. It is composed of n filters (w) of dimension $K \times K \times D$ with different weights that pass as a sliding window over the input (x, $W \times H \times D$) making the sum of the dot product of each possible position of the filter. This operation by position ($z[i, j, r]$) is represented by:

$$z[i,j,r] = \sigma(b + \sum_{p=0}^{K-1} \sum_{q=0}^{K-1} \sum_{r=0}^{D-1} w[p,q,r] \cdot x[i+p, j+q, r]), \tag{2.3}$$

where b is the bias and σ is the activation (linear or non-linear function). Each layer filter generates a new matrix with the values ($z[i, j, r]_n$) that are concatenated and serve as input to the next layer. The dot product can be seen in Figure 2.2.

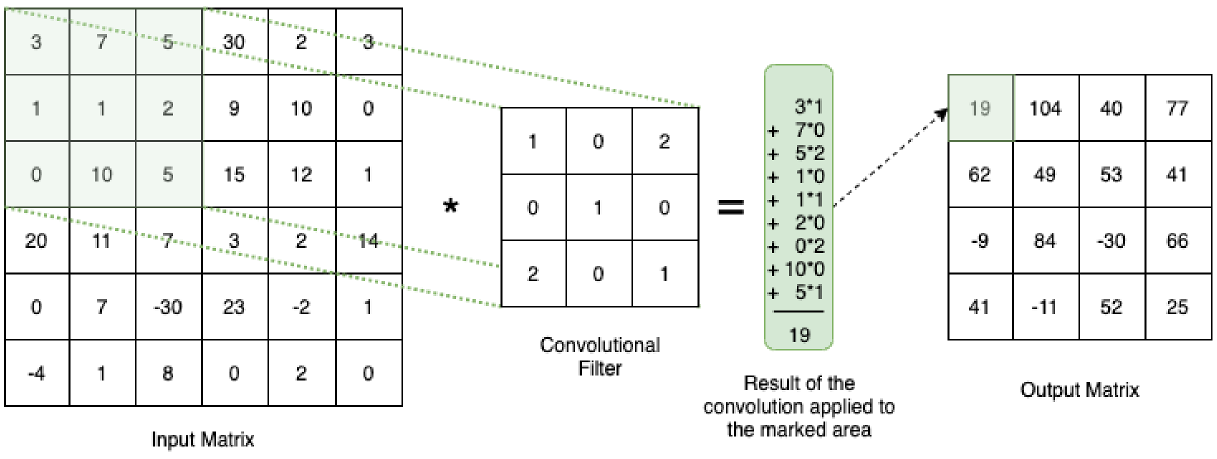


Figure 2.2: Example of convolutional operation in a 2D Matrix.

This operation is in fact a cross-correlation since the mathematical convolution requires the mirroring of the filter, however, the term conventional became a standard in the area of Deep Learning. In convolutional layers, the different filters are responsible for extracting different information from the input image so that it is possible to define its class at the end of the process. Thus, network training changes the weights to identify the characteristics that define the entry.

Convolutional layers are typically followed by pooling layers. The pooling layer does not receive any training, it has a window that slides over the input performing the predefined pooling operation and returning a value per window as shown in Figure 2.3. The main pooling operations used are max-pooling, which returns the highest value of the window, and average-pooling, which returns the average of the window. This layer is responsible for reducing the input size and consequently speeding up the computation and making the features more robust.

At the end of the convolutional blocks, the multidimensional output is flattened into a one-dimensional vector (V) size of T that is used as input for the fully connected layers. These layers are composed of n neurons with different weights (w), each of which is multiplied by each input value, generating an output $z[n]$ as represented by:

$$z[n] = \sigma b[n] + \sum_{j=0}^{T-1} \sigma(V[j] \times w[n]), \tag{2.4}$$

where b is the bias and σ is the activation (linear or non-linear function). After this process, the predicted class of the input image is obtained, so that CNN can do correct predictions, its weights must be updated according to the wrong predictions. This process is called a training phase where, for each training epoch, images are used as input to the network and at the end of the process outputs a prediction. This prediction is used in a loss function that returns a value that, through an optimizer, updates the network weights. This process can be exemplified by Figure 2.4.

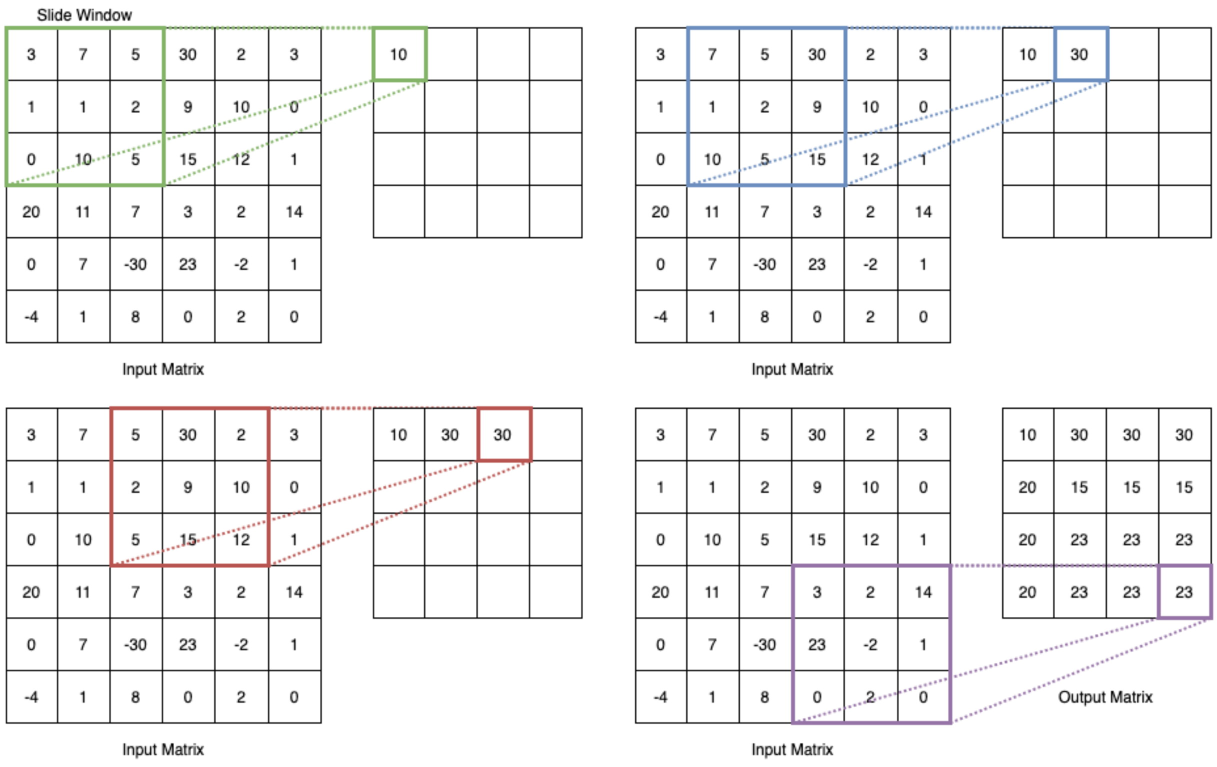


Figure 2.3: Example of max-pooling operation in a 2D Matrix.

The loss function is used to estimate how far the network prediction is from the expected value. The most used function for classification models is the cross-entropy (CE), and can be defined as:

$$L_{CE} = -\sum_{i=1}^{c} \mathcal{T}_i log_2(P_i), \tag{2.5}$$

where $\mathcal{T}$ are labels and P are predictions of the i^{th} class of c.

Optimizers are algorithms used to minimize the loss function by updating the network weights. There are several optimizers, the most used are Stochastic Gradient Descent (SGD) [20], Adam [21], and RMSprop [37].

In recent years, numerous CNN models have emerged for image classification tasks. They all follow the structure explained above, with the addition of new techniques. Some of the main models are:

- **AlexNet** was proposed by [23] and attracted attention for being the first CNN model to win the ILSVRC (2012). It has five convolutional, three max-poolings, and three fully connected layers. These layers are distributed as shown in Figure 2.5. The convolutional layers have kernels of different sizes (eleven, five, three, and three respectively) and this was the first model to use ReLU activation;

- **VGG** was proposed by [35] and differs from AlexNet in its simplicity. It is composed of convolutional, max-pooling, and fully connected layers with all convolutional

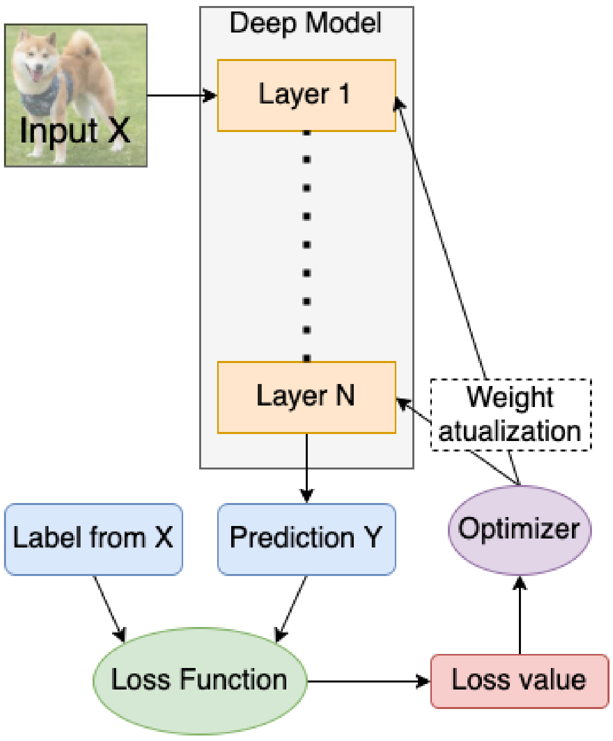


Figure 2.4: Pipeline of a training step.

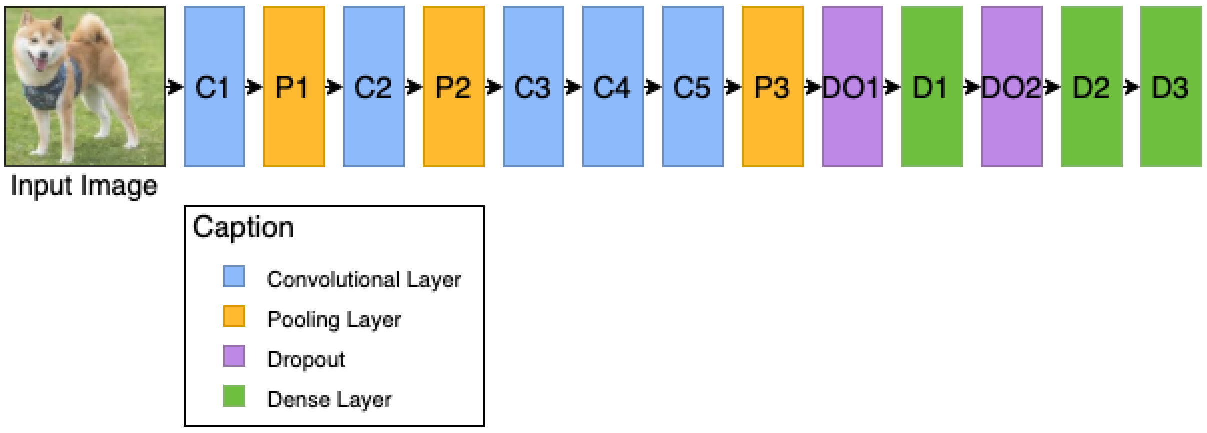


Figure 2.5: AlexNet [23] architecture.

layers having a kernel of size three. By stacking convolutional layers with smaller kernels, the model achieves the same effective receptive field as one layer with a larger kernel, adding more depth and more non-linearity. An example of the 16-layer network (VGG16) can be seen in Figure 2.6. Another difference presented by this model is that the dimensionality reduction is only done by the pooling layers due to the existence of padding in the convolutional layers;

- **ResNet** was proposed by [17] to solve the vanishing gradient problem during back-propagation in deeper networks. To solve this problem, the model proposed residual blocks composed of two or three convolutional layers stacked with shortcut connections. These blocks include the activation of the previous block to compose the

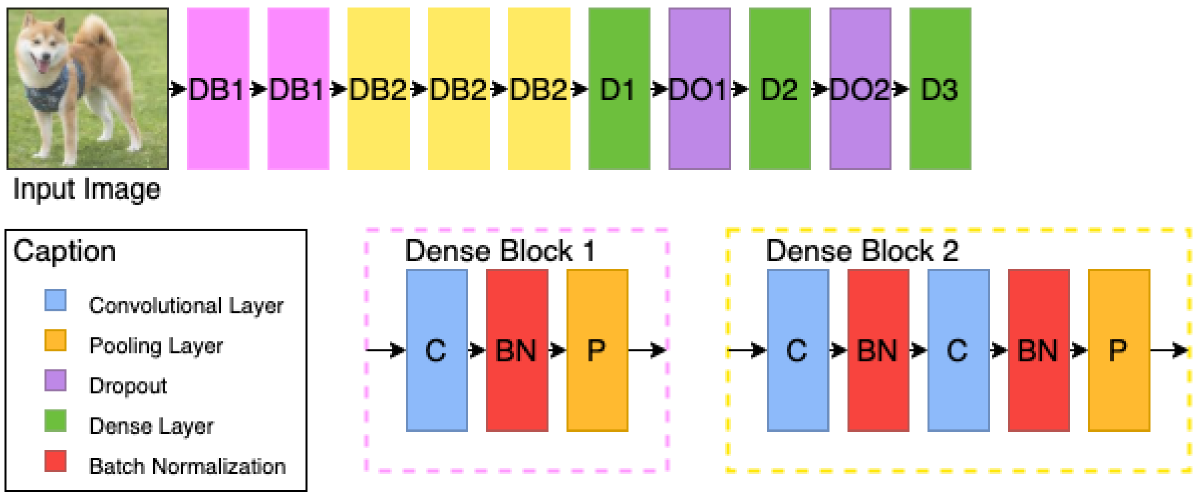


Figure 2.6: VGG11 [35] architecture.

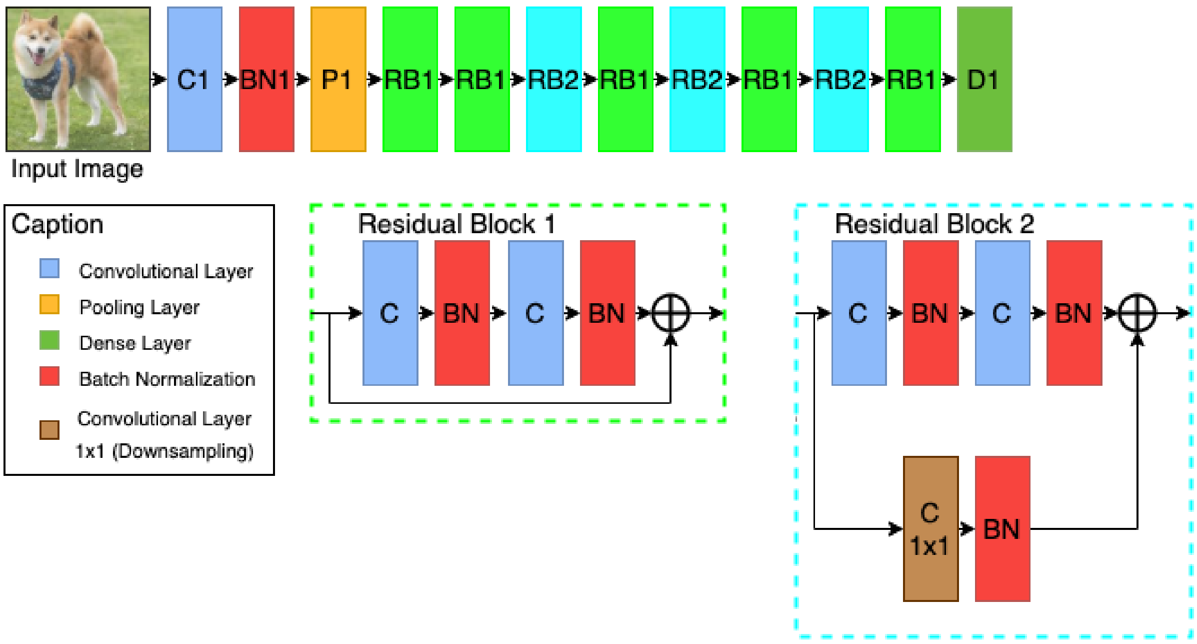


Figure 2.7: ResNet18 [17] architecture.

input of the next block, as shown in Figure 2.7;

- **Wide ResNet (WRN)** was proposed by [43] and is a variation of ResNet that aims to increase the length of the convolutional layers to reduce the depth without compromising the accuracy. The WRN-*d*-*k* has the parameters *d* (depth) and *k* (widening factor), with layers being *k* times wider than layers in the ResNet. This model is shown in Figure 2.8;

- **DenseNet** proposed by [18] maintains the idea of shortcut connections but with a new approach. The model is composed of dense blocks connected by transition blocks and a fully connected layer at the end as shown in Figure 2.9. Dense blocks have several convolutional layers with shortcut connections to all subsequent convolutional layers in the block. In these connections, previous feature maps are concate-

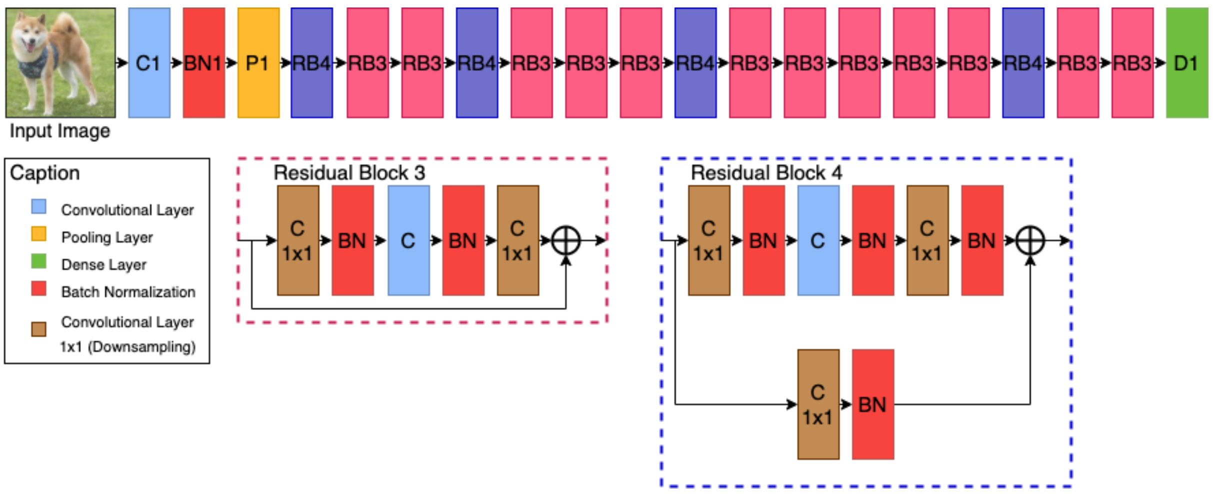


Figure 2.8: WRN-50-2 [43] architecture.

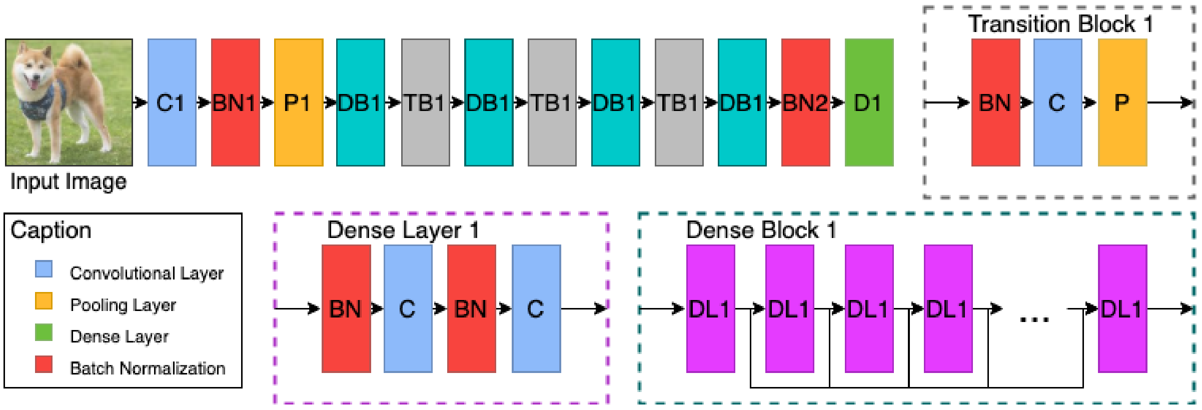


Figure 2.9: DenseNet121 [18] architecture.

nated with the output of the current layer. Transition blocks apply a convolution with a size of one kernel to reduce output depth.

2.2 Generative Models

Generative models seek to determine a distribution that is capable of describing some data. Modelling the distribution of given data makes it possible to estimate the probability that other data will follow the same distribution. The main generative model used in this work is the Gaussian Mixture Model (GMM) due to its multimodal capabilities. Other standard methods were used as a baseline to compare the impact of different methods. Examples of these generative models applied on a toy dataset are shown in Figure 2.10. The generative models used in this work include:

- **Gaussian Mixture Model (GMM)** was proposed by [3] as a probabilistic model

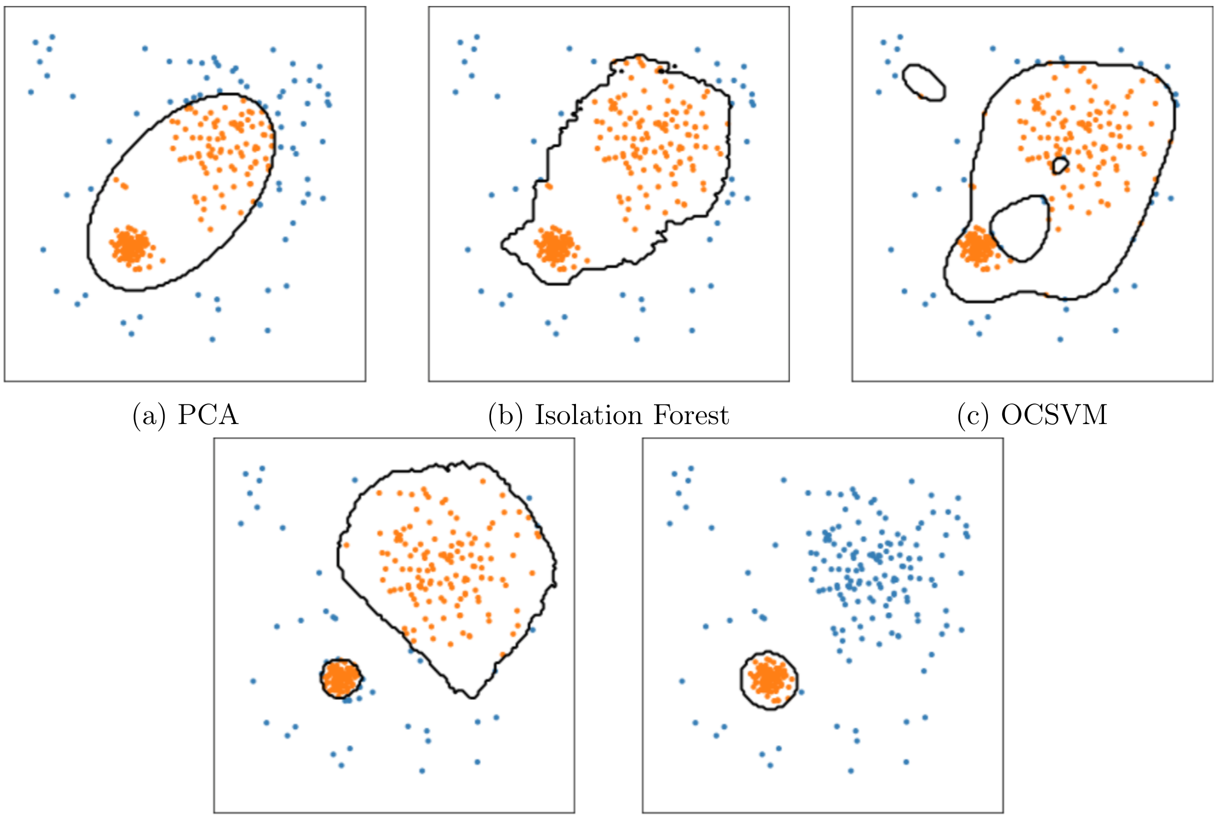

(a) PCA (b) Isolation Forest (c) OCSVM (d) LOF (e) GMM

Figure 2.10: Examples of some generative models applied on a toy dataset.

that assumes that data is generated by a mixture of different gaussians. By this definition, GMM can describe multimodal data while the other exemplified generative models only find unimodal distributions. The model uses the expectation-maximization (EM) algorithm [9] to find the local maximum likelihood by calculating for each point the probability that it was generated by each component of the model. The score is defined by the average log-likelihood.

- **Principal Component Analysis (PCA)** was proposed by [11] and is a method used for data dimensionality reduction. The method performs a linear transformation to highlight the variance and reduce the covariance of the data through dimensionality reduction. With this transformation, the method can generate a score based on the average log-likelihood;

- **Isolation forest** [28] aims to detect anomalies by isolating data from the rest of the dataset. This is an unsupervised method that works like a decision tree so that features are randomly selected and partitioned into a random value between the largest and smallest existing values, separating outliers from other data. The fit score generated is the opposite of the anomaly score, which is the average of the number of splits needed to isolate the data;

- **One-Class Support Vector Machine (OCSVM)** was proposed by [34] as an unsupervised method to detect outliers. The method detects soft boundaries of the input dataset from hyperplane separation. The score of new data is given based on the raw dataset distribution;
- **Local Outlier Factor (LOF)** was proposed by [4] as an unsupervised method for anomaly detection by measuring the local deviation of each sample from its neighbours. The method uses the K-Nearest Neighbour to calculate the distance and use that distance to estimate its density. The score is defined by how isolated the input is from its neighbours;

2.3 Open Set Recognition

As the number of unknown classes impacts directly on an Open Set Recognition task, the openness O is defined to evaluate these problems. The openness O is defined as:

$$O = 1 - \sqrt{\frac{2 \times C_{TR}}{C_{TA} + C_{TE}}}, \tag{2.6}$$

where C_{TA} is the number of classes to be recognized in the task, C_{TR} is the number of classes used in training and, C_{TE} is the number of classes used in testing. A problem with openness being equal to zero is that the problem is completely closed whereas, with a higher value, the problem is more open. For OSR problems in can be seen that $C_{TA} \subseteq C_{TR} \subseteq C_{TE}$. Considering C_{TA} should be a subset of C_{TR}, the openness of a problem should only depend on the KKCs from C_{TR} and UUCs from C_{TE}. Because of that, [13] defined the openness as:

$$O = 1 - \sqrt{\frac{2 \times C_{TR}}{C_{TR} + C_{TE}}}. \tag{2.7}$$

OSR is characterized by the ability to classify correctly inputs of known and unknown classes. While image classification is a closed problem, $C_{TA} = C_{TR} = C_{TE}$ and the openness is zero, an OSR task $\mathcal{T}$ is an image classification task with openness greater than zero, where $\mathcal{T}$ needs to classify correctly inputs from D into C_{TR} classes and identify inputs that do not belong to any C_{TA} classes as an unknown entry C_{TE}. OSR task S is composed of a dataset D, and a set of known classes C_{TA}. In this case, the dataset D is composed of inputs x with labels y in the set of classes C_{TR} and other inputs x' with labels y' in the set of classes C_{TE}.

To solve the OSR task some methods are proposed in the last years. These methods use different approaches, such as: define a threshold to separate KKC from UUC; generate images to compose the UUC spaces; and use reconstruction techniques to detect UUCs. To compare the evaluation with other OSR techniques, some methods are selected as the baseline:

- **OpenMax** [2] adapt the SoftMax layer to handle open set scenarios. Initially, the activations of the penultimate layer of a pre-trained model are extracted. With them, a mean activation vector and a probability distribution function for each class are calculated. Then the activations of the penultimate layer of the model are recalibrated by estimating the probability distribution score of the distance between the mean activation vector of the class and the input fit to the model. A threshold on this fitting score defines if an input belongs to this known class;
- **G-OpenMax** [12] extends the conventional OpenMax associating a GAN model with it to synthesize objects of unknown classes. Since objects are created, G-OpenMax follows the same process of OpenMax considering one more class for unknown objects. With that, the model provides a decision score for known and unknown classes;
- **ODIN** [27] is a simple method that does not require any change to a pre-trained CNN. This method consists of temperature scaling and image pre-processing. The temperature scale is a Softmax Score parameter used to calibrate the prediction confidence after the training, which aids in separating KKCs and UUCs images. Pre-processing inputs with small perturbations are used to reduce the Softmax Score, and this perturbation has a stronger effect on KKCs than on UUCs, making them more separable. This strategy allowed ODIN to work similarly to OpenMax using a threshold over class probabilities to discern between KKCs and UUCs;
- **OODNN** [10] adds an estimation branch parallel to the classification branch of any CNN. This branch is composed of one or more fully connected layers with an output c between zero and one. The SoftMax prediction probability p is adjusted to consider c and the loss is changed to maximize c in addition to p. After training, UUCs objects are detected if c is less than or equal to the defined threshold. An example of this model is shown in Figure 2.11;
- **OSRCI** [30] uses a GAN for data augmentation by generating images that do not belong to any known class, as represented in Figure 2.12. With this technique, it is possible to transform an Open Set Recognition problem into a closed set classification with an additional class composed of unknown objects. This method is composed of a GAN, used to generate these images, and a classifier model. Initially, the classifier is trained with K known classes and GAN generates a set of images that will be incorporated as a new class in the dataset forming an augmented dataset.

