TU WIEN Informatics

Open Set Klassifizierung von Nummerntafeltypen in Bildern mittels Deep Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Sebastian Haushofer, BSc
Matrikelnummer 01525970

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Walter Kropatsch, O.Univ.Prof. Dipl.-Ing. Dr.techn.
Mitwirkung: Jiří Hladůvka, Univ.Ass. Dr.techn.

Wien, 31. Jänner 2021

Sebastian Haushofer	Walter Kropatsch

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Open Set Classification in the Domain of License Plate Type Images using Deep Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Sebastian Haushofer, BSc
Registration Number 01525970

to the Faculty of Informatics

at the TU Wien

Advisor: Walter Kropatsch, O.Univ.Prof. Dipl.-Ing. Dr.techn.
Assistance: Jiří Hladůvka, Univ.Ass. Dr.techn.

Vienna, 31st January, 2021

Sebastian Haushofer	Walter Kropatsch

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Sebastian Haushofer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 31. Jänner 2021

Sebastian Haushofer

Acknowledgements

Above all, I want to thank my family and friends who always support me and give me a lot of energy and motivation. Besides, I want to express my gratitude to my supervisors Jiří Hladůvka and Walter Kropatsch for answering my technical questions and providing valuable feedback on my work. Last but not least, I am grateful to everybody around the world who inspires me by coming up with new and original ideas. This includes artists and scientists of all kind.

Kurzfassung

Bei der Erkennung des Nummerntafeltyps handelt es sich um ein Klassifizierungsproblem auf einer offenen Menge von Klassen. Das Ziel ist zwischen verschiedenen Hintergründen von Nummerntafeln, welche an Fahrzeugen angebracht sind, zu unterscheiden. Wenn der Nummerntafeltyp bekannt ist, lässt sich damit auf den Herkunftsstaat, in dem das Kennzeichen registriert ist, rückschließen.
Moderne Techniken, die Klassifikation auf einer offenen Menge lösen, nutzen Deep Learning, um einen eingebetteten Raum als Zwischenprodukt zu trainieren. In diesem wird anschließend die Klassifikation durchgeführt. Der eingebettete Raum ist im Wesentlichen das Ergebnis einer nichtlinearen Merkmalsreduktion. In ihm ist jede Klasse durch einen kompakten Cluster repräsentiert, der von den anderen Klassen trennbar ist. Der eingebettete Raum wird nur mit einer Teilmenge aller bekannten Klassen trainiert. Die übrigen bekannten Klassen werden integriert, ohne das Neuronale Netzwerk neu zu trainieren. Instanzen von Klassen, die für das gesamte System unbekannt sind, werden mittels Thresholding identifiziert.
Ein wesentlicher Beitrag dieser Arbeit ist eine Analyse der Klassenverteilungen im eingebetteten Raum, welche von bisherigen Arbeiten weitgehend außer Acht gelassen wurde. Für diesen Zweck werden einige neue Bewertungskriterien vorgestellt, welche es ermöglichen die Qualität verschiedener gelernter Räume zu vergleichen. Die identifizierenden Zeichen einer Nummerntafel sind eine Herausforderung für die maschinelle Erkennung des Nummerntafeltyps, da Neuronale Netzwerke dazu neigen sie als Merkmale zu lernen. Um dieses Problem zu lösen, werden die identifizierenden Zeichen während des Trainings per Zufall ausmaskiert. Außerdem wird mit Hilfe sogenannter Saliency Maps untersucht, auf welche Merkmale das trainierte Neuronale Netzwerk besonders sensibel reagiert.
Ein bestimmter Klassifikator, welcher häufig in eingebetteten Räumen verwendet wird, nimmt Verteilungen mit homogener Varianz als Vorbedingung an. Die Analyse der Verteilungen im eingebetteten Raum zeigt, dass diese Annahme für den verwendeten Nummerntafeltyp Datensatz nicht erfüllt ist. Die Experimente zeigen, dass die Klassenverteilungen auch einen signifikanten Unterschied zu multivariaten Normalverteilungen, welche die Form von Hyperellipsen besitzen, aufweisen. Die Saliency Maps visualisieren, dass das Neuronale Netzwerk für den Großteil der Nummerntafeltypen vernünftige Merkmale lernt. Allerdings werden für wenige Klassen primitive Merkmale ausgewählt, welche von einem Menschen nicht in Betracht gezogen werden würden.

Abstract

License plate type recognition is a classification problem on an open set of classes. Its purpose is to distinguish between different backgrounds of license plates mounted on vehicles. Knowing the license plate type allows to identify the country in which the respective license plate is registered.
Modern approaches which solve open set classification problems make use of deep learning to train an intermediate embedding space in which a subsequent classification is performed. This embedding space essentially represents the result of a non-linear feature reduction. Within it, each class is represented by a compact cluster which is separable from the other classes. The embedding space is only trained with a subset of the known classes. The remaining known classes are incorporated without retraining the network. Instances of classes which are unknown to the system are identified using thresholding.
The main contribution of this work is the analysis of the class distributions within the embedding space, which has been largely neglected by previous work. For this purpose, new benchmarks which allow a qualitative comparison between different learned embedding spaces are introduced. The identifying characters of a license plate are challenging for license plate type recognition because the model tends towards learning specific characters as features. To overcome this problem, randomized masking of the characters during training is proposed. Furthermore, investigations about which features the trained network is sensitive to are carried out using gradient-based saliency map techniques.
A commonly used embedding space classifier assumes class distributions of homogeneous variance as a prior under the open set restriction. The analysis of the class distributions in the embedding space shows that this prior is not met for the used license plate type dataset. The experiments reveal that the class distributions even show a significant difference from multivariate Gaussians, which are capable of modeling more complex distributions in the shape of hyper-ellipsoids. The computed saliency maps visualize that the model learns reasonable features for most of the license plate types. However, for a few of them, primitive features which a human would not consider for classification are exploited.

Contents

CHAPTER 1

Introduction

1.1 Problem Statement

License Plate Type Recognition (LPTR) is a subtask for solving the broader problem of License Plate Recognition (LPR). The goal of LPR is to read the characters written on a License Plate (LP) of a vehicle, as well as the country or state of origin for which the LP is registered. These two properties are a unique identifier of a vehicle in most settings. In many countries such as the USA, an LP does not only contain characters but also a background image which is shared amongst many different LPs of a certain state. This background image is also known as the License Plate Type (LPT) and may contain additional fixed characters, coats of arms, symbols, or textures. In general, the LPT is defined as the parts of the LP which remain once the characters identifying the LP are masked out. Some example LPs and their corresponding LPTs from the US state of Virginia are shown in Figure 1.1.
The aim of LPTR is to identify the LPT based on a given image of an LP, which fundamentally makes it a classification problem. This task is challenging for two reasons. Firstly, there are hundreds of different LPTs and any system solving the LPTR task should be able to classify new LPTs, which have not been considered when initially developing the system. This should be achieved with as little adaptation of the system, cost and time effort as possible. Another goal of an LPTR system is to recognize if a certain LPT is unknown to itself, as opposed to assigning an incorrect class label to it. Because of these two properties, LPTR is a classification problem on an open set of classes. The second aspect which makes LPTR challenging is that captured images of LPTs have very high variance along many different dimensions in real world applications. These include illumination, contrast, reflections, occlusions, motion blur, dirt, image resolution and other noise in general. The images shown in Figure 1.1 are idealized. Real world data typically looks like the images shown in Figure 1.2. In fact, these images are part of the dataset which has been used for this work.

(a) Original LPs containing the identifying characters.

(b) LPTs of the LPs shown in (a), obtained by masking out the relevant characters.

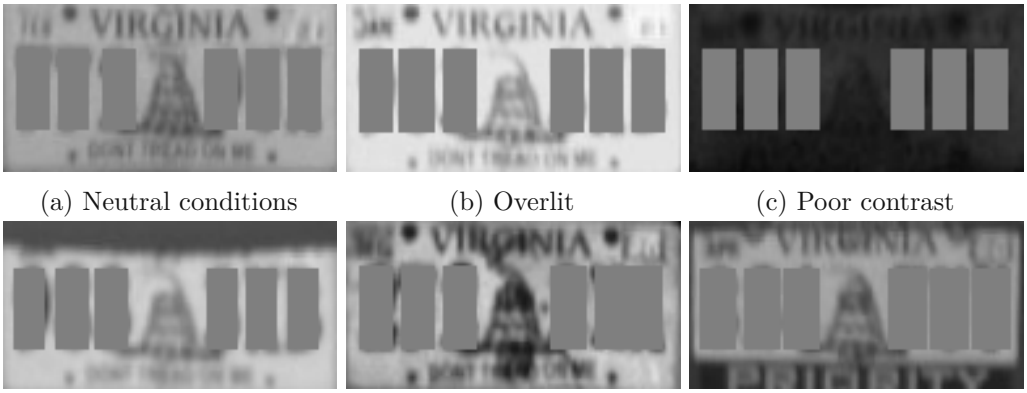
Figure 1.1: Some example LPs and their corresponding LPTs from the US state of Virginia. Images by Kustermann [Kus20].

1.2 Motivation

LPR systems are heavily used for tolling and speed control stations. To reduce costs of human evaluation, people have put a lot of effort and money into developing Optical Character Recognition (OCR) systems to solve the LPR task automatically over the past decades [AAP+08]. In many cases LPTR is very promising for further improving the reliability and performance of such a system.

First of all, the LPT implies the state of origin of an LP. This is due to the fact that there is a many-to-one correspondence between LPTs and states. Using a database which stores the LPTs for each state of interest, it is possible to identify the state of origin, given the LPT.

As the LPT depends on the number of identifying characters written on it and their relative positions, knowing the LPT makes it possible to perform sanity checks on the



(a) Neutral conditions (b) Overlit (c) Poor contrast

(d) Blurred (e) Dirty (f) Occluded

Figure 1.2: Different images of the same LPT from the real world dataset used in this work. The LPT shown in these images is the same as the one at the bottom left in Figure 1.1. The images show some of the variances which occur in the dataset.

read LP characters and vice versa. Notice that this property is only useful if the LPTR system is capable of identifying the LPT without requiring any special handling (e.g. masking) of the characters in advance. Otherwise, this leads to a chicken-egg situation, as character validation depends on the LPT and classifying the LPT depends on the characters.
Lastly, LPTs may contain additional information which would not be accessible by traditional OCR means otherwise. For example, in the USA there are certain groups of people who receive special rates for tolling stations. These groups include veterans or persons with disabilities. A group is indicated by special symbols or signs which are part of the LPT. Just as with the states, there is a many-to-one correspondence between LPTs and a certain group.

1.3 Who this Thesis is for

This thesis is written for anyone who is interested in computer vision, pattern recognition, deep learning and/or machine learning. It might be of particular interest to people who work on classification problems on open sets.
The thesis is written in such a way that a graduate student of computer science, who has taken an introduction course to machine learning, should be able to understand it. Thus, there is quite a stack of prior knowledge in different fields the reader should be familiar with. However, in all the different domains in which prior knowledge is assumed, the fundamental principles should be enough to follow along. More advanced topics are explained within the thesis.
Regarding mathematics, knowledge about basic concepts of linear algebra, multivariable calculus and statistics is assumed. Particularly important for this thesis are transformations between different coordinate systems, eigenvalues and eigenvectors, partial

derivatives, Probability Density Function (PDF)s and Cumulative Distribution Function (CDF)s, and common probability distributions. Furthermore, as the title of the thesis suggests, deep learning and machine learning play a central role in general. Basic knowledge about artificial neural networks, especially the Convolutional Neural Network (CNN) architecture, is assumed. This includes the common layer types (convolutional, pooling, fully connected, dropout, batch normalization), activation functions, loss functions, regularization, optimizers, gradient descent and back-propagation. Regarding pattern recognition and classical machine learning, Principal Component Analysis (PCA), multivariate Gaussians and various common metrics for evaluating classifiers are especially relevant. All evaluation metrics which have been used in this work are briefly explained in Section 3.3.
For all of these fields, any introductory textbook about the respective topic that is commonly used for teaching at universities should cover the necessary aspects. However, we offer some recommendations for books which we consider particularly well fit. For a great modern introduction to deep learning, the reader may have a look at the "Deep learning" book by Goodfellow et al. [GBC16]. As a reference book for pattern recognition and machine learning in general, we recommend Bishop's "Pattern recognition and machine learning" [Bis06]. Both books also cover a brief refresher of most of the mathematical foundations mentioned above. Two more great books worth mentioning are "Introduction to linear algebra" by Strang [Str16] and "Vector calculus, linear algebra, and differential forms: a unified approach" by Hubbard and Hubbard [HH15].

1.4 Methodological Approach

First of all, it is justified why deep learning is well suited for solving LPTR and what benefits it has over traditional approaches. Then, the basic principle behind using deep learning for solving the problem is explained. At the end of this section, an illustration of how such a deep learning based system can be used in practice is given.

1.4.1 The Difficulties with Traditional Approaches

Although all images of the same LPT should look the same once the respective LP's identifying characters are masked out, this rather theoretical idealization does not hold for real world data. This is due to different kinds of variances of the captured images, as mentioned in Section 1.1. In an idealized world, all images of LPTs are captured directly from the front with the same camera, without any perspective distortion and at neutral illumination conditions without any specular highlights. Under these circumstances, a pixel-wise comparison between two different images of the same LPT would show no difference at all.
A common metric for computing pixel-wise similarity between any pair of two images is the Normalized Cross Correlation (NCC). Due to the given variances illustrated in Figure 1.2, this naive approach alone is not capable of solving the LPTR task reliably, given more than just a handful of different LPTs. For example, tiny alignment differences obtained

by shifting one of the images slightly along the x or y-axis would already cause faulty results. A problem like this can be overcome with traditional computer vision approaches for aligning the images. Many other problems caused by variances such as noise may also be solved by hard-coded computer vision solutions designed by a human expert. However, there will most likely always be edge cases which have not been considered during the initial development phase of the system, which are potential error sources.
A particularly hard problem in general are occlusions of any kind. All methods which are based on simple pixel-wise comparisons are not capable of overcoming a problem like this. This is because such algorithms have no understanding of what characterizes the LPT shown in a given image. For a human being, an LPT remains uniquely identifiable as long as enough of its characteristics distinguishing it from other LPTs are not occluded. Being able to solve occlusions therefore requires deeper understanding of the underlying data than just raw pixels. Researchers have developed many algorithms to compute handcrafted features from images. The main idea is that certain combinations of computed features uniquely identify a specific class of interest. A major disadvantage of handcrafted features is that different features are useful for different datasets. A human expert must find the right features which are capable of solving the underlying problem. This is both time consuming and expensive.

1.4.2 Why Deep Learning?

Deep learning is very promising for a problem like LPTR because it extracts relevant features identifying a certain object by itself during training. It does not require any sort of expert human knowledge to define meaningful features for the problem at stake. Deep learning only needs a precise definition of the task to be solved in form of a differentiable optimization problem and a lot of training data as input. However, it must be noted that collecting and selecting the right training data is often very challenging, as unwanted biases can easily occur within the data. Notice that the previously mentioned problem of edge cases which have not been considered during development does not disappear. Deep learning will only learn to solve a task for the data it is trained with. Although it can learn to generalize to similar data, its generalization capabilities for edge cases are quite limited. Thus, the network will most likely have troubles interpreting edge cases correctly if they are not part of the training set. However, once the developer becomes aware of these cases, retraining the network with the according data can often solve the problem. If it does, the relevant features will then be discovered automatically during retraining. This gives another advantage over traditional approaches, where the chosen handcrafted features might not be expressive enough to describe edge cases correctly. In this scenario, selecting new or additional handcrafted features is necessary to overcome the problem. For example, if occlusions are not considered at first, simple features such as contours might be enough for correctly classifying different objects. But once occlusions are introduced, characterizing parts of the contours may not be visible anymore, which could lead to errors.

1.4.3 Deep Learning for Open Set License Plate Type Recognition

To summarize the previous section, deep learning is capable of overcoming or at least mitigating the problems with all sorts of variances if enough training data is available. For an LPTR classification system, there are two important requirements, as described in Section 1.1.

New Classes

The LPTR system must be able to incorporate new classes which have not been considered during the initial development of the system. Furthermore, adding these new classes should be quick and inexpensive. Therefore, retraining the network after a limited number of new classes has been added should be avoided whenever possible. Not using new classes for retraining the network is regarded as a given constraint for the LPTR system in this work. Every class which is initially available and which is used for training the network is defined as Train Class (TRC) in this work. Contrarily, each class which is added to the system afterwards is defined as Test Class (TEC). Together, the TRCs and TECs make up the known classes. Overcoming the constraint explained above can be achieved by using techniques from Few-Shot Learning (FSL). Wang et al. [WYKN20] give a great survey on the topic.

The goal of FSL is to correctly classify instances of classes for which only few training samples are available. This is achieved by using an additional dataset for training. The main idea is to learn features from the additional dataset which can also be used for recognizing the classes which only consist of a few samples. In FSL, this additional dataset often is only a means for training. In the LPTR context, the additional dataset consists of the TRCs. The classes with few samples correspond to the TECs. Various techniques for FSL, which also use the classes with few samples for training the network in addition to the extra dataset, exist. However, this violates the LPTR constraint. Approaches which comply to the constraint are based on embedding spaces. This work focuses on such methods.

The basic idea behind using embedding spaces for image classification with deep learning is the following [WYKN20]: A CNN is trained to learn a non-linear function mapping $\mathbb{R}^m \mapsto \mathbb{R}^n$ where $n \ll m$ from the input pixels of an image to a lower dimensional embedding space, which represents the characteristics of different classes. For the actual classification, a simple classifier is applied in the embedding space instead of using it on the raw pixels. The embedding space and its classifier are used due to the open set restriction. For each new TEC, it is only necessary to transform its representative images into the embedding space and to compute a classifier with the obtained embeddings. With this setup, the CNN calculating the embeddings does not require retraining. Notice that in this context, TEC refers to classes which are unknown during embedding space training. It does not refer to classes which are unknown to the entire system. TECs still require training samples in order to compute the parameters of the embedding space classifier. The exact type of classifier which is used in the embedding space is interchangeable, but it is usually a statistical model of simple nature. Different options are described in Section 3.2. These classifiers do not require iterative training. Their

parameters can be estimated directly. Therefore, updating an embedding space classifier for recognizing additional TECs is simple and quick, and does not violate the LPTR constraint.
It must be noted that the TECs are not used in the traditional FSL context in this work. This is because there is no restriction to the maximal number of TEC samples which can be used for training the embedding space classifier. In contrast, FSL can be defined more rigorously by limiting the exact number of samples which are available per class. In this work, more samples of a TEC are always desirable. This closely mimics how a LPTR system would be used in the real world, as data for new TECs would be collected over time. Not using all of this data if it was available would give a clear disadvantage. One benefit of having more data for each TEC is that it allows to use more complex embedding space classifiers, which is further discussed in Section 1.5.2 and Section 3.2.