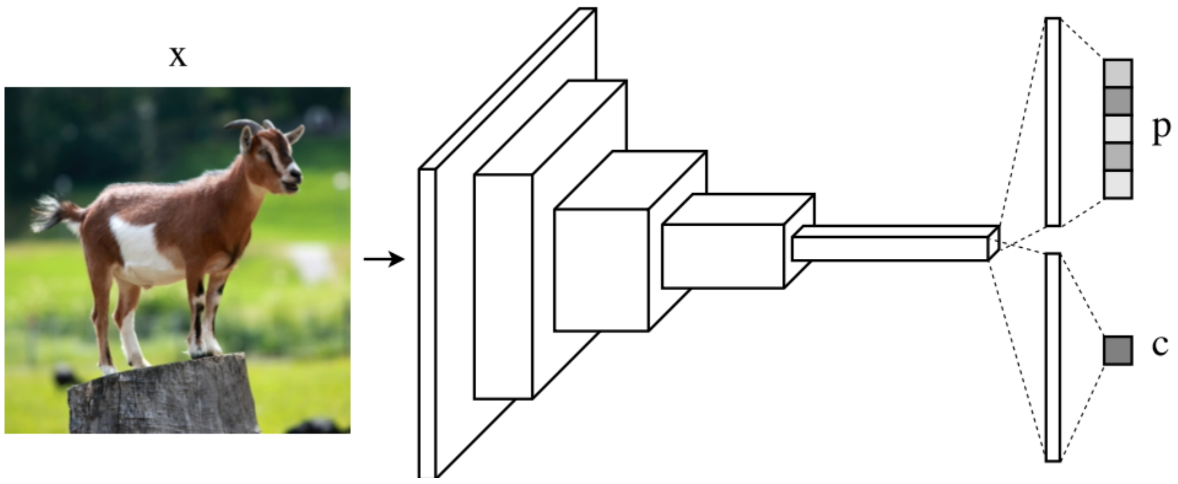


Figure 2.11: Example o OODNN model. This example shows the extraction of the prediction p as usual in any CNN model and the addition of the score c that is used to change the loss calculation. **Adapted from [10]**.

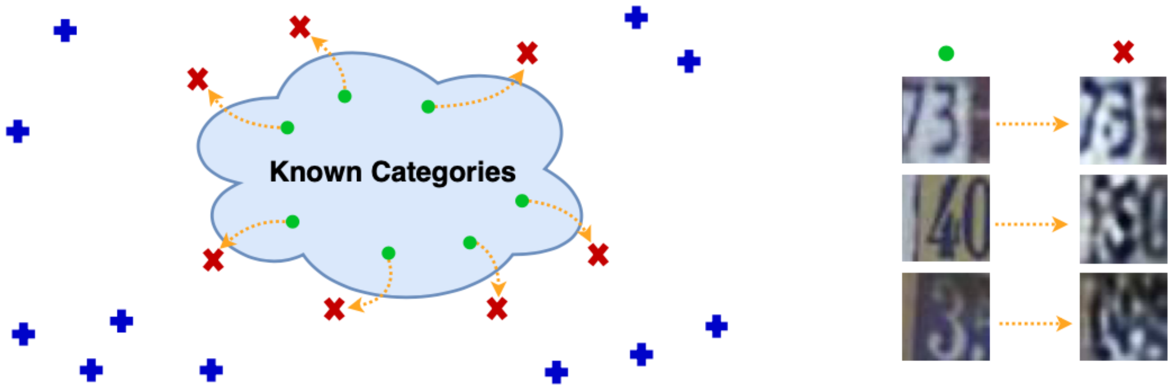
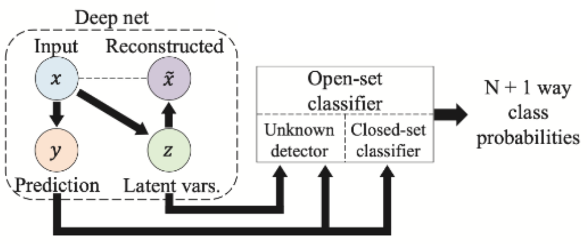


Figure 2.12: Green dots are examples of known objects and from them, red x are generated to represent unknown classes. Blue + are other unknown images that can be extended from the red ones. **Souce: [30]**

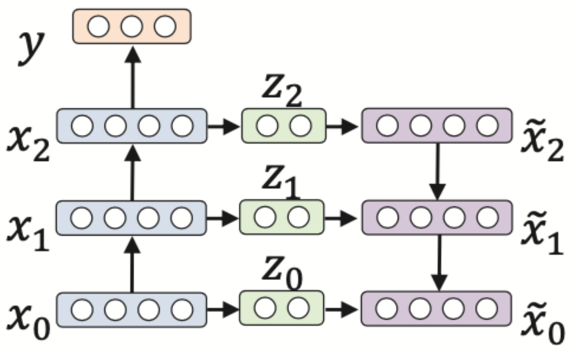
After that, a new classifier, with $K+1$ classes, is initialized with the same weights as the K classes model and is trained with the augmented dataset. With this new classifier model, it is possible to classify all K known classes and classify any unknown object as the new class;

- **CROSR** [41] creates a network to classify and reconstruct an input jointly, as shown in Figure 2.13a. The architecture is a mixture of the dimensionality reduction obtained by Autoencoders and the hierarchical reconstruction of LadderNets as shown in Figure 2.13b. This model makes it possible to use the latent space of reconstruction to create an unknown object detector and maintain the accuracy of the known object classification;

- **C2AE** [31] uses the reconstruction errors of an AE to separate KKC objects from UUC as well as CROSR. For this, the method divides the training into two parts: the



(a) Model Overview.



(b) Network.

Figure 2.13: CROSR Network. In *a* the overview of the model is represented, showing the model formed by input x, reconstruction $\tilde{x}$, prediction y and latent vars. z, and how each one of them is used for the OSR task. In *b* more detail of the layers of the network represented in *a* is shown. **Adapted from [41]**.

first, the closed set classification, where the encoder learns how to classify a known input; the second, the encoder has its values frozen and the decoder, conditioned to a class matching layer, is trained to reconstruct the image when the classes match and poorly reconstruct it when they are different. Following these steps, the model learns how to reconstruct KKC conditioned to the correct class as well as poorly reconstruct unknown objects. This model uses the latent space of reconstruction to detect unknown objects;

- **CGDL** [36] uses a conditional Gaussian distribution to force different latent features to approximate Gaussian models to obtain the best possible separation of unknown objects. The model uses a ladder architecture that has an encoder followed by a classifier of known classes and a decoder used for image reconstruction. This model extracts the latent distribution from all layers of the encoder and uses their values to compose the analogue layer of the decoder. In this way it seeks to reduce the reconstruction and classification loss and the Kullback–Leibler divergences of the latent spaces, forcing the space distributions to approximate to different multivariate Gaussians. The detection of unknown objects is done by the reconstruction error together with the latent distribution of the last layer of the encoder.

Chapter 3

Related Work

The concept of OSR was introduced by [33] and their proposed approach, as well as other early studies, were based on shallow decision models such as threshold-based or support-vector-based methods (1-vs-Set Machine). Recent trends in the literature coupled OSR's failure recognition capabilities with the versatility and modeling power of Deep Learning. Early strategies for deep OSR [2, 12, 27] consisted of incorporating the UUCs detection directly into the prediction of the DNN. For instance, OpenMax [2] performed OSR by reweighting SoftMax activation probabilities to account for a UUC during test time. Many improvements to OpenMax were further proposed such as the association of Generative Adversarial Networks (GANs) [15] to provide synthetic images to the method (G-OpenMax [12]). Out-of-Distribution Detector for Neural Networks (ODIN) [27] inserted small perturbations in the input image to increase the separability in the SoftMax predictions between in-distribution and out-of-distribution (OOD) data. This strategy allowed ODIN to work similarly to OpenMax [2] and to operate close to the label space, using a threshold over class probabilities to discern between KKCs and UUCs.

More recently, OSR for deep image classification has incorporated input reconstruction error in supervised DNN training as a way to identify OOD samples [10, 30, 41, 31, 36] by employing generative strategies. Confidence learning for out-of-distribution detection in neural networks (OODNN) [10] adds an estimation branch after the penultimate layer, parallel to the classification branch of any CNN. This branch generates a score that is maximized with the loss function along with the prediction and is used for OOD detection based on a threshold value. Open Set Recognition with Counterfactual Images (OSRCI) [30] uses dataset augmentation by generating images that do not belong to any known class with a GAN. Classification-Reconstruction learning for Open Set Recognition (CROSR) [41] jointly trains a supervised DNN for classification and an AutoEncoder (AE) to encode the input into a bottleneck embedding and then decodes it to reconstruct. The magnitude of the reconstruction error can then be used to delineate between known and unknown classes. Class Conditional AutoEncoder (C2AE) [31], similarly to CROSR, uses the reconstruction error of the input from an AE to discern between KKC and UUC samples. For that, the method uses a matching layer to learn how to reconstruct known classes when conditioned to the correct class while poorly reconstruct objects when con-

Table 3.1: Overview of open set recognition methods.

Method	Deep Model	Type	Year	Plug and Play	Maximize Unknown Detection on Training
1-vs-Set Machine	No	Threshold based	2012	No	No
OpenMax	Yes	Threshold based	2016	Yes	No
G-OpenMax	Yes	Image Generation	2017	No	Yes
ODIN	Yes	Threshold based	2017	Yes	No
OODNN	Yes	Threshold based	2018	No	Yes
OSRCI	Yes	Image Generation	2018	No	Yes
CROSR	Yes	Reconstruction Based	2019	No	Yes
C2AE	Yes	Reconstruction Based	2019	No	Yes
CGDL	Yes	Reconstruction Based	2020	No	Yes
T/E/S Learning	Yes	Image Generation	2021	No	Yes
GeMOS	Yes	Threshold based	2021	Yes	No

ditioned to wrong classes. Conditional Gaussian Distribution Learning (CGDL) [36] uses a Variational AutoEncoder (VAE) with a Ladder architecture to model the bottleneck representation of the input images and their activations according to a vector of gaussian means and standard deviations in a lower-dimensional high semantic-level space. The unknown samples are recognized using the reconstruction error and the latent space.

In 2021, the teacher-explorer-student (T/E/S) learning [19] was proposed as another image generation method. In this method, the teacher network extracts knowledge about known objects and delivers this knowledge to the student. With this learned information, the explorer network generates unknown samples to feed the student network. By repeating this process, the student network can classify known and unknown samples.

Comparing these methods, it is possible to see that there was a transition where initially methods based on threshold were used, and then it started to have two strands, one following the idea of generating images to compose the unknown class and the other using reconstruction models. The methods of these new streams tend to be difficult to train, in addition to being slower as they have a high computational cost both in terms of memory and processing. This is because many of these methods use complex structures to perform the reconstruction or GANS for image generation.

Table 3.1 lists several characteristics of the OSR models to compare the models and view where the GeMOS method fits in the literature. Regarding the use of deep models, the default is that they are used, except for the initial methods in the literature. As for the types, the methods were grouped into three categories: threshold based, methods that use a standard classification process and use the output to define unknown objects; image generation, methods that use synthetic images generated to compose the class of unknown objects; and reconstruction based, methods that use a classifier to perform the classification and a reconstructor that has its score used to detect unknown objects. Regarding the types, the methods were initially threshold based and over time more methods started to appear that use techniques of image generation and reconstruction, GeMOS is a threshold based method. Regarding the possibility of the method being plug and play, being able to be coupled to pre-trained classification models, most methods do not have this capability, only two methods and GeMOS are capable. Another point analyzed is the use of mechanisms to maximize the unknown detection during training. The methods capable of doing this are image generation methods, which create synthetic images and use them during training; reconstruction based methods, which use the reconstruction rate for detection; and some threshold based methods that change the loss. GeMOS and most of the threshold based methods only receive unknown objects during the test phase and do not maximize unknown detection on training.

Chapter 4

Methodology

This chapter presents the methodology used to propose GeMOS method. The proposed method is described in Section 4.1. The implementation details are in Section 4.2.

4.1 GeMOS

GeMOS is a novel approach composed of a CNN that extracts feature-level information from KKCs, and of generative models that use these features to assign a score for each KKC sample and identify UUCs.

GeMOS uses a pre-trained CNN for image classification as a backbone in a closed set scenario. This backbone is used to extract feature-level information from the intermediary activations (e.g. $a^{(L_3)}$, $a^{(L_4)}$, $a^{(L_5)}$, $a^{(L_6)}$ from Figure 4.1) to feed the generative models. This CNN could be any classification model trained in a closed set scenario. For the tests, ResNet18, WRN-10-28, and DenseNet121 are used. Convolutional layer activations are flattened using a function ϕ – usually an average-pooling – and concatenated into a single vector ($a^{\star}$) for each sample. Vector $a^{\star}$ can then be fed to the generative model G, which learns the most representative feature maps to maximize the reconstruction of this feature vector. The inherent dimensionality reduction of G increases the likelihood that only the most relevant activations are taken into account to compute sample scores.

In order to solve the OSR task, GeMOS employs a set $\mathcal{G} = \{k_0, k_1, \ldots, k_{C-1}\}$ of generative models, with C representing the number of KKCs. During the training process, each generative model k_c is trained with the activation values ($a^{\star}$) of samples correctly predicted to be from the corresponding class c in the training set. Multiple generative models were tested, such as Principal Component Analysis (PCA), One-Class Support Vector Machine (OCSVM), Isolation Forest (IF), and Local Outlier Factor (LOF). All of these traditional methods for dimensionality reduction and/or OOD detection assume unimodal distributions within each class, thus, it was also inserted Gaussian Mixture Models (GMM) in the experiments to test a multimodal strategy. One should notice,

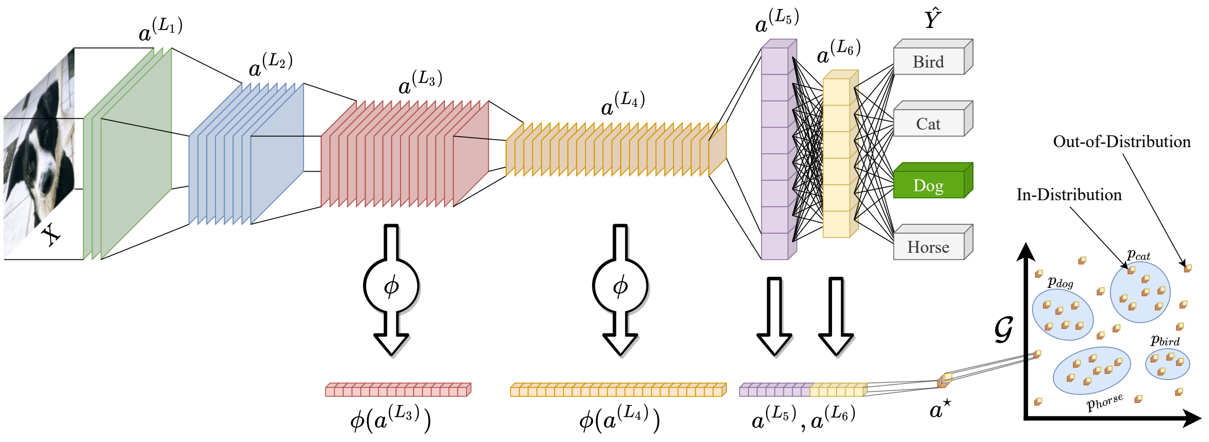


Figure 4.1: Scheme of the GeMOS pipeline. A CNN model is used to classify and extract the intermediary activations $a^{(L_3)}$, $a^{(L_4)}$, $a^{(L_5)}$ and $a^{(L_6)}$. In this example, $a^{(L_3)}$ and $a^{(L_4)}$ are 2D activation maps from convolutional layers, while $a^{(L_5)}$ and $a^{(L_6)}$ are originally 1D. In order to concatenate these features, $a^{(L_3)}$ and $a^{(L_4)}$ are pooled into a 1D vector using a function ϕ (e.g. average-pooling) and concatenate them with $a^{(L_5)}$ and $a^{(L_6)}$, resulting in a feature vector $a^{\star}$. The $a^{\star}$ vector can then be used as input to G.

however, that any generative model that produces a likelihood score for its samples can be used.

Closed set image recognition is performed by the CNN, while OSR is introduced into the framework by setting thresholds on scores returned by $\mathcal{G}$. Each $k_c \in \mathcal{G}$ fits one class and can generate a likelihood score $l_{i,c} \approx p_c$ for an input x_i to express how similar the corresponding vector $a_i^{\star}$ is to the probability p_c of class c, predicted by the CNN. To identify if x_i belongs to an unknown class, a threshold is applied to $l_{i,c}$ to define if the input is in-distribution or out-of-distribution. The threshold to a single cutoff value has not been defined as it depends on the hyperparameters of the generative model, its scoring function, and the failure tolerance of the application. This is because there is always a trade-off between KKC accuracy and UUC identification performance. Some scenarios might require minimal loss in the performance of the KKCs (e.g. medical image analysis), while others can tolerate a higher penalty in the performance of known classes to identify more reliably OOD samples. It is noteworthy that, as the model can be coupled to any CNN, it depends on its ability to separate well-known classes, since the features used to train generative models come internally from CNN.

A pipeline of the method can be described in Algorithm 1. The Algorithm initially needs a pre-trained CNN of a network to be trained with the desired dataset. After that, each piece of data from the testing set of the closed dataset should be used as input of the network. For each image that is correctly classified by the CNN, internal activations are extracted, flattened, concatenated into a single vector, and appended to a dataset of the classified class. The number of internal layers used in the extraction should be defined considering the model, and the problem, since simpler problems need less information to

Algorithm 1 GeMOS - Training pipeline

```
Require: Pre-trained CNN with a dataset D
  for each Tuple $(x, y)$ in train set of $D$ do
    Feed CNN with $x$
    if $y =$ CNN Prediction class then
      extract internal activations $a_1$ to $a_n$
      flatten these activations
      concatenate each activation into $a^\star$
      insert $a^\star$ into a dataset of $y$ $D_y$
    end if
  end for
  for each predicted class do
    train one generative model $G_y$ with the dataset $D_y$ of the corresponding class
  end for
```

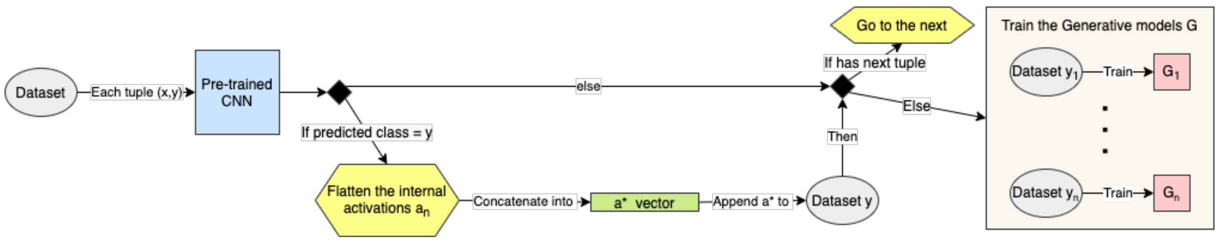


Figure 4.2: GeMOS train method pipeline. For each tuple (x,y) of the dataset used to train the CNN, where x is the input and y the label, if the input is correctly predicted by the CNN, the internal activations are flattened and concatenated in vector $a^\star$. After that, the vector is appended into the corresponding class dataset (y). After processing all the training images, a generative model for each class is trained using the corresponding dataset.

define boundaries for KKCs than more complex problems. After feeding the network with all testing sets one dataset for each class with the activation of the correctly predicted samples of the class is created. For each dataset, one generative model is trained to define the referent class. In the end, the method is ready for inference inputs. This process can also be seen in Figure 4.2.

The inference process is defined as: given an image as input, it is fed into the CNN model, and with that, the activations are extracted, flattened, and concatenated into a single vector. With the CNN classification, the corresponding generative model was fed with the activation vector generating a probability score representing the chance that the input belongs to the defined class. Based on a pre-defined threshold, the method should consider the input as a defined class or as an OOD sample. This process is shown in Figure 4.3.

A main advantage of GeMOS in comparison with end-to-end trainable deep models is the fact that shallow generative models can be attached to any pre-trained deep closed set model with minimal additional computation time. Contrarily to other state-of-the-art methods that rely on AE reconstruction error [41, 31], GeMOS focuses on generative

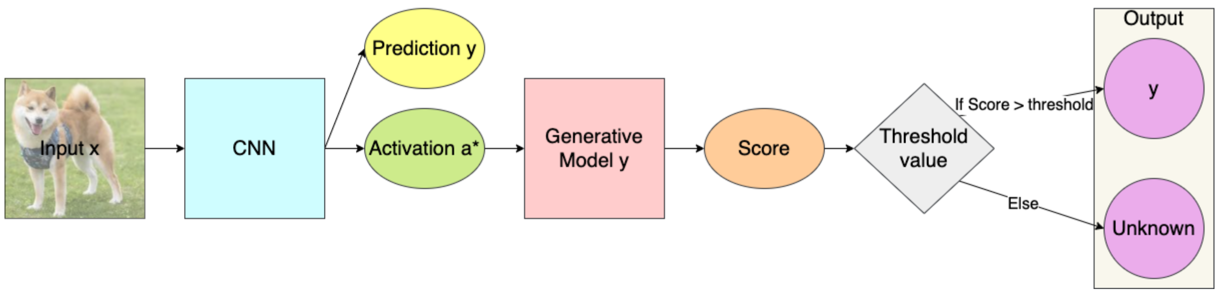


Figure 4.3: GeMOS method inference pipeline. Given an input x, it is fed into the CNN model that returns an activation vector $a^\star$ and a prediction y. The corresponding generative model of the predicted class y was fed with $a^\star$ generating a probability score that represents the chance that the input belongs to the defined class. Based on a pre-defined threshold, the method should consider the input as a defined class or as an OOD sample.

models that do not require GPU for training, rendering it an ideal candidate for a plug-and-play alternative to opening pre-trained CNNs.

4.2 Implementation Details

The method was developed in an Ubuntu 18.04.3 kernel version 5.4.0 environment with one NVIDIA TITAN X 12GB for processing the models, a 64-bit Intel i9 7920X machine with 64GB of RAM, and Python 3.6.8 with the following libraries for the development:

- *PyTorch 1.5.0*: It is responsible for all CNN implementation, including all training, test, validation, inference process, and extraction of the internal activations. It also includes all datasets used in the experiments;
- *SKLearn 0.0*: It is responsible for all metrics and generative model implementation;
- *NumPy 1.18.4*: It is used for mathematical calculations of multidimensional arrays;
- *MatPlotLib 3.2.1*: It is used to plot images and graphics.

The GeMOS method implementation is available on GitHub[1]. It is possible to see more details of the implementation using different datasets.

[1] https://github.com/marcosvendramini/GeMOS

Chapter 5

Experimental Setup

This Chapter presents the configurations and the evaluation protocol used during the experiments. In Section 5.1 the datasets used as in-distribution (D^{in}) and out-of-distribution (D^{out}) are presented. In Section 5.2 the metrics used to evaluate the method are described. In Section 5.3 the evaluation protocol used for us and all of the baseline methods is presented.

Figure 5.1: Random examples from MNIST dataset.

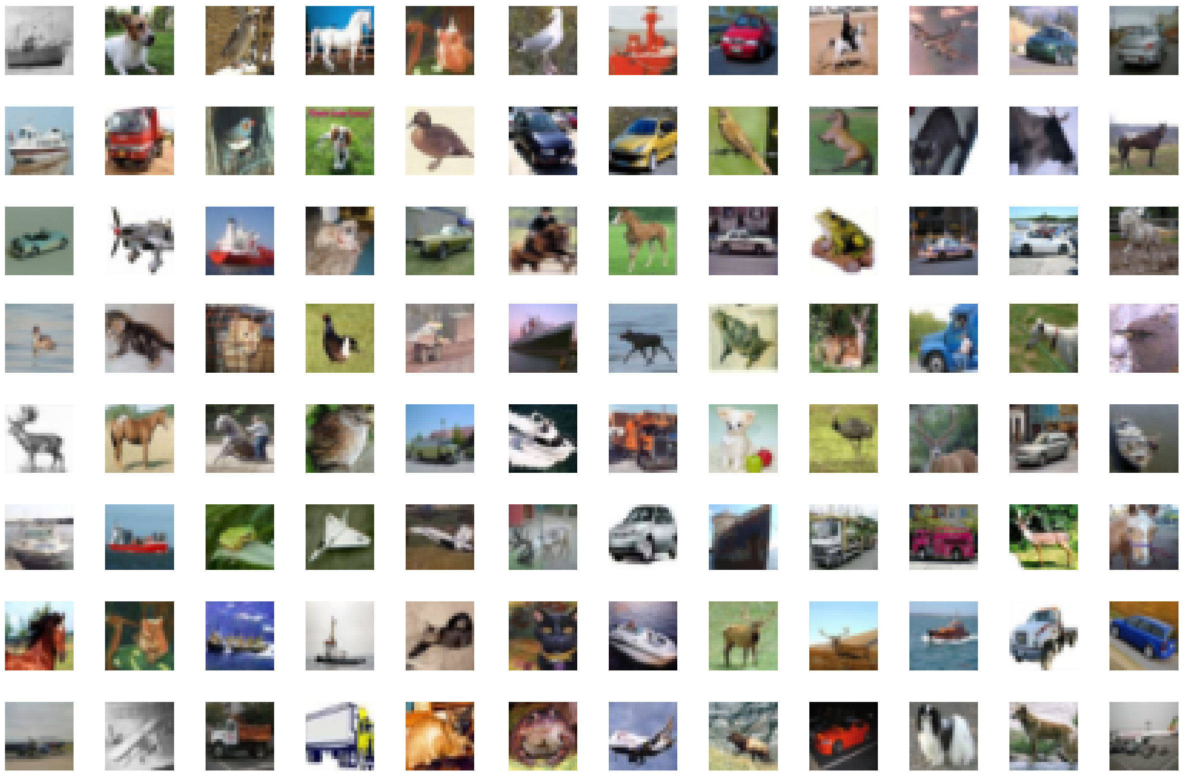


Figure 5.2: Random examples from CIFAR10 dataset.

5.1 Datasets

In OSR problems, multi-class datasets are usually used as a benchmark. The main approach used in these problems is as follows: given a dataset, some classes are randomly selected as a training set, which is considered the method's KKCs, while the others are used only during validation as UUCs. Another approach is to use a dataset to train the model considering all its classes as known and another dataset as UUCs.

5.1.1 MNIST

The MNIST dataset [26] is composed of images of handwritten digits. This dataset has 70,000 images from ten classes (numbers 0-9), each image has 28x28 grayscale pixels, and has the digit centred. Examples of this dataset are shown in Figure 5.1.

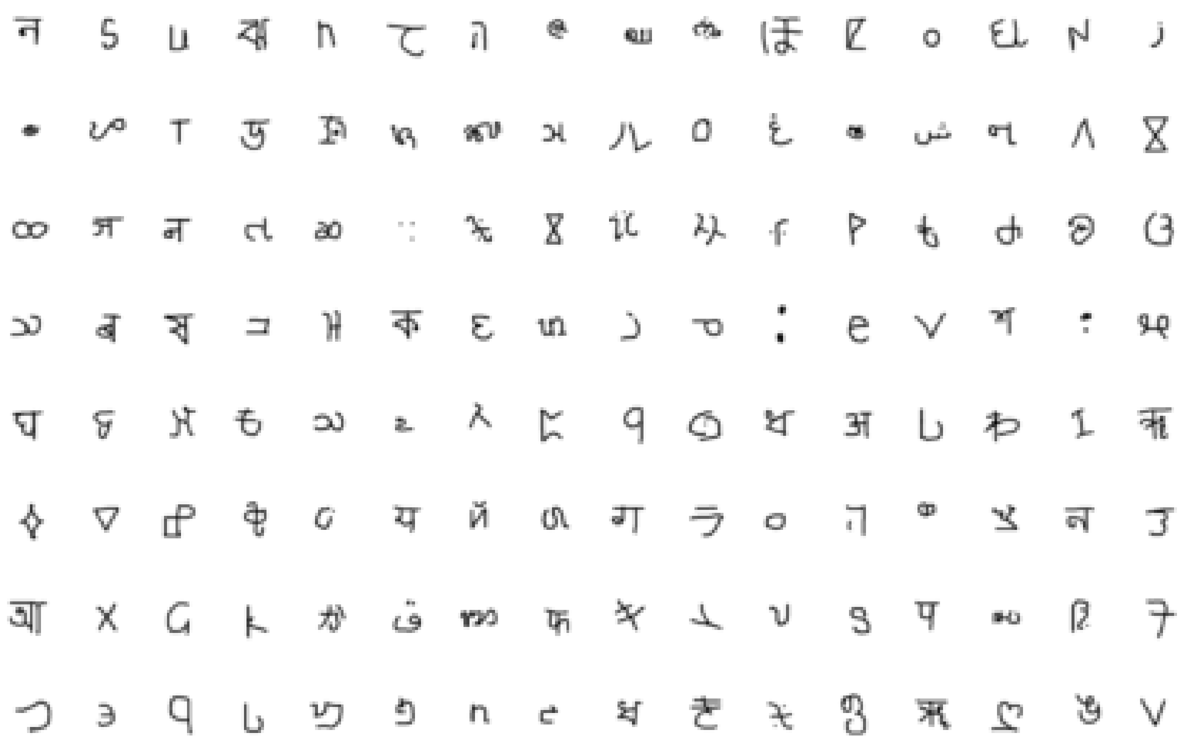


Figure 5.3: Random examples from Omniglot dataset.

5.1.2 CIFAR10

The CIFAR10 dataset [22] is composed of images of animals and vehicles. This dataset has 60,000 images from ten classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), each image is RGB, and has 32x32 pixels. Examples of this dataset are shown in Figure 5.2.

5.1.3 Omniglot

The Omniglot dataset [24] is composed of images of handwritten characters from 50 different alphabets. This dataset has 32,460 images from 1,623 classes, each image is grayscale, and has 105x105 pixels. Examples of this dataset are shown in Figure 5.3.