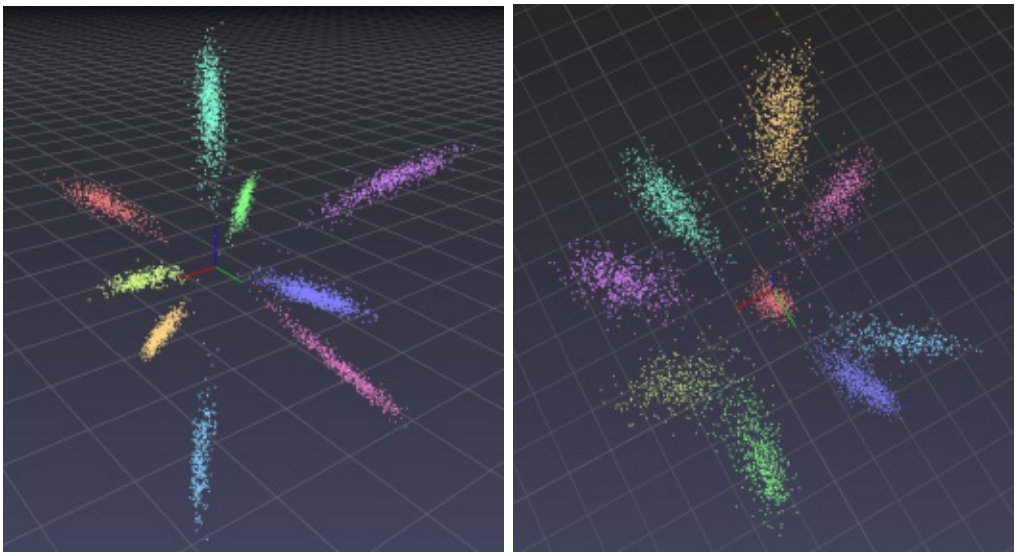
Unknown Classes
The LPTR system should also recognize instances of classes which are unknown to itself. In this context, unknown class means that it is neither a TRC nor a TEC, i.e. an LPT which the system has not seen for training the embedding space classifier. Geng et al. [GHC20] give a comprehensive survey on the underlying problem, which is known as Open Set Recognition (OSR). The aim of OSR is to distinguish between known and unknown classes, which fundamentally makes it a binary classification task. Performing OSR can be achieved by setting a threshold for the required confidence in the classifier's prediction for a known class. If the highest computed class confidence is below this threshold, the input is identified as belonging to an unknown class. The threshold is computed empirically using the training datasets of the known and unknown classes. Notice that all samples of the unknown classes can have the same label, as long as it is distinguishable from the labels of the known classes. Combing OSR with embedding spaces is straight forward, as the embedding space classifiers directly output class confidences. In fact, it is also common to use embedding spaces solely for OSR [GHC20]. Embedding spaces allow to perform thresholding on other parameters than confidence, such as distance, which is further discussed in Section 3.3.1.

Learning an embedding space which is capable of distinguishing TECs from each other as well as from TRCs, while still recognizing unknown classes, is challenging. There are various algorithms to train an embedding space which use different loss functions. Recent state-of-the-art approaches are elaborated in Section 2.2. However, the underlying idea remains the same for all of them.

1.4.4 Desired Properties of Embedding Spaces for Classification

The natural question to ask at this point is how a qualitative embedding space must be structured after training. The classification performed within this space should be as simple as possible and TECs must also be recognizable. A desired representation for every class is a compact and homogeneous cluster which has large empty gaps to the clusters

of other classes in the embedding space. Intuitively, large gaps between clusters of TRCs allow the cluster of a TEC to be located somewhere in between existing clusters. This enables distinguishing it from all other classes. In other words, it is desirable to only fill a fraction of the available high dimensional embedding space with the TRCs. Moreover, each TRC should have its own dedicated subspace, preferably of uniform variance, to enable the usage of simpler classifiers. The majority of the embedding space should be left empty after training to leave space for TECs. For these reasons, it is desirable that the classification problem is linearly separable within the embedding space. Two different three dimensional embedding spaces are shown in Figure 1.3. The embedding space in Figure 1.3a is qualitatively better than the one in Figure 1.3b for open set problems, as it is more compact and has larger gaps between different TRCs.



(a) This embedding space is obtained by using the Arcface softmax extension [DGXZ19], which is explained in Section 2.2.1. The resulting cluster of each class is more compact and the gaps between different classes are larger than in (b).

(b) The cluster of each class is spread out more widely than in (a), leaving smaller gaps between distinct classes. This embedding space has been trained with a traditional softmax classifier enforcing no explicit constraints on the embeddings.

Figure 1.3: Points within two 3D embedding spaces trained with the same CNN and dataset, using a different loss function. Each colored point represents the embedding of one image and each color one class. The used dataset is a subset of the license plate type dataset described in Section 4.1 and consists of 9 classes.

1.4.5 License Plate Type Recognition using Embedding Spaces in Practice

Using an embedding space based classification algorithm in a real world LPTR system can be achieved as follows, once the embedding space has been trained. First, all classes of interest for the specific application are identified. These classes can contain both TRCs and TECs. Then, for every class, a certain amount of embeddings are computed, given a set of representative images for the respective class. The embeddings of each class are then used to compute the relevant parameters of a simple classifier, which can be as trivial as the mean vector of all embeddings. Notice once again that TRC refers to the training of the embedding space. Both TRCs and TECs consist of a train and test set for the embedding space classifiers. The calculated parameters of each classifier are stored in a database and associated with the according class label. All steps mentioned so far only have to be done once in advance, during the system setup.
During operation, the system is given an image for which the according class should be computed. To do so, the embedding of the input image is calculated and then given to all precomputed classifiers stored in the database. Assuming that the system is only confronted with images of classes known to its database, the class with the highest score always wins and the according label is assigned to the input image. In scenarios where images of classes unknown to the database can also be an input, i.e. an open set, the system must be able to identify that these images belong to an unknown class. As mentioned before, this is particularly relevant for LPTR in practice, as many rare LPTs exist for which little to no data could be collected in advance. Identifying unknown classes can be achieved by thresholding [GHC20]. This is further discussed in Section 3.3.1.

1.5 Aim of the Work

One obvious goal is to develop a system which is capable of solving the LPTR task as described in Section 1.1 as well as possible. Apart from that, many interesting scientific questions arise when thinking about this problem. We distinguish between those regarding LPTR specifically and those concerning embedding spaces. To the best of our knowledge, no scientific work which is specifically targeted at solving LPTR has been done yet. A lot of recent research on embedding spaces using deep learning has been conducted on face recognition problems [SKP15], [DGXZ19], [GZJ+15]. These algorithms are further discussed in Section 2.2. Many concepts which are introduced in the face recognition domain can be directly applied to LPTR. Vice versa, the analysis methods used in this work, which do not directly concern LPTR, can be applied to embedding space based methods in general. Face recognition is heavily used as point of reference in this work, as it operates under similar conditions as LPTR. Methodologically speaking, recognizing unknown LPTs is similar to verifying whether or not two faces are identical. Furthermore, classifying new faces without requiring expensive retraining is a standard use case in face recognition.

1.5.1 Research Questions in the License Plate Type Recognition Domain

The biggest question about LPTR using deep learning is how to deal with the identifying characters of an LP, as they vary between different LPs. The easiest idea which comes to mind is to simply ignore their presence. However, this could lead to severe problems if a certain character was placed at the same position for most images of a particular LPT in the training dataset. This is because it would introduce an unwanted bias. Consider the following scenario of a country or LPT implementing a policy which systematically iterates through characters over time. For example, the LP syntax could have the form $annn$, where a stands for a letter and n for a number. According to this system, the first registered LP would be $A000$. Then, each time a new LP was registered, the number would increase until it eventually reached $A999$. The next LP in order would be $B000$. A system assigning characters in a way like this can be problematic for developing a learning based LPTR algorithm if the training data has been collected over a short period of time. In our example, the CNN could learn the letter A as reliable indicator for identifying a certain LPT. This is because deep learning algorithms do not allow to formally implement hard, abstract constraints such as "ignore all characters when making a decision". In our example, using the presence of A as first character for making a decision is an unwanted bias. What makes this problem worse is its time dependent nature, as it may stay unrecognized for a long period of time. A good indicator for a system suffering from this problem could be a slowly decreasing performance over time. In our example, more and more vehicles which have a syntax that starts with a different letter than A would be registered.
The main research question associated with this problem is how to prevent the CNN from learning the presence of particular characters as reliable decision rules. Now, the next question is how to proof that the CNN does not learn such faulty behavior. Many people criticize neural networks for being black box systems, as the exact reasons why they make certain decisions are unknown. Proofing that a neural network has a property like "ignores all characters" formally is impossible with current mathematical methods. However, many techniques have been developed in recent years which try to explain based on which features a neural network makes a certain decision [SVZ13], [SCD+17], [AGM+18]. Those which are relevant to this work are discussed in Section 2.3. Thus, a crucial part for verifying that the CNN behaves as desired to some degree is to make use of these techniques.
Conveniently, the same algorithms can also be used to partially answer another relevant question: Which features does the CNN learn and extract from different LPTs? It is not clear in advance which of the LPT characteristics (symbols, textures, additional texts, positions of the identifying characters) the network will identify as discriminative to what degree.

1.5.2 Research Questions about Embedding Spaces

Most deep learning research on classification focuses on reaching better performance than previous methods regarding metrics such as Mean Average Precision (mAP) and accuracy. As time has shown, this approach is great for closing the performance gap between humans and machines in many areas such as image classification. However, this strategy does not bring science closer to a real understanding what exactly artificial neural networks learn in order to solve a certain task.
Regarding embedding spaces specifically, to the best of our knowledge, there is little to no research which analyzes what a learned embedding space actually looks like. The research about training an embedding space mentioned in Section 2.2 goes into detail about the constraints which are enforced on the embedding space, i.e. under which conditions it is obtained. However, it does not analyze how the data is actually distributed in the embedding space once training has finished. State-of-the-art systems such as triplet loss [SKP15] and Arcface [DGXZ19] neglect the shapes of the true distributions. Instead, they simply compute the mean embedding as center for each class, given a certain number of samples. For classification, they choose the nearest center to the input of interest and assign the respective class label to it. This method only guarantees to select the correct class if each sample's intra-class distance to the center is always smaller than its inter-class distances to the other centers. As long as this condition is fulfilled, the distributions can have arbitrary shapes. In order to fulfill this criterion for stretched or differently sized distributions, the gaps between distinct classes need to be sufficiently large. While this is fine for closed sets, it is problematic for open sets because TECs will lie within empty gaps between TRCs. Consequently, this condition is likely to be broken for stretched TRCs once TECs are added. Thus, it is desirable that all classes can be recognized independently of the size of the empty gaps between their distributions. This can be achieved with a more restrictive condition, where the classes' respective distributions must share the same variance in all directions. In this case, the class distributions are hyper-spherical and have the same reach. To summarize, under the open set restriction, a nearest center based classifier must assume the restrictive condition as a prior to guarantee flawless results. An example which violates both the more restrictive and the general condition is illustrated in Figure 1.4.

One of the main goals of this thesis is to analyze the class distributions in the embedding space. The results of this investigation depend on the used training algorithm, dataset and parameters. It is unclear in advance how similar a class's distribution is to the desired hyper-spherical shape. However, the assumption that a TRC has the shape of a hyper-ellipsoid seems plausible, just by looking at the shapes of the class distributions in Figure 1.3. Besides, the central idea of algorithms learning an embedding space is to minimize intra-class distances and maximize inter-class distances. Thus, each class must be represented by a single cluster in the embedding space, assuming the training has been successful and has converged. A cluster in the shape of a hyper-ellipsoid is a natural way for achieving this.
Consider the following scenario as an analogy for the emergence of a particular cluster during training. Imagine individual atoms or molecules in outer space flying towards

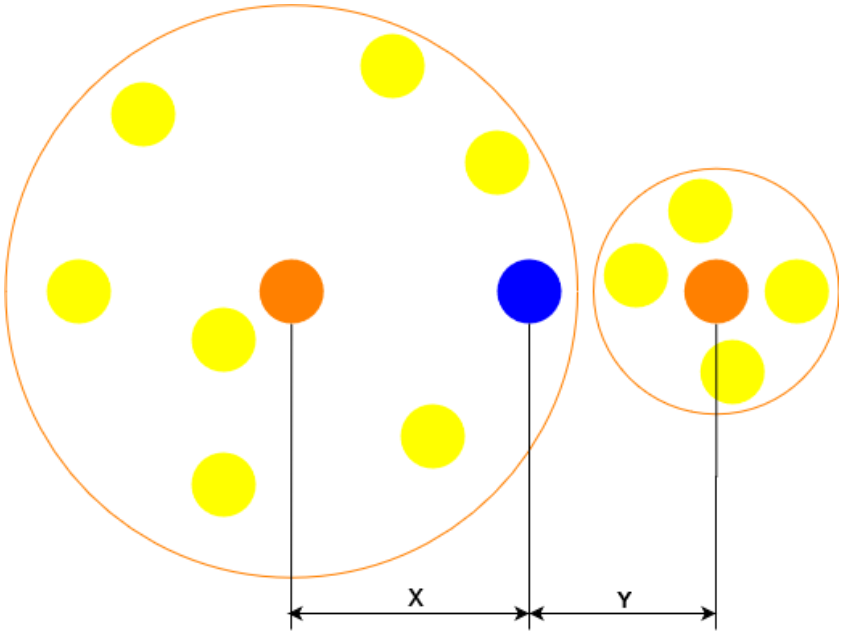


Figure 1.4: The image shows two distinct classes, where each class for itself has the same variance in all directions. However, the variance of the left class is larger. The yellow dots represent the samples based on which the parameters (mean and variance) of each class have been calculated. The blue circle symbolizes a new sample which should be classified as either belonging to the left or to the right class. The point's Euclidean distance to the right class Y is slightly smaller than its distance X to the left class. However, because the left class has higher variance, it is more likely that the blue point belongs to this class. If only the class means are considered without the variances, the blue circle will be incorrectly assigned to the right class.

each other due to the force of gravity. Each atom is equivalent to one training sample of the same class in this scenario. Gravity pulls each atom towards every other atom, which is equivalent to minimizing intra-class distances. The point of equilibrium for this situation is one joint spherical object. This object could be a planet or a sun, depending on the kind and amount of atoms. This analogy is incomplete though because additional forces pushing the atoms of one planet away from the atoms of other planets are missing. These forces would represent maximizing inter-class distances and could potentially cause an elongation along some axes of a planet. In our analogy, such an additional force could be a centrifugal force, caused by the rotation of the planet. Earth is slightly elliptical due to its rotation, for example.
The arguments about a cluster's shape presented above only apply to TRCs but not to TECs. The shape of a TEC has far greater uncertainty in the embedding space, as the network has not directly been trained on it. The more similar instances of a TEC are to those of the TRCs, the higher the probability that the TEC's shapes will be similar those of the TRCs. This is because the network knows how to interpret most underlying

patterns of a TEC if it is similar enough to the TRCs. It cannot be expected that a TEC which shares zero commonalities with the TRCs will form one compact cluster. In this case, the neural network will most certainly not be able to identify the class's patterns correctly, as it has not encountered them during training. Therefore, analyzing the shapes of TEC distributions seems promising for evaluating whether or not the network has learned to generalize well enough to make sense of the TECs. If it has, the shape of a TEC cluster should look similar to the one of a TRC. To summarize, another goal of this work is to see how strongly the shape of TEC distributions vary from those of TRC distributions. Note that for the reasons mentioned above, this property depends on the used data and does not generalize to arbitrary datasets.

CHAPTER 2

State of the Art

This chapter is split into three parts. First, the general setup for solving LPR and recent research, which uses deep learning for this purpose, are briefly discussed in Section 2.1. Then, various algorithms for training embedding spaces are elaborated in Section 2.2. The final part of this chapter (Section 2.3) summarizes different techniques for creating saliency maps, which visualize which parts of input images are most relevant to a trained network. The focus lies on gradient-based saliency techniques.

2.1 License Plate Recognition (LPR)

As LPTR is only a subtask of LPR, it is worth discussing how such a system is typically setup. Broadly speaking, LPR consists of two main subtasks. First, the LP has to be located in the image, which is known as License Plate Detection (LPD). This is necessary because it is not possible to only capture the LP without any additional background and perspective distortion in practice. In real world applications, images of LPs are taken with stationary cameras while the respective cars on which the LPs are mounted are driving. This leads to a lot of uncertainty about a car's precise position regarding image coordinates. As a result, the LP only fills a fraction of the captured image. This ensures that the car will be contained within the camera's view frustum in almost all situations. A captured image typically contains large parts of the car, the street and in many cases even other cars. LPTR operates on crops of LPs and therefore causally follows LPD. The second stage of LPR is responsible for segmenting and classifying the characters on the extracted LP.
Precise details about how these two steps are solved are not directly relevant for this work. However, this paragraph points out references for the interested reader. Until the recent advances in deep learning, LPR has been dominated by traditional computer vision approaches. Such systems made heavy use of thresholding, edge detection, morphological operations, color, texture and more. A great summary about how these techniques are

used for LPR is given by Anagnostopoulos et al. [AAP+08] and by Du et al. [DISB12]. In more recent years, most research regarding LPR has been carried out using deep learning. Laroca et al. [LZG+19], [LSZ+18] use a pipeline consisting of three different neural network models for solving the entire LPR task. They use the You Only Look Once (YOLO) network architecture, introduced by Redmon et al. [RDGF16], for each stage. In a nutshell, YOLO is capable of classifying several objects in an image. The underlying network also outputs the respective axes aligned bounding box in image coordinates for each classified object. In the approach by Laroca et al., the first network is in charge of detecting all vehicles in the scene to reduce the search space for plates. The second network then searches for license plates on each detected vehicle in the scene, individually. Interestingly, they apply what they call layout classification within this step. This can be viewed as a simple closed set variation of LPTR which coarsely distinguishes between different LP layouts. However, their classification system is only used as means to identify the approximate shape and overall layout of a LP for further processing. Its goal is not to distinguish between different LPTs. The final stage of their system processes each detected plate one-by-one and detects and classifies all characters on the plate. Knowing the plate layout from the previous step, it is possible to modify the read characters to fit the constraints implied by the layout. Note that this method performs all stages of the LPR task regarding computer vision using neural networks and does not require any traditional computer vision approaches. Jorgensen [Jør17] and Silva et al. [SJ17] develop very similar systems which are based on a YOLO network for LPD as well.

2.2 Embedding Spaces

Using intermediate embedding spaces is state of the art in both OSR [GHC20] and FSL [WYKN20]. This section discusses various methods for training embedding spaces. The goal of each technique is to create empty gaps between distinct classes by maximizing inter-class distances and minimizing intra-class distances.

2.2.1 Softmax Extensions

First, traditional classification using the softmax activation function is discussed from a geometric view point. This new perspective allows to modify the underlying concept, such that marginal constraints can be added between distinct classes, leading to more compact embedding spaces. Three state-of-the-art approaches on which the derivations and explanations in this section are based on and inspired by are introduced and compared [LWY+17], [WWZ+18] [DGXZ19].

2.2.1.1 Softmax from a Geometric Perspective

The standard way of training a classifier with a neural network is to use a separate output for each possible class j, i.e. one-hot-encoded labels. The raw output values computed by the network for a given input a are also known as logits. To obtain a confidence score p_j for each class based on the logits, the softmax activation function is used as defined in Equation 2.1. Intuitively, it can be understood as interpreting the output logits of the network as unnormalized log likelihoods. This observation follows directly from the equation, which first transforms the logits from log space to linear space and then normalizes over all classes N.

$$p_j = \frac{e^{logit_j}}{\sum_{k=1}^{N} e^{logit_k}} \tag{2.1}$$

The loss function L used for training, which is based on these confidences, is given by Equation 2.2. There are two different interpretations of this function, which are maximizing the log likelihood of the target class y and minimizing the cross entropy H between the target outputs z_j and the actual outputs p_j. The former is easy to see, as maximizing the log likelihood is equal to minimizing the negative log likelihood. The latter can be derived from the general equation for cross entropy, by setting $z_j = 0 | \forall j \neq y$ and $z_y = 1$, as shown in Equation 2.3.

$$L = -\log(p_y) \tag{2.2}$$

$$L = H(z,p) = -\sum_{j=1}^{N} z_j \log(p_j) = -\log(p_y) \tag{2.3}$$

Independently of the interpretation, the loss will decrease if p_y increases and will converge to 0 as p_y approaches 1. p_y in turn increases if either $logit_y$ increases or the logits of all other classes decrease, which is exactly what happens during training. The logits themselves are the final outputs computed over many neural network layers with non-linearities in between.
Let x be the activation feature vector of the final layer before the individual class outputs. To keep the equations clearer, indices are not considered for different inputs and x can represent the feature vector of *any* single input a. In this case, the individual output logits are computed by Equation 2.4. W_j represents the weights of the final fully connected layer which are used for computing the respective class's output $logit_j$. More specifically, W_j is the jth row vector of the last fully connected layer's weight matrix W. b represents the additive scalar bias. Note that Equation 2.4 is also written in terms of the dot product, which incorporates the angle $\varphi_j \in [0, \pi]$ between the vectors W_j and x.

$$logit_j = W_j \cdot x + b = \|W_j\| \cdot \|x\| \cdot \cos\varphi_j + b \tag{2.4}$$

The computation of the logits by these means without the bias is visualized in Figure 2.1.