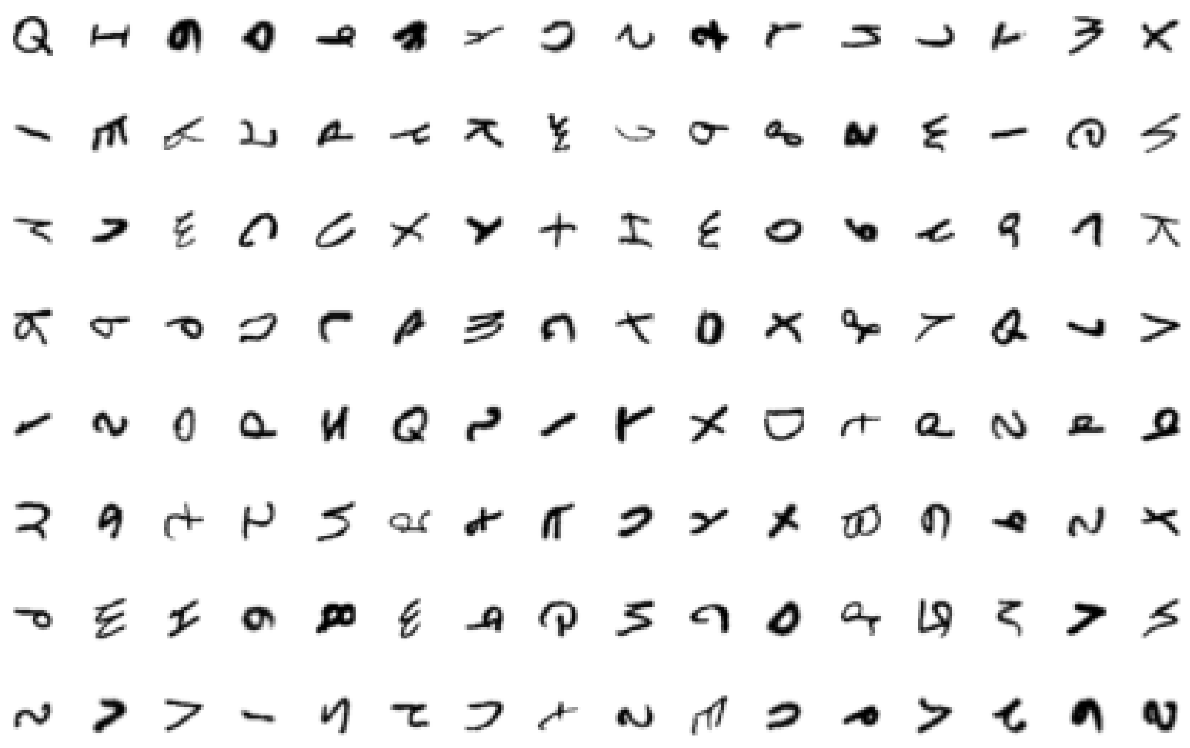


Figure 5.4: Random examples from EMNIST Letter dataset.

5.1.4 EMNIST Letters

The EMNIST Letters dataset [6] is composed of images of handwritten letters. This dataset has 145,600 images from 26 classes, each image is grayscale, and has 28x28 pixels. Examples of this dataset are shown in Figure 5.4.

5.1.5 KMNIST

The KMNIST dataset [5] is composed of images of handwritten characters to represent each of the ten rows of Hiragana. This dataset has 70,000 images from ten classes, each image is grayscale, and has 28x28 pixels. Examples of this dataset are shown in Figure 5.5.

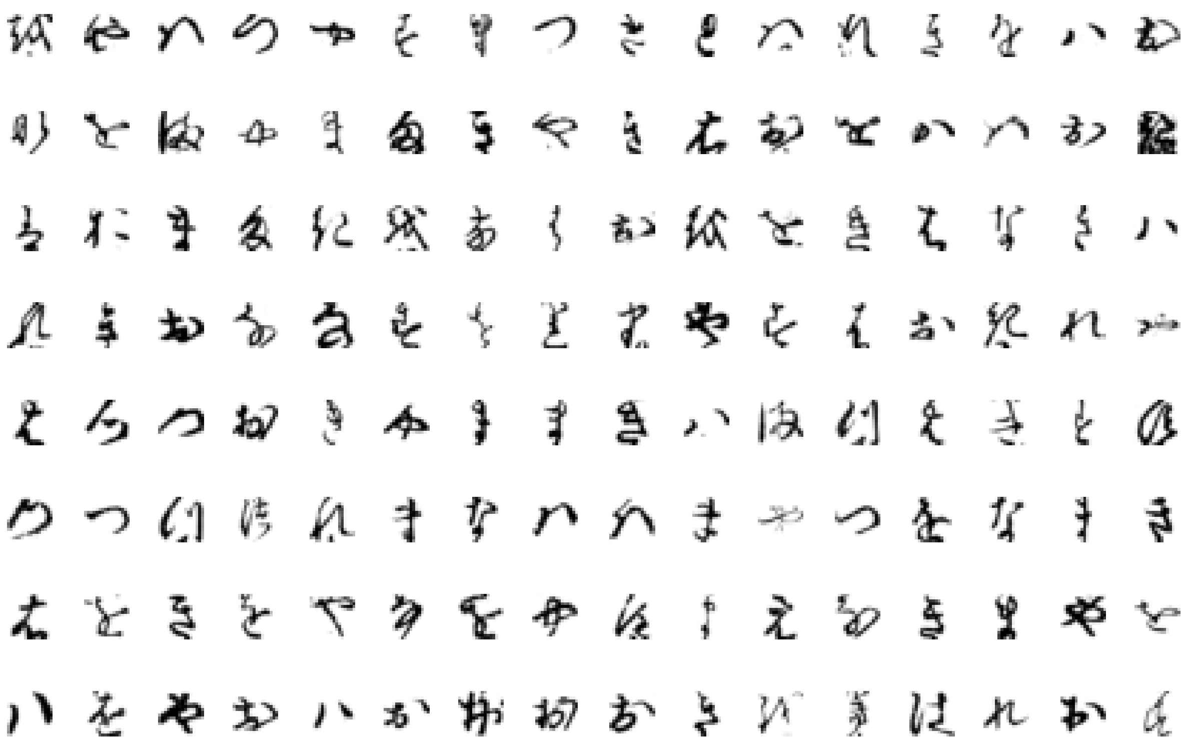


Figure 5.5: Random examples from KMNIST dataset.

5.1.6 CIFAR100

The CIFAR100 dataset [22] is composed of images of different objects. This dataset has 60,000 images from 100 classes, each image is RGB, and has 32x32 pixels. Examples of this dataset are shown in Figure 5.6.

5.1.7 LSUN

The LSUN dataset [42] is composed of images of scenes. This dataset has 10,000,000 images from ten classes (bedroom, kitchen, living room, dining room, bridge, tower, restaurant, conference room, classroom, and church outdoor), each image is RGB, and has 256x256 pixels. Examples of this dataset are shown in Figure 5.7.

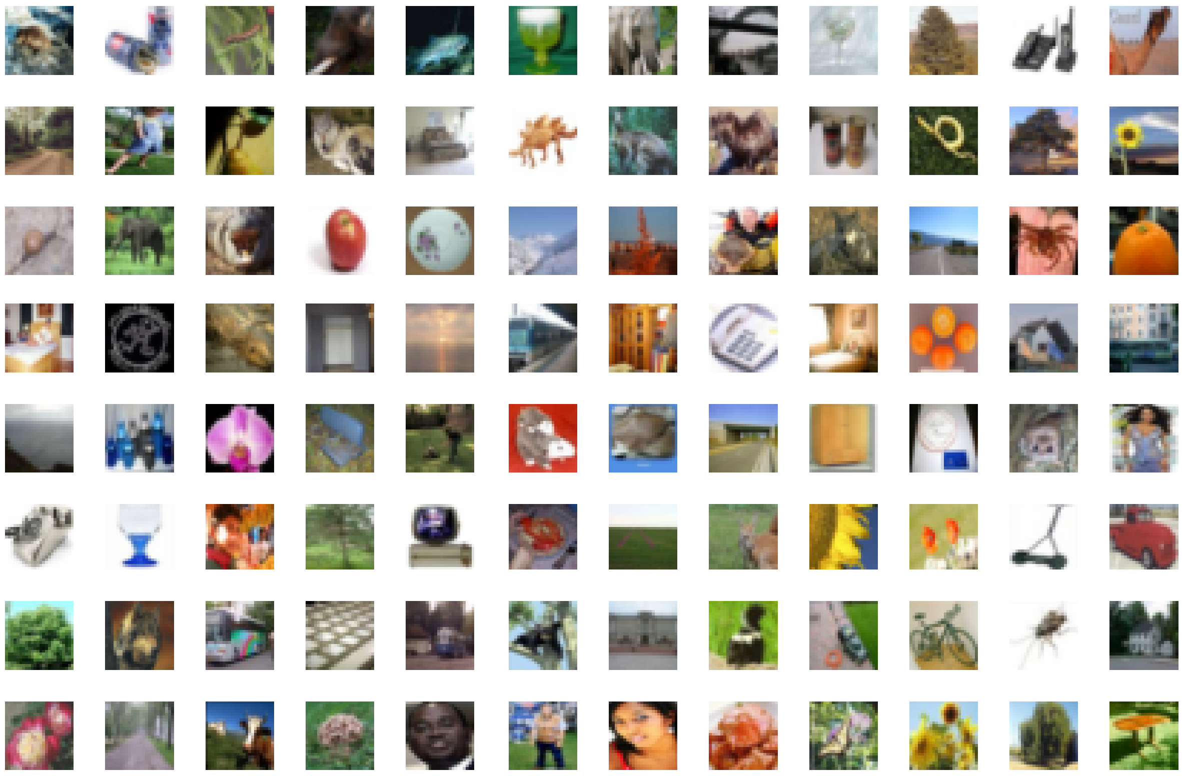


Figure 5.6: Random examples from CIFAR100 dataset.

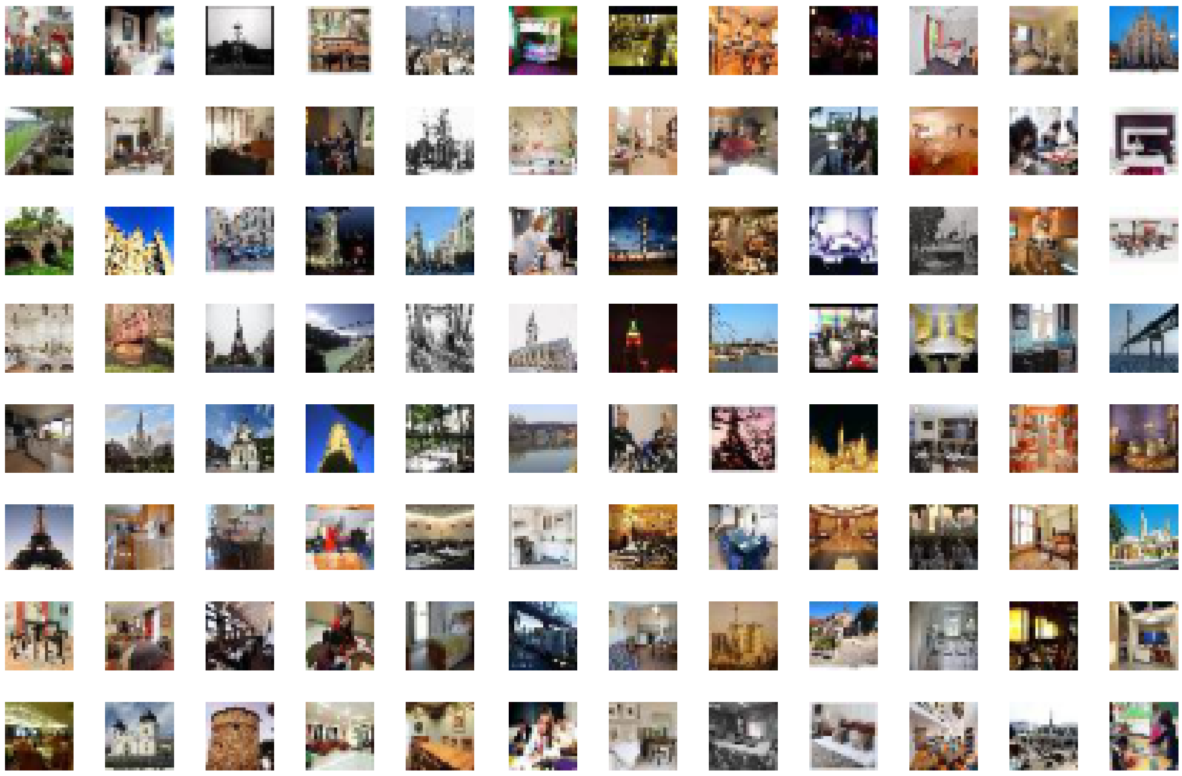


Figure 5.7: Random examples from LSUN dataset.

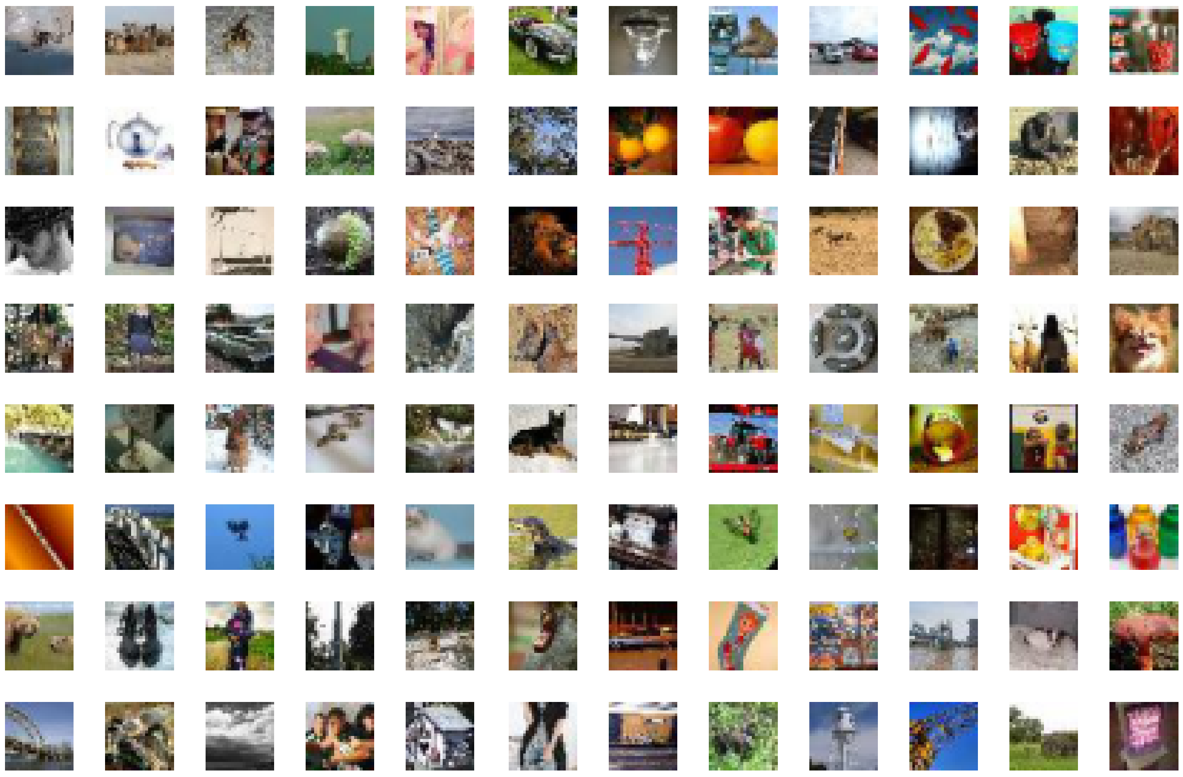


Figure 5.8: Random examples from Tiny ImageNet dataset.

5.1.8 Tiny ImageNet

The Tiny ImageNet [40] is a small set of ImageNet. This dataset has 120,000 images from 200 classes, each image is RGB, and has 64x64 pixels. Examples of this dataset are shown in Figure 5.8.

5.2 Metrics

A set of threshold-dependent and threshold-independent metrics are used to assess the performance of GeMOS over both KKCs and UUCs. These metrics are explained in the subsequent subsections.

5.2.1 Macro F1-score

The main threshold-dependent metric is the macro F1-score, commonly used to assess the performance of models on supervised tasks. This metric calculates the accuracy of the model given the same importance to each class. F1-score is the harmonic mean of precision and recall, represented as:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{5.1}$$

An F1-score equal to zero means either the precision or the recall is zero and one means the best precision and recall value.

By substituting the precision and recall for the confusion matrix fields, F1-score can be represented as:

$$F_1 = \frac{TP}{TP + 0.5 * (FP + FN)}, \tag{5.2}$$

where TP is True Positive values, FP is False Positive values, and FN is False Negative values.

Precision and recall are metrics used to assess the dataset performance. Precision represents the fraction objects that were correctly classified out of all positively classified objects; recall represents the fraction objects that were correctly classified out of all the objects of the class; true positive samples are inputs correctly classified; false positive samples are inputs wrongly classified as the class in evaluation; and false negative are inputs of the class in evaluation wrongly classified.

As the F1-score is a per class metric, in multiclass problems the Macro F1-score is reported. In these cases, F1-score is calculated for each of the classes and the final value is the average of the score of each class.

5.2.2 AUC

The Area Under Curve (AUC) in the classification task between KKCs and UUCs was the main threshold-independent metric, capturing information about the performance of the model without requiring a cutoff point that separates in-distribution from out-of-distribution samples.

The AUC score is derived from the ROC curve (Receiver Operating Characteristic). This curve shows the relationship between the true positive rate (TPR) and the false positive rate (FPR) at different thresholds. The TPR and the FPR are calculated according to:

$$TPR = \frac{TP}{TP + FN}, \tag{5.3}$$

$$FPR = \frac{FP}{FP + TN}, \tag{5.4}$$

where TN is true negative values and represents the correct rejections of a class.

The AUC calculates the area under the curve, generating a score between zero and one. The higher the AUC score, the better the model can distinguish positive from negative classes since the value represents the probability of the occurrence of a correct prediction. In this way, when the score is zero, the model classifies wrongly all input and, when the score is one the model classify correctly all inputs.

5.2.3 FPR95

FPR95 is a metric that measures FPR when TPR is higher than 95%. It is a threshold-independent metric and shows the percentage of OOD samples that were misclassified when the TPR is greater than or equal to 95%. Therefore the metric is 100% when all OOD samples have been misclassified and 0% when all are correctly classified.

5.2.4 Kappa score

Kappa score is a secondary threshold-dependent metric used to measure the level of agreement between two observers considering the possibility of the agreement occurring by chance. The Kappa score k is calculated according to:

$$k = \frac{p_o - p_e}{1 - p_e}, \tag{5.5}$$

where p_o is the empirical probability of agreement for each label and p_e is the randomly expected agreement between both observers. The p_e value is estimated using a per-annotator empirical prior over the labels as proposed by [1]. The Kappa score can variate between -1 and 1, where values lower than 0 represent no agreement values above 0.8 represent good agreement.

5.3 Evaluation Protocol

To verify the effectiveness of the method, some datasets are tested on different CNN models and with several generative models. For this, two types of experiments are defined: the first uses one dataset as D^{in} and the other as D^{out}, and the second uses the same dataset as D^{in} and D^{out}, splitting the dataset classes randomly between these two groups. The steps of each type will be described in the next subsections.

5.3.1 Protocol for D^{in} datasets different from D^{out}

For the tests with one dataset as D^{in} and the other as D^{out} follow the steps:

1. Train the CNN model with D^{in} dataset using the standard train-test split, normalization, and hyperparameters following the article of the dataset;
2. Feed the network with all train inputs, extract, flat, and concatenate the activations of each correctly predicted input, forming one dataset for each class composed of the activations of the inputs;
3. Train one generative model for each class using its dataset;
4. Feed the network with all test inputs of the D^{in} dataset and all D^{out} datasets, extracting, flattening, and concatenating each activation and classification class;
5. Feed the generative model of the classified class with the activation and get the probability score;
6. With all scores, the F1-score, AUC and, FPR95 metrics are generated. As the F1-score is a threshold-dependent metric, its values were acquired by dividing the threshold range (zero to one) by 0.02.

5.3.2 Protocol for D^{in} datasets equal to D^{out}

For the tests with the same dataset as D^{in} and D^{out} follow the steps with five-fold cross-validation:

Table 5.1: List of experiments with MNIST as D^{in} dataset.

CNN Model	D^{out} **Dataset**	**Openness**	**Generative Model**
ResNet-18	Omniglot	0.46	GMM4, GMM8
ResNet-18	EMNIST (letters)	0.34	GMM4, GMM8
ResNet-18	KMNIST	0.18	GMM4, GMM8
ResNet-18	MNIST (Random Split)	0.13	GMM4, GMM8, PCA8
WRN-28-10	MNIST (Random Split)	0.13	PCA8

1. Randomly split the classes between the datasets D^{in} (60%) and D^{out} (40%);
2. Train the CNN model with D^{in} dataset using the standard train-test split, normalization, and hyperparameters following the article of the dataset;
3. Feed the network with all train inputs, extract, flat, and concatenate the activations of each correctly predicted input, forming one dataset for each class composed of the activations of the inputs;
4. Train one generative model for each class using its dataset;
5. Feed the network with all test inputs of the D^{in} dataset and all D^{out} datasets, extracting, flattening, and concatenating each activation and classification class;
6. Feed the generative model of the classified class with the activation and get the probability score;
7. With all scores, the F1-score, AUC and, FPR95 metrics are generated. As the F1-score is a threshold-dependent metric, its values were acquired by dividing the threshold range (zero to one) by 0.02.

5.3.3 List of experiments

Using these steps, a list of experiments is defined to evaluate the method. These experiments are divided into two major groups, one using MNIST dataset as D^{in} and the other using CIFAR10 as D^{in}.

For both cases, different parameters were tested for each generative model. The GMM tests were carried out with two, four, eight, and sixteen mixtures components. For the PCA, tests were carried out keeping two, four, eight, and sixteen components. The IF tests were performed using 100, 200, 300, and 600 estimators in the ensemble. The LOF tests were performed considering 20, 40, 80, and 160 neighbours for the KNN. For the

Table 5.2: List of experiments with CIFAR10 as D^{in} dataset.

CNN Model	D^{out} **Dataset**	**Openness**	**Generative Model**
DenseNet-BC	CIFAR-100	0.59	GMM2, GMM4, GMM8, GMM16, PCA2, PCA4, PCA8, PCA16, IF100, IF200, IF300, IF600, LOF20, LOF40, LOF80, LOF160, OCSVM RBF, OCSVM POLY
DenseNet-BC	Tiny ImageNet (crop)	0.70	GMM2, GMM4, GMM8, GMM16, PCA2, PCA4, PCA8, PCA16
DenseNet-BC	Tiny ImageNet (resize)	0.70	GMM2, GMM4, GMM8, GMM16, PCA2, PCA4, PCA8, PCA16
DenseNet-BC	LSUN (crop)	0.18	GMM2, GMM4, GMM8, GMM16, PCA2, PCA4, PCA8, PCA16
DenseNet-BC	LSUN (resize)	0.18	GMM2, GMM4, GMM8, GMM16, PCA2, PCA4, PCA8, PCA16
DenseNet-BC	CIFAR-10 (Random Split)	0.13	GMM8, PCA8
WRN-28-10	CIFAR-100	0.59	GMM4, GMM8
WRN-28-10	Tiny ImageNet (crop)	0.70	GMM4, GMM8
WRN-28-10	Tiny ImageNet (resize)	0.70	GMM4, GMM8
WRN-28-10	LSUN (crop)	0.18	GMM4, GMM8
WRN-28-10	LSUN (resize)	0.18	GMM4, GMM8

OCSVM, tests were performed using the radial basis (RBF) and polynomial functions as the kernel to transform the nonlinear space into a higher dimensional linear space.

Using MNIST as a D^{in} dataset following the experiments listed in Table 5.1, the hyperparameters used to train the ResNet-18 and WRN-28-10 are learning rate: $5e^{-3}$ with a scheduler for 20 steps with ratio 0.2, number of training epoch: 30, and optimizer: Adam with a momentum equal to 0.9 and weight decay equal to $5e^{-5}$. The number of the referent parameters tested for each generative model is described in Table 5.1 after the name.

Using CIFAR10 as D^{in} dataset following the experiments listed in Table 5.2, the hyperparameters used to train the DenseNet-BC and WRN-28-10 are learning rate: 0.1 with a scheduler on epoch 150 and 250 with ratio 0.1, number of training epoch: 400, and optimizer: SGD with a momentum equal to 0.9 and weight decay equal to $5e^{-4}$. The number of the referent parameters tested for GMM, PCA, IF, and LOF model and the kernel type of OCSVM are described in Table 5.2 after the name.

MNIST methods use 670Mb from GPU memory to load the backbone model, and 3Mb for each GMM. To train the GMM of one class of MNIST takes 2 seconds. CIFAR10 methods use 630Mb from GPU memory to load the backbone model, and 180Mb for each GMM. To train the GMM of one class of CIFAR10 takes 120 seconds. The difference in memory and time is because CIFAR10 uses more internal layers of the network to train the generative models.

Chapter 6

Results and Discussion

This chapter shows the results of all experiments and a brief discussion about them. In Section 6.1 the utilization of unimodal and multimodal generative models is discussed. In Section 6.2 the experiments using one dataset as D^{in} and the other as D^{out} are discussed and compared with the baselines. In Section 6.3 experiments using the same dataset as D^{in} and D^{out}, dividing classes randomly are discussed and compared with the baselines.

Table 6.1: AUC and F1-score comparison for different generative models on D^{in} = CIFAR10/D^{out} = CIFAR100.

Model	**F1-Score↑**	**AUC↑**	**FPR95↓**
GMM2	0.83	0.90	0.51
GMM4	0.83	**0.91**	0.47
GMM8	**0.84**	**0.91**	**0.46**
GMM16	**0.84**	**0.91**	0.47
PCA2	0.82	0.89	0.57
PCA4	0.82	0.89	0.56
PCA8	0.83	0.90	0.54
PCA16	0.83	0.90	0.54
IF100	0.76	0.88	0.51
IF200	0.77	0.89	0.48
IF300	0.81	0.89	0.49
IF600	0.79	0.89	0.47
LOF20	0.81	0.89	0.52
LOF40	0.81	0.89	0.52
LOF80	0.83	0.90	0.52
LOF160	0.80	0.90	0.52
OCSVM RBF	0.82	0.89	0.55
OCSVM POLY	0.80	0.84	0.49

6.1 Unimodal versus Multimodal Generative Models

This section describes the tests comparing unimodal and multimodal generative models. These experiments are used to define which models will be used in the other experiments.

Table 6.1 shows F1, AUC, and PFR95 scores for GMM with two, four, eight, and sixteen components, PCA with two, four, eight, and sixteen components, IF with 100, 200, 300, and 600 components, LOF with 20, 40, 80, and 160 components, and OCSVM with RBF and POLY kernels using DenseNet as the backbone with CIFAR10 as D^{in} and CIFAR100 as D^{out}. Bold text marks the best result for each metric. GMM with eight and sixteen components presented the best results and, thus, GMM8 was the model chosen as the standard for the baseline comparisons. The optimal number of components on GMM may vary with dataset and task because that in some cases different numbers of components are still tested as well as PCA models. The other types of generative models do not achieve similar scores compared with GMM and PCA, besides having greater variation in different configurations.

The good performance of GMM is attributed to its multimodality capabilities, which is not commonly found in reconstruction based DNNs as AEs or VAEs or in other shallow generative models (e.g. PCA, IF, LOF and OCSVM). This is because real-world data, even within a single class, does not commonly follow unimodal distributions.

6.2 D^{in} not equal D^{out}

The tests were separated using one dataset as D^{in} and another dataset as D^{out} in subgroups by the D^{in} dataset. In Subsection 6.2.1 the experiments using the MNIST dataset as D^{in} are described and compared with the baselines. In Subsection 6.2.2 the experiments using the CIFAR10 dataset as D^{in} are described and compared with the baselines. All baseline results listed were reported in your original articles.

Table 6.2: F1-score, AUC and FPR95 for OSR with D^{in} = MNIST.

D^{out}	OMNIGLOT			EMNIST			KMNIST		
Metrics	**F1↑**	**AUC↑**	**FPR95↓**	**F1↑**	**AUC↑**	**FPR95↓**	**F1↑**	**AUC↑**	**FPR95↓**
SoftMax	0.60	-	-	-	-	-	-	-	-
OpenMax	0.78	-	-	-	-	-	-	-	-
ODIN	-	**1.00**	-	-	-	-	-	-	-
CROSR	0.79	-	-	-	-	-	-	-	-
CGDL	0.85	-	-	-	-	-	-	-	-
GeMOS(GMM4)	0.90	0.97	0.10	0.71	0.96	**0.13**	0.90	0.98	0.05
GeMOS(GMM8)	**0.91**	0.97	**0.09**	**0.74**	**0.97**	**0.13**	**0.92**	**0.99**	**0.03**

6.2.1 MNIST as D^{in}

Table 6.2 shows the F1, AUC, and FPR95 scores for MNIST as D^{in} and Omniglot, EMNIST and KMNIST as D^{out}. Bold text marks the best result for each metric. In this case, GeMOS was tested with ResNet-18 as the backbone and GMM with four and eight components. Since MNIST is a simpler dataset for this type of test, GMM with sixteen components and PCA were not tested.

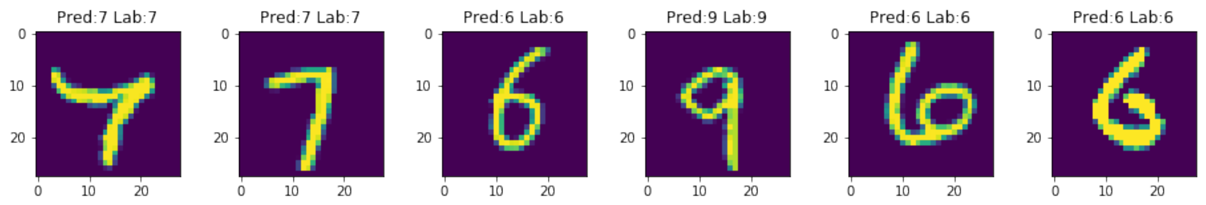
The results are reported in Table 6.2, with GeMOS(GMM8) presenting the best results in F1-score on Omniglot followed by CGDL, CRSOR, OpenMax, and SoftMax thresholding. The other metrics and dataset were not reported in the literature, except by the AUC on Omniglot where ODIN outperforms GeMOS. Comparing the results reported, GeMOS shows to be consistently better than other state-of-art architectures.

Examples of recognized samples from each dataset can be seen in Figure 6.1 for OMNIGLOT, Figure 6.2 for EMNIST, and Figure 6.3 for KMNIST. The best F1-score threshold was used in all cases of recognition examples and reported metrics. Some images were wrongly recognized mainly because of the trade-off between the classification of known classes and recognition of unknown classes. The normalized confusion matrix for these cases can be seen in Figure 6.4 for OMNIGLOT, Figure 6.5 for EMNIST, and Figure 6.6 for KMNIST.

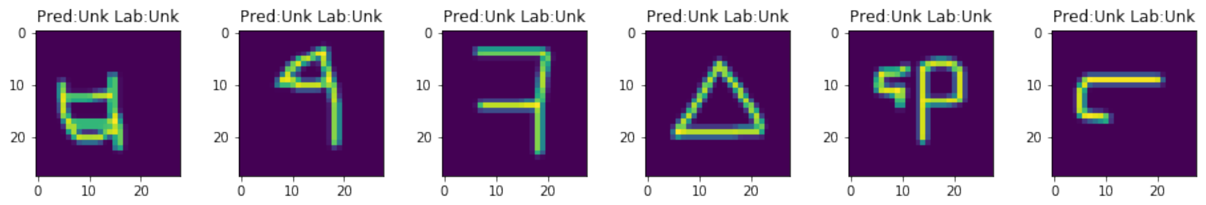
In Figure 6.4, it is possible to observe that classes “3”, “7” and “9” have the largest number of FN and that classes “0”, “1”, and “4” have the largest FP. Observing the hits (TP) unknown objects have the highest TP and the FP rate is much lower than the FN rate. The other classes that have a higher TP are ‘0” and “6”.

In Figure 6.5, it is possible to observe that classes “3” and “8” have the largest number of FN and that class “0” has the largest FP. Observing the hits (TP) unknown objects have the highest TP followed by the class “6” and the FP rate is much lower than the FN rate. Another point that stands out, in this case, is that the hits of the known classes, in this case, are much smaller than in the others.

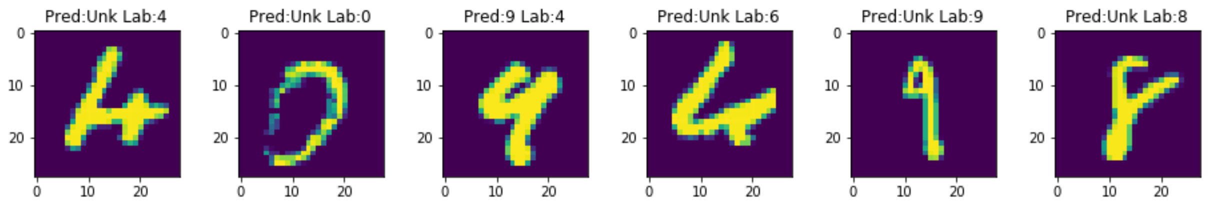
In Figure 6.6 it is possible to observe that classes “8” and “9” have the highest



(a) Correctly predicted for MNIST as D^{in} using ResNet-18 with GMM8.

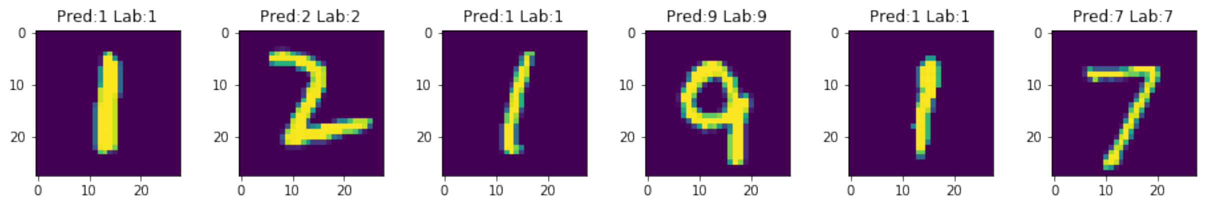


(b) Correctly predicted for OMNIGLOT as D^{out} using ResNet-18 with GMM8.

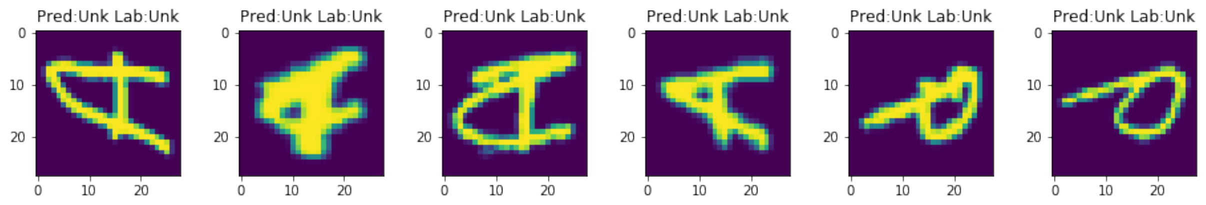


(c) Wrong predictions for MNIST as D^{in} and OMNIGLOT as D^{out} using ResNet-18 with GMM8.

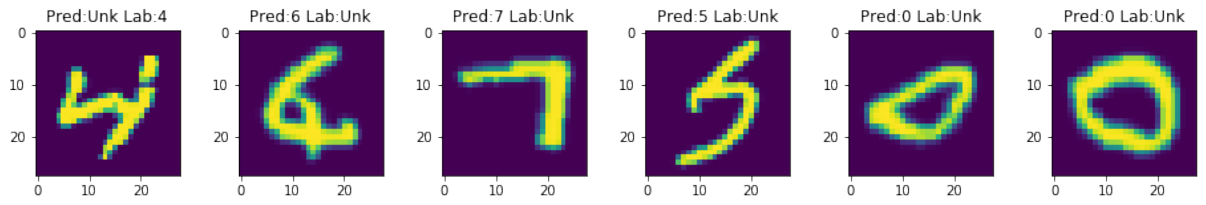
Figure 6.1: Prediction examples for MNIST as D^{in} and OMNIGLOT as D^{out} using ResNet-18 with GMM8.



(a) Correctly predicted for MNIST as D^{in} using ResNet-18 with GMM8.

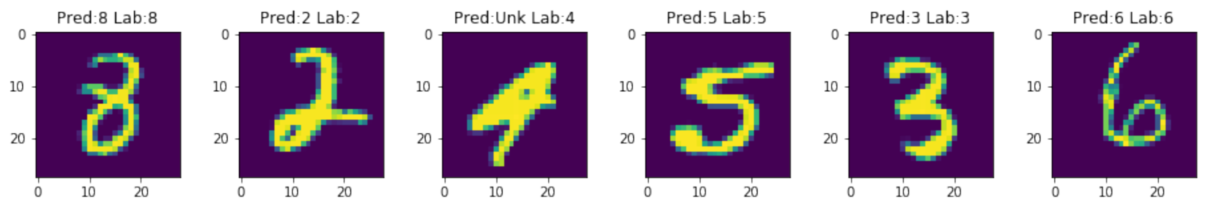


(b) Correctly predicted for EMNIST as D^{out} using ResNet-18 with GMM8.

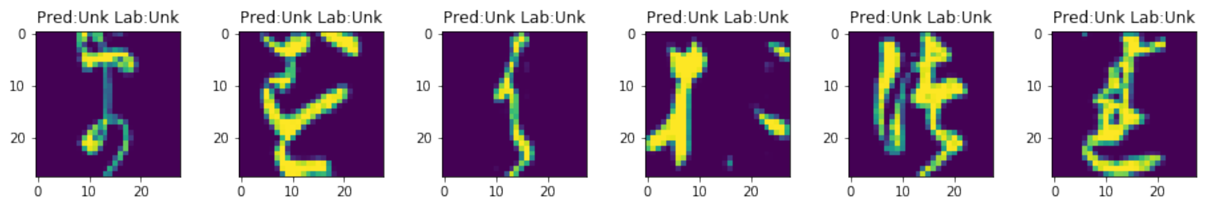


(c) Wrong predictions for MNIST as D^{in} and EMNIST as D^{out} using ResNet-18 with GMM8.

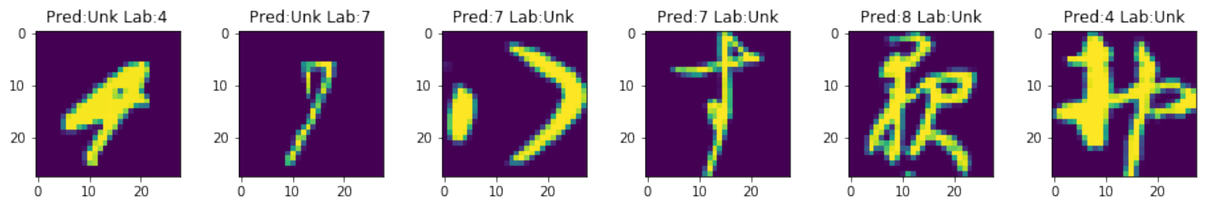
Figure 6.2: Prediction examples for MNIST as D^{in} and EMNIST as D^{out} using ResNet-18 with GMM8.



(a) Correctly predicted for MNIST as D^{in} using ResNet-18 with GMM8.



(b) Correctly predicted for KMNIST as D^{out} using ResNet-18 with GMM8.



(c) Wrong predictions for MNIST as D^{in} and KMNIST as D^{out} using ResNet-18 with GMM8.

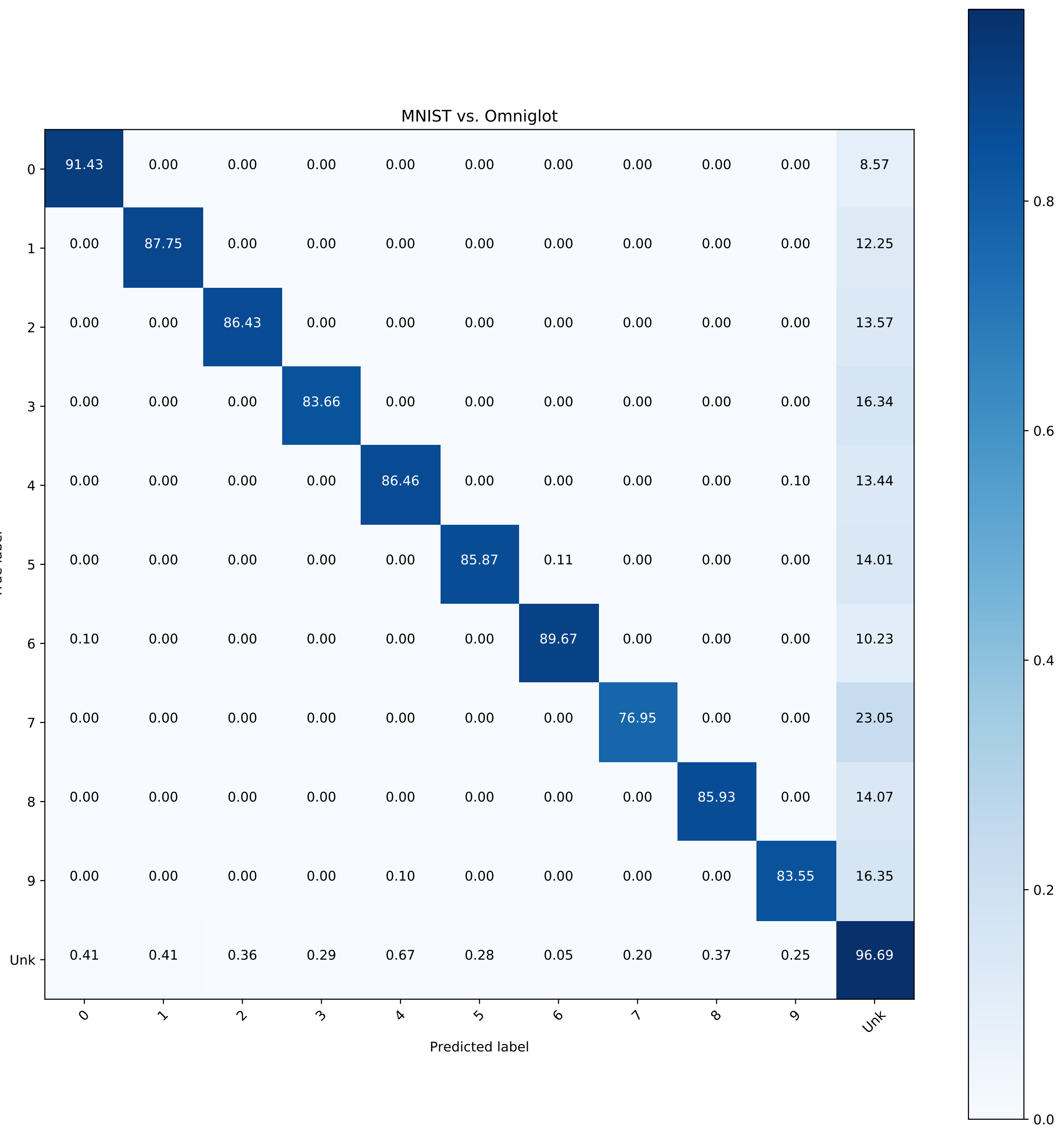
Figure 6.3: Prediction examples for MNIST as D^{in} and KMNIST as D^{out} using ResNet-18 with GMM8.

number of FN and that class “3” has the highest FP. Observing the hits (TP) unknown objects have the highest TP followed by class “6”, “3”, and “4” and the FP rate is much lower than the FN rate despite this case having the highest F1-score.

Comparing these three cases, the classes “3”, “8”, and “9” are the ones that are most confused with unknown objects. Among other classes, “6” tends to have the highest TP. Analyzing the FP, the rate is low, and it does not seem to have a class that stands out in all cases.

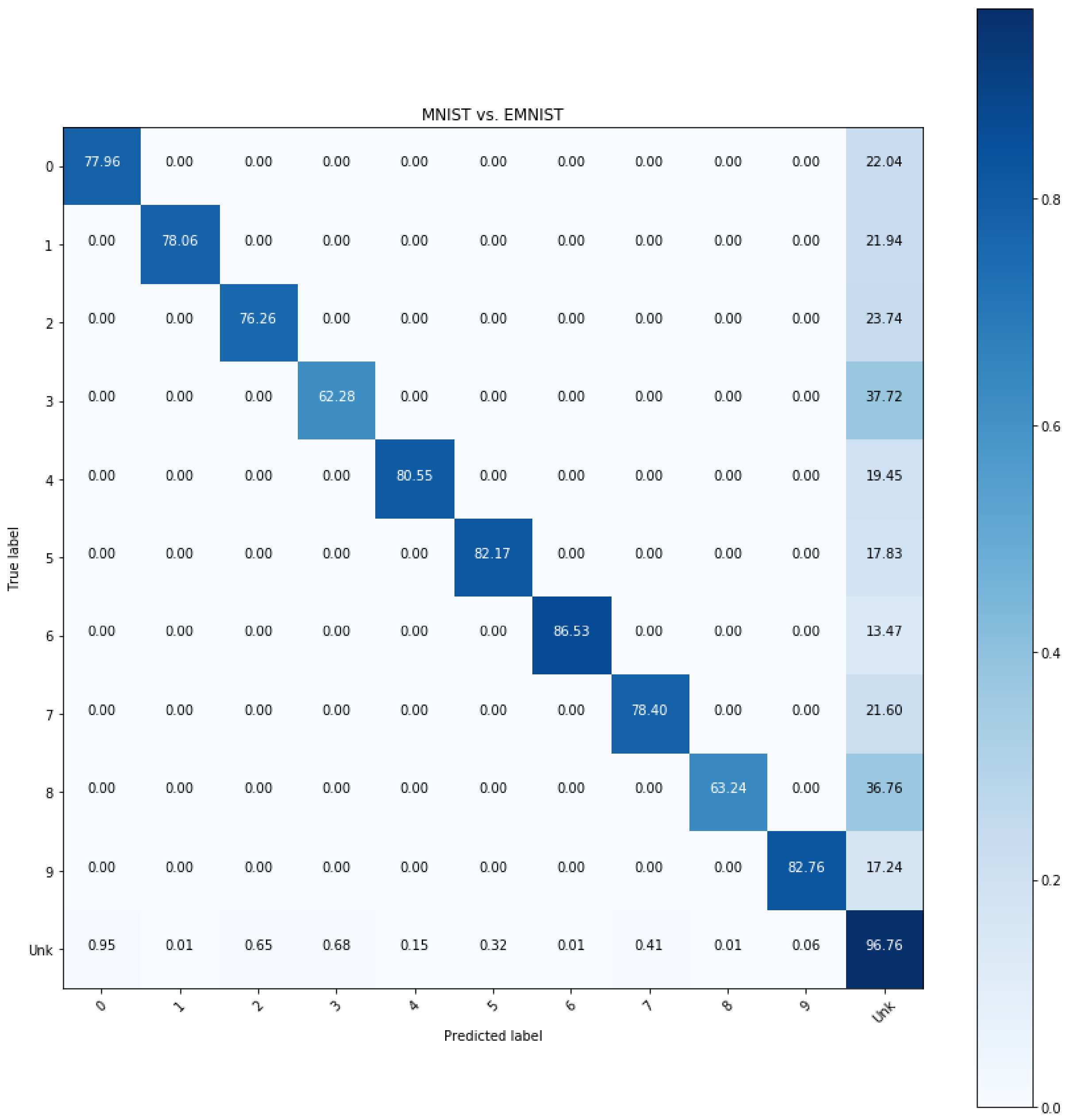
To visualize the impact of threshold on this trade-off between classification and recognition on MNIST, Figure 6.7, 6.8, and 6.9, shows the metrics by a range of threshold for OMNIGLOT, EMNIST, and KMNIST respectively. In these graphs are represented the accuracy of known objects (red dot line), the precision and recall of unknown objects (green and blue dot line respectively), the balanced accuracy (cyan dot line), F1 for unknown objects (yellow dot line), F1 macro (black dot line), and kappa score (red dot line).

In Figure 6.9, it is possible to observe that the metrics for unknown and known objects converge around the threshold of 0.8 while in Figure 6.7 and 6.8 this conversion is not so clear and happens around the threshold of 0.6 and 0.8. Another point that can be observed is that for OMNIGLOT and KMNIST the metrics tend to converge to one point while for EMNIST the F1 macro and the kappa score do not come close to the metrics of



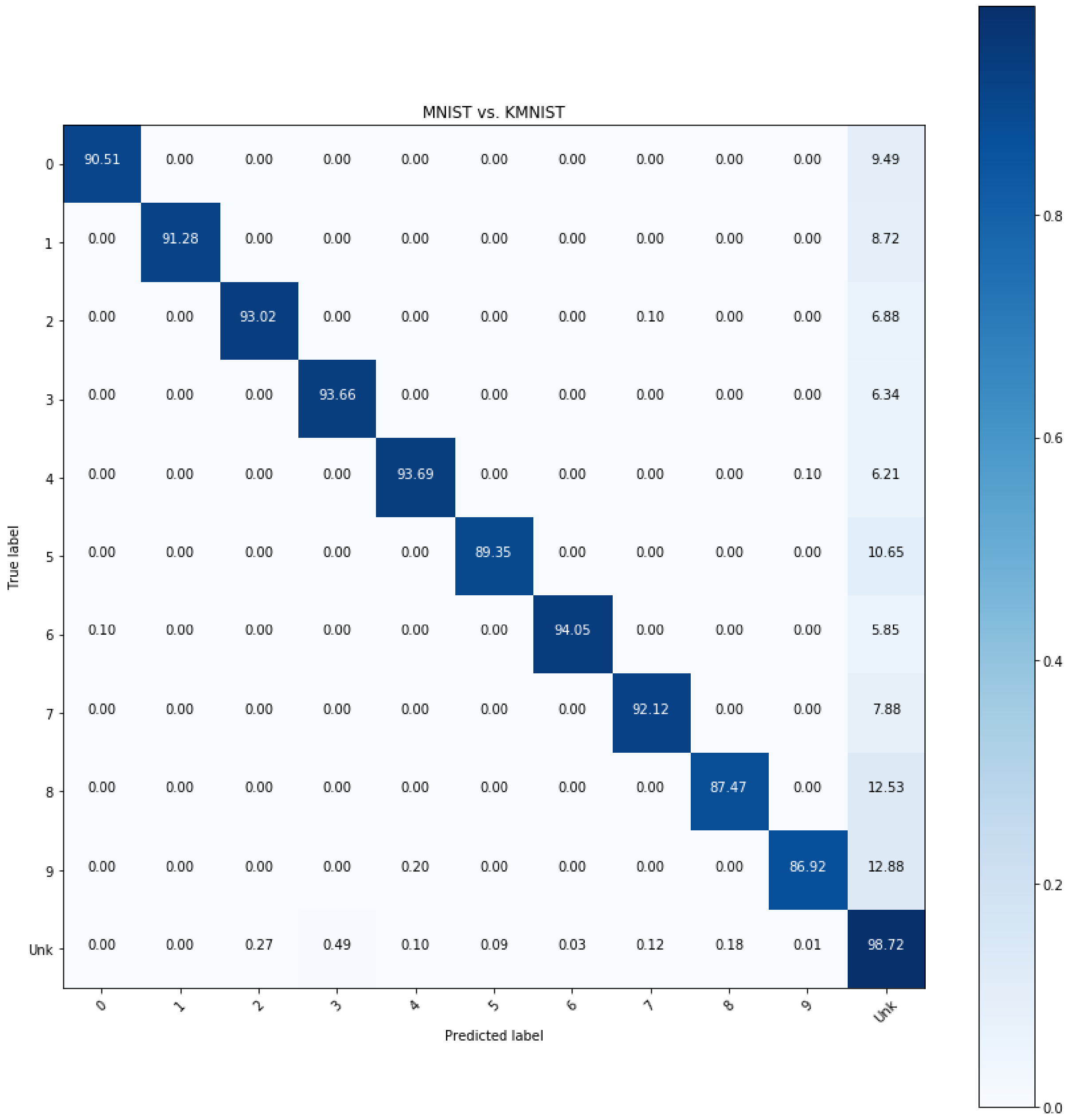
	0	1	2	3	4	5	6	7	8	9	Unk
0	91.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.57
1	0.00	87.75	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	12.25
2	0.00	0.00	86.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	13.57
3	0.00	0.00	0.00	83.66	0.00	0.00	0.00	0.00	0.00	0.00	16.34
4	0.00	0.00	0.00	0.00	86.46	0.00	0.00	0.00	0.00	0.10	13.44
5	0.00	0.00	0.00	0.00	0.00	85.87	0.11	0.00	0.00	0.00	14.01
6	0.10	0.00	0.00	0.00	0.00	0.00	89.67	0.00	0.00	0.00	10.23
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	76.95	0.00	0.00	23.05
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	85.93	0.00	14.07
9	0.00	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.00	83.55	16.35
Unk	0.41	0.41	0.36	0.29	0.67	0.28	0.05	0.20	0.37	0.25	96.69

Figure 6.4: Normalized confusion matrix for MNIST as D^{in} and OMNIGLOT as D^{out} using ResNet-18t with GMM8.



	0	1	2	3	4	5	6	7	8	9	Unk
0	77.96	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	22.04
1	0.00	78.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	21.94
2	0.00	0.00	76.26	0.00	0.00	0.00	0.00	0.00	0.00	0.00	23.74
3	0.00	0.00	0.00	62.28	0.00	0.00	0.00	0.00	0.00	0.00	37.72
4	0.00	0.00	0.00	0.00	80.55	0.00	0.00	0.00	0.00	0.00	19.45
5	0.00	0.00	0.00	0.00	0.00	82.17	0.00	0.00	0.00	0.00	17.83
6	0.00	0.00	0.00	0.00	0.00	0.00	86.53	0.00	0.00	0.00	13.47
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	78.40	0.00	0.00	21.60
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	63.24	0.00	36.76
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	82.76	17.24
Unk	0.95	0.01	0.65	0.68	0.15	0.32	0.01	0.41	0.01	0.06	96.76

Figure 6.5: Normalized confusion matrix for MNIST as D^{in} and EMNIST as D^{out} using ResNet-18 with GMM8.



	0	1	2	3	4	5	6	7	8	9	Unk
0	90.51	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9.49
1	0.00	91.28	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.72
2	0.00	0.00	93.02	0.00	0.00	0.00	0.00	0.10	0.00	0.00	6.88
3	0.00	0.00	0.00	93.66	0.00	0.00	0.00	0.00	0.00	0.00	6.34
4	0.00	0.00	0.00	0.00	93.69	0.00	0.00	0.00	0.00	0.10	6.21
5	0.00	0.00	0.00	0.00	0.00	89.35	0.00	0.00	0.00	0.00	10.65
6	0.10	0.00	0.00	0.00	0.00	0.00	94.05	0.00	0.00	0.00	5.85
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	92.12	0.00	0.00	7.88
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	87.47	0.00	12.53
9	0.00	0.00	0.00	0.00	0.20	0.00	0.00	0.00	0.00	86.92	12.88
Unk	0.00	0.00	0.27	0.49	0.10	0.09	0.03	0.12	0.18	0.01	98.72

Figure 6.6: Normalized confusion matrix for MNIST as D^{in} and KMNIST as D^{out} using ResNet-18 with GMM8.

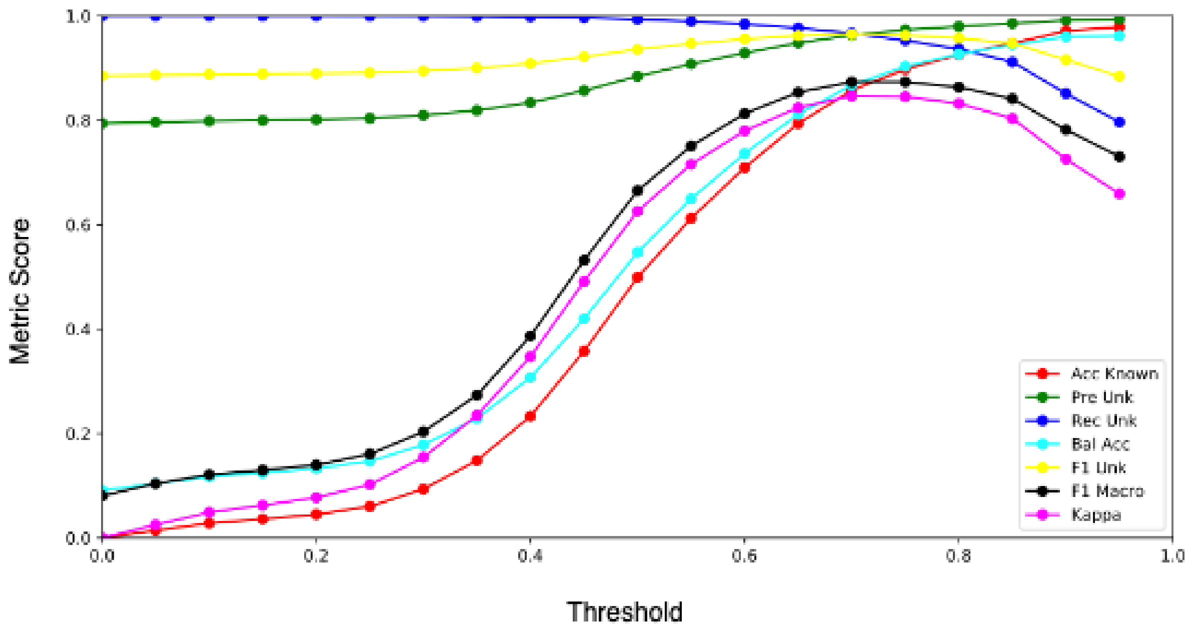


Figure 6.7: Plot with main metrics score (axis y) by threshold range (axis x) for MNIST as D^{in} and OMNIGLOT as D^{out} using ResNet-18 with GMM8.

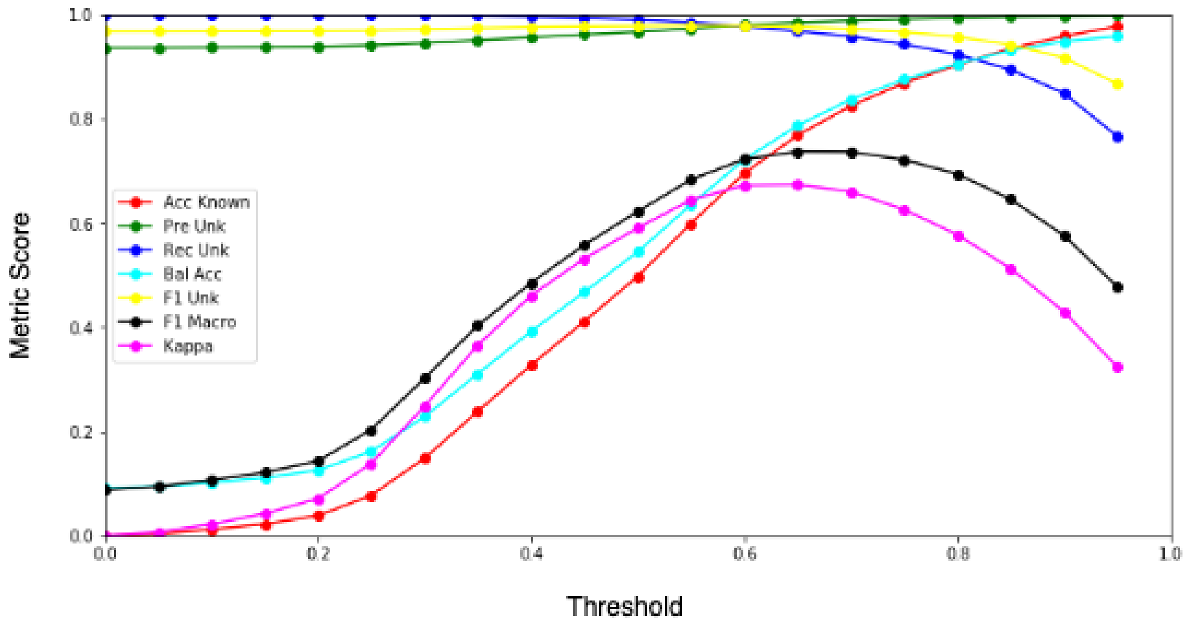


Figure 6.8: Plot with main metrics score (axis y) by threshold range (axis x) for MNIST as D^{in} and EMNIST as D^{out} using ResNet-18 with GMM8.

unknown objects.