Equation 2.1 implies that the highest confidence is always assigned to the logit with

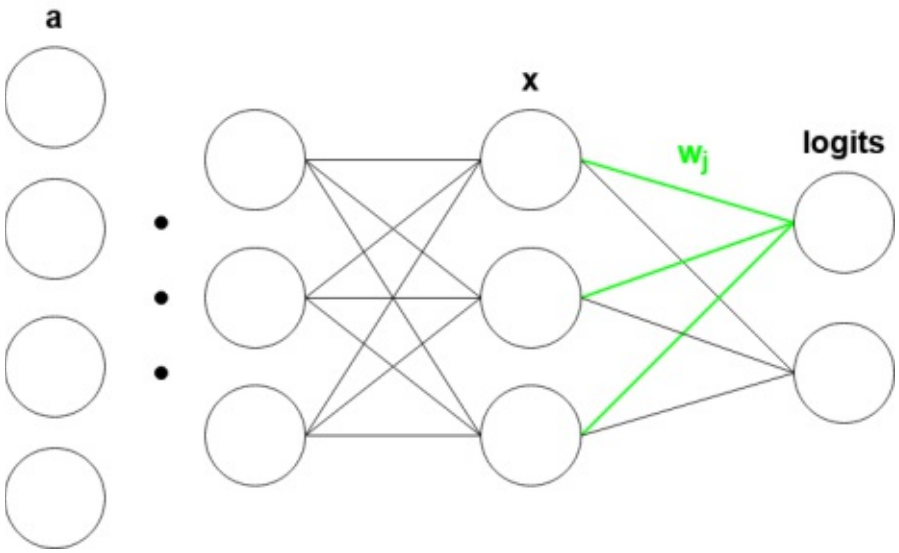


Figure 2.1: A schematic drawing of a typical architecture for a neural network classifier. The input a is transformed to a feature vector x over many layers. This vector is then multiplied by the vector W_j to obtain the respective output logit for class j.

the highest value. This is because the exponential function is strictly monotonically increasing over $\mathbb{R}$. Thus, an input a is assigned to the target class y if Inequality 2.5 holds.

$$\|W_y\| \cdot \|x\| \cdot \cos\varphi_y + b > \|W_j\| \cdot \|x\| \cdot \cos\varphi_j + b \quad |\forall j \neq y \tag{2.5}$$

Notice that when both $\|W_j\|$ and $\|x\|$ are set to 1, b is set to 0 and the logits are multiplied by a constant scalar s instead, the only remaining variable in Equation 2.4 is the angle φ_j. Therefore, it simplifies to Equation 2.6. The effects of s are discussed in Section 2.2.1.3.

$$logit_j = s \cdot \cos\varphi_j \tag{2.6}$$

By substituting Equation 2.6 into Inequality 2.5, it is easy to see that the classification decision only depends on the angles φ_j. To classify an input correctly, Inequality 2.7 must be fulfilled.

$$\cos\varphi_y > \cos\varphi_j \quad |\forall j \neq y \tag{2.7}$$

This can be further simplified to Inequality 2.8, by taking the inverse cosine function, which flips the $<$ operator, as it is strictly monotonically decreasing in the range $[-1, 1]$.

$$\varphi_y < \varphi_j \quad |\forall j \neq y \tag{2.8}$$

Given this inequality, one can observe that in order to fulfill the conditions, the angle φ_y of the target class must decrease, while the angles φ_j of all other classes should increase. In fact, this is exactly what the neural network optimizes for during training, as decreasing φ_y increases $logit_y$ and increasing φ_j decreases $logit_j$. Notice that the training

operates in angular space on the manifold of a hyper-sphere and not in Euclidean space because both $\|W_j\|$ and $\|x\|$ are normalized to 1. This is why the network must learn to distinguish between classes solely based on the angles φ_j. Intuitively, the training process can be imagined as pulling all x of the same class closer together while pushing them further away from the x of all other classes. More specifically, this process happens on the manifold of a hyper-sphere. At this point, the reader might notice the similarity between the desired properties for the space in which all x lie and those of an embedding space. In fact, the space of all x corresponds to an embedding space. One possible outcome of a converged training for various normalized x in an embedding space of dimensionality 3 is visualized in Figure 2.2. This figure shows the normalized versions of the same x shown in Figure 1.3a. Another way to think about the difference is that Figure 1.3a shows the values of x as they truly are before normalization, as opposed to Figure 2.2, which shows how they are interpreted during training.

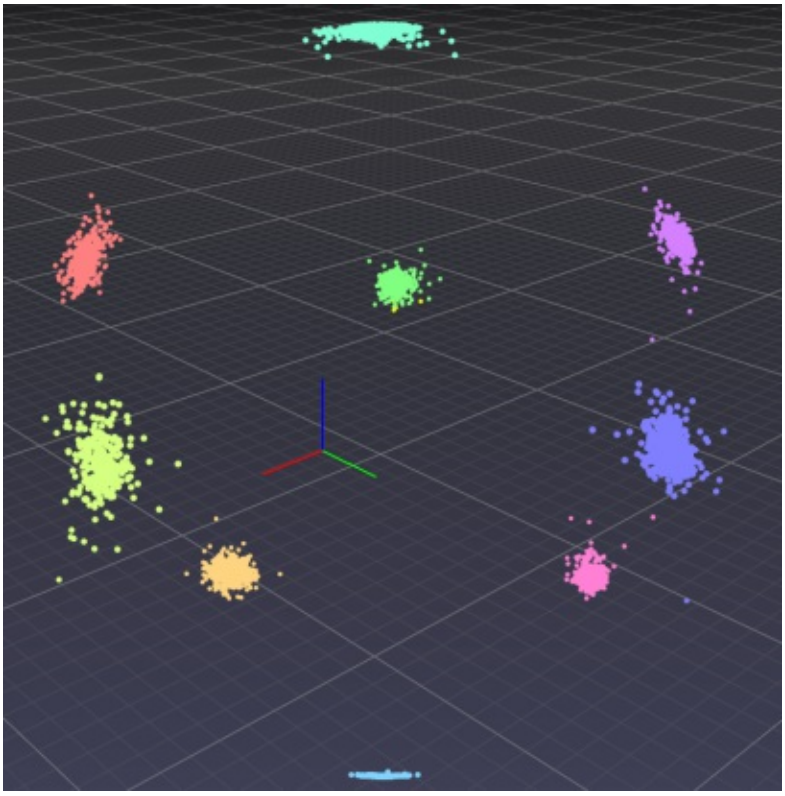


Figure 2.2: A 3D embedding space obtained by training a classifier using Arcface [DGXZ19]. Each point represents the normalized feature vector x of an input a and each class is illustrated by one color. Due to the normalization, all points lie on the surface of a sphere. During training, the angular distance of each x to its respective class center W_j is minimized.

The angle φ_j is given by Equation 2.9 and simplifies to Equation 2.10 if $\|W_j\| = \|x\| = 1$ holds.

$$\varphi_j = \arccos\left(\frac{W_j \cdot x}{\|W_j\| \cdot \|x\|}\right) \tag{2.9}$$

$$\varphi_j = \arccos\left(W_j \cdot x\right) \tag{2.10}$$

As W_j is a weight vector of the network, it is constant for all inputs x at any fixed point during training. The network tries to minimize the angles φ_j between all x which belong to class j and W_j. This allows to interpret W_j as the center of class j.

2.2.1.2 Incorporating the Margin

Leaving the computation of the logits as in Equation 2.6 will result in a separation between distinct classes without enforcing empty gaps in between them, as Inequality 2.8 illustrates. Thus, we want to add a penalty in form of a margin for the target class y, which incorporates the desired gaps. The underlying idea is to pretend that a feature vector x is further away from its target center W_y than it truly is. This forces the training to decrease the angle φ_y even further to classify the input correctly, leading to more intra-class compactness. As each class adds the margin to its respective samples, this leads to more compactness for all classes. As a consequence, empty gaps between distinct classes will emerge.
When making use of the margin all equations mentioned in Section 2.2.1.1 remain the same, except for the computation of the logits. Regarding the logits, the computation for the target class y changes, all other $logit_j \quad |\forall j \neq y$ remain unchanged and are computed as in Equation 2.6. Sphereface by Liu et al. [LWY+17] uses a multiplicative margin while both Cosface by Wang et al. [WWZ+18] and Arcface by Deng et al. [DGXZ19] apply additive margins in different ways. The respective formulas are given by Inequalities 2.11, 2.12 and 2.13. The reason why all three have the term *face* in their name is because they have originally been developed in the domain of face recognition.

$$logit_y = \|x\| \cdot \cos\left(m_s \cdot \varphi_y\right) \tag{2.11}$$

$$logit_y = s \cdot \left(\cos\left(\varphi_y\right) - m_c\right) \tag{2.12}$$

$$logit_y = s \cdot \cos\left(\varphi_y + m_a\right) \tag{2.13}$$

Notice how Arcface given by Inequality 2.11 does not multiply by a constant scalar s but by $\|x\|$ instead. This method was the first of the three approaches and the others later discovered that fixing the value for $\|x\|$ by using the constant s instead gives better results. This is explained with more detail in Section 2.2.1.3. Now the question is, which of the three margins works best empirically. To understand their difference intuitively, a geometric interpretation is helpful. Let us consider a binary classification problem with classes A and B for simplicity and visualization purposes. It is easy to generalize the following observations to multiple classes. For binary classification, the decision region for class A is given by Inequalities 2.14, 2.15 and 2.16 for Sphereface, Cosface and Arcface,

respectively. All three can be derived in the same way as Inequality 2.8. To obtain the decision region for B instead of A, φ_B and φ_A simply have to be exchanged with one another. The resulting decision regions are visualized in Figure 2.3.

$$\varphi_A \cdot m_s < \varphi_B \tag{2.14}$$

$$\arccos\left(\cos\left(\varphi_A\right) - m_c\right) < \varphi_B \tag{2.15}$$

$$\varphi_A + m_a < \varphi_B \tag{2.16}$$

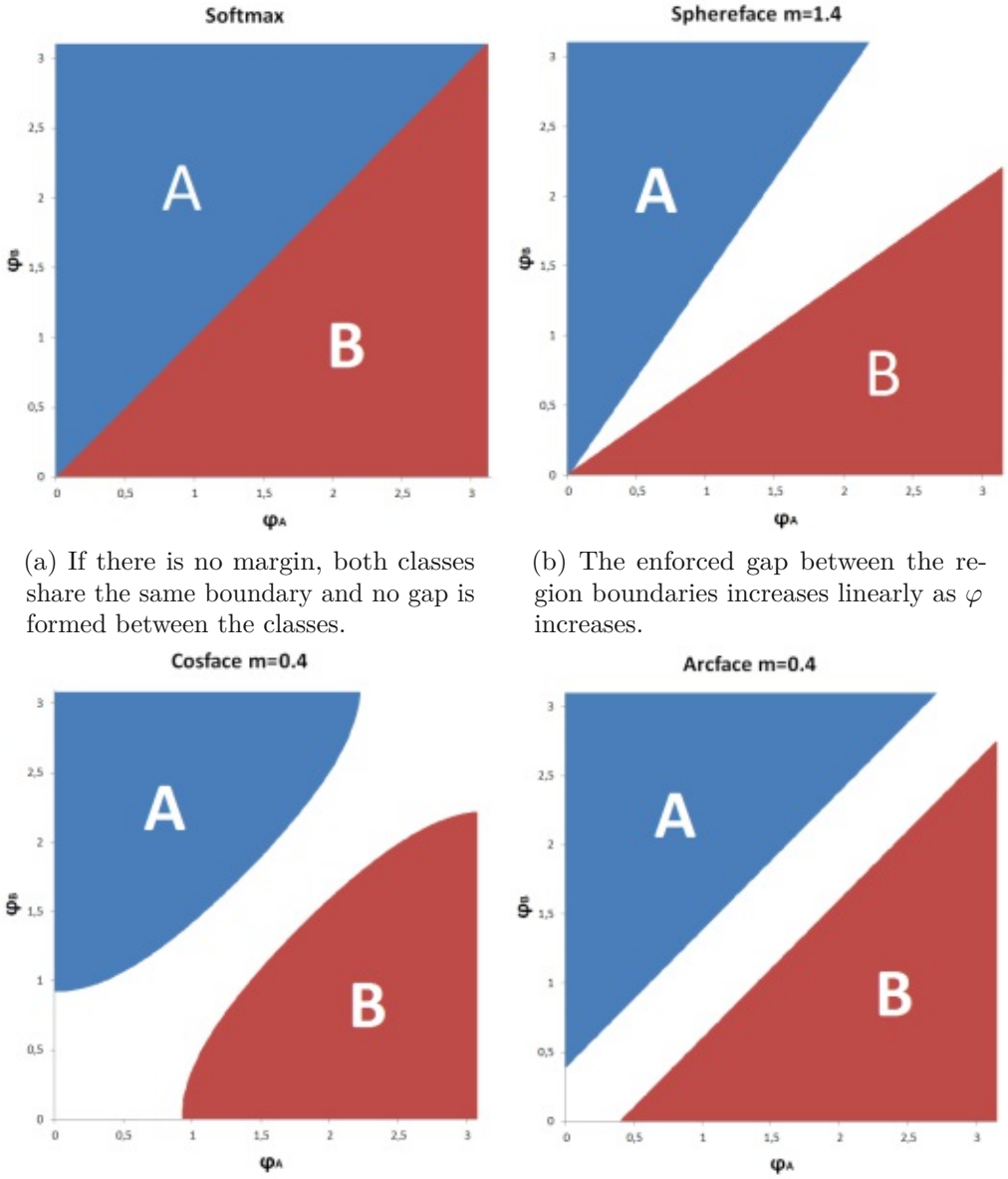
The enforced gap between any pair of two classes in a multi-class scenario behaves equivalently to the binary classification setup. However, this is difficult to visualize, as each added class introduces another dimension. Furthermore, it seems intuitive that the enforced gap should be independent of the exact value of the target class's angle φ_y. For example, why should it be desirable that a small angle requires less margin than a larger angle as enforced by Sphereface, shown in Figure 2.3b. Consider that if φ_y is small, all other angles φ_j should still be significantly larger. Empirical experiments by Deng et al. [DGXZ19] confirm the intuition that the constant gap enforced by Arcface indeed gives better results than the gaps which emerge from Sphereface or Cosface. In their study, the quality of the learned embedding space is evaluated in the open set face recognition context for various datasets. The authors also experiment with applying all three different types of margins at the same time, as they are not mutually exclusive. Their results indicate that the Arcface margin alone works best.

2.2.1.3 Impact of the Scale Parameter

Finally, let us consider the effect of the scale parameter s. Wang et al. [WWZ+18] formally derive a lower bound of its value which depends on the number of classes. We try to offer an intuitive, alternative explanation for why this value is necessary and how it can be chosen.
Without applying any scaling to the logits, their value will always be within the range $[-1, 1]$ if both W_j and x have unit length. If we assume that the target class y scores the maximal value 1 while all other $N-1$ classes take the minimal value -1, the resulting confidence p_y can be computed by Equation 2.17. It can be derived by plugging the respective values into Equation 2.1. Notice that Equation 2.17 assumes the best case scenario, yielding the highest possible value for p_y. In practice, the unscaled $logit_y$ will be smaller than 1 and the other unscaled logits will be larger than -1.

$$p_y = \frac{e^s}{e^s + (N-1) \cdot e^{-s}} \tag{2.17}$$

Figure 2.4 shows various results for p_y when varying N and s. For example, if s is fixed at 1, the value for p_y quickly converges towards 0 as N increases. This is clearly undesirable because p_y should approach 1 in this scenario. The figure also demonstrates that as s increases the value for p_y rises if N is fixed. This behavior can be explained by the nature



(a) If there is no margin, both classes share the same boundary and no gap is formed between the classes.

(b) The enforced gap between the region boundaries increases linearly as φ increases.

(c) A non-linear decision boundary which corresponds to wider gaps for particularly small and large φ.

(d) The gap between both classes has the same constant value which is independent of φ.

Figure 2.3: Decision regions for a binary classification problem for the traditional softmax classifier and its extensions which make use of a margin parameter. The blue regions represent the combinations of angle values which classify the respective input as A and the red regions illustrate the equivalent for B. The algorithm and margin value used are written above each respective graph.

of the exponential function. Linearly increasing inputs lead to exponentially increasing outputs, hence the name. Consequently, if the range of input values is increased by applying the scale factor, the outputs grow further apart. This allows the target logit to outweigh the sum of the other logits when the normalization is applied.

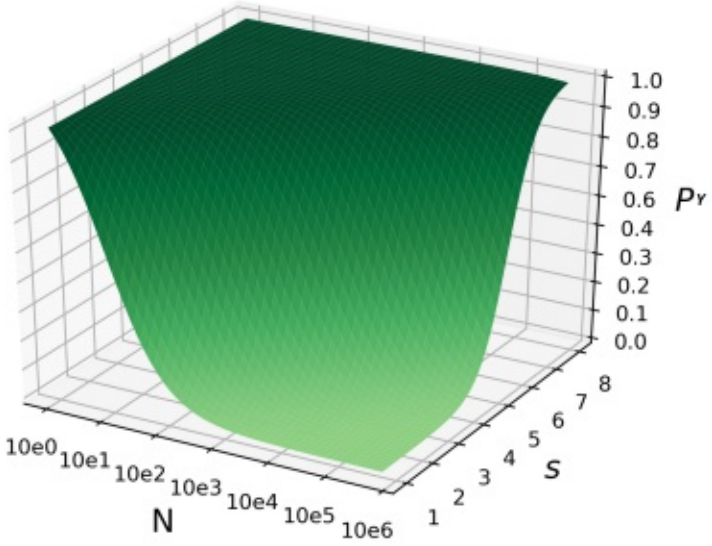


Figure 2.4: The maximal achievable target confidence score p_y for a classifier, assuming the scale s and the number of classes N are parameters. This graph represents the best case scenario, yielding the theoretical maximal values for p_y, as defined in Equation 2.17. Notice that the N axis shows values on a log scale.

To calculate the required scale factor s for obtaining a certain minimum target confidence p_y, given N classes, Equation 2.17 has to be solved for s. Let us consider a more realistic scenario for computing this value, by assuming a logit value of 0 instead of -1 for all non-target classes. A logit of 0 is equivalent to an angle φ of $\pi/2$. Replacing e^{-s} in Equation 2.17 by 1 and solving for s yields Equation 2.18. Solving it for $p_y = 0.99$ and $N = 100$ yields a minimal scale of 9.2, for example. This heuristic equation can be used to calculate an approximately fitting value for s.

$$s = \ln\left(\frac{p_y \cdot (1 - N)}{p_y - 1}\right) \tag{2.18}$$

2.2.2 Triplet Loss

Another popular method for learning an embedding space is to optimize directly on the embeddings without using an additional bottleneck classification layer as in Section 2.2.1. In order to do so, constraints for intra-class compactness and inter-class gaps are incorporated into the loss function, which operates on concrete vectors in the embedding space. Broadly speaking, there are two different types of loss functions of this kind, which

are contrastive loss and triplet loss. Examples of the former are given by [TYRW14] and by Sun et al. [SCWT14], using Siamese networks. However, here we focus on the latter, which has been introduced by Schroff et al. [SKP15]. This is because its results outperform contrastive loss in most domains, such as face recognition [MD18]. It is also the most widely used method which operates directly on the embedding space.
The main idea behind triplet loss is to make use of three different inputs at once for a single loss value computation by putting them in relation to one another. The inputs are the anchor a_a, positive a_p and negative a_n. Their respective representations in the embedding space are x_a, x_p and x_n. The anchor can be viewed as the equivalent of the target class in classification. The positive represents an input of the same class as the anchor and the negative an input of a different class. The aim is to pull the positive closer to the anchor and to push the negative further away from the anchor. The concept of triplet loss is illustrated in Figure 2.5.