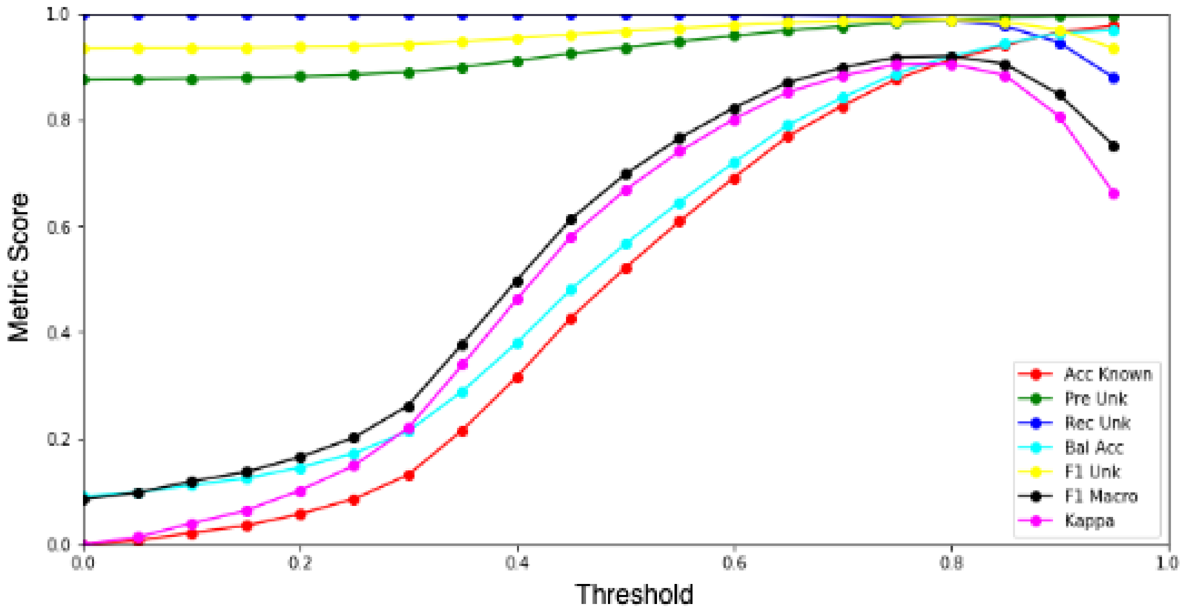


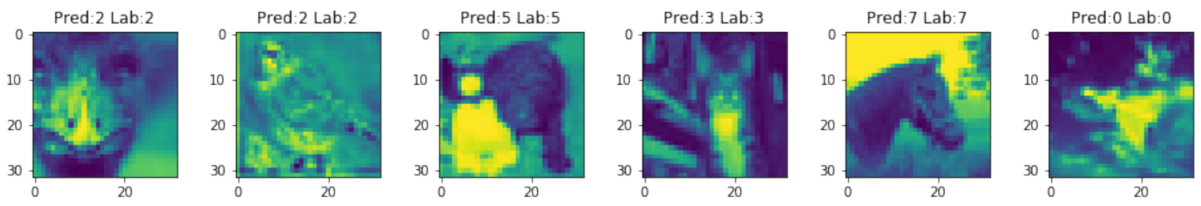
Figure 6.9: Plot with main metrics score (axis y) by threshold range (axis x) for MNIST as D^{in} and KMNIST as D^{out} using ResNet-18 with GMM8.

Table 6.3: F1-score, AUC and FPR95 for OSR with D^{in} = CIFAR10.

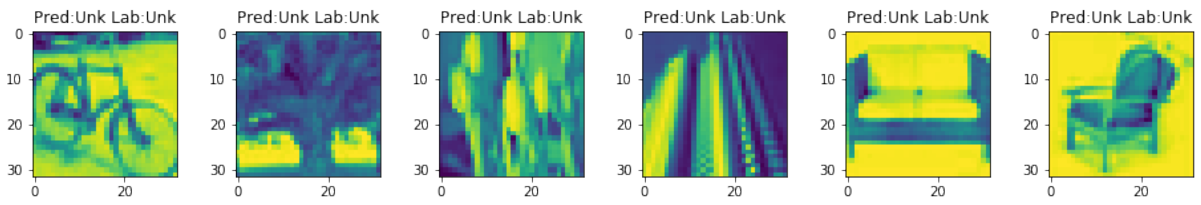
D^{out}	CIFAR100			Tiny ImageNet (crop)			Tiny ImageNet (resize)			LSUN (crop)			LSUN (resize)		
Metrics	**F1↑**	**AUC↑**	**FPR95↓**	**F1↑**	**AUC↑**	**FPR95↓**	**F1↑**	**AUC↑**	**FPR95↓**	**F1↑**	**AUC↑**	**FPR95↓**	**F1↑**	**AUC↑**	**FPR95↓**
SoftMax	-	-	-	0.64	-	-	0.65	-	-	0.64	-	-	0.65	-	-
OpenMax	-	-	-	0.66	-	-	0.68	-	-	0.66	-	-	0.67	-	-
ODIN	-	0.90	0.47	-	0.94	**0.04**	-	0.92	0.07	-	0.96	**0.09**	-	0.95	0.04
OODNN	-	-	-	-	0.96	-	-	0.95	-	-	**0.97**	-	-	0.96	-
CROSR	-	-	-	0.72	-	-	0.74	-	-	0.72	-	-	0.75	-	-
C2AE	-	-	-	0.84	-	-	0.83	-	-	0.78	-	-	0.80	-	-
CGDL	-	-	-	0.84	-	-	0.83	-	-	0.81	-	-	0.81	-	-
GeMOS (DN-GMM2)	0.83	0.90	0.51	**0.92**	0.97	0.23	0.90	0.95	0.34	**0.92**	0.96	0.29	0.91	0.96	0.27
GeMOS (DN-GMM4)	0.83	**0.91**	0.47	**0.92**	0.97	0.22	0.88	0.96	0.31	**0.92**	**0.97**	0.25	0.91	0.97	0.23
GeMOS (DN-GMM8)	**0.84**	**0.91**	**0.46**	0.91	0.97	0.21	0.89	0.95	0.32	**0.92**	**0.97**	0.25	0.91	0.97	0.22
GeMOS (DN-GMM16)	**0.84**	**0.91**	0.47	0.90	0.96	0.25	0.87	0.94	0.37	0.91	0.96	0.27	0.90	0.96	0.24
GeMOS (DN-PCA2)	0.82	0.89	0.58	0.85	0.91	0.56	0.84	0.90	0.58	0.84	0.91	0.64	0.87	0.93	0.46
GeMOS (DN-PCA4)	0.82	0.89	0.56	0.86	0.92	0.55	0.84	0.91	0.58	0.86	0.92	0.61	0.87	0.94	0.43
GeMOS (DN-PCA8)	0.83	0.89	0.54	0.87	0.93	0.47	0.85	0.92	0.51	0.87	0.93	0.54	0.88	0.95	0.35
GeMOS (DN-PCA16)	0.83	0.90	0.54	0.88	0.94	0.42	0.87	0.93	0.47	0.87	0.93	0.49	0.89	0.95	0.39
GeMOS (WRN-GMM4)	0.79	0.88	0.56	**0.92**	**0.99**	0.05	**0.92**	**0.99**	**0.04**	0.91	**0.97**	0.19	**0.93**	**0.99**	**0.02**
GeMOS (WRN-GMM8)	0.80	0.89	0.55	**0.92**	0.98	0.08	**0.92**	**0.99**	0.05	0.90	0.96	0.23	**0.93**	**0.99**	**0.02**

6.2.2 CIFAR10 as D^{in}

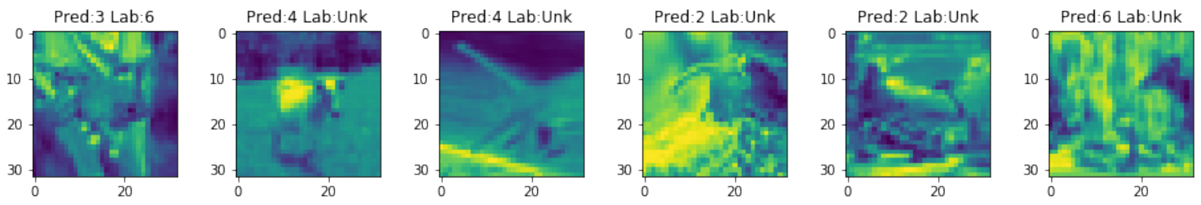
Table 6.3 shows the F1, AUC, and FPR95 scores respectively for CIFAR10 as D^{in} and CIFAR100, Tiny ImageNet (crop), Tiny ImageNet (resize), LSUN (crop), and LSUN (resize) as D^{out}. Bold text marks the best result for each metric. For the GeMOS results, the network used on each test is referenced before the generative model name. GeMOS



(a) Correctly predicted for CIFAR10 as D^{in} using DenseNet with GMM8.



(b) Correctly predicted for CIFAR100 as D^{out} using DenseNet with GMM8.



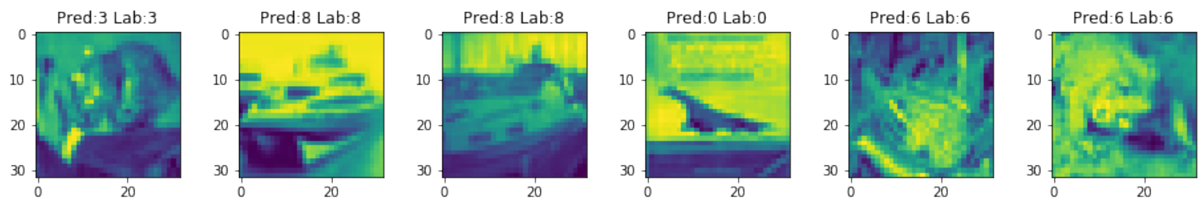
(c) Wrong predictions for CIFAR10 as D^{in} and CIFAR100 as D^{out} using DenseNet with GMM8.

Figure 6.10: Prediction examples for CIFAR10 as D^{in} and CIFAR100 as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.

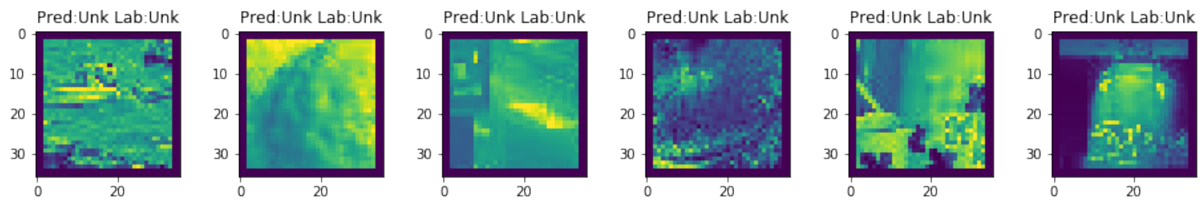
outperforms all other methods in all datasets and presents the best results using DenseNet and WRN with GMM4 and GMM8 except by the FPR95 on Tiny ImageNet (crop) and LSUN (crop) where ODIN achieves the best score. Since CIFAR10 is considerably more complex than MNIST, these results show consistency in the OOD detection capabilities of GeMOS even in harder scenarios. GeMOS scores in different backbones do not vary much since both backbones have high accuracy in the closed set task, showing that the method can work with distinct pre-trained CNNs.

Examples of recognized samples from each dataset can be seen in Figure 6.10 for CIFAR100, Figure 6.11 for Tiny ImageNet (crop), Figure 6.12 for Tiny ImageNet (resize), Figure 6.13 LSUN (crop), and Figure 6.14 for LSUN (resize). The best F1-score threshold was used in all cases of recognition examples and reported metrics. Some images were wrongly recognized mainly because the trade-off between classification known classes and recognition unknown classes. The normalized confusion matrix for these cases can be seen in Figure 6.15 for CIFAR100, Figure 6.16 for Tiny ImageNet (crop), Figure 6.17 for Tiny ImageNet (resize), Figure 6.18 LSUN (crop), and Figure 6.19 for LSUN (resize). For theses cases the class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.

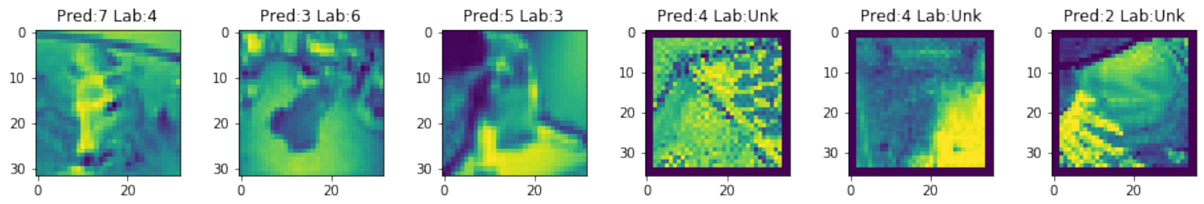
In Figure 6.15, it is possible to observe that class “3 - cat” has the highest FN. About TPs, class “3 - cat” is the one that has the lowest score. Classes “3 - cat” and “5 -



(a) Correctly predicted for CIFAR10 as D^{in} using DenseNet with GMM8.



(b) Correctly predicted for Tiny ImageNet (crop) as D^{out} using DenseNet with GMM8.



(c) Wrong predictions for CIFAR10 as D^{in} and Tiny ImageNet (crop) as D^{out} using DenseNet with GMM8.

Figure 6.11: Prediction examples for CIFAR10 as D^{in} and Tiny ImageNet (crop) as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.

dog" have the highest FN.

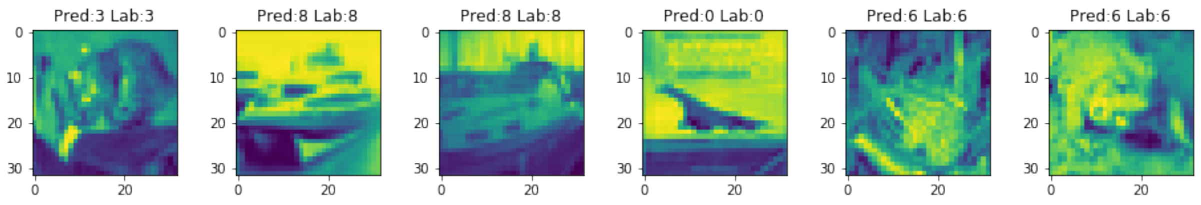
In Figure 6.16, it is possible to observe that class "3 - cat" also has the highest FN. About TP, class "3 - cat" is the one with the lowest score and the unknown objects have the highest score. Classes "3 - cat" and "6 - frog" have the highest NF.

Figure 6.17 is similar to the 6.16, in it is possible to observe that class "3 - cat" also has the highest FN. About TP, class "3 - cat" is the one with the lowest score and the unknown objects have the highest score. Class "6 - frog" has the highest NF.

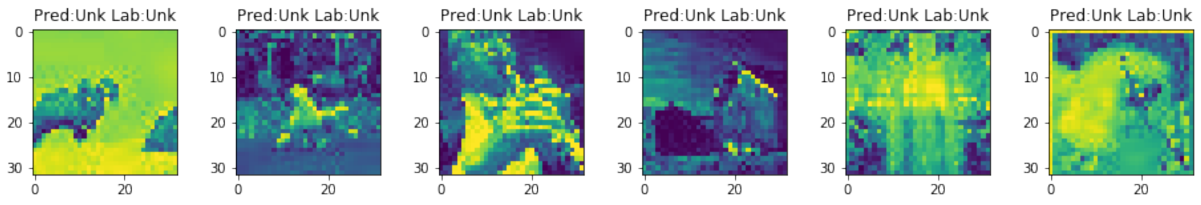
In Figure 6.18, it is possible to observe that classes "2 - bird" and "3 - cat" have higher FN. About TP, class "3 - cat" is the one that has the lowest score and the unknown objects have the highest score. Class "3 - cat" also features the highest NF.

Figure 6.19 is similar to the Figure 6.18, in which classes "2 - bird" and "3 - cat" have higher FN. About TP, class "3 - cat" is the one that has the lowest score and the unknown objects have the highest score. Class "9 - truck" has the highest NF.

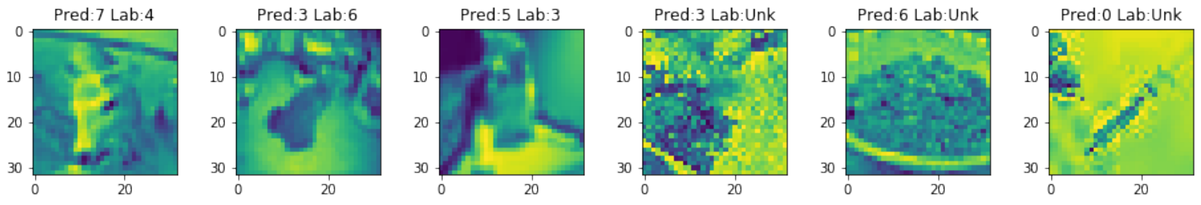
Comparing these confusion matrices, it is observed that class "3 - cat" is the one that presents the lowest correctness in all cases. It is noteworthy that the change in the pre-processing of the image from crop to resize does not change the behavior of the result in both Tiny ImageNet and LSUN. Another point noted is that the CIFAR100 as a D^{out} dataset has the least recognition of unknown objects.



(a) Correctly predicted for CIFAR10 as D^{in} using DenseNet with GMM8.



(b) Correctly predicted for Tiny ImageNet (resize) as D^{out} using DenseNet with GMM8.

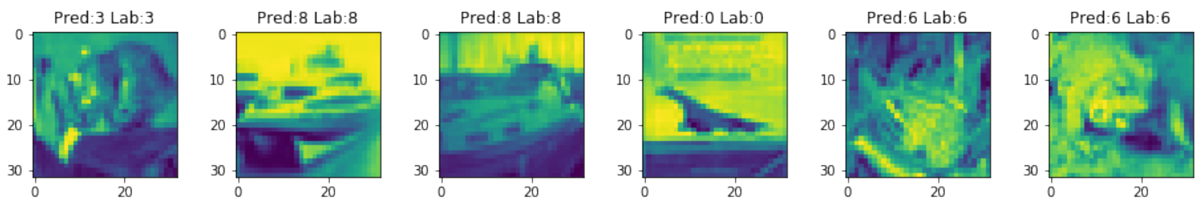


(c) Wrong predictions for CIFAR10 as D^{in} and Tiny ImageNet (resize) as D^{out} using DenseNet with GMM8.

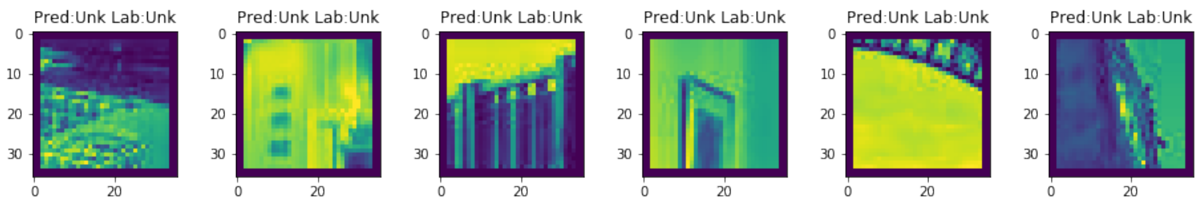
Figure 6.12: Prediction examples for CIFAR10 as D^{in} and Tiny ImageNet (resize) as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.

To visualize the impact of threshold on this trade-off between classification and recognition on CIFAR10, Figure 6.20, 6.21, 6.22, 6.23, and 6.24, shows the metrics by threshold for CIFAR100, Tiny ImageNet (crop), Tiny ImageNet (resize), LSUN (crop), and LSUN (resize) respectively. In these graphs are represented the accuracy of known objects (red dot line), the precision and recall of unknown objects (green and blue dot line respectively), the balanced accuracy (cyan dot line), F1 for unknown objects (yellow dot line), F1 macro (black dot line), and kappa score (red dot line).

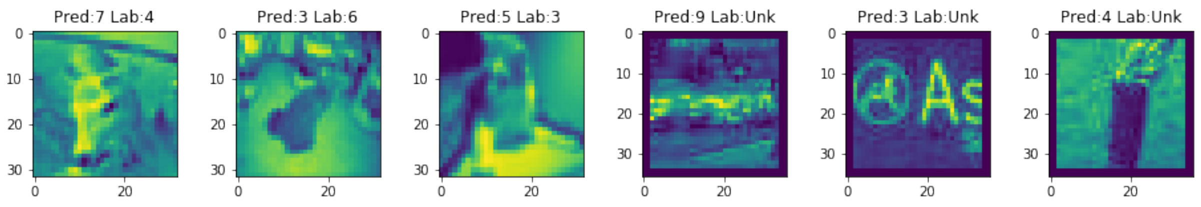
In Figure 6.20, 6.21, 6.22, 6.23, and 6.24, it is possible to observe that the metrics for unknown and known objects converge at some point. In Figure 6.20 the metrics converge around the threshold of 0.7 while in the others, the conversion happens around the threshold of 0.8. In Figure 6.20, it is possible to observe that the recall of unknown objects falls much earlier in other cases, in addition to reaching a lower value at the end of the threshold range. all CIFAR10 graphics have similar behavior.



(a) Correctly predicted for CIFAR10 as D^{in} using DenseNet with GMM8.



(b) Correctly predicted for LSUN (crop) as D^{out} using DenseNet with GMM8.



(c) Wrong predictions for CIFAR10 as D^{in} and LSUN (crop) as D^{out} using DenseNet with GMM8.

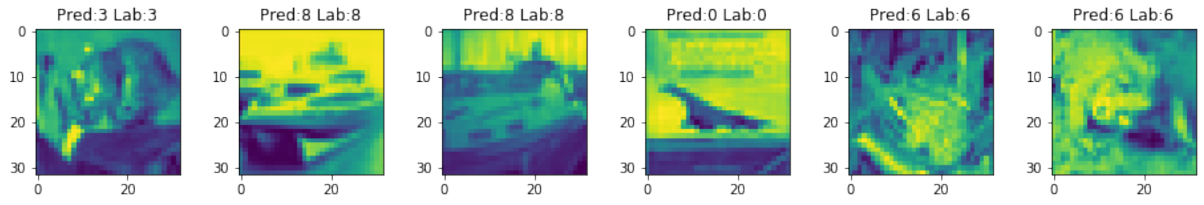
Figure 6.13: Prediction examples for CIFAR10 as D^{in} and LSUN as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.

6.3 D^{in} equal D^{out}

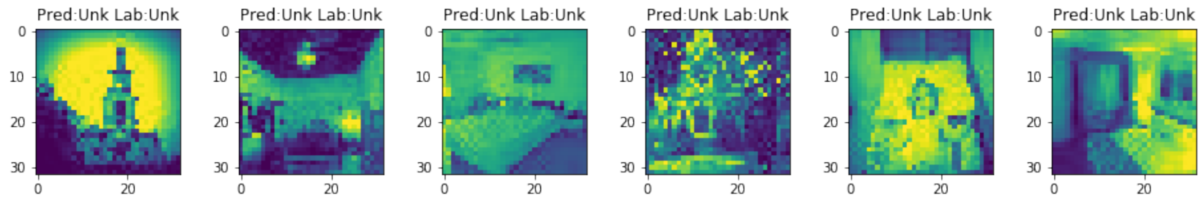
The tests using the same dataset as D^{in} and D^{out} are separated in subgroups by the dataset. In Subsection 6.3.1 the experiments using the MNIST dataset are described and compared with the baselines. In Subsection 6.3.2 the experiments using the CIFAR10 dataset are described and compared with the baselines. All baseline results listed were reported in your original articles.

6.3.1 MNIST

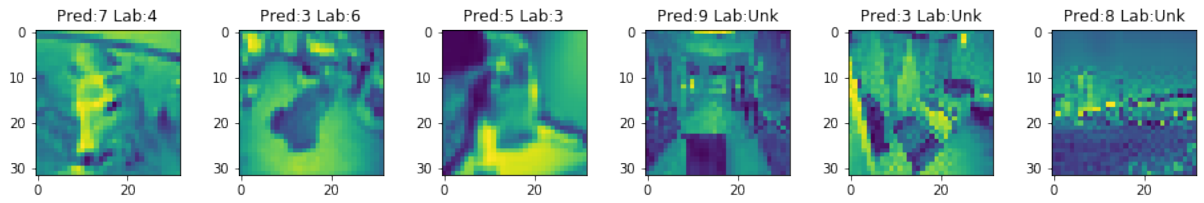
Table 6.4 shows AUC for MNIST as D^{in} and D^{out}. Bold text marks the best result for each metric. For the GeMOS results, the network used on each test is referenced before the generative model name. That case is harder to recognize UUC because these



(a) Correctly predicted for CIFAR10 as D^{in} using DenseNet with GMM8.



(b) Correctly predicted for LSUN (resize) as D^{out} using DenseNet with GMM8.

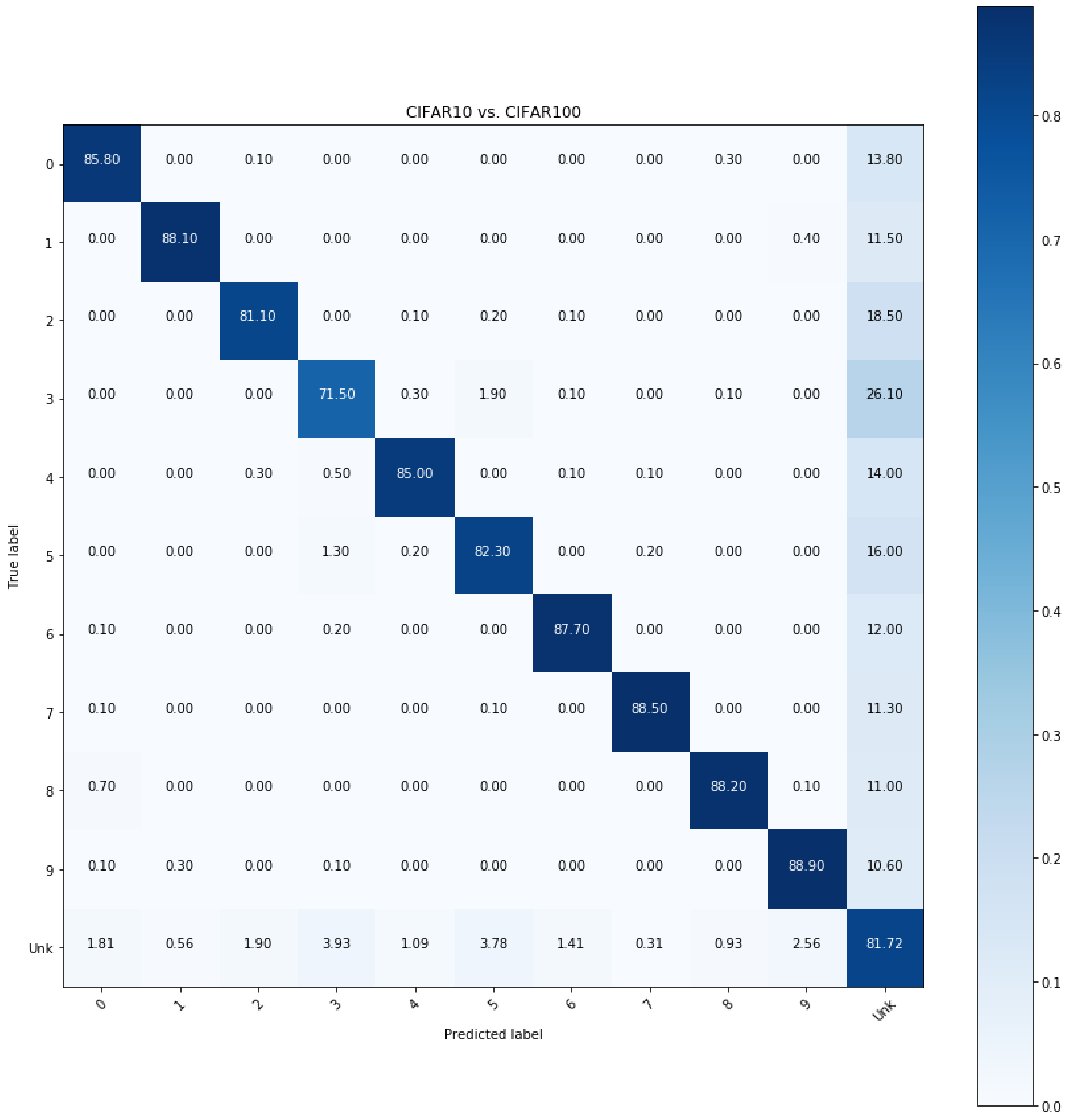


(c) Wrong predictions for CIFAR10 as D^{in} and LSUN (resize) as D^{out} using DenseNet with GMM8.

Figure 6.14: Prediction examples for CIFAR10 as D^{in} and LSUN (resize) as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.

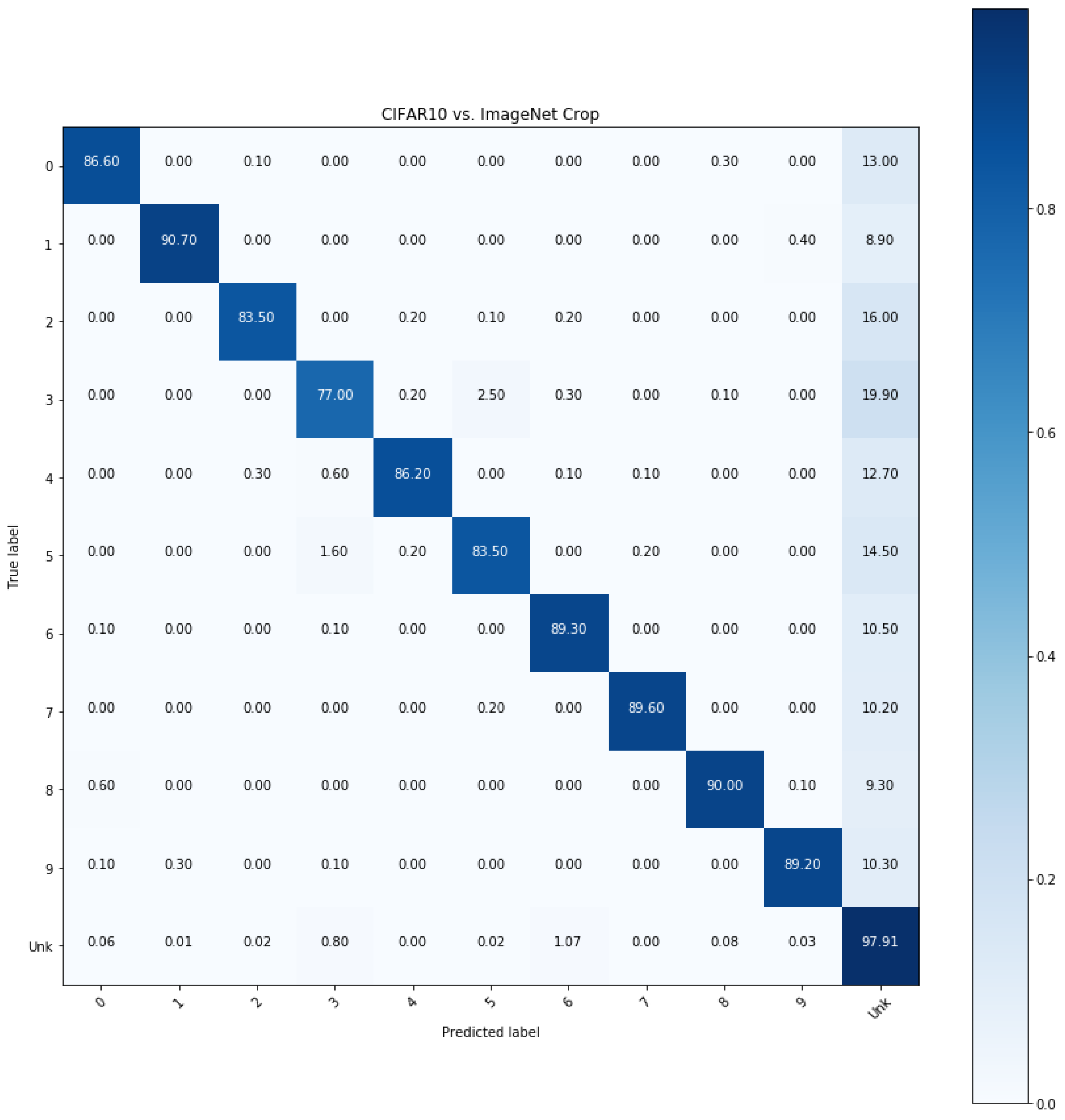
Table 6.4: AUC for OSR for MNIST.

Method	**MNIST↑**
SoftMax	0.978 ± 0.002
OpenMax	0.981 ± 0.002
G-OpenMax	0.984 ± 0.001
OSRCI	0.988 ± 0.001
C2AE	0.989 ± 0.002
CGDL	**0.994 ± 0.002**
GeMOS (RN-GMM4)	0.973 ± 0.009
GeMOS (RN-GMM8)	0.971 ± 0.015
GeMOS (WRN-PCA8)	0.952 ± 0.013
GeMOS (RN-PCA8)	0.954 ± 0.013



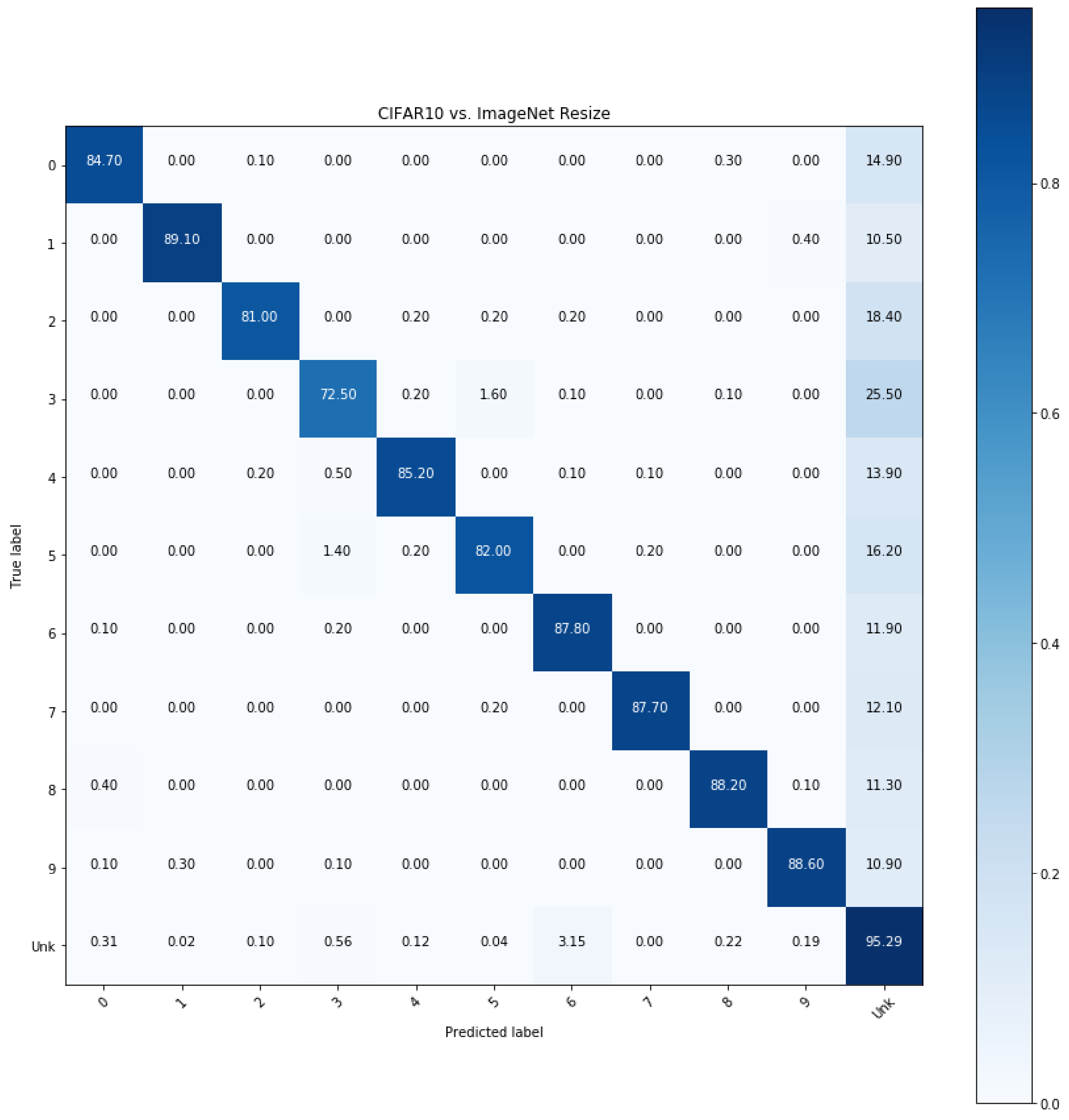
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	Unk
0	85.80	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.30	0.00	13.80
1	0.00	88.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	11.50
2	0.00	0.00	81.10	0.00	0.10	0.20	0.10	0.00	0.00	0.00	18.50
3	0.00	0.00	0.00	71.50	0.30	1.90	0.10	0.00	0.10	0.00	26.10
4	0.00	0.00	0.30	0.50	85.00	0.00	0.10	0.10	0.00	0.00	14.00
5	0.00	0.00	0.00	1.30	0.20	82.30	0.00	0.20	0.00	0.00	16.00
6	0.10	0.00	0.00	0.20	0.00	0.00	87.70	0.00	0.00	0.00	12.00
7	0.10	0.00	0.00	0.00	0.00	0.10	0.00	88.50	0.00	0.00	11.30
8	0.70	0.00	0.00	0.00	0.00	0.00	0.00	0.00	88.20	0.10	11.00
9	0.10	0.30	0.00	0.10	0.00	0.00	0.00	0.00	0.00	88.90	10.60
Unk	1.81	0.56	1.90	3.93	1.09	3.78	1.41	0.31	0.93	2.56	81.72

Figure 6.15: Normalized confusion matrix for CIFAR10 as D^{in} and CIFAR100 as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.



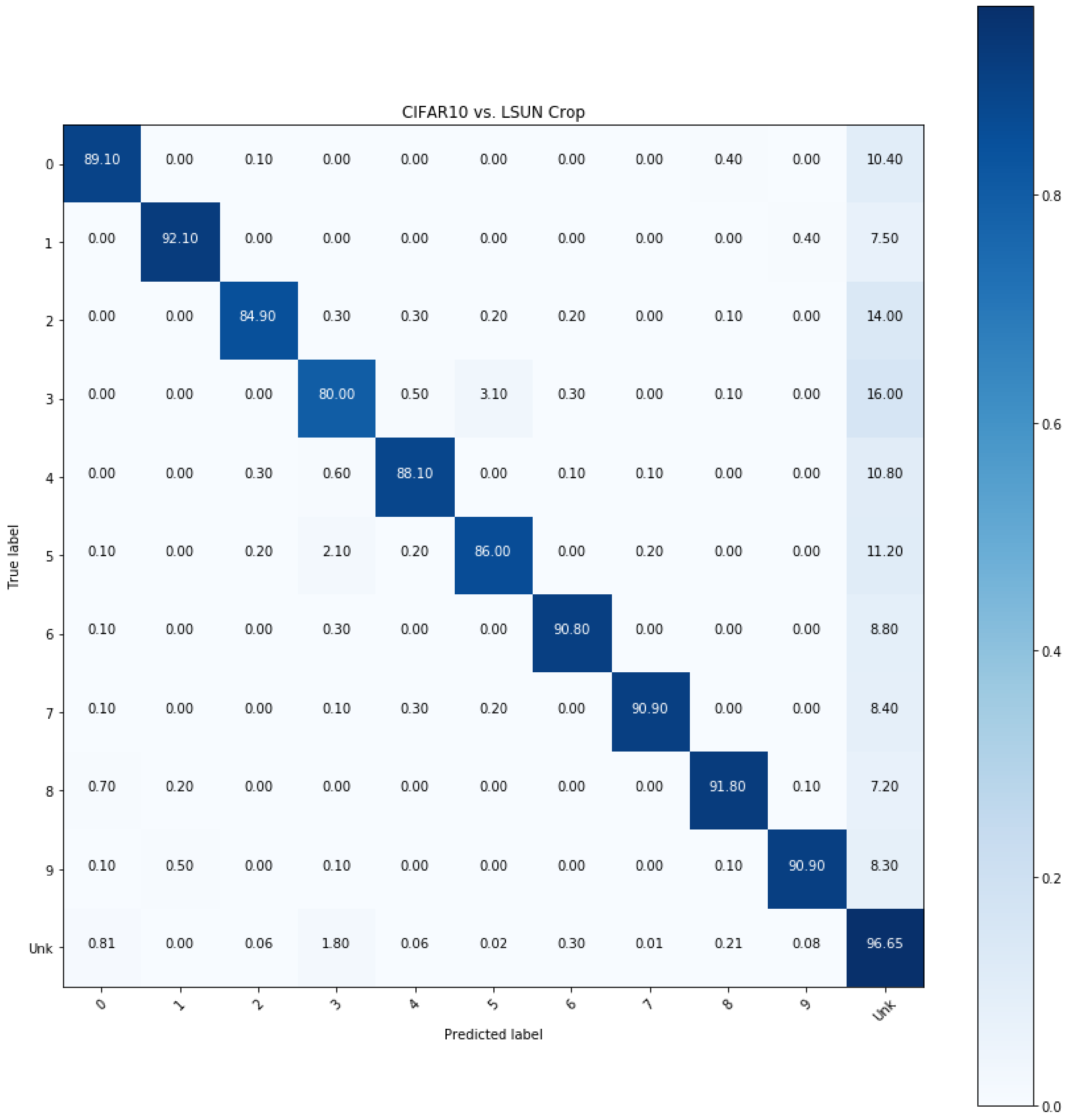
	0	1	2	3	4	5	6	7	8	9	Unk
0	86.60	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.30	0.00	13.00
1	0.00	90.70	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	8.90
2	0.00	0.00	83.50	0.00	0.20	0.10	0.20	0.00	0.00	0.00	16.00
3	0.00	0.00	0.00	77.00	0.20	2.50	0.30	0.00	0.10	0.00	19.90
4	0.00	0.00	0.30	0.60	86.20	0.00	0.10	0.10	0.00	0.00	12.70
5	0.00	0.00	0.00	1.60	0.20	83.50	0.00	0.20	0.00	0.00	14.50
6	0.10	0.00	0.00	0.10	0.00	0.00	89.30	0.00	0.00	0.00	10.50
7	0.00	0.00	0.00	0.00	0.00	0.20	0.00	89.60	0.00	0.00	10.20
8	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	90.00	0.10	9.30
9	0.10	0.30	0.00	0.10	0.00	0.00	0.00	0.00	0.00	89.20	10.30
Unk	0.06	0.01	0.02	0.80	0.00	0.02	1.07	0.00	0.08	0.03	97.91

Figure 6.16: Normalized confusion matrix for CIFAR10 as D^{in} and Tiny ImageNet (crop) as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	Unk
0	84.70	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.30	0.00	14.90
1	0.00	89.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	10.50
2	0.00	0.00	81.00	0.00	0.20	0.20	0.20	0.00	0.00	0.00	18.40
3	0.00	0.00	0.00	72.50	0.20	1.60	0.10	0.00	0.10	0.00	25.50
4	0.00	0.00	0.20	0.50	85.20	0.00	0.10	0.10	0.00	0.00	13.90
5	0.00	0.00	0.00	1.40	0.20	82.00	0.00	0.20	0.00	0.00	16.20
6	0.10	0.00	0.00	0.20	0.00	0.00	87.80	0.00	0.00	0.00	11.90
7	0.00	0.00	0.00	0.00	0.00	0.20	0.00	87.70	0.00	0.00	12.10
8	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	88.20	0.10	11.30
9	0.10	0.30	0.00	0.10	0.00	0.00	0.00	0.00	0.00	88.60	10.90
Unk	0.31	0.02	0.10	0.56	0.12	0.04	3.15	0.00	0.22	0.19	95.29

Figure 6.17: Normalized confusion matrix for CIFAR10 as D^{in} and Tiny ImageNet (resize) as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	Unk
0	89.10	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.40	0.00	10.40
1	0.00	92.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	7.50
2	0.00	0.00	84.90	0.30	0.30	0.20	0.20	0.00	0.10	0.00	14.00
3	0.00	0.00	0.00	80.00	0.50	3.10	0.30	0.00	0.10	0.00	16.00
4	0.00	0.00	0.30	0.60	88.10	0.00	0.10	0.10	0.00	0.00	10.80
5	0.10	0.00	0.20	2.10	0.20	86.00	0.00	0.20	0.00	0.00	11.20
6	0.10	0.00	0.00	0.30	0.00	0.00	90.80	0.00	0.00	0.00	8.80
7	0.10	0.00	0.00	0.10	0.30	0.20	0.00	90.90	0.00	0.00	8.40
8	0.70	0.20	0.00	0.00	0.00	0.00	0.00	0.00	91.80	0.10	7.20
9	0.10	0.50	0.00	0.10	0.00	0.00	0.00	0.00	0.10	90.90	8.30
Unk	0.81	0.00	0.06	1.80	0.06	0.02	0.30	0.01	0.21	0.08	96.65

Figure 6.18: Normalized confusion matrix for CIFAR10 as D^{in} and LSUN (crop) as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.

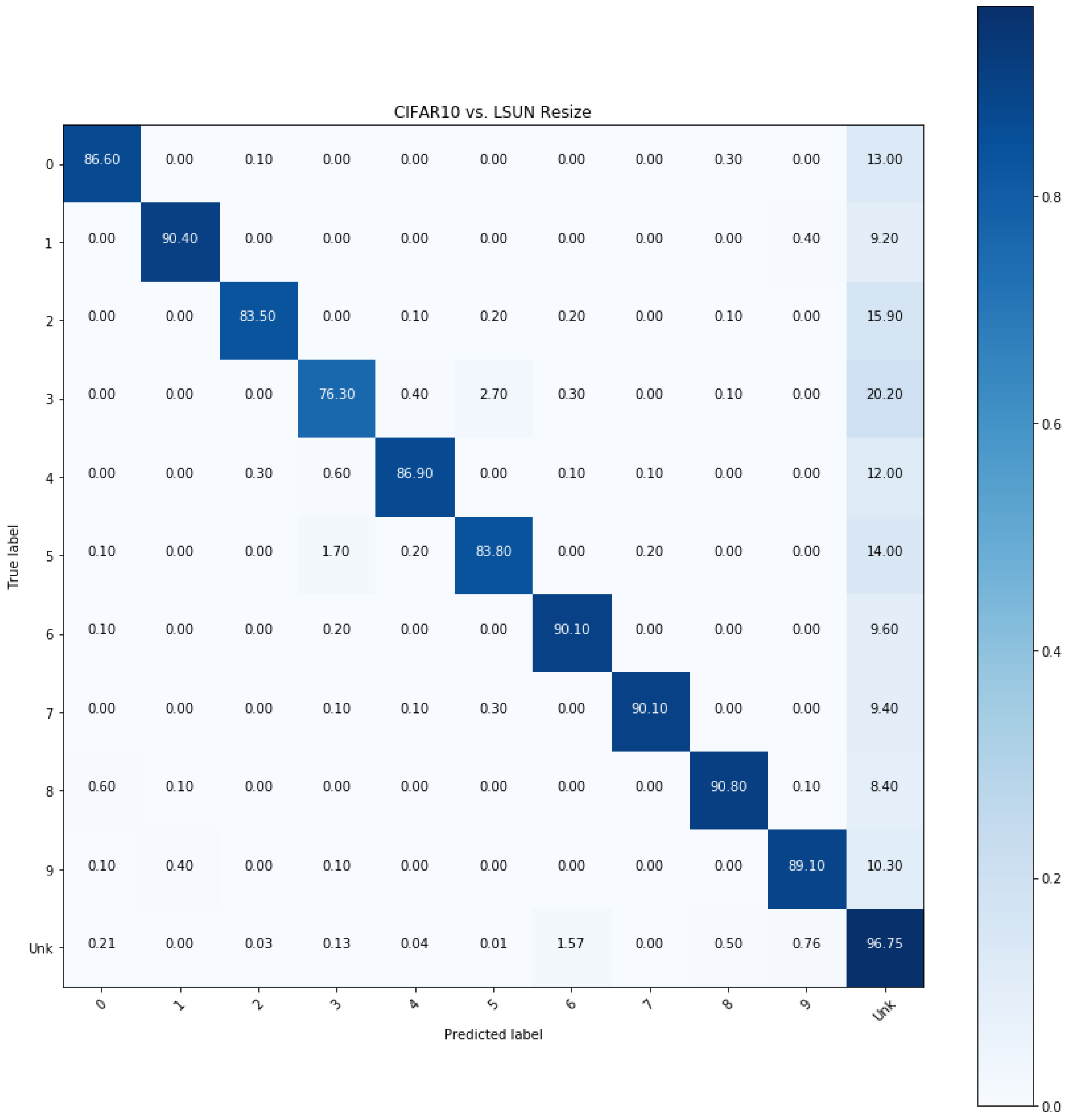


Figure 6.19: Normalized confusion matrix for CIFAR10 as D^{in} and LSUN (resize) as D^{out} using DenseNet with GMM8. The class number 0-9 represents respectively: airplane; automobile; bird; cat; deer; dog; frog; horse; ship; and truck.

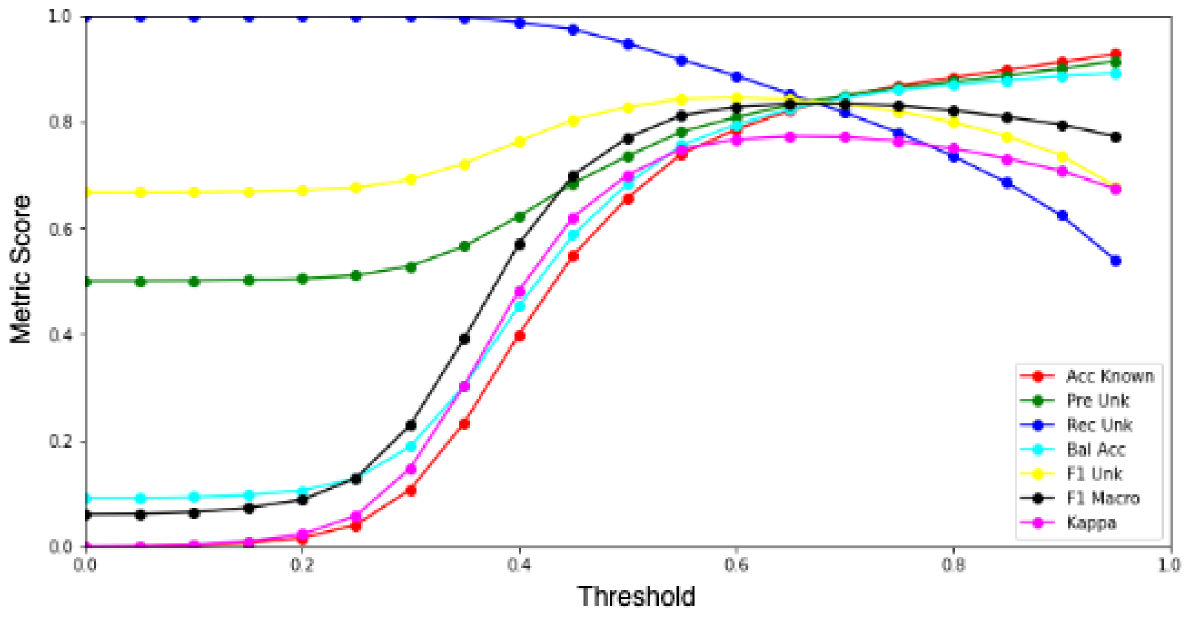


Figure 6.20: Plot with main metrics score (axis y) by threshold range (axis x) for CIFAR10 as D^{in} and CIFAR100 as D^{out} using DenseNet with GMM8.

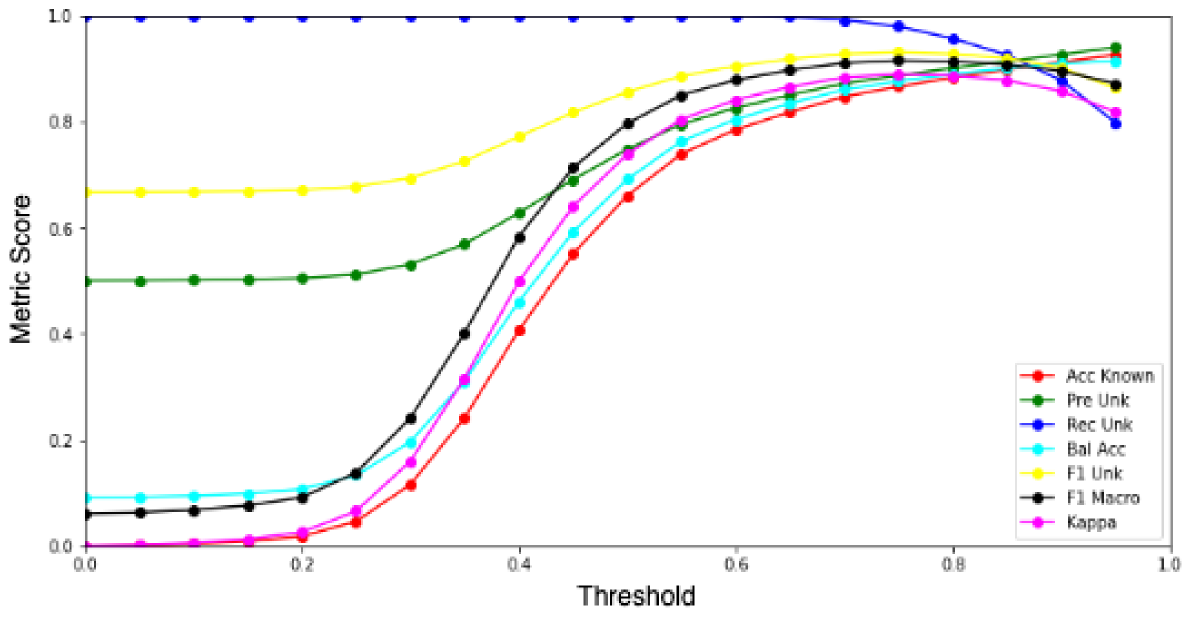


Figure 6.21: Plot with main metrics score (axis y) by threshold range (axis x) for CIFAR10 as D^{in} and Tiny ImageNet (crop) as D^{out} using DenseNet with GMM8.

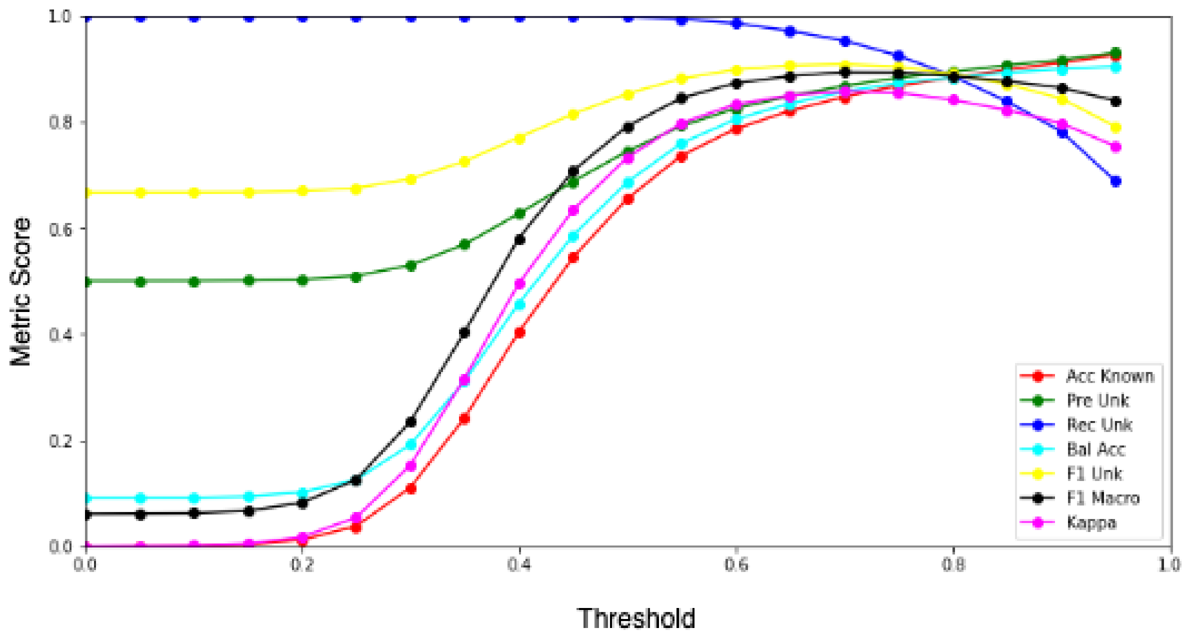


Figure 6.22: Plot with main metrics score (axis y) by threshold range (axis x) for CIFAR10 as D^{in} and Tiny ImageNet (resize) as D^{out} using DenseNet with GMM8.

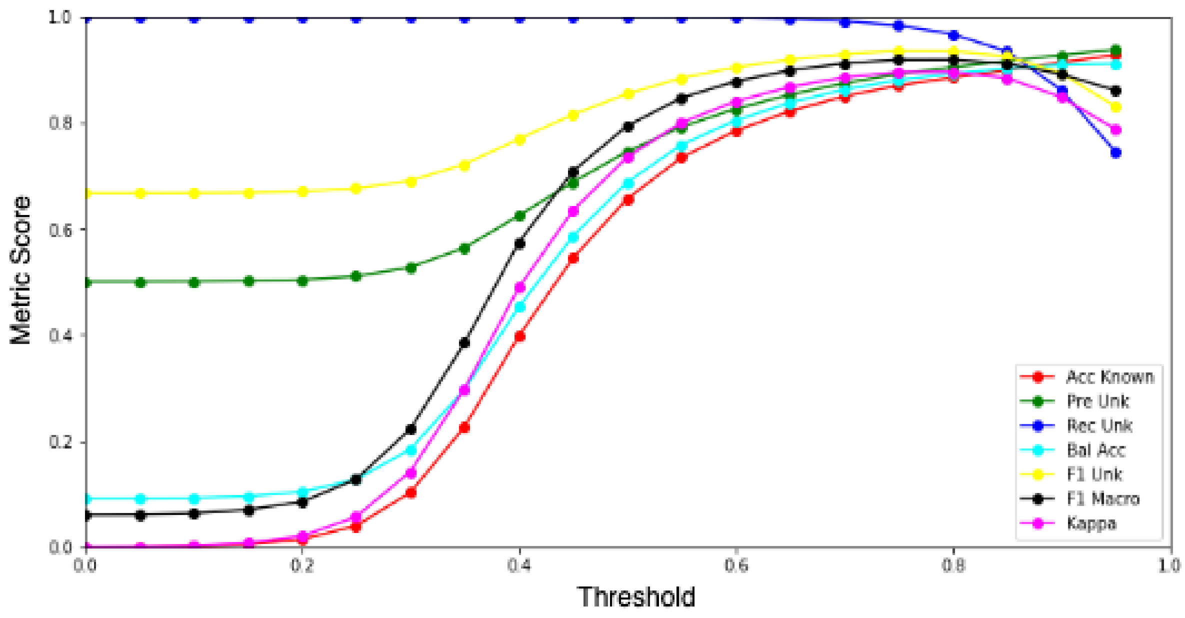


Figure 6.23: Plot with main metrics score (axis y) by threshold range (axis x) for as D^{in} and LSUN (crop) as D^{out} using DenseNet with GMM8.