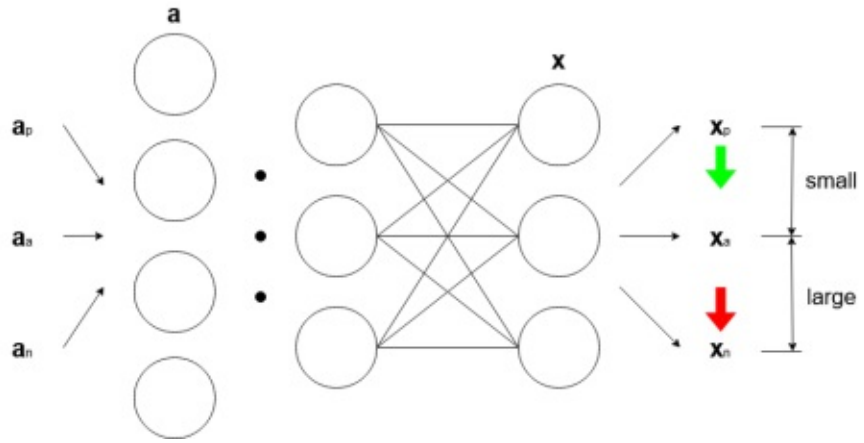


Figure 2.5: A schematic drawing of the principle behind triplet loss. The inputs, anchor a_a, positive a_p and negative a_n are transformed to their respective embedding representations x_a, x_p and x_n, using the same neural network. The green arrow indicates that the network aims to pull x_p closer to x_a, whereas the red arrow symbolizes that x_n is being pushed away from x_a, during training.

The desired outcome is that x_p is a certain amount m closer to x_a than x_n, for all possible configurations of the triplet $T(x_a, x_p, x_n)$. The hyper-parameter m represents the margin or desired gap between distinct classes. Mathematically, this is expressed by Equation 2.19.

$$dist(x_a, x_p) + m < dist(x_a, x_n) \quad |\forall (x_a, x_p, x_n) \in T \tag{2.19}$$

Originally, the distance between different embeddings has been measured in Euclidean space, but Deng et al. [DGXZ19] later pointed out that better results are obtained in angular space. This is because all x are normalized for computing the triplet loss, which constraints them to lie on the manifold of a hyper-sphere, just like all methods described

in Section 2.2.1. Due to the constant curvature of the hyper-sphere's manifold, it is more natural to measure distances with angles. To account for different metrics and to generalize, $dist(a, b)$ is used as a placeholder for an appropriate distance metric in this section. As mentioned, two examples are the Euclidean distance $dist(a, b) = \|a - b\|$ and the angular distance $dist(a, b) = arccos(a \cdot b), \|a\| = \|b\| = 1$. The loss function which achieves the desired constraints is given by Equation 2.20. The max function is used for clamping, in order to avoid negative loss in cases where the constraint is already fulfilled.

$$L = max(0, dist(x_a, x_p) + m - dist(x_a, x_n)) \tag{2.20}$$

The challenging aspect about triplet loss is to select triplets from which the optimization benefits most. For example, if a specific triplet already fulfills the constraint of Equation 2.19, there is no loss. This results in no improvement and change of the network's parameters. The training benefits most from hard positives and hard negatives. In this context, the hardest positive is the positive furthest away from the anchor $argmax_{x_p} dist(x_a, x_p)$ and the hardest negative is the negative closest to the anchor $argmin_{x_n} dist(x_a, x_n)$. Schroff et al. [SKP15] note that consistently choosing a negative which is closer to the anchor than the positive $dist(x_a, x_n) < (x_a, x_p)$ can result in convergence to a poor local minima.
Finding the optimal triplets for each parameter update during training scales poorly, as it requires searching through the entire dataset each time. What makes it worse is that the complexity of finding the best triplet for n samples is not $O(n)$ but $O(n^3)$. This is because all permutations need to be considered for the triplet. Therefore, online search strategies are often used for triplet mining in practice. One simple method is to increase the batch size and to only pick a subset of the samples within the batch which contains quality triplets. Some more advanced strategies for online triplet mining are given by Hermans et al. [HBL17].

2.2.3 Auto-encoders

One of the most popular and researched unsupervised deep learning methods is the auto-encoder. This section is about the supervised auto-encoder by Gao et al. [GZJ+15], which adds supervision in the form of class annotations to a traditional auto-encoder. The goal of the supervision is to add a constraint for intra-class compactness in addition to the reconstruction target, in order to learn a qualitative embedding space.
The aim of a traditional auto-encoder is to learn an encoding function $f(a) = x$ and a decoding function $g(x) = o$ such that the loss $L = \|a - o\|^2 = \|a - g(f(a))\|^2$ is minimized. This means that the reconstructed output should be as similar to the input as possible. An auto-encoder only makes sense if the dimensionality of the encoding x is significantly smaller than the dimensionality of the input a, i.e. the information is compressed. The architecture of an auto-encoder is schematically illustrated in Figure 2.6.

Typically, auto-encoders are used for compression, feature reduction or clustering, which means that only the encoder is of interest after training. The decoder is only a remedy

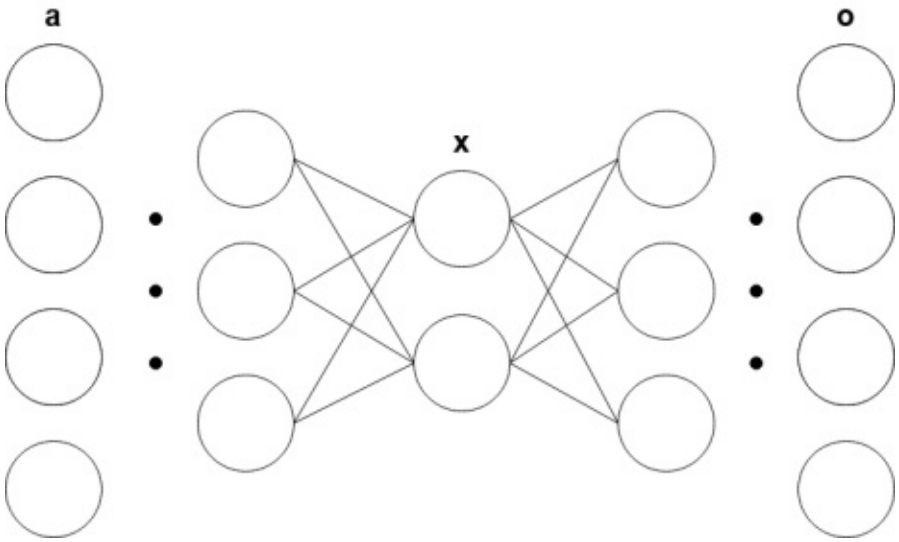


Figure 2.6: A schematic drawing of an auto-encoder. The input a is compressed into x, which in turn is used for reconstructing o. The aim of training is to make o as similar to a as possible.

for learning. The elegance of the auto-encoder originates from its unsupervised nature, as its only target is to reconstruct the input as closely as possible. Furthermore, it can be trained in a layer-by-layer fashion. First, an encoding $f_1(a) = x_1$ and decoding $g_1(x_1) = o$ are learned. Afterwards, another encoding $f_2(x_1) = x_2$ and decoding $g_2(x_2) = x_1$ are trained. This process can be repeated iteratively until a desired compression factor is reached.

Notice that for open set problems, the encoded space represents the embedding space. However, no constraints on the desired shape of the embedding space are enforced by this traditional target function formulation. The only goal is to reconstruct inputs, but there are many different ways of achieving this. For example, it is plausible that inputs which show the same type of variance, such as Gaussian noise, are summarized by similar embeddings. However, for classification it is desired that inputs of the same class are described by similar embeddings as opposed to inputs exposed to the same variance. The main idea behind the supervised auto-encoder by Gao et al. [GZJ+15] is to add a class similarity constraint in addition to the reconstruction target. To do so, a reference input a_j^r which represents the perfect instance of a particular class j, with as little variance as possible, is used. The goal of training is slightly modified, such that the reconstructed output o_j of any input a_j should match the reference input a_j^r as closely as possible, instead of the actual input a_j. In addition, the encoding x_j should be as similar as possible to the reference encoding x_j^r. The complete resulting loss function is summarized in Equation 2.21, where λ is a hyper-parameter controlling the weighting between the two loss components. The authors also add another loss component which aims for sparsity in the network's activations. However, as we do not use this term for any of the other

losses in Section 2.2, we also omit it here. Notice that a key difference compared to the other losses in the section is that this approach only enforces intra-class compactness. It does not explicitly aim for increased inter-class distances.

$$L = \lambda \|a_j^r - g(f(a_j))\|^2 + (1 - \lambda)\|f(a_j^r) - f(a_j)\|^2 \tag{2.21}$$

The supervised auto-encoder shares many similarities with the denoising auto-encoder [VLL+10]. The latter is an unsupervised method which adds artificial noise to the inputs. Similar to the former, it tries to match the reconstructed output of the noisy input to the unnoisy (reference) input. The main difference is that the supervised auto-encoder does not use noisy inputs, but any input of the same class as the reference input. This also explains the reason why it requires supervision while the denoising auto-encoder does not.

2.3 Saliency Maps

The biggest critic neural networks receive amongst experts and the public in general, is that they are black box or at least dark-gray box systems. Their decisions and inner workings are not fully understandable for a human because there are simply too many parameters to be considered.
Trying to mitigate this issue, an entirely new branch studying the interpretability and comprehensibility of neural networks has emerged. One key question which this research is trying to answer is "Which parts of an image are actually important when the network makes a decision?". For example, in the context of LPTR, one could expect that special symbols and textures of an LP are good indicators for classification. This question can be answered by saliency map techniques, which this section and work mainly focuses on. A different subbranch of this research called feature visualization views the problem from a different perspective and asks "Which features maximize the activation of a single neuron or a group of neurons in general?". By "in general", features which do not necessarily have to be part of an actual image of the dataset are meant. This question is briefly discussed in Section 2.3.5, but left for more rigorous exploration for future work.

2.3.1 Why Saliency Maps?

Because neural networks learn patterns from data, any bias in the training data can trick the network into believing that the wrong patterns are relevant for making a certain decision. Ribeiro et al. [RSG16] show an example, where a classifier is trained to distinguish between huskies and wolfs. But the training data is biased, such that all images of wolfs show them in snow scenes whereas the huskies are never shown in such a setting. Testing the trained model with data showing huskies in snow scenes and wolfs in different scenes shows that the network has learned to classify between huskies and wolfs depending on whether or not snow is present in the background. In this context, this means that huskies in snow scenes are classified as wolfs and the other way around. This

is clearly undesirable and unacceptable. The problem emerges because the network is only given the task which needs to be solved during training. There are no constraints on how to solve the problem. The snow bias in the example above is quite obvious and arguably not too difficult to identify. However, not all biases are this easy to spot and may hide deep within the data. The usage of saliency map techniques often allows to find such learned biases via careful human inspection.

2.3.2 Vanilla Saliency Maps

Saliency maps as introduced by Simonyan et al. [SVZ13] are a popular and simple method for investigating to which parts of the input image the network reacts strongly after training. Consequently, they can be used to spot learned biases by careful human inspection to some degree.
The idea behind vanilla saliency maps for a classification network is to first compute which class j has the highest score $logit_{max} = max(logit_j)$ after a regular forward pass. Then, the gradients which emerge solely from back-propagating the signal of the neuron with the highest class score are computed. The result of this step contains a partial derivative G^a_{xy} for every pixel a_{xy} of the input image a, given that it is a grayscale image. Mathematically, the partial derivative for a pixel a_{xy} is given by Equation 2.22.

$$G^a_{xy} = \frac{\partial logit_{max}}{\partial a_{xy}} \tag{2.22}$$

The absolute value of this partial derivative $|G^a_{xy}|$ is a good approximation for how much the respective pixel influences the largest logit. Together, all $|G^a_{xy}|$ make up the saliency map. An intuitive interpretation for why these values correspond to saliency attributions is the following: Changing a pixel's value whose corresponding partial derivative is large will affect the largest logit more significantly than adapting a pixel with a lower partial derivative by the same amount. Therefore, it is more important for the classification decision. To summarize, a saliency map consists of the absolute values of the partial derivatives of the respective pixels of the input image. In practice, these values are rescaled to a different range, such that they can be interpreted as a heatmap by a human.
When computing the gradients based on the highest class score, it is important to use the raw class logit before the subsequent softmax layer. This is because increasing the softmax score of a certain class can be achieved by reducing the logits of other classes. This is undesirable in this context because we are interested in which parts of the image cause a large logit for one specific class, as opposed to which parts decrease the logits of other classes.
For our biased wolf husky example, a typical saliency map for a wolf classification would highlight the pixels showing the snow and disregard the regions of the image showing the actual wolf.

2.3.3 Gradient-weighted Class Activation Mapping (Grad-CAM)

While a vanilla saliency map is a great first step towards understanding fractions about which features in a given image a neural network reacts to, it still does not reveal much about what is going on within the hidden layers of the network.
Selvara et al. [SCD+17] have introduced an approach called Grad-CAM, which tackles this problem. Grad-CAM can be viewed as an extension or generalization of vanilla saliency maps. Instead of only looking at the partial derivatives G^a_{xy} at the input image a, the gradients G^l_{uv} of a hidden layer A^l are considered. In this context, l refers to the lth layer of the network and u and v specify the spatial position within the layer. Notice that any G^k_{uv} represents a gradient as opposed to a partial derivative because there are several feature maps per layer. This is why all activations of a convolutional layer together are also known as feature volume, whose width U_l and height V_l decrease for deeper layers of the network.
Grad-CAM applies smoothing by averaging the partial derivatives of each feature map to retrieve a weighting vector w_l for each layer l, as illustrated in Equation 2.23.

$$w_l = \frac{1}{U_l \cdot V_l} \sum_{u}^{U_l} \sum_{v}^{V_l} \frac{\partial logit_{max}}{\partial A^l_{uv}} \tag{2.23}$$

Consequently, each feature map can be interpreted as either having a positive or a negative influence on the largest logit. This smoothing is particularly reasonable for deeper layers of the network, where features become increasingly complex and are likely to be detected by entire feature maps as opposed to individual feature wise convolutions. Finally, to retrieve a single 2D saliency map M^l for each layer l, the dot product between the respective activations and weight vector is computed for every spatial position (u, v), as in Equation 2.24.

$$M^l_{uv} = ReLU(A^l_{uv} \cdot w_l) \tag{2.24}$$

Looking at this equation, one notices two things. Firstly, Grad-CAM does not only construct a heatmap solely based on the gradients, but additionally also makes use of the activations A^l. Secondly, negative pixel values of the resulting heatmap are truncated by the Rectified Linear Unit (ReLU) operation.
For a given image, the activations within a feature volume represent how important certain image regions are for making a classification decision and not what the image looks like at these regions. Notice that for vanilla saliency maps the input pixels have the same role as the activations for Grad-CAM. Using these activations for constructing a Grad-CAM saliency map is the biggest difference compared to computing the respective vanilla saliency map. For the latter, the values of the input pixels are disregarded because they could potentially be misleading. Consider a scenario where the partial derivative for a very bright input pixel is small. If this partial derivative was multiplied by the pixel's intensity, the resulting saliency attribution would falsely become quite large. The opposite argument can also be made for a dark pixel with a large partial derivative. In a nutshell, the problem with using pixel intensities directly for vanilla saliency maps is that dark pixels can be part of important features whereas bright pixels might not be.

For hidden layers, especially those at the end of the network, the situation is different, as large activations correspond to detected features. The larger an activation and its partial derivative, the more important the respective feature is for a classification decision. If an activation is positive but its partial derivative is negative, this indicates that the detected feature is important for a different class than the one with the largest logit. As all feature volumes A^l are assumed to be the result of a preceding ReLU activation, the other two possible cases where an activation is negative do not need to be considered.
In practice, the Grad-CAM saliency map is only computed for the last convolutional layer because it represents the most high level features which have been detected. As a consequence, the resolution of the heat map is significantly smaller than for a vanilla saliency map, which can make it appear very pixelated.

2.3.4 Other Saliency Map Techniques

It must be noted that there are many other algorithms for computing saliency maps based on gradients, such as Integrated Gradients by Sundararajan et al. [STY17]. However, recent research by Adebayo et al. [AGM+18] has shown that many of these methods are actually not very reliable. In their experiments, they have performed sanity checks by computing saliency maps for both completely untrained classifiers and partially corrupted classifiers, for which a subset of the trained weights have been exchanged by random variables. Furthermore, they have investigated saliency maps which have been created using a network trained on randomly labeled data. Their results demonstrate that many saliency techniques such as Integrated Gradients show little change under such corruptions compared to the reference saliency maps created from a properly trained network. They conclude that many of the algorithms which fail their sanity checks share similarities with regular edge detectors. Only Grad-CAM and vanilla saliency maps pass their sanity checks. This is also one of the reasons why these techniques have been chosen for this work.
For the sake of completion, it must be said that saliency techniques which are not based on gradients exist too. Examples for these are Layer-wise Relevance Propagation by Bach et al. [BBM+15] and Deep Learning Important Features by Shrikumar et al. [SGK17]. The major disadvantage of such methods is that they require additional algorithms for back-propagating their relevance signals to the input. In contrast, the gradient back-propagation required for gradient-based saliency techniques is always available, as it is needed for training anyway. The biggest benefit of such methods is that they allow to back-propagate relevance even if the respective gradients are zero. However, these techniques are not used within this work and therefore not further discussed. The interested reader is encouraged to read the papers mentioned within this paragraph for more information.

2.3.5 Feature Visualization

While saliency techniques reveal which image regions are considered for a classification decision, they do not show which images maximally activate a specific neuron or com-

binations of neurons. An entire subbranch called feature visualization tries to answer this question. A good introduction is given by Olah et al. [OMS17], for example. The basic idea of feature visualization is to use gradient ascent on the input image instead of gradient descent on the network weights, to maximize a specific neuron activation of the network. After convergence, the resulting input image shows which features the neuron of interest reacts to most strongly.
More recent work on this topic by Carter et al. [CAS+19] focuses on creating 2D embeddings for frequently occurring activations and maps objects of each class to their own region within this embedding space. For the mapped features, the input images which create the according activations are created and shown in the 2D embedding space in a so called activation atlas. This technique allows to identify how similar objects are positioned relative to each other in the 2D embedding space. It must be noted that the embeddings do not directly correspond to an activation within the network. Instead, they are computed using t-SNE [MH08], a non-linear dimensionality reduction technique, on the last convolutional layer.

CHAPTER 3

Methodology

While Chapter 2 summarizes recent state-of-the-art methods related to this work, this chapter discusses other techniques which have been used. The majority of this chapter deals with classification on an open set using embedding spaces in general and is not specific to LPTR. First, various training aspects such as architectural choices for the neural network and hyper-parameter tuning are discussed in Section 3.1. Afterwards, different options for performing classification within a learned embedding space are elaborated in Section 3.2. Then, the used evaluation metrics are described in Section 3.3. These include both performance benchmarks on the test dataset and metrics for evaluating the embedding space. Finally, an extension for gradient-based saliency maps, which allows to also compute them for TECs, is introduced in Section 3.4.