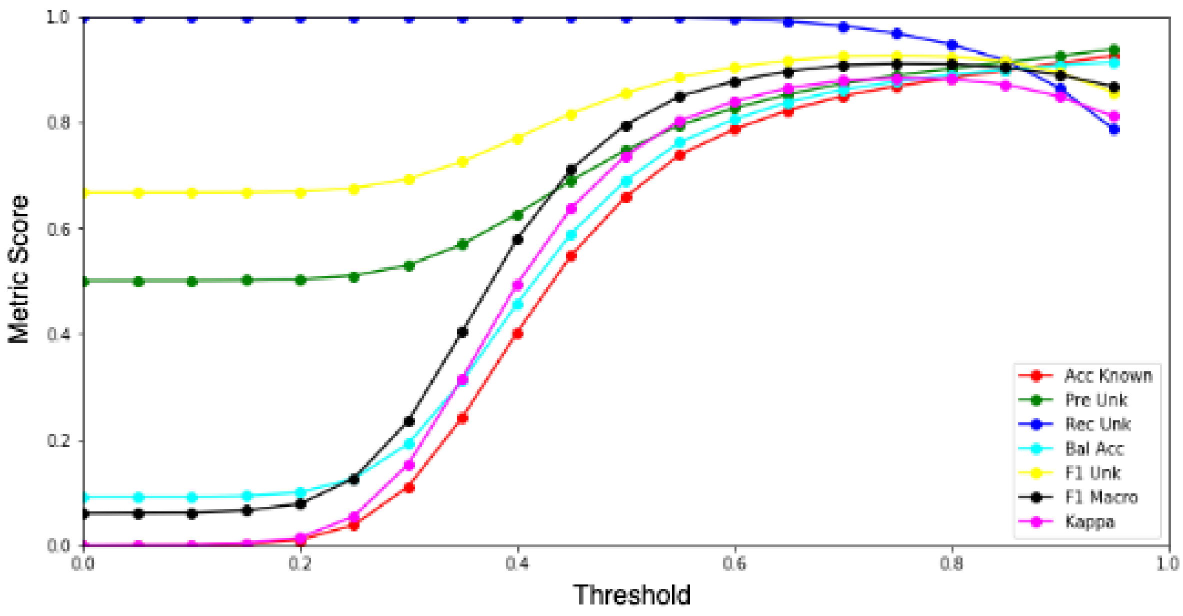
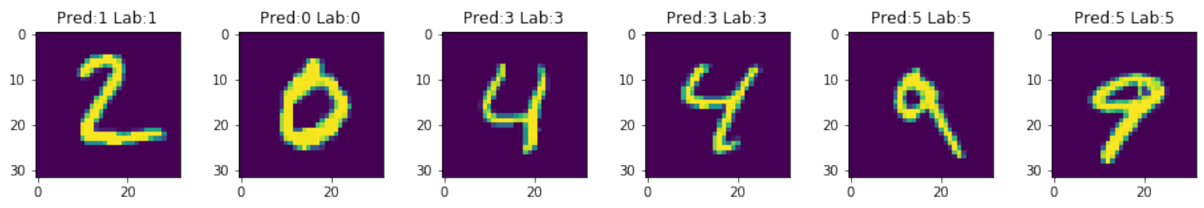


Figure 6.24: Plot with main metrics score (axis y) by threshold range (axis x) for CIFAR10 as D^{in} and LSUN (resize) as D^{out} using DenseNet with GMM8.

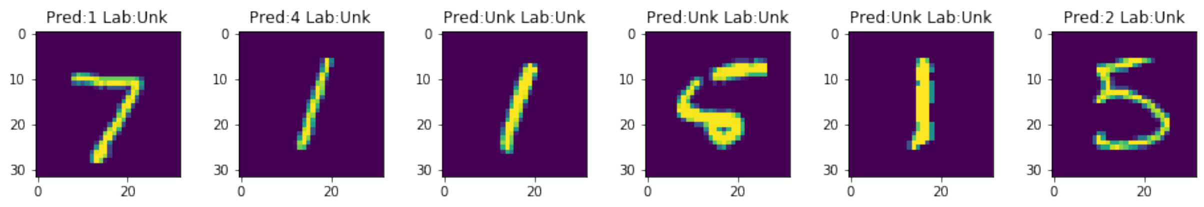
classes have the same global distribution as KKC. The GeMOS method has a lower score in this case because the backbone models are under-fitted when trained with part of these datasets. GeMOS with ResNet and GMM4 has the best result comparing the experiments but it is still lower than other baselines and has a higher standard deviation.

The normalized confusion matrix of each fold for ResNet with PCA8 is reported in the Figures 6.26, 6.27, 6.28, 6.29, and 6.30. The first fold has the classes "0", "2", "3", "4", "6", and "9" as D^{in}, the second fold has the classes "0", "1", "2", "4", "5", and "7" as D^{in}, the third fold has the classes "1", "2", "4", "5", "6", and "9" as D^{in}, the fourth fold has the classes "2", "3", "4", "6", "8", and "9" as D^{in}, and the fifth fold has the classes "1", "2", "4", "5", "7", and "9". The class selection is randomized for each fold. Examples of recognized samples from the first fold can be seen in Figure 6.25.

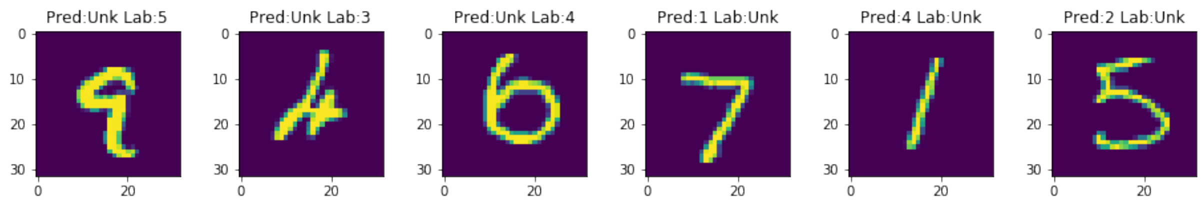
Comparing the Figures 6.26, 6.27, 6.28, 6.29, and 6.30 it is possible to notice that the recognition of unknown objects varies more than the classes between the folds, besides having the lowest TP. Class "0" varies between 95 and 98, class "1" between 96 and 97, class "2" between 92 and 97, class "3" between 90 and 94, class "4" between 93 and 95, class "5" between 94 and 95, class "6" between 92 and 95, class "7" between 95 and 96, class "8" between 95 and 96, class "9" between 91 and 95, and unknown objects between 79 and 91. Compared to the confusion matrices of cases with D^{in} different from D^{out}, it is possible to observe that using only one dataset, FP have much higher rates, where for D^{in} different from D^{out} the FP value did not exceed one while in cases with D^{in} equal to D^{out} there are several cases with FP between two and nine.



(a) Correctly predicted for the numbers “0”, “2”, “3”, “4”, “6”, and “9” of MNIST as D^{in} using ResNet with PCA8 (classes 0-5 respectively).



(b) Correctly predicted for the numbers “1”, “5”, “7”, and “8” of MNIST as D^{out} using ResNet with PCA8.



(c) Wrong predictions for MNIST as D^{in} and D^{out} using ResNet with PCA8.

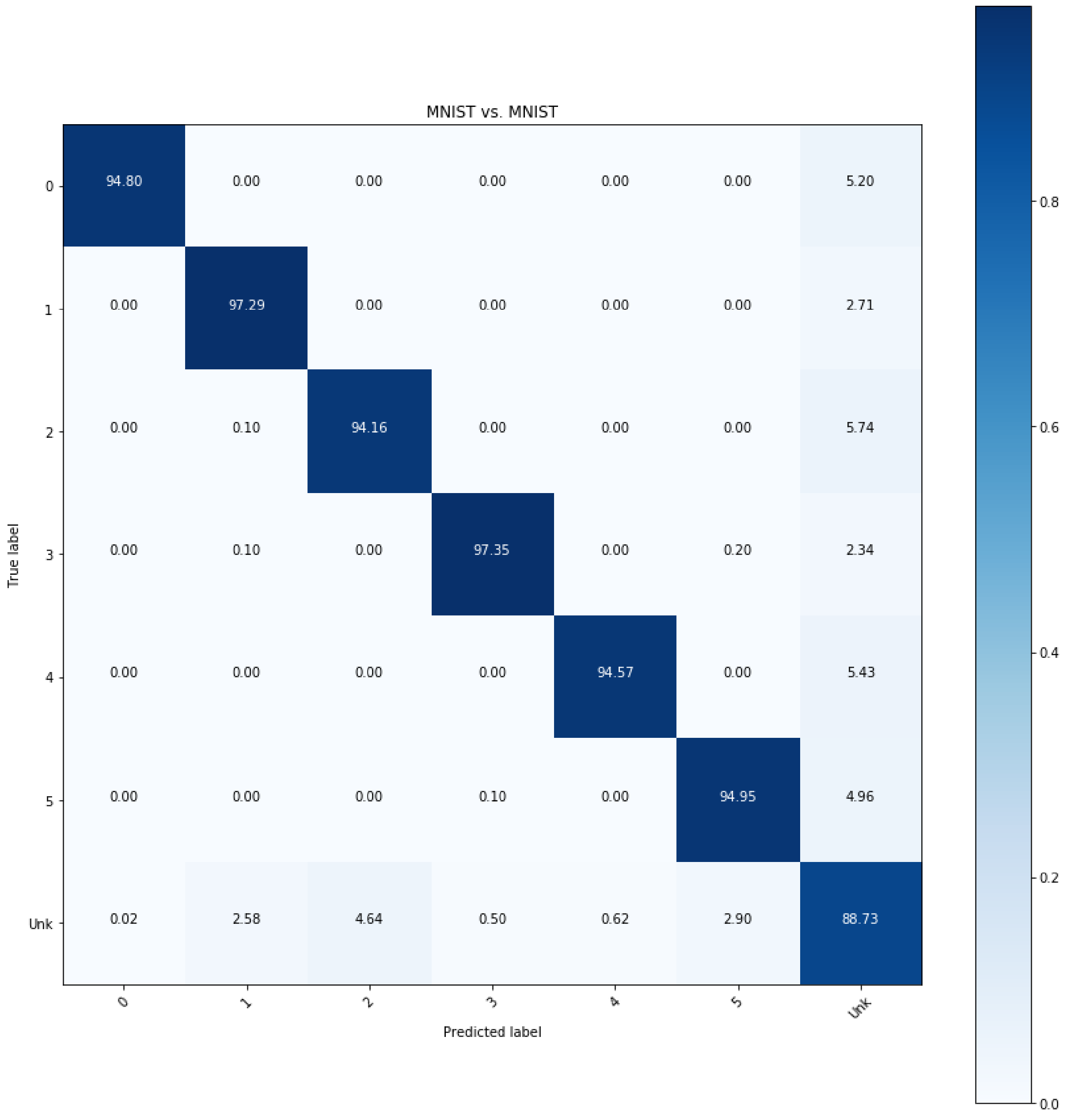
Figure 6.25: Prediction examples for the first fold of MNIST as D^{in} and D^{out} using ResNet with PCA8.

Table 6.5: AUC for OSR for CIFAR10.

Method	**CIFAR10↑**
SoftMax	0.677 ± 0.032
OpenMax	0.695 ± 0.032
G-OpenMax	0.675 ± 0.035
OSRCI	0.699 ± 0.029
C2AE	0.895 ± 0.008
CGDL	**0.903 ± 0.009**
GeMOS (DN-GMM8)	0.782 ± 0.047
GeMOS (DN-PCA8)	0.683 ± 0.063

6.3.2 CIFAR10

Table 6.5 shows the AUC for CIFAR10 as D^{in} and D^{out}. Bold text marks the best result for each metric. For the GeMOS results, the network used on each test is referenced before the generative model name. For CIFAR10, the problems found in the experiments with the MNIST were aggravated due to the greater difficulty of the dataset. Thus, the GeMOS method still does not reach the highest result due to the model's underfit, but it



True label \ Predicted label	0	1	2	3	4	5	Unk
0	94.80	0.00	0.00	0.00	0.00	0.00	5.20
1	0.00	97.29	0.00	0.00	0.00	0.00	2.71
2	0.00	0.10	94.16	0.00	0.00	0.00	5.74
3	0.00	0.10	0.00	97.35	0.00	0.20	2.34
4	0.00	0.00	0.00	0.00	94.57	0.00	5.43
5	0.00	0.00	0.00	0.10	0.00	94.95	4.96
Unk	0.02	2.58	4.64	0.50	0.62	2.90	88.73

Figure 6.26: Normalized confusion matrix for first fold with MNIST as D^{in} and D^{out} using ResNet with PCA8. The classe 0-5 are respectively the numbers: “0”, “2”, “3”, “4”, “6”, and “9”. The “Unk” group is composed by the number: “1”, “5”, “7”, and “8”.

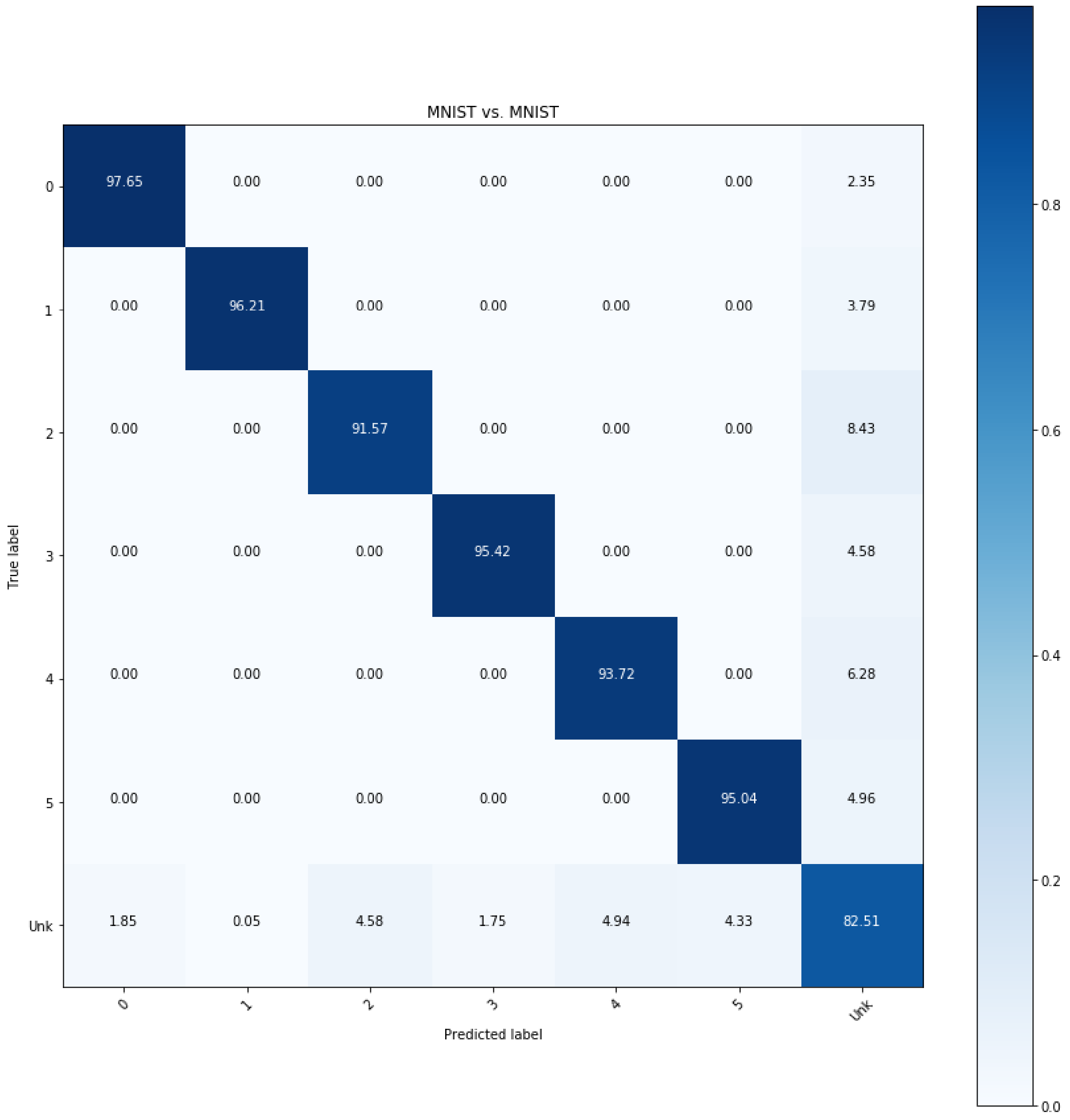


Figure 6.27: Normalized confusion matrix for second fold with MNIST as D^{in} and D^{out} using ResNet with PCA8. The classe 0-5 are respectively the numbers: “0”, “1”, “2”, “4”, “5”, and “7”. The “Unk” group is composed by the number: “3”, “6”, “8”, and “9”.

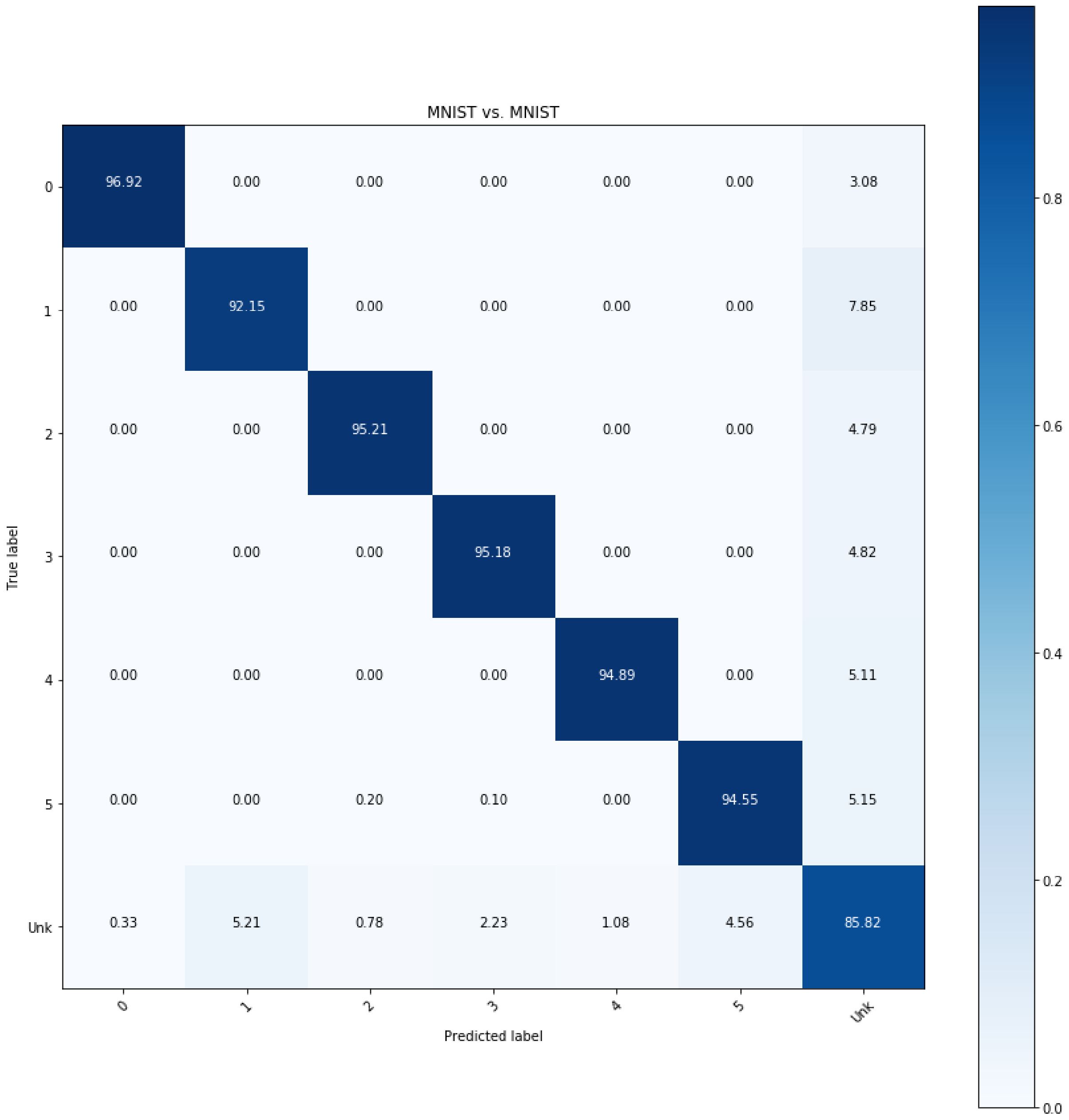


Figure 6.28: Normalized confusion matrix for third fold with MNIST as D^{in} and D^{out} using ResNet with PCA8. The classe 0-5 are respectively the numbers: “1”, “2”, “4”, “5”, “6”, and “9”. The “Unk” group is composed by the number: “0”, “3”, “7”, and “8”.

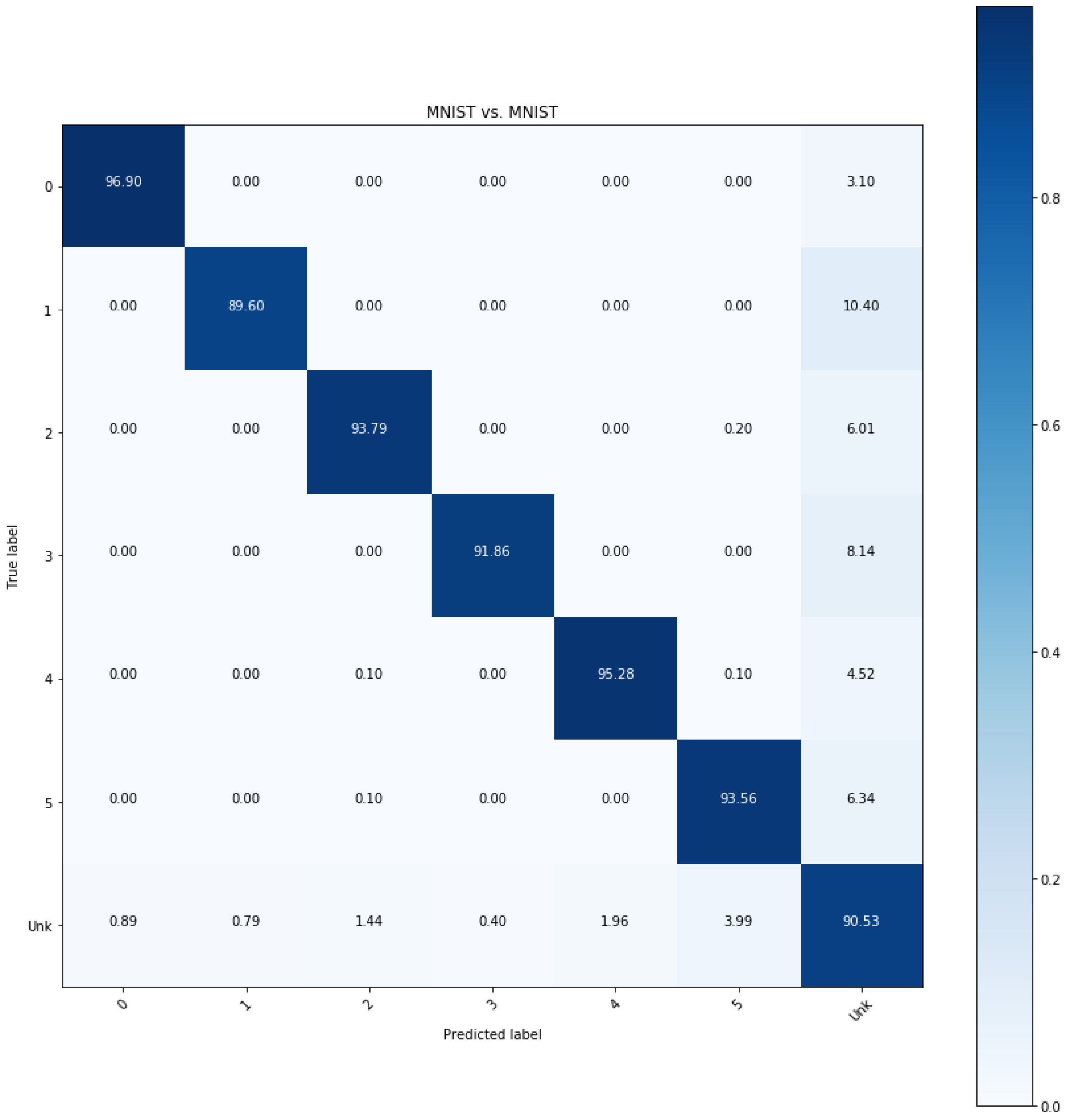


Figure 6.29: Normalized confusion matrix for fourth fold with MNIST as D^{in} and D^{out} using ResNet with PCA8. The classe 0-5 are respectively the numbers: “2”, “3”, “4”, “6”, “8”, and “9”. The “Unk” group is composed by the number: “0”, “1”, “5”, and “9”.

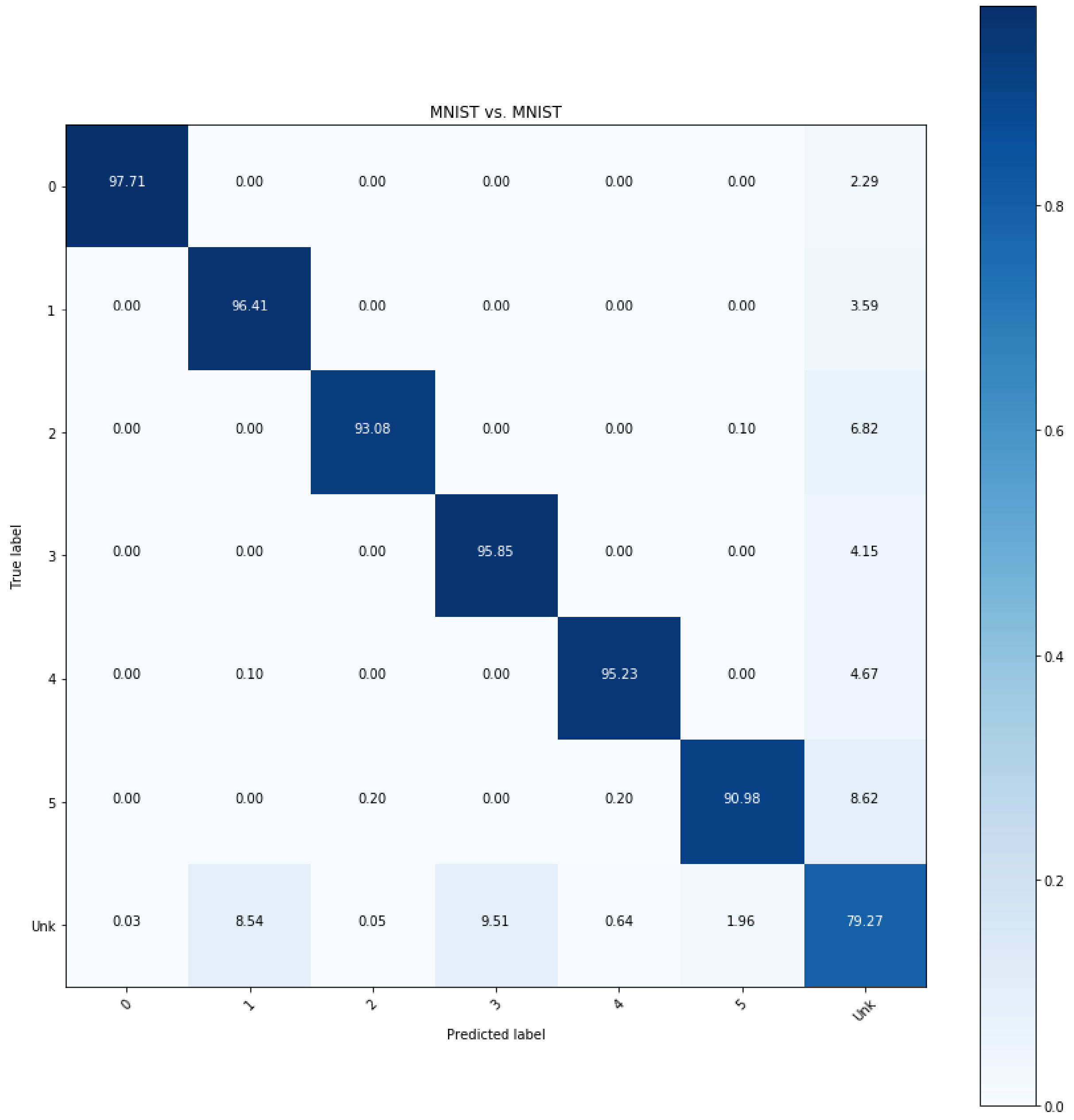
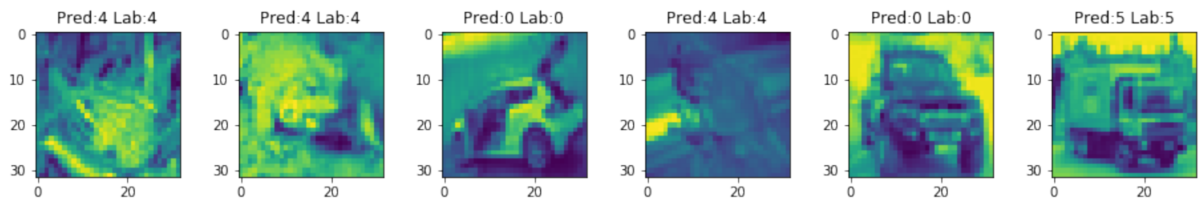
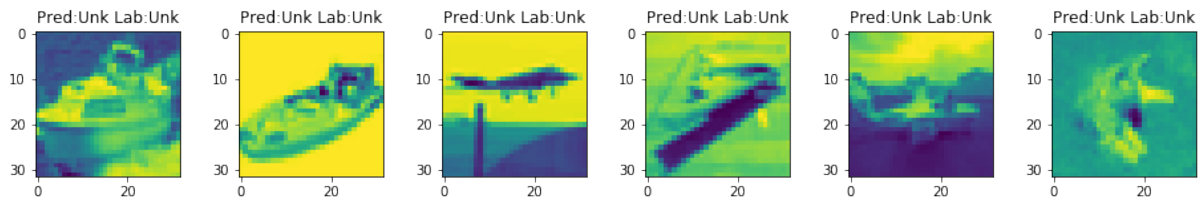


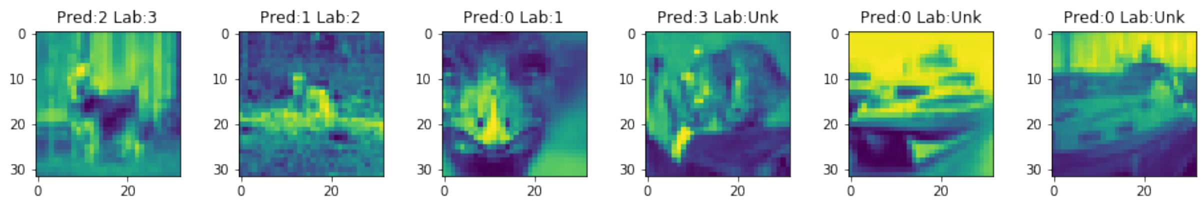
Figure 6.30: Normalized confusion matrix for fifth fold with MNIST as D^{in} and D^{out} using ResNet with PCA8. The classe 0-5 are respectively the numbers: “1”, “2”, “4”, “5”, “7”, and “9”. The “Unk” group is composed by the number: “0”, “3”, “6”, and “8”.