3.1 Training the Embedding Space

In Section 2.2 many ways for training an embedding space effectively have been introduced. However, nothing has been said about the underlying network which transforms the inputs to feature vectors of the embedding space. The first part of this section discusses different architectural options for this purpose. Then, the selection of the training algorithm for this work is justified. Finally, the heuristic which has been used for choosing good hyper-parameters for network training is discussed.

3.1.1 Neural Network Architecture

Over the past ten years a lot of innovations have happened in the design of CNN architectures. A great overview of many recent developments is given by Gu et al. [GWK+18]. Two popular and commonly used types of model architectures are the Visual Geometry Group Network (VGG) by Simonyan et al. [SZ14] and the ResNet by He et al. [HZRS16a]. For this work, both types of architectures have been evaluated and compared. The main

reason for testing two different CNN types is to test if the obtained results regarding the properties of a learned embedding space strongly depend on the choice of architecture type.

3.1.1.1 VGG Type

The VGG's main building block consists of a few convolutional layers followed by a max pooling layer. Many of such blocks are concatenated, where each one reduces the feature map size to a quarter, due to the pooling. Typically, the number of feature maps per convolutional layer is doubled each time the feature map size is decreased. This is done to counterbalance the decrease of the feature vector's size after the pooling. At the end of the network, some fully connected layers are used, which finally lead to the individual class outputs. Between the convolutional layers, ReLUs are used as non-linearities. Additionally, we use Batch Normalization (BN) [IS15] layers after the ReLUs for normalization. The final VGG architecture used in this thesis is summarized in Table 3.1. Many different options with a varying number of convolutional layers and therefore learnable parameters have been evaluated. Empirical experiments have shown that this architecture is a good compromise between accuracy and complexity for the used LPTR dataset. Each network input is an image crop of an LP with a height of 64 pixels and a width of 128 pixels. The output is an embedding of dimensionality 32. The tuning of this value is further discussed in Section 3.1.2.2.

3.1.1.2 ResNet Type

The ResNet has been designed for building very deep networks, with up to several hundred convolutional layers. The main idea behind ResNet is to use a building block which adds an identity (shortcut) connection in parallel to the regular convolutional layers, as a second path. The output of a block is the sum of both paths. This way, gradients can be back-propagated to the early blocks of the network effectively and do not vanish. This is because of the multivariable chain rule, which states that gradients reaching the same neuron from different paths are summed up. As a gradient remains unchanged along the identity function towards any preceding neuron of an earlier block, it can hardly vanish completely.
Let x_l be the activation after the lth block. The activation after the $(l+1)$th block is then given by $x_{l+1} = x_l + f(x_l)$, where $f(x_l)$ represents the function learned by the convolutional layers. Reordering the equation yields $f(x_l) = x_{l+1} - x_l$, which is equivalent to the difference (the residual) between the output and the input of the respective block. The shortcut connection makes the learning task for the network easier, as it can learn $f(x) = 0$, which effectively ignores the entire block. ResNet uses a ReLU activation function and BN layer after every convolutional layer.
In the original paper, the authors applied the ReLU and BN at the end of a block, after the addition of the identity. This breaks the unaltered gradient flow between two consecutive shortcut connections mentioned above because the ReLU and BN are applied in between. In a later paper [HZRS16b], the same authors investigated an alternative

layer name	output shape	layer shape	parameters
conv0x	$64 \times 128 \times 32$	$[3 \times 3, 32] \times 2$	9568
maxpool0	$32 \times 64 \times 32$	2×2	0
conv1x	$32 \times 64 \times 32$	$[3 \times 3, 32] \times 2$	18496
maxpool1	$16 \times 32 \times 32$	2×2	0
conv2x	$16 \times 32 \times 32$	$[3 \times 3, 32] \times 2$	18496
maxpool2	$8 \times 16 \times 32$	2×2	0
conv3x	$8 \times 16 \times 64$	$[3 \times 3, 64] \times 2$	55424
maxpool3	$4 \times 8 \times 64$	2×2	0
conv4x	$4 \times 8 \times 64$	$[3 \times 3, 64] \times 2$	73856
maxpool4	$2 \times 4 \times 64$	2×2	0
fc0	128	512×128	65664
fc1	32	128×32	4128

Table 3.1: Structure of the used VGG type architecture. An entry in the *layer shape* column with content $[a \times a, b] \times c$ can be understood as following: A convolutional block of c individual convolutional layers, where each layer has b feature maps and a kernel width and height of a. In total, the network has 245632 weight parameters. The outputs are not individual class scores, but form a feature vector in the embedding space.

placement of the according ReLU and BN units. The ResNet building block resulting from the empirically established best placement is also used in this work and is visualized in Figure 3.1.

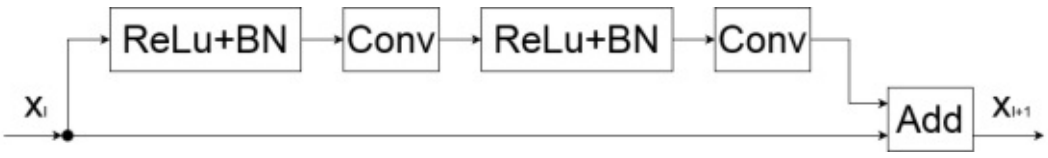


Figure 3.1: A schematic drawing of a single ResNet block. The upper path applies two convolutional layers. The lower path is the identity function. As only the upper path contains BN and ReLU units, gradients are propagated over the identity unaltered.

To reduce the size of a feature map within a block, it is not applicable to use the identity function anymore. To overcome this problem, a 1×1 convolutional layer with a stride of 2 is used instead, which effectively reduces the output size to a quarter. Notice that this in

fact alters the gradients over the shortcut connection, but only by a linear transformation. In the convolutional path of a block, the feature map size is reduced by applying the convolution using a stride of 2 in one of the layers. In large networks, several consecutive residual building blocks are used before the feature map size is reduced again. However, for our LPTR dataset, the architecture in Table 3.2, which reduces the output size within every residual block, has shown to be sufficient empirically.

layer name	**output shape**	**layer shape**	**parameters**
conv0	$64 \times 128 \times 32$	$3 \times 3, 32$	320
conv1x	$64 \times 128 \times 32$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 1$	18496
conv2x	$32 \times 64 \times 32$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 1$	19552
conv3x	$16 \times 32 \times 32$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 1$	19552
conv4x	$8 \times 16 \times 64$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 1$	57536
conv5x	$4 \times 8 \times 64$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 1$	78016
conv6	$4 \times 8 \times 128$	$1 \times 1, 128$	8320
avgpool	$2 \times 4 \times 128$	2×2	0
fc	32	1024×32	32800

Table 3.2: Structure of the used ResNet [HZRS16a] architecture. The entries of shape $\begin{bmatrix} a \times a, b \\ a \times a, b \end{bmatrix} \times c$ within the *layer shape* column represent c consecutive residual blocks. The conceptual shape of one such block is shown in Figure 3.1. a defines the width and height of each convolutional kernel and b the number of feature maps per convolutional layer. In total, the network has 234592 weight parameters. The outputs are not individual class scores, but form a feature vector in the embedding space.

3.1.2 Training Procedure

3.1.2.1 Choice of Training Algorithm

While considering all the algorithms discussed in Section 2.2, it has been decided to chose Arcface for this work because of its elegance, simplicity and effectiveness. The

biggest advantage of Arcface and its other softmax based cousins over other methods is their ability to compute the loss function from a single decoupled input. In contrast, triplet loss always requires two auxiliary reference inputs to the anchor input for this purpose, i.e. a triplet. The additional computational effort which is required to select good triplets make the algorithm less attractive. The supervised auto-encoder also needs the auxiliary reference input of the target class for loss computation. The reason why Arcface works so well without requiring any additional inputs is because the centers of all classes are implicitly stored in the weights of the last fully connected layer. Because the class centers are just parameters of the network, they adapt automatically during training and are always available as reference for loss computation.
As a baseline for comparisons regarding performance and the shape of the embedding space, a traditional softmax classifier has been used. Notice that the traditional softmax classifier enforces no constraints on the embedding space, except for class wise separability. Therefore, distinct classes can have arbitrarily small gaps in between them. For this reason, such an embedding space serves as a good basis for comparisons. When the shape of an embedding space is investigated, it is important to consider how large the enforced marginal gap between distinct classes was for training. This is because this parameter effectively controls the compactness of each class's distribution and thus its shape. For Arcface, the margin m is a single hyper-parameter. To guarantee fair conditions for the embedding space analysis, it is the only hyper-parameter which has been varied for the results discussed in Chapter 4.
To summarize, embedding spaces which are analyzed are obtained by using the Arcface algorithm with different margin settings. Additionally, the embedding space resulting from a traditional softmax classifier training is investigated for comparison.

3.1.2.2 Hyper-parameter Tuning

There are several training hyper-parameters which need to be tuned before conducting more careful experiments. This is done to improve performance and to avoid spending too much time on the analysis of settings which are later discarded. Hyper-parameters for training an embedding space that are independent of the training algorithm are the embedding space's dimensionality d, the network architecture and the optimizer. Specific to Arcface are the scale s and the margin m.
The numerical hyper-parameters have been adjusted by independent exhaustive searches, each one with the other hyper-parameters fixed. The metric used for measuring the quality of a specific configuration is its resulting mean accuracy score on the test set, further explained in Section 3.3.1. A parameter wise exhaustive search has no guarantees for yielding the optimal configuration, but heavily reduces complexity compared to a full exhaustive search. The complexity of an independent exhaustive search which iterates over x values for each of the n hyper-parameters is $O(nx)$. In contrast, the full exhaustive search, which enumerates all possible inter-parameter combinations, has a complexity of $O(x^n)$, due to the curse of dimensionality. This makes the full exhaustive search computationally intractable, even for small n. This is because trying just a single configuration is very expensive, as it requires training and evaluating the underlying

neural network.
For hyper-parameters whose scale is bounded by a specific range, the x sampled values are picked equally spaced apart on a linear scale. An example for this is the margin m, which must lie within the range $[0, \pi]$. However, if a hyper-parameter can become arbitrarily large or if there is great uncertainty about the upper or lower bound of its range, choosing the n samples equally spaced apart on a logarithmic scale is a better option. Consider a scenario, where it is uncertain if a good candidate value lies within the range $[1, 10]$, $[1, 100]$ or $[1, 1000]$. If samples are picked from a linear scale, the search will not sample two out of the three ranges effectively, independently of the chosen distance between two consecutive samples. For example, a sampling distance of 10 would give 10 samples for the range $[1, 100]$, but only 1 sample for the range $[1, 10]$ and a total of 100 samples within the range $[1, 1000]$. Thus, a logarithmic scale with base 10 is required to sample each interval equally, in this scenario. On the logarithmic scale, the samples are picked equally spaced apart and are then converted back to the linear scale via exponentiation. Once an approximately good value has been found with this method, an additional search on the linear scale can be carried out on the surrounding sub-interval, to fine tune the hyper-parameter. Logarithmic hyper-parameter sampling has been used for tuning the scale s and embedding space dimensionality d.
Many more sophisticated approaches for hyper-parameter tuning exist. One such technique is Bayesian optimization, which is explored visually by Agnihotri et al. [AB20]. However, such techniques are not a key aspect for this work, which is why they are not further discussed.

3.2 Classification in the Embedding Space

The purpose of training the embedding space is that the subsequent classification can also be applied on the TECs and unknown classes. The natural question which arises is which kind of classifier should be used. Independently of the choice, the goal is to build the respective model, given training samples x_j for each class j. Notice that in this context training sample does not refer to the training of the embedding space. Instead, it solely refers to the computation of an input's respective embedding, which is then used for computing the parameters of the classifier.
One straight forward approach for classification in the embedding space is the k-Nearest Neighbors (kNN) classifier. However, kNN scales poorly for an increasing number of classes or samples, as it requires to store every single sample in a database for reference. For this reason, it is not a serious option for LPTR because performance and scalability are important. Another method is classification based on the nearest class center. Here, the distance to each center is measured according to a metric which does not consider variance, e.g. the Euclidean distance. This approach has been commonly used by recent work, such as Arcface. Other examples are given in Section 2.2. This type of classifier only guarantees to yield the correct result under the open set constraint if all classes share the same variance along all dimensions, as explained in Section 1.5.2. To test if this condition holds and to see if performance benefits from a more complicated classifier,

a multivariate Gaussian classifier is evaluated in addition to the nearest center classifier in this work. The intuition behind this is that a Gaussian is more expressive and allows to model a distribution in the shape of a hyper-ellipsoid. One Gaussian is used for each class.
Each class should receive a normalized confidence score p_j as output, as opposed to a binary decision output. This is also necessary for defining a threshold value for unknown classes. A classifier should also be capable of measuring similarity sim and distance $dist$, as both can be used for thresholding. Converting from distance to similarity and vice versa is simple. It is convenient to define the final class confidences in terms of softmax, as many similarities represent unnormalized log likelihoods. However, as the range of similarity values can vary for different classifiers, it is practical to implement a scale parameter s, as discussed in Section 2.2.1.3. The final score p_j for each class j out of N classes is then given by Equation 3.1.

$$p_j = \frac{e^{s \cdot sim_j}}{\sum_{k=1}^{N} e^{s \cdot sim_k}} \tag{3.1}$$

3.2.1 Cosine Classifier

For Arcface, the angular distance is the metric of choice for the nearest center based classifier. This is because points lie on the manifold of a hyper-sphere during training. The angular distance essentially captures the same information as the cosine similarity and converting between these two metrics is straight forward. The classifier's only parameter is the center of each class, which is calculated as the mean μ_j of its respective training samples x_j. The cosine similarity sim_{φ_j} for a test input x is then given by Equation 3.2.

$$sim_{\varphi_j} = \frac{\mu_j \cdot x}{\|\mu_j\| \cdot \|x\|} \tag{3.2}$$

Converting this value to the according angular distance can be done with Equation 3.3. Notice that it is almost equivalent to Equation 2.9, which is used for training. The only difference is that instead of the weights W_j, which are explicitly stored for the TRCs during training, the class centers μ_j are used. This allows to compute class scores for the TECs as well.

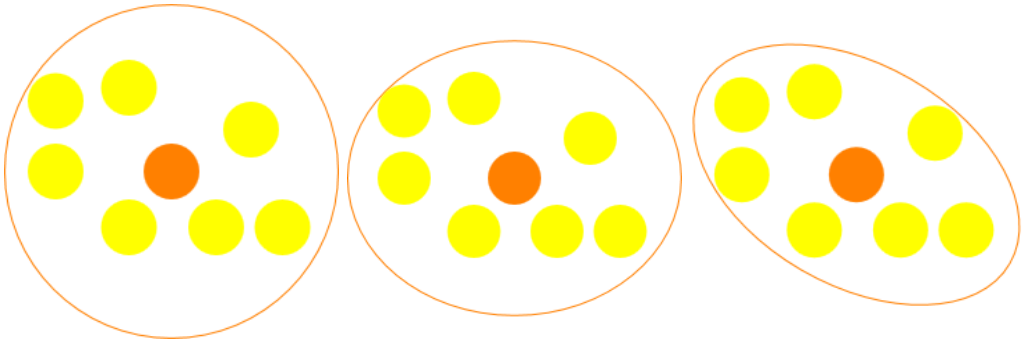
$$dist_{\varphi_j} = arccos(sim_{\varphi_j}) = arccos\left(\frac{\mu_j \cdot x}{\|\mu_j\| \cdot \|x\|}\right) \tag{3.3}$$

A good value for the scaling factor s is one which is similar to the value used during training with Arcface. This is because the underlying computation of the confidence scores using the cosine classifier is the same as during training.

3.2.2 Gaussian Classifier

Geometrically, the iso-contours of a multivariate Gaussian are equivalent to hyper-ellipsoids. Essentially, a Gaussian models different amounts of variance along orthogonal axes or directions. This means that every direction is its own linearly independent

dimension. Although the axes are all orthogonal, they do not necessarily have to be aligned with a Cartesian reference coordinate system, but can be arbitrarily rotated, as in Figure 3.2c. Two special cases of Gaussians are same variance along all dimensions and arbitrary variances which are aligned with the reference coordinate system. These cases are shown in Figure 3.2a and Figure 3.2b, respectively.



(a) Spherical Gaussian (b) Diagonal Gaussian (c) Full Gaussian

Figure 3.2: Three different bivariate Gaussians obtained from the same samples. The orange dot shows the mean while the orange circle represents the covariance matrix of the Gaussian. The yellow points show the samples from which the Gaussian is obtained. In (a), the variance is constrained to be equal in all directions. In (b), the axes of the largest variances are aligned with the axes of the reference coordinate system. (c) shows the general case without constraints.

3.2.2.1 Geometric Interpretation of the Mahalanobis Distance

The PDF of the multivariate Gaussian $X \sim \mathcal{N}(\mu, \Sigma)$ is given by Equation 3.4, where Σ represents the covariance matrix, d the dimensionality and the μ the mean vector.

$$p(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) \tag{3.4}$$

As it is important to understand the meaning of this equation for the analysis of the embedding space, an intuitive geometric interpretation is given in this section.
Let us start by considering the PDF of the univariate Gaussian $X \sim \mathcal{N}(\mu, \sigma^2)$, which is given by Equation 3.5.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{3.5}$$

The part of the exponent given by Equation 3.6 corresponds to a transformation to the standard normal distribution $X \sim \mathcal{N}(0, 1)$ and thus measures how many standard

deviations away x is from μ. The value $dist(x)$ is also known as the *z-score*.

$$dist(x)^2 = \left(\frac{x-\mu}{\sigma}\right)^2 \tag{3.6}$$

For the multivariate Gaussian, this distance computation must be generalized to multiple dimensions. To do so, the Euclidean distance metric is used. Let us consider the diagonal covariance matrix (Figure 3.2b) first and generalize to the full Gaussian afterwards. For a diagonal Gaussian, the Euclidean distance from the mean is given by Equation 3.7, where Λ represents a diagonal matrix with the variances σ_i^2 as its entries.

$$dist(x)^2 = \sum_{i=1}^{d} \left(\frac{x_i-\mu_i}{\sigma_i}\right)^2 = (x-\mu)^T\Lambda^{-1}(x-\mu) \tag{3.7}$$

This equation already looks quite similar to part of the exponent in Equation 3.4. At this point, the only thing missing for a full Gaussian is an arbitrary orientation of its axes relative to a fixed reference coordinate system. This can be implemented by rotating x, using the orthonormal rotation matrix Q. More precisely, Q defines the orientation of the full Gaussian's axes in the reference coordinate system. Therefore, $y = Q^{-1}(x-\mu)$ effectively transforms x to the Gaussian's local coordinate system. This transformation simplifies the full Gaussian $X \sim \mathcal{N}(\mu, \Sigma)$ to a diagonal Gaussian $Y \sim \mathcal{N}(0, \Lambda)$. If this rotation is incorporated into Equation 3.7, it can be simplified to Equation 3.8, given that $Q^T = Q^{-1}$ for an orthonormal matrix.

$$dist(x)^2 = (Q^T(x-\mu))^T\Lambda^{-1}(Q^T(x-\mu)) = (x-\mu)^T(Q\Lambda^{-1}Q^T)(x-\mu) \tag{3.8}$$

This distance is also known as the Mahalanobis distance $dist_M$ and is the multivariate equivalent to the z-score. Thus, it also answers the question how many standard deviations away a point lies from the mean.
The expression $Q\Lambda^{-1}Q^T$ can be summarized as a single matrix Σ^{-1}. Its inverse $\Sigma = (Q\Lambda^{-1}Q^T)^{-1} = Q\Lambda Q^T$ corresponds to the Eigendecomposition of a symmetric matrix. In fact, Σ is not only symmetric, but positive definite, as all variances must be positive in the non-degenerate case of a Gaussian. From the Eigendecomposition it follows that Λ represents both the variances and the eigenvalues.
A formal proof that Σ is equivalent to the (sample) covariance matrix and not any other positive definite symmetric matrix is omitted here. It can be proven by a maximum likelihood estimator for Σ, as done by Magnus and Neudecker [MN19], for example. Intuitively it is not surprising, after all the covariance matrix measures all pair wise linear correlations between the variables. These can also be described by a single rotation from the reference variables' original coordinate system.
Substituting Σ^{-1} into Equation 3.8 yields Equation 3.9, the general formulation for the Mahalanobis distance, as it occurs in the exponent of Equation 3.4.

$$dist_M(x)^2 = (x-\mu)^T\Sigma^{-1}(x-\mu) \tag{3.9}$$

The purpose of the normalization divisor $\sqrt{(2\pi)^d|\Sigma|}$ in Equation 3.4 is to ensure that it integrates to 1 over $\mathbb{R}^d$. This is a formal requirement for all PDFs. It can be intuitively understood by a Riemann sum approximation of the integral, which ensures that the sum of the probabilities of all infinitesimal small intervals equals 1.

3.2.2.2 Gaussians for Classification

To use Gaussians for classification in the embedding space, each class must be represented by its own multivariate Gaussian distribution. Its PDF effectively measures similarity, as a larger value corresponds to being closer to the center. In practice, the natural logarithm of the PDF is nicer to work with because it simplifies the calculation. This is illustrated in Equation 3.10.

$$ln(p(x)) = -\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) - \frac{1}{2}ln|\Sigma| - \frac{d}{2}ln(2\pi) \tag{3.10}$$

This equation could be used directly for computing the similarity score of each class. However, the last term is constant for the Gaussians of all classes and therefore can be omitted, yielding Equation 3.11 for the similarity score of class j.

$$sim_j(x) = -\frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j) - \frac{1}{2}ln|\Sigma_j| \tag{3.11}$$

Using the log likelihood also allows to multiply the similarity score with the scale factor s before applying the softmax. The scaling is particularly important for high dimensional Gaussians because the log PDF values approach negative infinity with increasing dimensionality. To counterbalance this, s needs to have rather small values $s \ll 1$ for Gaussians.
The maximum likelihood estimator for the covariance matrix Σ of a full Gaussian is given by Equation 3.12. It measures all unnormalized pair wise correlations based on M samples x_i. μ is the sample mean of all x_i.

$$\Sigma = \frac{1}{M-1}\sum_{i=1}^{M}(x_i-\mu)(x_i-\mu)^T \tag{3.12}$$

This general case is the most promising choice because class variances can neither be expected to be equal in all directions nor to be aligned with the embedding space's reference coordinate system. However, the former, i.e. a spherical Gaussian, is still useful for two reasons. Firstly, it captures the average variance of a class and thus allows to compute how strongly the average size varies between different classes. Secondly, it serves as a baseline for performance comparisons with the full Gaussian. The spherical Gaussian's performance is bounded by the performance of the full Gaussian if the latter fits the true data distribution of a class more accurately. The maximum likelihood estimator for the scalar variance σ of the spherical Gaussian is defined by Equation 3.13. This formula effectively computes the variance for each dimension and averages over all

dimensions d and samples M.

$$\sigma = \frac{1}{(M-1)d} \sum_{i=1}^{M} (x_i - \mu)^T (x_i - \mu) \tag{3.13}$$

To define a non-degenerate Gaussian of dimensionality d, $d+1$ linearly independent samples are required. To understand this intuitively, imagine a straight line in $2D$ space. It can be described by any point on the line plus another point for specifying its direction. Notice that the number of points which are required to define the line does not change, independently of the dimensionality $D \geq 2$ of the space it lies in. A line will still be a line in $3D$ or any higher dimension. The only thing that changes is the number of coordinates the two points consist of. A $2D$ plane in $3D$ can be defined by a 3rd additional point, which fixes the orientation of the plane. This observation can be generalized to hyper-planes by induction, showing that a d dimensional hyper-plane can be described by $d+1$ independent points. This property is independent of the dimensionality $D > d$ in which the hyper-plane is placed.
A Gaussian behaves the same way as the hyper-plane in this scenario. For example, if only 3 independent samples are available to specify a $3D$ Gaussian, there will only be variances in 2 different directions instead of all 3 if the orientation of the Gaussian is chosen accordingly. In general, if $M < d+1$ independent samples are used to define a d dimensional Gaussian, there will only be $M-1$ distinct eigenvalues or variances. In this case, Σ does not have full rank and the PDF does not exist, as there is no Σ^{-1}.
This property makes it more challenging to use the full Gaussian for classification in high dimensional embedding spaces. As the required dimensionality of the embedding space grows with the number of distinct classes, this problem becomes more troublesome for large datasets. A way to mitigate it, is to use data augmentation to compute several embedding space samples from a single input. The spherical Gaussian does not suffer from this scaling problem, as it is well defined by two independent samples, regardless of its dimensionality.

3.2.2.3 Gaussians and Metric Spaces

Notice that the embeddings are interpreted in angular space during training using Arcface, but a multivariate Gaussian operates in Euclidean space by definition. Therefore, the distance of a point x from the embedding space center matters for a Gaussian. For this reason, the unnormalized values x are used as inputs for computing its parameters instead of the normalized ones.
An alternative option would be to transform the normalized embedding space coordinates of x into hyper-spherical coordinates. For a d dimensional vector, this would yield $d-1$ angles plus the radius r, which is constant due to the normalization. Computing a Gaussian based on these angular values would eliminate the curvature, introduced by defining points on the hyper-sphere's surface in Euclidean coordinates. However, the transformation would cause distortions around the poles of the hyper-sphere. To illustrate this, consider the map of our globe on a flat piece of paper, as the result of a Mercator

projection. It will make the north and south pole appear huge, while the size of Africa is heavily underrepresented. In fact, there is no projection of a sphere onto the flat plane which does not cause distortions on the global level.
Therefore, interpreting the unnormalized embedding space values directly as Euclidean coordinates is an effective option. This is not a restriction, as a classification problem which is separable solely based on angular distances must also be solvable using the respective Euclidean distances. In fact, the classification can even become easier when interpreting the unnormalized embeddings in Euclidean space, as it effectively uses the radius as additional parameter.

3.3 Evaluation Metrics

There are two types of evaluation metrics which need to be distinguished in this work. Performance benchmarks measure results in terms of accuracy on the dataset, whereas embedding space metrics deal with the structure and properties of the embedding space after training with the dataset. This section describes the theory and motivation behind various metrics, which can be applied independently of the dataset. Specific results for these metrics are discussed in Chapter 4 and are based on the used LPTR dataset.

3.3.1 Performance Benchmarks

All performance benchmarks measure which data is classified correctly under which conditions in some way. How exactly this is done depends on the specific metric. In this work, each performance metric is not just applied once on the entire dataset, but four times on different subsets. This is done to distinguish between performance on the TRCs and TECs more carefully. The easiest category is performance solely on the closed set TRCs. The supplementary measurement is based only on the TECs. The third subset measures performance on both TRCs and TECs. The final category only takes TEC instances as inputs, but allows to classify each input as either a TRC or a TEC. This may seem bizarre at first glance. Why would one want to classify an input as an object which is known not to be part of the dataset? The purpose of this is to see how well TRCs and TECs can be distinguished in the embedding space. If their class distributions don't share any overlaps, the performance of classifying TECs as TECs only and TECs as TRCs or TECs should be the same.

Accuracy is one of the most important metrics and measures the percentage of the correctly classified data independently of the confidence in the classification. In its traditional formulation, every sample of the dataset receives equal weight. This can lead to biases, in cases where the number of samples for different classes varies strongly. For example, consider a ternary classification problem where a single class covers 90% of all samples. A classifier which always outputs this class will have 90% accuracy, although it

might not compute anything. Mean accuracy overcomes this problem by computing the accuracy of each class individually and then averages over all classes afterwards. This way, each class receives the same total weight, independently of its number of samples. In the example above, the mean accuracy would equal $\frac{1}{3} * (90 + 0 + 0) = 30\%$ because the classifier never predicts the other two classes. As the number of samples per class varies quite heavily for the LPTR dataset used in this work, mean accuracy is always evaluated instead of accuracy.

Confusion Matrix is the natural multi-class generalization for the distinction between true positives, false positives, true negatives and false negatives. For each class, the confusion matrix captures how many instances of another class were classified as that class, hence the matrix. In the context of TRCs and TECs it is particularly interesting because it allows to inspect the proportions of mis-classifications between TRCs and TECs. For example, if the training has not generalized optimally, TRCs can be expected to be confused with TECs more often than with other TRCs. This is because the network has been explicitly trained to distinguish TRCs from TRCs but not TRCs from TECs. One major disadvantage of the confusion matrix is that it scales quite poorly because N distinct classes result in N^2 entries. This makes it intractable for human inspection for thousands of classes or more.

Precision Recall Curve is a plot which shows the relationship between precision and recall for different confidence thresholds. For low thresholds, the recall is highest and the precision lowest, as no predictions are discarded. The opposite is true for very high thresholds because only predictions which receive a high confidence are believed to be correct. For multi-class problems, each class has its own Precision Recall Curve (PRC). However, a single PRC can be computed by averaging over all classes.

Mean Average Precision (mAP) is one of the most popular scalar performance benchmarks for object classification. It can be interpreted as an aggregation of several PRCs because it effectively computes their average area under the curve. More precisely, the Average Precision (AP) of a specific class is calculated by integrating the class's precision over all respective recall values. mAP then computes the mean of the APs over all classes N, as given by Equation 3.14. p_j and r_j represent precision and recall of class j, respectively. In practice, the AP is computed numerically, e.g. by sampling the precision for a number of recall values which are equally spaced apart.

$$mAP = \frac{1}{N} \sum_{j=1}^{N} AP_j = \frac{1}{N} \sum_{j=1}^{N} \int_0^1 p_j(r_j) dr_j \tag{3.14}$$

Distinction between Known and Unknown Classes is about measuring how many percent of the instances of known classes and unknown classes can be identified as known and unknown, respectively. Essentially, this OSR problem corresponds to a

binary classification task. The accuracy depends on the confidence threshold below which inputs are considered as belonging to an unknown class. A low threshold will result in a high recall for known classes, but a small recall for unknown classes. The opposite is true for a high threshold. The optimal threshold value maximizes the mean accuracy of the underlying binary classification problem.
The thresholding can also be performed based on distances [GHC20]. The advantage of a distance threshold is that it is independent of the distances to all classes but the closest. As the confidence is indirectly based on the distances to all known classes, its value can be misleading. In a nutshell, a larger minimal distance can score a higher confidence than a smaller minimal distance. This is because the confidence solely depends on the ratio between the computed similarity scores of all classes.
Consider the following binary classification example which illustrates this. In scenario A, the distances have the values $dist_1^A = 5$ and $dist_2^A = 10$. If similarity is computed as the reciprocal of the distance $sim = dist^{-1}$, the larger confidence score equals $p_1^A = \frac{5^{-1}}{5^{-1}+10^{-1}} = \frac{2}{3}$. In scenario B, the distances equal $dist_1^B = 30$ and $dist_2^B = 90$, yielding $p_1^B = \frac{30^{-1}}{30^{-1}+90^{-1}} = \frac{3}{4}$ as largest confidence. This example illustrates that $\max p^B > \max p^A$, although $\min dist^B > \min dist^A$.
This alone does not necessarily explain a distortion if distinct classes are expected to have different distance distributions. For example, the larger distance values in scenario B could just be more likely to occur than the smaller distance values in scenario A. However, if all classes share the same distance distribution, this can be problematic. Therefore, distance thresholds sometimes are preferable over confidence thresholds. This is the case for the Mahalanobis distance of the multivariate Gaussian, which is further discussed in Section 3.3.2.
Note that in this work the OSR performance is only evaluated using the entire dataset. This means that instances of TRCs, TECs and unknown classes are used as inputs for evaluation. This is done because it mimics the LPTR task in the real world most closely. Besides, the mean accuracy of the OSR is evaluated independently of the other metrics. This means that it is the only benchmark which also uses instances of unknown classes for evaluation. For all other metrics, the inputs are restricted to images of TRCs and TECs only.

3.3.2 Embedding Space Metrics

This second category of metrics measures properties of the learned embedding space. For most of these metrics, evaluation on two subgroups, i.e. only TRCs and only TECs is sufficient, as no performance benchmarks are measured. The purpose of this distinction is to see if and how the embedding space metrics differ for TRCs and TECs. If the network has learned to generalize features of TECs well enough, there should be no significant difference for the TRC and TEC measurements. The metrics are compared for different values of the margin hyper-parameter to get a deeper understanding of its effects.

Simple Angular Distance Statistics are used to compare the TRCs' and TECs' approximate average sizes in the embedding space based on the margin hyper-parameter. We introduce two benchmarks, the Mean Average Angular Distance (MAAD) and the Standard Deviation of the Average Angular Distance (StdAAD). To compute them, the Average Angular Distance (AAD) to the class center is computed for each class at first. Then, the MAAD is calculated by Equation 3.15, where N equals the number of classes, M_j the number of available samples of class j and x_{ij} the ith sample of the jth class.

$$MAAD = \frac{1}{N}\sum_{j=1}^{N} AAD_j = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{M_j}\sum_{i=1}^{M_j} dist_{\varphi_j}(x_{ij}) \tag{3.15}$$

The StdAAD measures the average deviation from the MAAD and is given by Equation 3.16. In other words, it can be used to see how strongly the sizes of different classes vary.

$$StdAAD = \sqrt{\frac{1}{N-1}\sum_{j=1}^{N}(AAD_j - MAAD)^2} \tag{3.16}$$

Plotting the MAAD and StdAAD for different margins for both TRCs and TECs separately allows to inspect how the average size of a class is affected by the margin. It is expected that the MAAD decreases with an increasing margin, as it makes the classes more compact. For a particular dataset, is unclear in advance how strongly the MAAD varies for TRCs and TECs and how the StdAAD changes for an increasing margin.
Notice that we have defined the MAAD and StdAAD specifically for angular distances. This is because Arcface learns based on angles. Defining both metrics based on angles therefore allows to evaluate the underlying distributions as seen by the training. Defining the MAAD and StdAAD for other metric spaces is straight forward.

Mean Distance PDF extends the angular distance statistics by not only considering the mean and standard deviation but the entire distribution. This distance distribution is computed as a continuous PDF, which is obtained from a limited number of samples using a Gaussian Kernel Density Estimation (KDE), given by Equation 3.17. M represents the number of samples, x_i the individual samples and h the standard deviation of the Gaussian, which controls the amount of smoothing.

$$p(x) = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{\sqrt{2\pi h^2}} exp\left(-\frac{1}{2}\frac{(x-x_i)^2}{h^2}\right) \tag{3.17}$$

The mean distance PDF is calculated as the average of the individual distance PDFs obtained from the test set of all TRCs or TECs. It cannot only be computed for the angular distance but also for the Euclidean distance or even the Mahalanobis distance if Gaussians are used.
The Mahalanobis distance PDF of a multivariate Gaussian is static and its properties are very counter-intuitive at first glance. This is because the PDF's mode is not located at the center like in the univariate case. Instead, the mode moves further and further away

from the center with increasing dimensionality. An intuitive explanation for this is that for a point the number of possibilities to be a certain distance away from the center grows with the size of the manifold at that distance. For example, for a $2D$ spherical Gaussian, there is an entire circle of infinitely many points which share the same distance towards the center. For a $3D$ spherical Gaussian, there is a whole sphere of points for every distance. In contrast, at the center there is only one point which has zero distance to the center, which is itself. The probability to sample a point which is a certain distance away from the center is the Gaussian's PDF value of any point at that distance multiplied by the size of the manifold at that distance.
To formulate this mathematically, let us consider this from a different perspective. We will start with standard multivariate Gaussian and generalize to arbitrary Gaussians afterwards. A d-dimensional standard multivariate Gaussian $X \sim \mathcal{N}(0, I)$ has no correlations and therefore its PDF can also be described as the product of the PDFs of d orthogonal standard univariate Gaussians $Y \sim \mathcal{N}(0, 1)$. The distribution over the Euclidean distances from the origin of X is then given by Equation 3.18 and is equivalent to the definition of the χ distribution.

$$\chi = \sqrt{X^T X} = \sqrt{\sum_{i=1}^{d} Y_i^2} \tag{3.18}$$

This means that the distance PDF for a standard univariate Gaussian is given by the according PDF of the χ distribution. A similar derivation which uses the χ^2 distribution instead of the χ distribution is given by Wang et al. [WSM15]. The χ distribution's PDF for k degrees of freedom and values $x \geq 0$ is given by Equation 3.19, where $\Gamma(z)$ represents the gamma function, given by $\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx$. The gamma function can be understood as the generalization of the factorial function over $\mathbb{R}$. The interested reader can find more detailed information about these functions in a book by Forbes et al. [FEHP11], which is specifically about probability distributions.

$$p(x) = \frac{x^{k-1} e^{-\frac{x^2}{2}}}{2^{\frac{k}{2}-1} \Gamma(\frac{k}{2})} \tag{3.19}$$

If the χ distribution's PDF is used as Euclidean distance PDF, x corresponds to a point's distance from the center and k to the dimensionality d of the standard multivariate Gaussian X. To generalize this to an arbitrary multivariate Gaussian distribution, the Mahalanobis distance of a point needs to be used as input for the PDF instead of the points' Euclidean distance. This is because the Mahalanobis distance distribution of any multivariate Gaussian is equivalent to the Euclidean distance distribution of the standard multivariate Gaussian of the same dimensionality. As explained in Section 3.2.2.1, for a Gaussian, the Mahalanobis distance measures how many standard deviations away a point is from its center. Furthermore, the Mahalanobis distance's iso-contours have the shape of hyper-ellipsoids. Therefore, its underling computations can also be viewed as a transformation from any multivariate Gaussian distribution to X. Some examples of the Mahalanobis distance PDF for different dimensionality values are plotted in Figure 3.3.

Notice that due to the transformation, this PDF looks exactly the same for all possible Gaussians which share the same dimensionality.

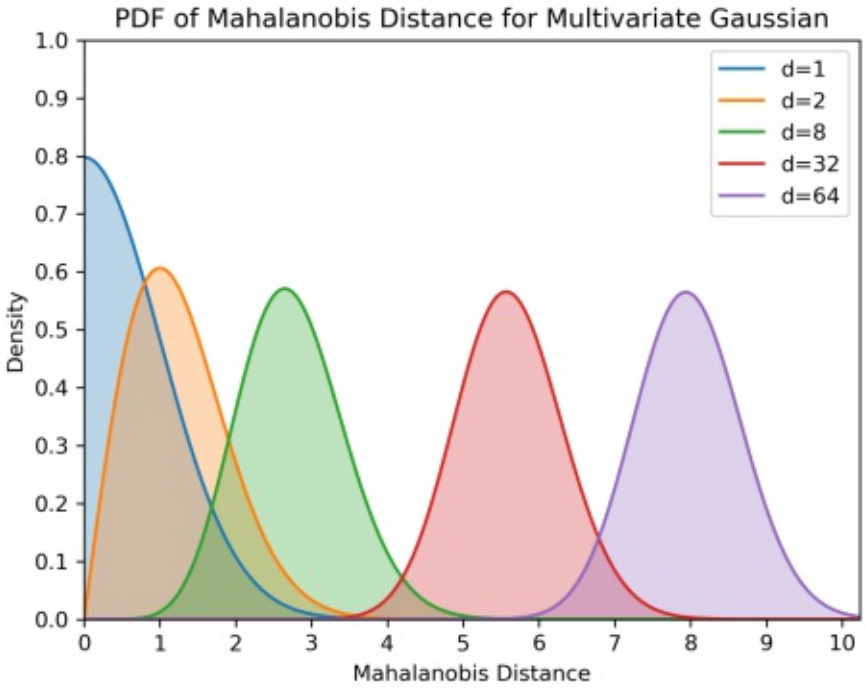


Figure 3.3: The Mahalanobis distance PDFs for multivariate Gaussian distributions with a different dimensionality d. As the dimensionality increases, the distribution approaches a univariate Gaussian with its center further and further away from the center.