(a) Correctly predicted for the classes “1 - automobile”, “2 - bird”, “4 - deer”, “5 - dog”, “6 - frog”, and “9 - truck” of CIFAR10 as D^{in} using DenseNet with GMM8 (classes 0-5 respectively).



(b) Correctly predicted for the classes “0 - airplane”, “3 - cat”, “7 - horse”, and “8 - ship” of CIFAR10 as D^{out} using DenseNet with GMM8.



(c) Wrong predictions for CIFAR10 as D^{in} and D^{out} using DenseNet with GMM8.

Figure 6.31: Prediction examples for the third fold of CIFAR10 as D^{in} and D^{out} using DenseNet with GMM8.

still manages to surpass most baselines. The best result was obtained by CGDL followed by C2AE and GeMOS with DenseNet and GMM8. The standard deviation of GeMOS is higher than the baselines but the results still consistently better than most of them.

The normalized confusion matrix of each fold for DenseNet with GMM8 is reported in the Figures 6.32, 6.33, 6.34, 6.35, and 6.36. The first fold has the classes “0 - airplane”, “2 - bird”, “3 - cat”, “4 - deer”, “6 - frog”, and “9 - truck” as D^{in}, the second fold has the classes “0 - airplane”, “1 - automobile”, “2 - bird”, “4 - deer”, “5 - dog”, and “7 - horse” as D^{in}, the third fold has the classes “1 - automobile”, “2 - bird”, “4 - deer”, “5 - dog”, “6 - frog”, and “9 - truck” as D^{in}, the fourth fold has the classes “2 - bird”, “3 - cat”, “4 - deer”, “6 - frog”, “8 - ship”, and “9 - truck” as D^{in}, and the fifth fold has the classes “1 - automobile”, “2 - bird”, “4 - deer”, “5 - dog”, “7 - horse”, and “9 - truck”. The class selection is randomized for each fold. Examples of recognized samples from the third fold can be seen in Figure 6.31.

Comparing the Figures 6.32, 6.33, 6.34, 6.35, and 6.36 it is possible to observe that the recognition of unknown objects has a much smaller TP than all the other cases besides having a much bigger variation. As in the MNIST experiments, the FP is much larger than in the experiments with D^{in} different from D^{out}. FN also presents higher values than previous experiments. Even with the increase in FN, the classification pattern remains

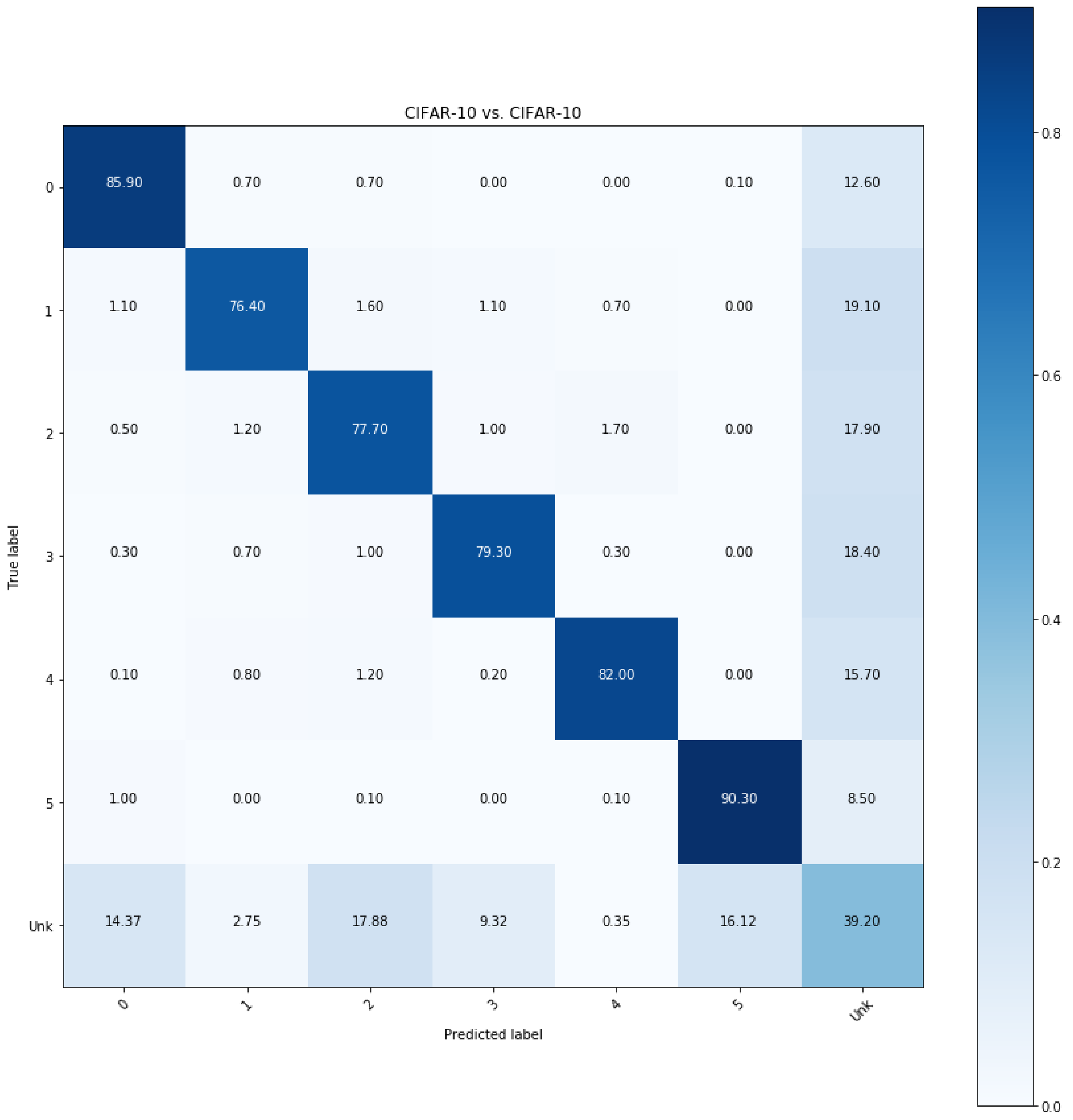


Figure 6.32: Normalized confusion matrix for first fold with CIFAR10 as D^{in} and D^{out} using DenseNet with GMM8. The classe 0-5 are respectively the numbers: “0 - airplane”, “2 - bird”, “3 - cat”, “4 - deer”, “6 - frog”, and “9 - truck”. The “Unk” group is composed by the number: “1 - automobile”, “5 - dog”, “7 - horse”, and “8 - ship”.

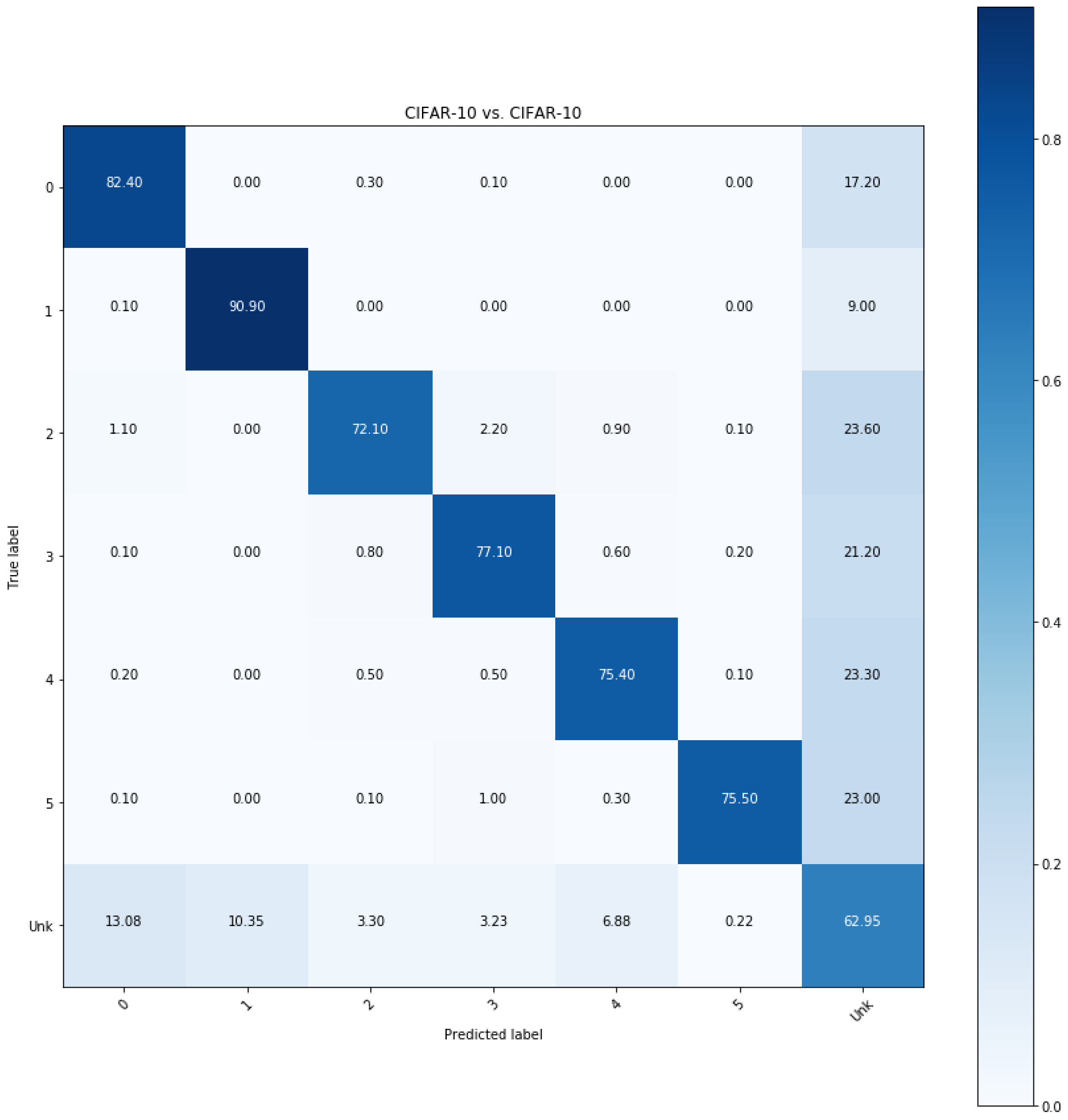


Figure 6.33: Normalized confusion matrix for second fold with CIFAR10 as D^{in} and D^{out} using DenseNet with GMM8. The classe 0-5 are respectively the numbers: “0 - airplane”, “1 - automobile”, “2 - bird”, “4 - deer”, “5 - dog”, and “7 - horse”. The “Unk” group is composed by the number: “3 - cat”, “6 - frog”, “8 - ship”, and “9 - truck”.

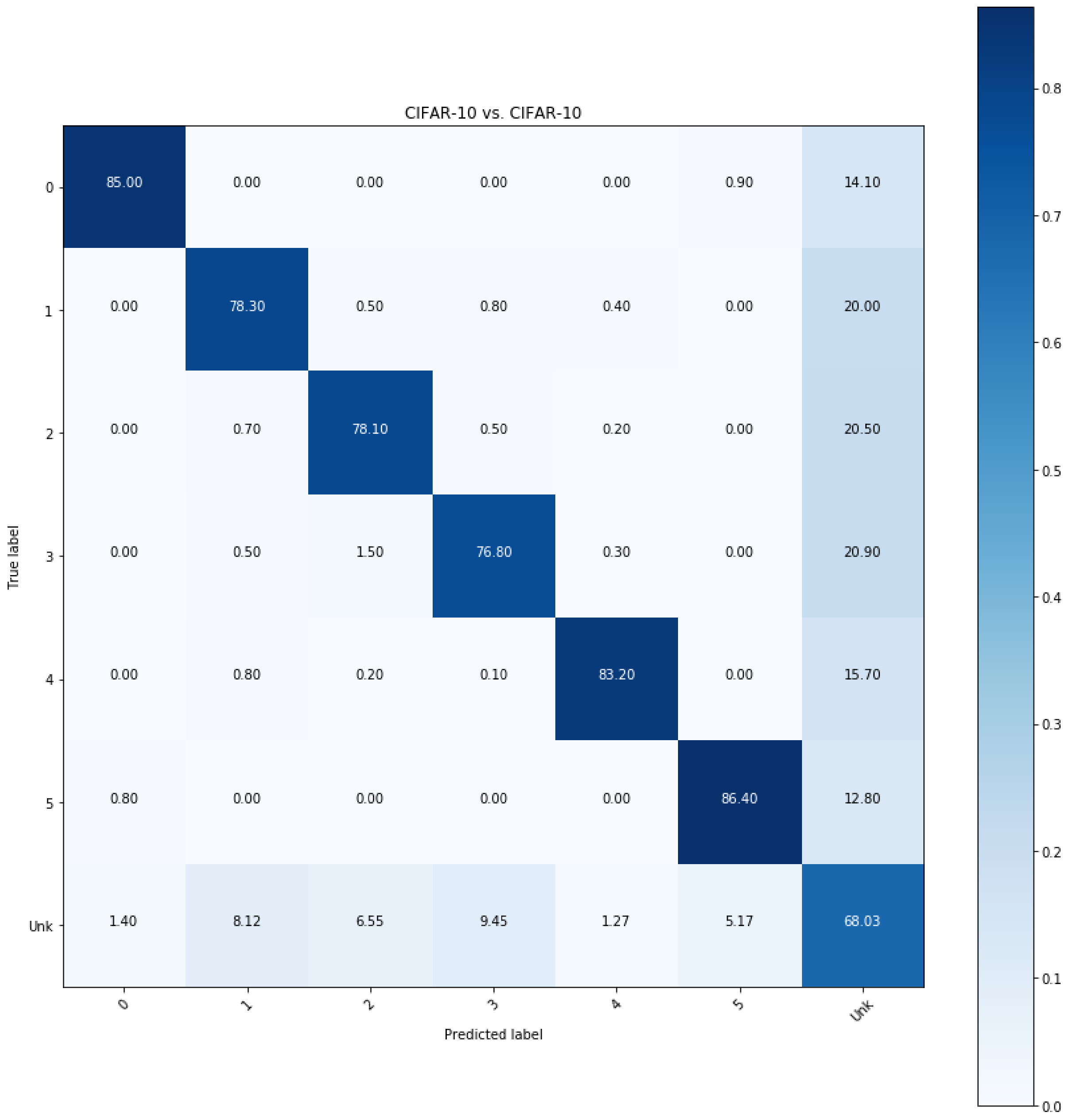


Figure 6.34: Normalized confusion matrix for third fold with CIFAR10 as D^{in} and D^{out} using DenseNet with GMM8. The classe 0-5 are respectively the numbers: “1 - automobile”, “2 - bird”, “4 - deer”, “5 - dog”, “6 - frog”, and “9 - truck”. The “Unk” group is composed by the number: “0 - airplane”, “3 - cat”, “7 - horse”, and “8 - ship”.

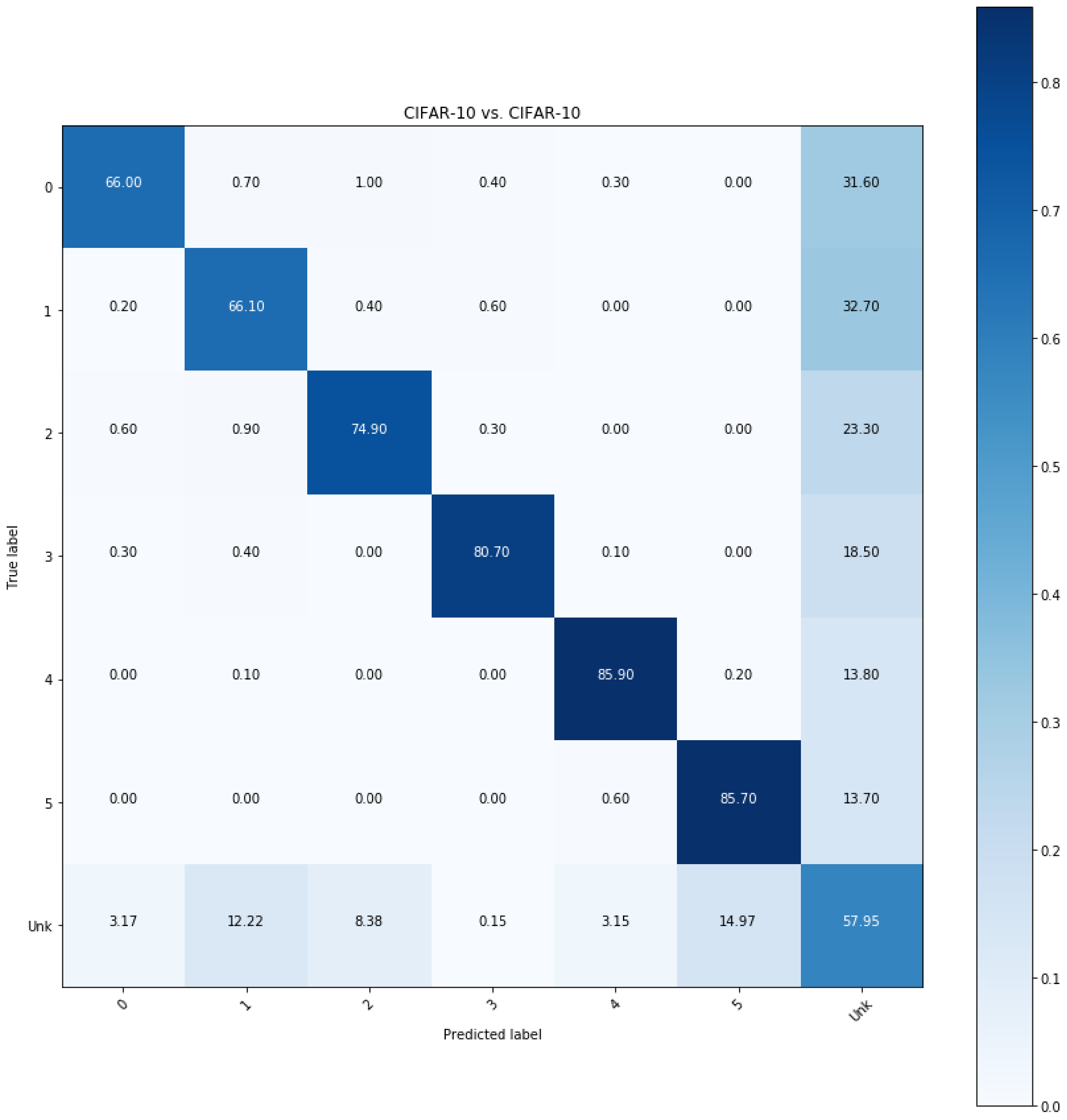


Figure 6.35: Normalized confusion matrix for fourth fold with CIFAR10 as D^{in} and D^{out} using DenseNet with GMM8. The classe 0-5 are respectively the numbers: “2 - bird”, “3 - cat”, “4 - deer”, “6 - frog”, “8 - ship”, and “9 - truck”. The “Unk” group is composed by the number: “0 - airplane”, “1 - automobile”, “5 - dog”, and “7 - horse”.

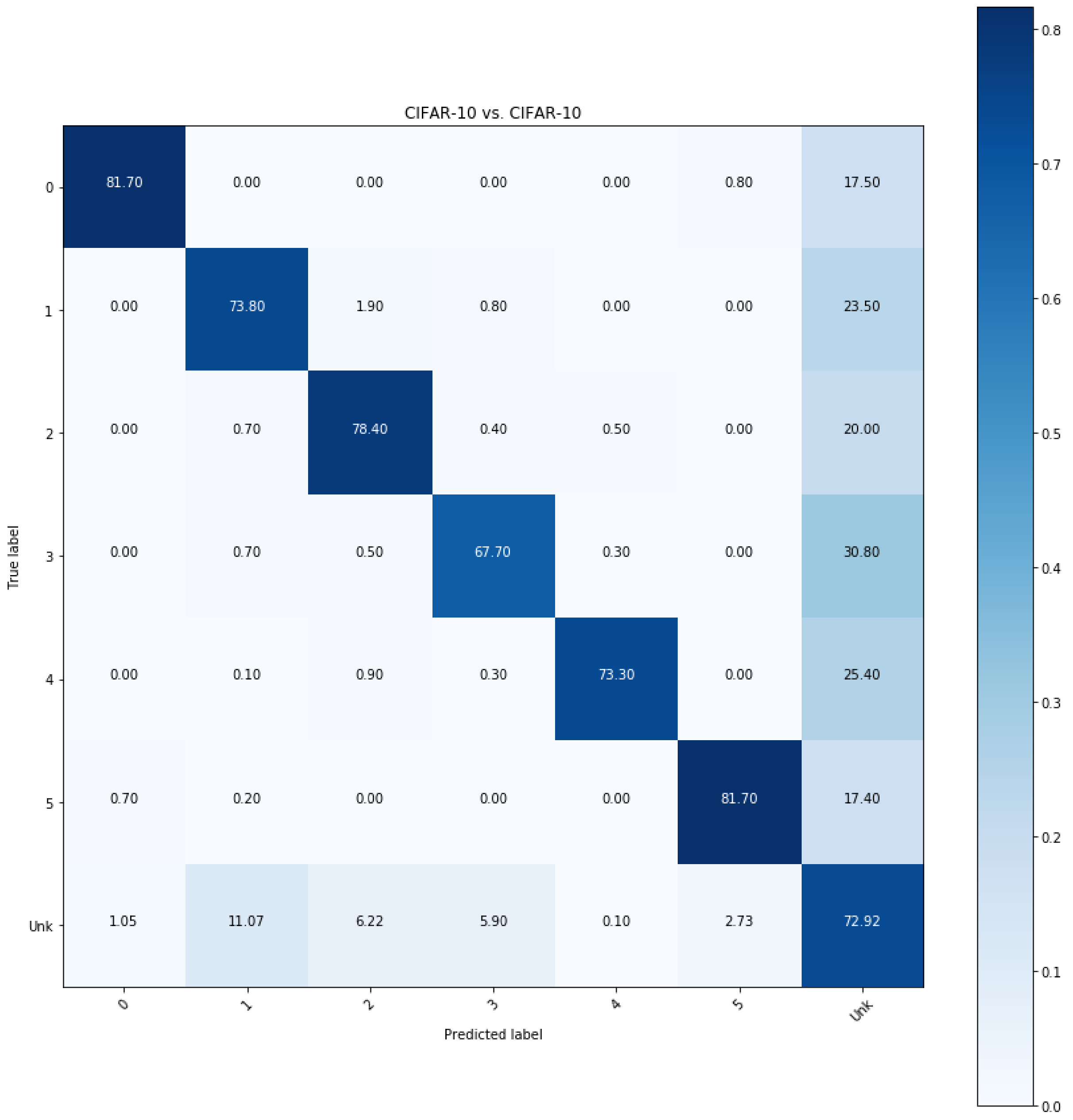


Figure 6.36: Normalized confusion matrix for fifth fold with CIFAR10 as D^{in} and D^{out} using DenseNet with GMM8. The classe 0-5 are respectively the numbers: “1 - automobile”, “2 - bird”, “4 - deer”, “5 - dog”, “7 - horse”, and “9 - truck”. The “Unk” group is composed by the number: “0 - airplane”, “3 - cat”, “6 - frog”, and “8 - ship”.

the same, so that class “3 - cat” still has the lowest TP.

Chapter 7

Conclusion and Future Work

In this work GeMOS was proposed, a novel method for Open Set Recognition (OSR). OSR is a challenging task because it needs to differentiate outliers from unknown classes while classifying a group of known classes without considering a fixed number of classes and without knowledge about unknown objects. Attention in this area is growing because, in real-world cases, it is often not possible to access information about all possible classes.

GeMOS uses generative models to identify OOD samples based on feature vectors extracted from pre-trained closed set CNNs. The consistency of the proposed method was assessed by using different backbones and generative models tested in several datasets used as D^{in} and D^{out}.

The ablation study showed that GMMs perform consistently better than unimodal shallow methods in OSR. The results were compared to state-of-the-art OSR methods mimicking their exact test protocol, with GeMOS consistently achieving great results. GeMOS' performance can be attributed to the fact that the intermediary feature vectors from the CNN are used to feed the generative model instead of looking only at the input [41, 31] or output spaces [2].

Experimental evaluation with different backbones shows that the model architecture does not influence the final result as long as the model is well trained for a problem's domain. Comparing the generative models it is possible to see that the GMM model is superior to the others because of its multimodality capabilities.

Experiments with a different dataset as D^{in} and D^{out} showed good results for MNIST as D^{in} and also in more complex scenarios such as using CIFAR10 as D^{in}. For MNIST as D^{in}, EMNIST scores as D^{out} were lower than OMNIGLOT and KMNIST. Using CIFAR10 as D^{in}, the results of CIFAR100 as D^{out} were lower than Tiny ImageNet (crop and resize) and LSUN (crop and resize). Among the classes, considering the largest F1-macro, the accuracy of the unknown classes tends to be higher than the other classes, except in the case of CIFAR10 as D^{in} and CIFAR100 as D^{out}. Despite this, the accuracy between the known and unknown classes varies with the threshold and can be manipulated based on the problem.

In the tests with the same dataset as D^{in} and D^{out} it is harder to recognize OOD because these classes have the same global distribution as the other objects. In these cases, the GeMOS method produces lower scores because the backbone models trained with part of the dataset classes are under-fitted.

The main drawback of GeMOS observed in our exploratory experiments was the high reliance on the performance of the closed set model, where a dip of around 5% in KKC accuracy severely compromised the OSR detection capabilities of the algorithm. However, given a state-of-the-art pre-trained CNN in the KKCs, GeMOS can yield equivalent results or even outperform more costly alternatives. Additionally, GeMOS has the advantage of being able to be coupled to any pre-trained CNN, while most reconstruction based methods (e.g. C2AE and CGDL) require expensive training of additional neural network architectures. Coupled with other pre-trained CNNs GeMOS has a low-cost to train the method since it is not necessary to have a GPU to train the generative models.

7.1 Future Work

For future work, initially, a detailed test of the impact of the accuracy of pre-trained networks in the studied cases is proposed, carrying out experiments with underfitted networks at different points. After that, the method could be applied to other domains like geoprocessing images to expand the tests beyond the standard protocol and could be applied to real world problems and GeMOS will be adapted to apply it in other vision tasks like segmentation problems.

Other ideas for future work are: apply the method in semi-supervised models, inserting a few OOD samples in the training process to help the detection of the method; expand the method to Open World problems adding incremental learning for OOD classes that become known over time; and use GAN or VAE models to generate synthetic samples and insert those samples into the training process of our pipeline.

Bibliography

[1] Ron Artstein and Massimo Poesio. Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596, 2008.

[2] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572, 2016.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[5] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018.

[6] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[7] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[8] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[9] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[10] Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

[11] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[12] ZongYuan Ge, Sergey Demyanov, Zetao Chen, and Rahil Garnavi. Generative openmax for multi-class open set classification. *arXiv preprint arXiv:1707.07418*, 2017.

[13] Chuanxing Geng and Songcan Chen. Collective decision for open set recognition. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[14] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[15] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[16] Robert M Haralick, Karthikeyan Shanmugam, and Its' Hak Dinstein. Textural features for image classification. *IEEE Transactions on systems, man, and cybernetics*, (6):610–621, 1973.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[19] Jaeyeon Jang and Chang Ouk Kim. Teacher-explorer-student learning: A novel learning method for open set recognition. *arXiv preprint arXiv:2103.12871*, 2021.

[20] Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[24] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[25] Fahad Lateef and Yassine Ruichek. Survey on semantic segmentation using deep learning techniques. *Neurocomputing*, 338:321–348, 2019.

[26] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[27] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[28] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[29] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[30] Lawrence Neal, Matthew Olson, Xiaoli Fern, Weng-Keen Wong, and Fuxin Li. Open set learning with counterfactual images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 613–628, 2018.

[31] Poojan Oza and Vishal M Patel. C2ae: Class conditioned auto-encoder for open-set recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2307–2316, 2019.

[32] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[33] Walter J Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E Boult. Toward open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(7):1757–1772, 2012.

[34] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[36] Xin Sun, Zhenning Yang, Chi Zhang, Keck-Voon Ling, and Guohao Peng. Conditional gaussian distribution learning for open set recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13480–13489, 2020.

[37] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[38] Michael E Tipping and Christopher M Bishop. Mixtures of probabilistic principal component analyzers. *Neural computation*, 11(2):443–482, 1999.

[39] Marcos Vendramini, Hugo Oliveira, Alexei Machado, and Jefersson A. dos Santos. Opening deep neural networks with generative models. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 1314–1318, 2021.

[40] Jiayu Wu, Qixiang Zhang, and Guoxi Xu. Tiny imagenet challenge. *Technical Report*, 2017.

[41] Ryota Yoshihashi, Wen Shao, Rei Kawakami, Shaodi You, Makoto Iida, and Takeshi Naemura. Classification-reconstruction learning for open-set recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4016–4025, 2019.

[42] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[43] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.