PCA is closely related to a multivariate Gaussian distribution by definition. The Gaussian's major axes are aligned with the principal components of the data from which the Gaussian has been estimated. One way to view principal components is that they capture the directions of maximal variance, which is exactly what a Gaussian does. The alternative viewpoint on PCA describes the coordinate system of a linear transformation, which is defined by the principal components, and leads to the minimal linear reconstruction error if a number of components are discarded. In this work, there is no interest in reducing dimensionality using PCA. If this was necessary or desired, it could be done directly during training of the embedding space. This would leave even greater flexibility, as the reduction would not be restricted to be linear.
Instead, the subject of interest is solely the distribution of the standard deviations along the principal components. If viewed as the major axes of a multivariate Gaussian, principal components measure how curved or spherical its corresponding hyper-ellipsoid is. Calculating the principal components is equivalent to computing the major axes of a

Gaussian and their lengths. More mathematically, the variances $diag(\Lambda)$ along all major axes Q need to be calculated. As explained in Section 3.2.2.1, these can be obtained from the Eigendecomposition of the covariance matrix $\Sigma = Q\Lambda Q^T$. The respective standard deviations are then computed as $diag(\sqrt{\Lambda})$. After sorting, these values can be plotted against their rank. The resulting figure can then be interpreted in two ways. The first viewpoint simply shows the lengths of the true distribution's principal components. Alternatively, the elongation of the hyper-ellipsoid corresponding to the Gaussian estimate of the true distribution is visualized. To retrieve a single plot which consolidates this for all TRCs or TECs, the sorted standard deviations of each distribution can be normalized and then averaged over all classes.

Similarity of Estimated Class Distributions to Gaussians determines how well the test data distributions fit the respective Gaussian estimates, which are based on the training data.
For this purpose, the mean Mahalanobis distance PDF estimated from a test set is particularly interesting because the PDF it is expected to match is known, as explained above. Therefore, it is possible to determine if there is a statistically significant difference between the distribution given by test data and the one to be expected. It must be noted that even if there is no statistically significant difference, this does not imply that the test data follows a Gaussian distribution. In this case, a test for multivariate normality has to be applied in addition. The first test alone is not enough for verifying that the test set data follows the estimated Gaussians because the Mahalanobis distance PDF looses representative power. This is due to its reduction from the Gaussian's real dimensionality d to a single dimension. However, if there is a statistically significant difference, it implies that the test data does not follow a perfect multivariate Gaussian distribution.
Notice that even if a statistically significant difference has been confirmed, this does not imply that this difference is important. In this context, it merely states that it is extremely unlikely that the computed Mahalanobis distances of the test set originate from the estimated distribution. Statistical significance makes no statement about how different two distributions are from each other, only that they are unlikely to be identical. For example, if there is zero uncertainty about the shape of two almost identical distributions, there will still be a statistically significant difference between them.
In cases where a statistically significant difference has been confirmed, it is still interesting to see how different the according distributions are. There are many different ways to measure similarity between two distributions. Cha [Cha07] gives a great comprehensive overview of different options for measuring similarity using distances. In this work, the Jensen-Shannon Distance (JSD), which is a statistical distance metric, has been used. For a more rigorous description of the JSD than provided here, please refer to Briet et al. [BH09].
Two nice properties of the JSD are that it is always well defined and that its base 2 interpretation lies within the range $[0, 1]$. The latter allows to directly convert it to a similarity score by computing $sim_{JSD} = 1 - JSD$. The JSD is an extension to the Kullback-Leibler Divergence (KLD) and enforces symmetry to obtain a mathematically

valid metric space. The KLD essentially describes the average amount of redundant information contained within the encodings that define a message, compared to the theoretical minimum of the average amount of information required for each encoding. This minimum is is also known as entropy. For this reason, the KLD is commonly referred to as relative entropy, as it describes the additionally required information due to its non-optimality. In our context, there are no discrete encodings, but continuous distributions instead. Defining the KLD in terms of distributions is a generalization, as the length of each encoding represents a probability. Typically, specific encodings are derived from the underlying probability distribution. The KLD is given by Equation 3.20, where P and Q represent the continuous random variables of the true PDF and the estimated PDF, respectively.

$$KLD(P||Q) = \int_{-\infty}^{\infty} p(x) log \frac{p(x)}{q(x)} dx \tag{3.20}$$

Using this, the JSD is then defined by Equation 3.21 and can be understood as a symmetric, smoothed and averaged version of the KLD.

$$JSD(P||Q) = \sqrt{\frac{KLD(P||M) + KLD(Q||M)}{2}} \quad | \quad M = \frac{P+Q}{2} \tag{3.21}$$

The similarity score computed based on the JSD is only an approximation of the true similarity between the test set's distribution and the estimated Gaussians. Again, this is because the similarity is computed based on the 1D Mahalanobis distance distribution as opposed to the full d dimensional Gaussian distribution. The major advantage of this dimensionality reduction is that the resulting 1D PDF can be visually interpreted by a human. This allows a human to estimate similarity visually without computing a similarity score. Furthermore, the overall shape of the distance distribution reveals insight about the overall structure of the true distribution. For example, if several modes exist, it implies that the true distribution consists of more than one compact cluster.

Separability of Distinct Classes assesses whether or not any overlaps exist between the distributions of different classes in the embedding space. Assuming that each class can be described by a Gaussian, this question is equivalent to asking if the classes' respective hyper-ellipsoids overlap. Theoretically, all Gaussians spread infinitely far in all dimensions they cover. However, a confidence ellipsoid which contains $p < 100\%$ of a class's data can be computed. This is also known as the Standard Deviational Hyper-Ellipsoid (SDHE) and is described with great detail by Wang et al. [WSM15]. In a nutshell, the SDHE can be derived by considering the χ distribution of the Mahalanobis distance, as explained above. More precisely, the inverse CDF of the χ distribution can be used to compute the Mahalanobis distance threshold above which $p\%$ of the Gaussian's data is contained. Multiplying this value by the standard deviations $diag(\sqrt{\Lambda})$ of the Gaussian's major axes yields the lengths of the $p\%$ confidence SDHE axes.
Calculating the smallest distance between two hyper-ellipsoids is a non-linear problem. An iterative algorithm based on local approximations via hyper-spheres is given by

Lin and Han [LH02], for example. For N classes, $O(N^2)$ pair wise shortest distances exist. This makes the problem intractable for large datasets, as the smallest distance computation needs to be done for each pair of Gaussians. One way to mitigate complexity is to approximate each hyper-ellipsoid by a hyper-sphere whose radius is equivalent to the length of the hyper-ellipsoid's axis of median length. The median is used instead of the mean because it is robust to extreme values. Another advantage of this method is that it captures an approximation of the true distribution, which is not directly based on the assumption that the true distribution follows a Gaussian. Calculating the distance between the centers of two hyper-spheres A and B is as trivial as $D_{AB} = \|\mu_A - \mu_B\|$, where μ presents the center of a hyper-sphere. All pair wise center-to-center distances can be summarized in a symmetric matrix D. Notice that $diag(D) = 0$, as all hyper-spheres do not have any distance to themselves.
We propose a metric called Gap/Radius Ratio (GRR) (Equation 3.22), which measures how filled or empty the embedding space is by the distributions of a number of classes. r_j refers to the radius of the hyper-sphere approximation of the jth class. Essentially, the GRR measures the average ratio between the distance between the centers of two classes and the sum of their respective radii. A GRR of 1 states that the average gap between two classes is zero. In other words, the hyper-sphere approximations of the classes just barely touch each other. GRR values below 1 indicate that the hyper-sphere approximations on average are more likely to overlap with each other than to lie apart from each other with a gap in between them. For GRR values above 1, the opposite is the case. Notice that the GRR is bounded by the range $[0, \infty)$.

$$GRR = \frac{1}{\binom{N}{2}} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{D_{ij}}{r_i + r_j} \tag{3.22}$$

It must be noted that the GRR is not a precise measure for embedding space emptiness, as it approximates the classes' true distributions by hyper-spheres. The GRR as introduced in Equation 3.22 only yields an approximation for the true value. This is because an overlap between two distinct classes cannot be captured if it occurs outside of the bounds given by the respective hyper-sphere approximations. Similarly, an overlap between two classes which does not really exist is captured if the two classes do not stretch out as widely as their hyper-sphere approximations in the overlapping region.

3.4 Saliency Map Extensions

Vanilla saliency maps and Grad-CAM have been used for investigating the classification decisions of the trained network, as explained in Section 2.3. In this section, we first introduce an extension which also allows to compute saliency maps for TECs. Then, we argue why an average saliency map per LPT is preferable over an individual saliency map per image.

3.4.1 TEC Saliency Maps

All saliency map techniques described in Section 2.3 require the ability to compute the logits of the classes of interest. Gradient-based methods additionally need to be differentiable because gradients need to be back-propagated to create a saliency map. Another advantage of training an embedding space with a softmax extension like Arcface is that saliency maps for TRCs can be computed out of the box. This is because the necessary outputs for the logits already exist, as they are required for training. Notice that this is not the case for techniques like triplet loss.
We propose a simple method for computing gradient-based saliency maps for TECs. As explained in Section 2.2.1.1, each Arcface row vector W_j of the last fully connected layer can be interpreted as the class center of the jth TRC in the embedding space. One way to compute saliency maps for TECs as well is to use such a weight vector for every TEC too. The class center of every TEC can be computed empirically based on the available training data. In this context, empirically refers to computing the respective embeddings of all samples of a TEC and averaging the obtained values to retrieve the center. Notice that these calculations only require the forward pass of the network. The weights of the network are not altered in any way by this procedure. The computed centers can then directly be used as weights for the last fully connected layer in order to create TEC saliency maps. Both, extending the original fully connected layer encoding the TRC centers and replacing it by the TEC centers is possible.
Notice that the proposed method is specifically designed for both Arcface and the usage of the cosine classifier. Theoretically, it is also possible to compute the logits using the Gaussian classifier and Equation 3.11, as it is also differentiable. However, in this case the logit calculations can no longer be represented by a single fully connected layer after the embedding layer. This is because each logit computation is more complex than a single dot product for the Gaussian classifier.

3.4.2 Average Saliency Maps

In general, saliency maps of different images need to be considered individually and independently of one another. This is because one class can have very distinct appearances. For example, an image of a cat could show it from the front or from the side. Depending on the view, a cat looks completely different, but nevertheless both images belong to the same class. Thus, combining two such images by an overlay does not make sense in general, as pixels at corresponding locations can represent very different features.
However, in the world of LPTs there is not a lot of diversity because different instances of the same LPT are almost identical by definition, except for their identifying characters and other neglectable personalizations. To determine the general behavior of the network regarding a specific LPT, it is a good approximation to overlay the saliency maps of all its available samples to compute the average for every pixel. Obviously, this requires that all images of an LPT have the same size. If the saliency map of a single image is considered individually, there is a danger of over-fitting due to occlusions or noise, for

example. The average saliency map smooths out such variances to some degree. Besides, it heavily reduces the number of saliency maps which need to be inspected by a human. Instead of having to look at one saliency map per image, there is only one per LPT.

CHAPTER 4

Results

This chapter discusses the results which have been achieved on the used LPTR dataset. At first, the LPTR dataset on which the experiments have been conducted and the experimental setup are described in Section 4.1. Afterwards, a solution for handling the identifying characters of the LPs during training is proposed in Section 4.2. Then, the results of various performance benchmarks and the associated embedding space metrics are given in Sections 4.3 and 4.4, respectively. The theory behind all used benchmarks is described in Section 3.3. Next, observations about what the network has learned during training are described based on the saliency maps of various LPTs in Section 4.5. Finally, the results are summarized and discussed in Section 4.6 and Section 4.7, respectively.

4.1 Experimental Setup

The results described in this chapter are all based on the ResNet architecture shown in Table 3.2. The overall conclusions regarding embedding space properties which have been obtained from the VGG architecture from Table 3.1 are equivalent. However, the performance of this architecture is slightly worse compared to the ResNet, which is why its results are not further discussed.
For training, the Adam optimizer introduced by Kingma et al. [KB14] has been used with a learning rate of 0.001. The other hyper-parameters have been tuned as described in Section 3.1.2.2. A good configuration for the used LPTR dataset is an embedding space dimensionality of 32 and a scale factor of 16. The margin hyper-parameter is kept variable for the benchmarks.
The LPTR dataset consists of 7500 annotated images of 128 distinct LPTs from the USA. The TRCs contain 69 of these LPTs and a total of 5800 images, while the TECs cover the remaining 59 LPTs and 1700 images. Notice that although the number of samples of the TRCs is much greater, the number of classes is not. This is because the LPTs of which most samples are available have been used for network training. This does not

introduce a more significant bias in the TRC selection than a random split between TRC and TECs would. Careful human inspection has shown that the LPTs of which most samples are available are distributed almost uniformly over both different states and the LPTs' main visual features, which a human would use for distinguishing between them. Additionally, 700 images of LPTs which are neither part of the TRCs nor the TECs have been used as unknown class instances. It must be noted that the LPTs which occur in the unknown classes belong to the same states of origin as the LPTs of the known classes. Therefore, many images within the unknown classes look quite similar to the images within the TRCs and TECs. Images of other arbitrary objects are not contained within the unknown classes. This setup has been decided because the adequate prior closely mimics the problem during real world operation. In a full LPR system, inputs are only passed to the LPTR subsystem if there is very little uncertainty that the inputs show an LP and not something else. The first task of the LPTR subsystem then is to decide whether or not the image shows a known LPT. Also, for a given region, unknown LPTs from that region are the most likely inputs for the respective system.
The used train/test set ratio is 70/30 for all classes. The dataset statistics are summarized in Table 4.1.

	Total	**TRC**	**TEC**	**Unknown Class**
#Classes	128	69	59	-
#Images	8200	5800	1700	700
#Training Images	5740	4060	1190	490
#Test Images	2460	1740	510	210

Table 4.1: Statistics of the used LPTR dataset.

The LPT images which are used as neural network inputs have a size of 128×64 pixels. This resolution is a good trade-off between preserving image detail and performance regarding computational speed. As the image quality of the used dataset is not the best, a much larger resolution would require upsampling anyway. More precisely, the original resolution of the average LP before resampling is about 20% larger.

4.2 Data Augmentation and Character Masking

4.2.1 Traditional Data Augmentation Techniques

Each data augmentation operation which is mentioned in this section is performed according to a random value within a given interval. Within these bounds, all values are equally likely. First of all, contrast and brightness modifications of $\pm 20\%$ are used. Next, a shift along the x and y axes by $\pm 10\%$ is implemented. Furthermore, rescaling is performed by zooming in or out of the image by $\pm 15\%$. To avoid artificial computation of pixels when zooming out of an image, for example mirroring the edge pixels, the LPTs are stored at a zoomed out scale. This way, all rescaling operations can be implemented as

inward zooms and no artificial data has to be generated for pixels outside of the original bounds. As a consequence, the mean zoom value within the augmentation interval is not 0, but a positive number, which depends on the maximal zoom-out and shift factor.
Other traditional augmentation techniques, such as different variants of noise or mirroring, are not used. The former has been tried, but has not lead to an increase in performance. Presumably, because the dataset already contains a lot of noise as it is. The latter does not make any sense, as no mirrored LPs can be found in the real world without legal consequences.
Data augmentation is also used to create several embeddings per training image in order to obtain more samples for computing the embedding space classifiers. This is particularly important for the Gaussian classifier, as it requires at least $d+1$ independent samples. In our setup this means that 33 samples are needed per class. Notice that the training set for some of the TECs only consists of a little more than 10 images. Consequently, estimating a Gaussian distribution for these classes would not be possible without data augmentation. In our experiments we create 10 embeddings per training image for computing the embedding space classifier's parameters.

4.2.2 Character Masking

Another augmentation strategy which needs to be considered is specific to LPTR and deals with the masking of the identifying characters of LPs. The idea behind masking is to avoid learning the presence of specific characters as decision indicators, as explained in Section 1.5.1. In this work, masking out a character refers to putting a rectangular bounding box of uniform color over it, such that the character is no longer visible. The used color is fixed and is set to approximately match the average pixel intensity over the entire dataset. The width and height and position of each bounding box are part of the annotated ground truth. The bounding boxes' sizes and positions are also varied randomly by a small amount during augmentation to further improve the generalization capabilities of the network.
The simplest strategy is to mask out all characters for every image during training. This gives quite good results on the test set, as long as the characters are also masked out for these images. In contrast, if the characters are not masked out for the test set, performance drops enormously. The opposite effects on the test set can be observed if no characters are masked out at all during training. This indicates that the masks/characters, depending on what has been used for training, are important features to the network and that they are required for many of its decisions. It is not surprising that the network learns the masks as reliable indicators, as they are very primitive, yet effective features. After all, a bounding box is perfectly described solely by its edges. For the used dataset, exploiting the number of bounding boxes in an image and their relative positions towards each other is enough to identify the underlying LPT, in some cases. Note that the used TRCs only include 69 distinct LPTs. For many more classes than this, it will become increasingly harder to distinguish between different LPTs based on their respective number of identifying characters and their positioning. This in turn forces the neural network to learn more complex features in order to be able to successfully distinguish

between the LPTs.
However, the goal is to avoid learning the shape of specific masks altogether, independently of the dataset size. Therefore, it should be verified that neither specific characters nor specific masks are the main features which are used to classify input images. One way to confirm this to some degree is to ensure that performance on the test set is equally good whether or not an LP's identifying characters are masked out. Notice that even if this is the case, it does not mean that the contents of the image regions containing the identifying characters or masks are not relevant at all. Instead, it merely implies that the network is able to handle both cases and therefore must have learned to generalize to different features in such regions to some degree. Using the presence of some abstract object at a certain position as feature is actually reasonable. For example, a human may use the information that any character is present at a specific location as an indicator for classification. In cases where two LPTs only differ by their numbers of characters, this is the only way to distinguish between them.
A simple strategy which has proven to be effective and efficient is to independently and randomly decide if masking is applied for each character of an LP during training. The probability of applying the masking has been set to 50%, such that keeping the original character and masking it out are equally likely. To generate even more data and boost generalization further, this procedure is applied every time an image is used as input for training, as opposed to once per image in total. Consequently, the same original image can be subject to many non-identical maskings, yielding different training inputs. The number of characters which are masked out for an LP follow a binomial distribution. This means that it is most likely that half of an LP's characters are masked out and rarest that all or zero characters are masked out.
This strategy is very effective because the network cannot rely on the presence of a specific character or mask at a certain position due to the randomization of the masking. Besides, it is very inefficient to learn the relative positioning between different masked characters, as there are too many different configurations. For example, there are $2^7 = 128$ possible masking options for an LP that contains 7 characters. Because of the random process, it is easier for the network to learn other features, which are shared amongst all inputs and thus are more robust.
As desired, using this strategy, there is no significant difference in the accuracy on the test set whether or not all identifying characters of the LPs are masked out. To verify this, the McNehmar test statistic has been computed, for which the null hypothesis could not be rejected for an alpha of 0.05. A brief introduction to this test is given by Adedokun et al. [AB12]. In a nutshell, it can be used to compare the accuracies of two paired classification results based on a contingency table. Each sample can either by classified correctly by one of the classifiers, none of them, or both of them. These four possibilities make up the entries of the 2x2 contingency table. For the null hypothesis, the McNehmar test uses the assumption of marginal homogeneity for the table, which implies that both classifiers have equal accuracy. The p-value can then be computed based on a chi-square test statistic. For example, the p-value for using the Gaussian classifier on the test set of the TECs equals 0.58. Consequently, the null hypothesis could

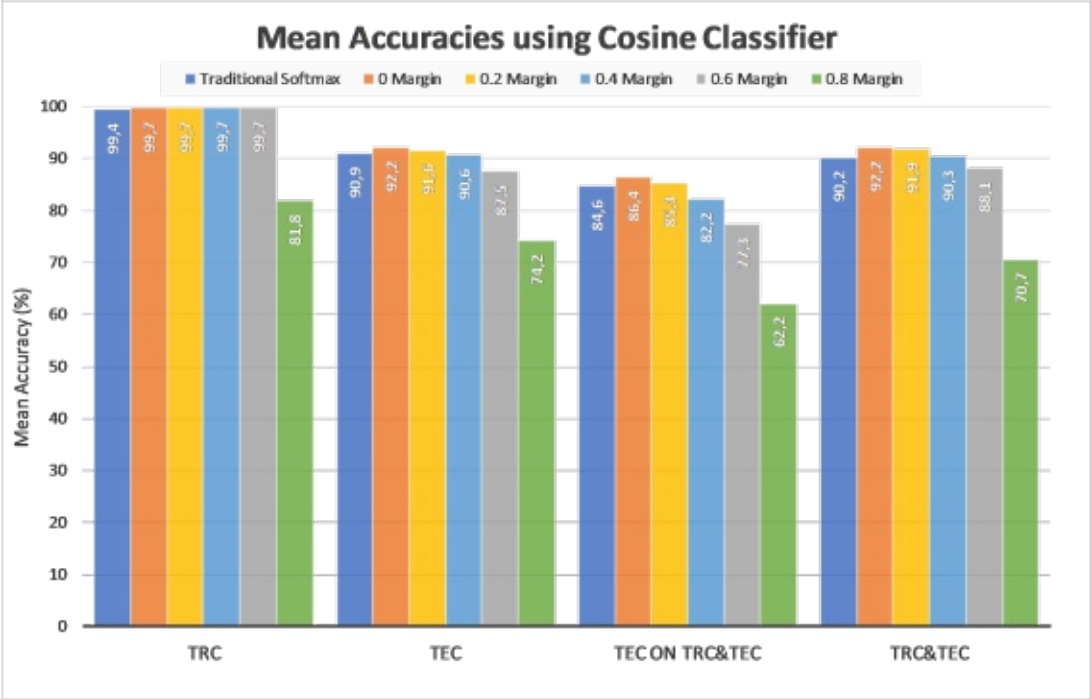
not be discarded, as desired.

4.3 Performance Benchmarks

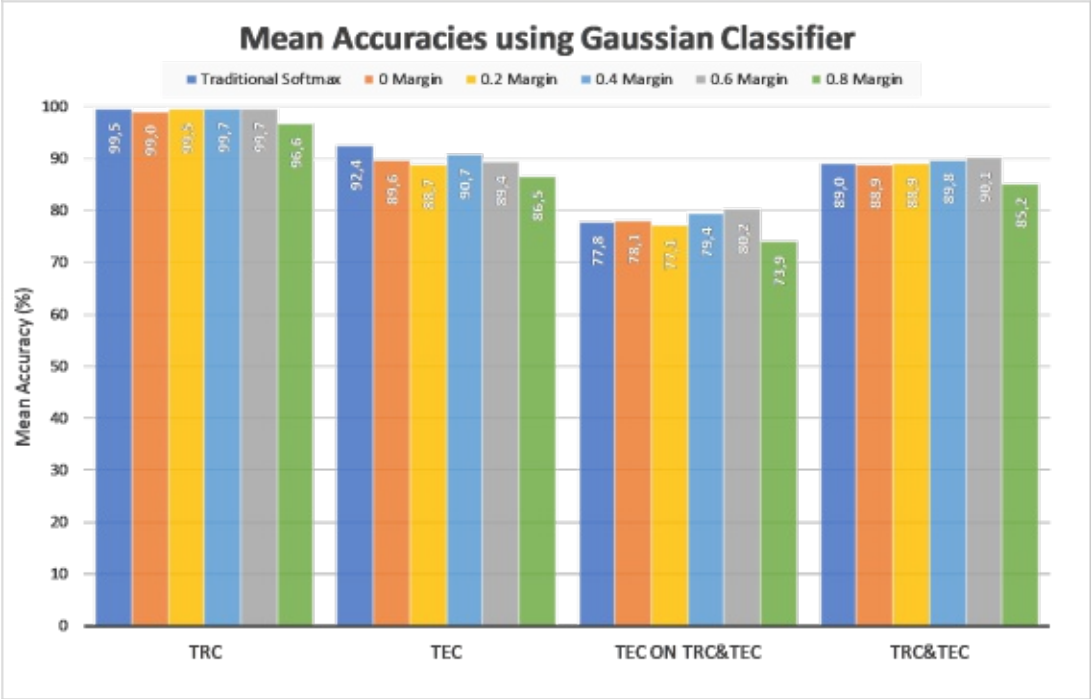
4.3.1 Mean Accuracy

Mean accuracy summarizes performance intuitively, using just a single number. This allows to compare different settings for Arcface's margin hyper-parameter and different dataset subsets efficiently. Figure 4.1a and Figure 4.1b show the results for the Cosine and Gaussian classifier, respectively. The left most bar of each subset category represents the accuracy obtained by training a traditional softmax classifier.

For the cosine classifier, accuracy is highest at 0 margin for all subsets and starts to decrease monotonically with increasing margin afterwards. In contrast, the Gaussian classifier has its peak at a margin of 0.4 or 0.6, depending on the subset which is evaluated. Surprisingly, the Gaussian's performance is not better than the Cosine's. In fact, considering the best margin setting for each classifier individually, the latter slightly outperforms the former. For small margins $m <\sim 0.6$, the Cosine's performance is superior to the Gaussian's.
The worse performance of the Gaussian classifier suggests that the TECs true distributions either are not clean Gaussians or that their parameters have not been estimated accurately enough. The latter can occur if not enough samples are available for estimating the Gaussian, which can lead to overfitting. The incorrectly captured TEC distributions become especially apparent when considering the evaluation on the third subset, where the Gaussian's performance is poorest compared to the Cosine's.
However, it can also be observed that the TECs do not seem to follow hyper-spherical distributions with similar variances either. This is because the Cosine's performance drops with an increasing margin, whereas the Gaussian's increases up to a certain margin. This suggests that the TECs become less round with increasing margin, as the Gaussian classifier is able to capture their true distributions more accurately than the cosine classifier.
Surprisingly, the traditional softmax classifier performs almost as well as the sophisticated Arcface classifier with fine tuned parameters. However, studies by Deng et al. [DGXZ19] show that Arcface outperforms a traditional softmax classifier by a large margin on big face recognition datasets such as MegaFace [KSSMB16]. MegaFace has 690 thousand classes, whereas the LPTR dataset only has slightly more than a hundred. This suggests that large differences in performance only start to become apparent for much bigger datasets. Sadly, creating a larger LPTR dataset for this work is not possible, as human annotation is expensive. For this work, this leaves no way for empirically confirming that the performance of Arcface would also become more superior to a traditional softmax classifier for a larger LPTR dataset.



(a) Cosine classifier



(b) Gaussian classifier

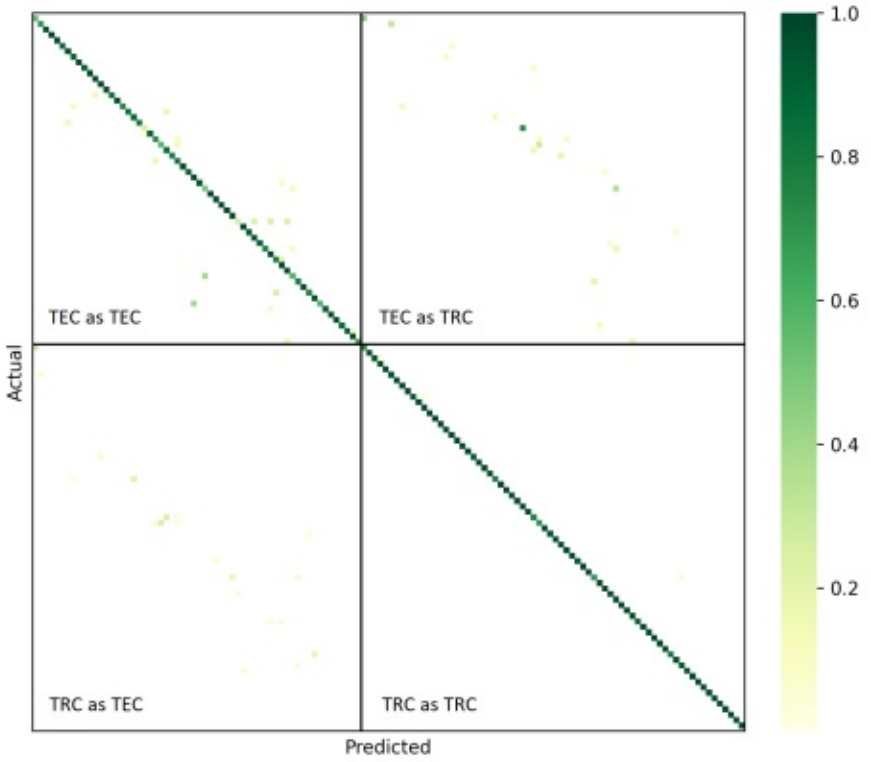
Figure 4.1: The mean accuracies obtained by different embedding space classifiers and by training with different margin values. There are a total of four different dataset subsets, for which each parametrization is evaluated. TRC and TEC refer to only evaluating on the TRCs and TECs, respectively. TEC on TRC&TEC means that only TEC inputs are used which can be classified as both TRC or TEC instances. TRC&TEC represent the entire dataset.

4.3.2 Confusion Matrix

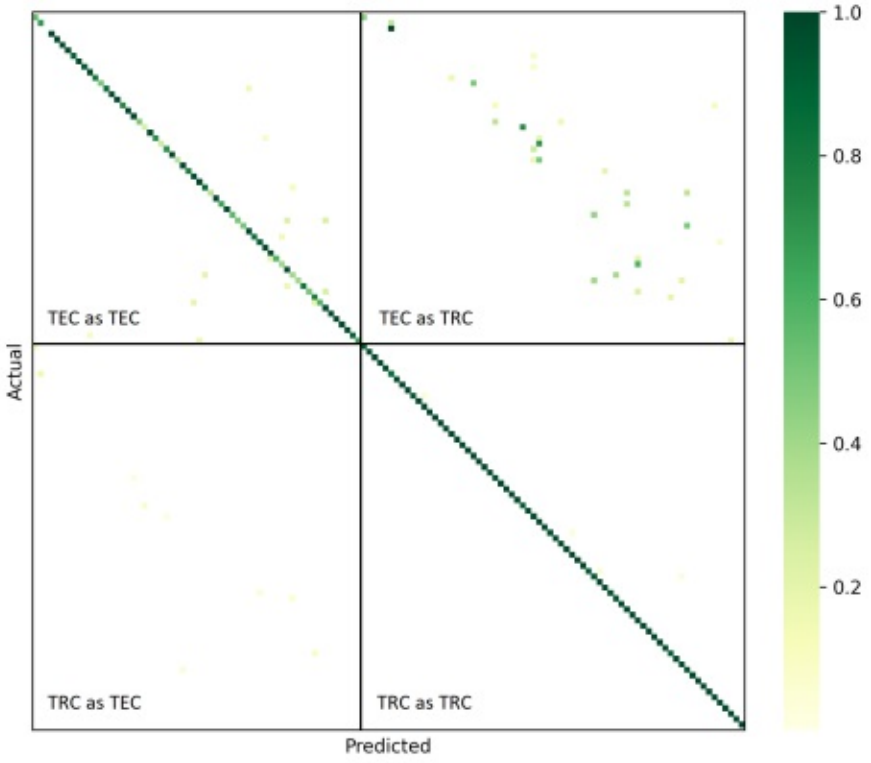
A more detailed view on the essentially same information captured by the mean accuracy is given by the confusion matrix. For the used dataset with slightly more than a hundred classes, it is still accessible for in depth inspection by a human. The confusion matrices for the Cosine and the Gaussian classifier are shown in Figure 4.2a and Figure 4.2b, respectively. Both images show that there are almost no confusions between different TRCs. This is to be expected, as the network has been directly optimized for this purpose. The Gaussian classifier confuses TECs as TRCs more often than TECs as other TECs. However, it almost never confuses TRCs as TECs. Both is surprising, as the confusions of TECs as other TECs, TECs as TRCs and TRCs as TECs are expected to be distributed almost uniformly, as it is the case for the cosine classifier. A closer look at Figure 4.2a reveals that for the cosine classifier the percentage of confusion of TRCs as TECs is slightly lower than in the other categories. This is because the TECs have far fewer samples than the TRCs, which can lead to a higher confusion even if just a few samples are classified incorrectly. This is not unlikely to happen by chance a few times for 59 TECs.
For the Gaussian classifier, the disproportionally large number of confusions of TECs as TRCs (Figure 4.2b) could be explained by incorrect distribution estimates for the TECs. More precisely, samples which belong to a certain TEC falsely achieve just a small confidence score for the respective class compared to some other TRC, which leads to a false classification. The true distributions of the TRCs on the other hand are estimated more precisely, which leads to almost no confusions of TRCs as TECs. If this hypothesis was true, it would mean that the TRCs are described more precisely by Gaussians than the TECs. This is further investigated using the embedding space metrics in Section 4.4.
For both classifiers, the confusions are not distributed uniformly in each quadrant, but form clusters in the proximity of the diagonals (Figure 4.2). This is because the order of the LPTs is not arbitrary, but sorted by state. In general, LPTs of the same state are more similar to each other than to LPTs of other states, as they often share many features. Evaluating the accuracy on state correctness rather than LPT correctness supports this observation. For example, the state accuracy on the TEC test set using the cosine classifier is 98.9%, whereas the LPT correctness only equals 92.2%.

4.3.3 Precision Recall Curve and mAP

For the used dataset, the computed mAP values are very similar to the respective mean accuracy scores. For this reason, a specific plot illustrating mAP under various settings is omitted. The PRCs for the optimal configurations for the cosine classifier and the Gaussian classifier are shown in Figure 4.3a and Figure 4.3b, respectively.
The area under each curve of Figure 4.3 defines the respective mAP on the dataset. The cosine classifier achieves a mAP of 90.2% on all classes. In contrast, the Gaussian classifier only reaches a mAP of 89.3%. Just as for the mean accuracy, the mAP is worst on the subset which allows to classify each TEC input as any TRC or TEC.

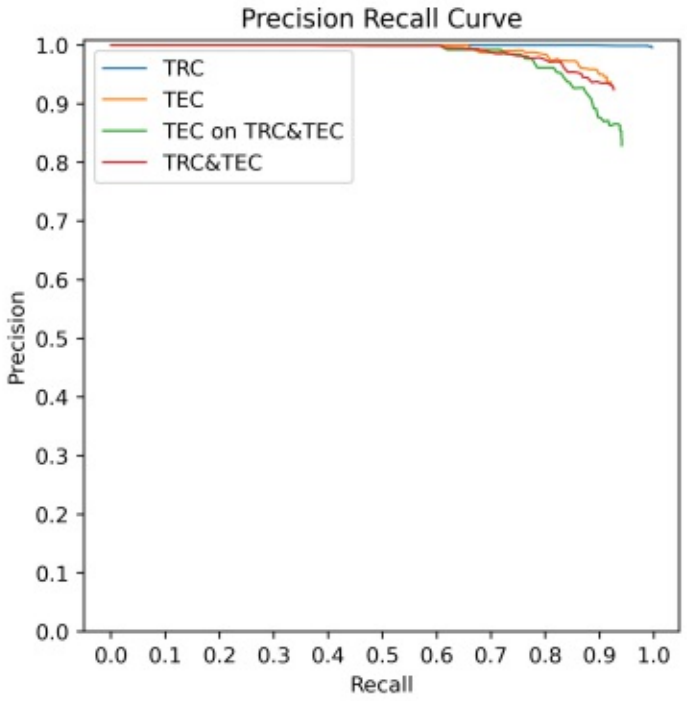


(a) Cosine classifier

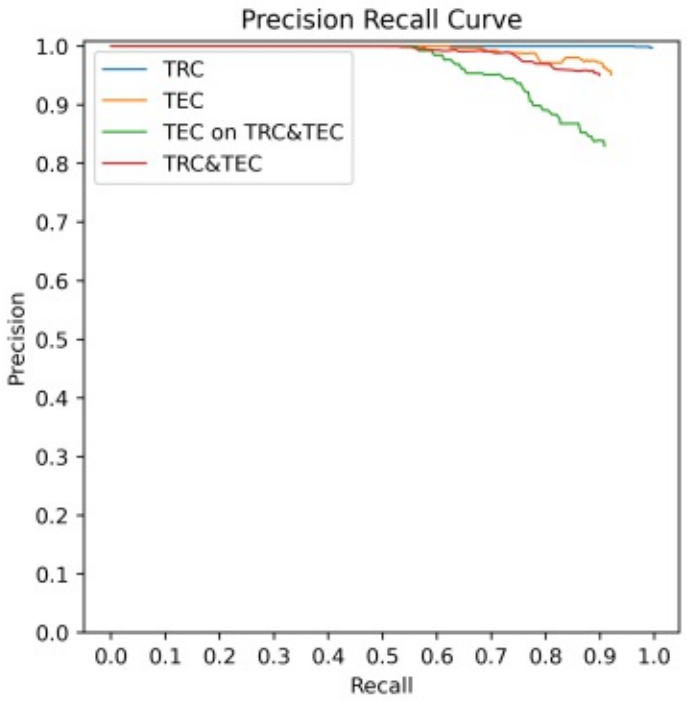


(b) Gaussian classifier

Figure 4.2: The confusion matrices of different embedding space classifiers resulting from their respective optimal margin configurations for training. The entries indicate percentages as opposed to an absolute number of samples. The greener a specific entry is, the more percent of a class's total test samples it represents. The x-axis represents the predicted classes and the y-axis the target classes.



(a) Cosine classifier



(b) Gaussian classifier

Figure 4.3: The PRCs of different embedding space classifiers. The best performance is reached on the TRCs only, where the curves of both classifiers are close to optimal. The plots highlight the best recall which is actually achieved by the trained LPTR system. Therefore, artificial sample points for a recall of 100% with precision values close to 0% are omitted.

4.3.4 Distinction between Known and Unknown Classes

The underlying OSR problem for distinguishing between known and unknown classes is based on a threshold value. The variable used for thresholding is obtained from the embedding space classifier and is either based on distance or confidence. For distance thresholding, the smallest distance to all known classes is used, whereas the largest confidence score is used for confidence thresholding.
The advantages of the distance threshold are that it does not introduce any additional parameters and that it is transparent for human inspection. This is because for both the Mahalanobis distance and the angular distance the expected range of values are familiar for known classes. The confidence threshold on the other hand requires the additional scale parameter s from Equation 3.1 for computing confidence. Confidence cannot be interpreted as measure of similarity directly, as it depends on the ratios between the similarities of all known classes. The benefit of the confidence threshold is that it can give better results than the distance threshold for an optimal configuration of the scale parameter. This is the case for the cosine classifier, as illustrated in Figure 4.4. This diagram shows the mean accuracy for the underlying binary classification problem for various margin values and threshold configurations.

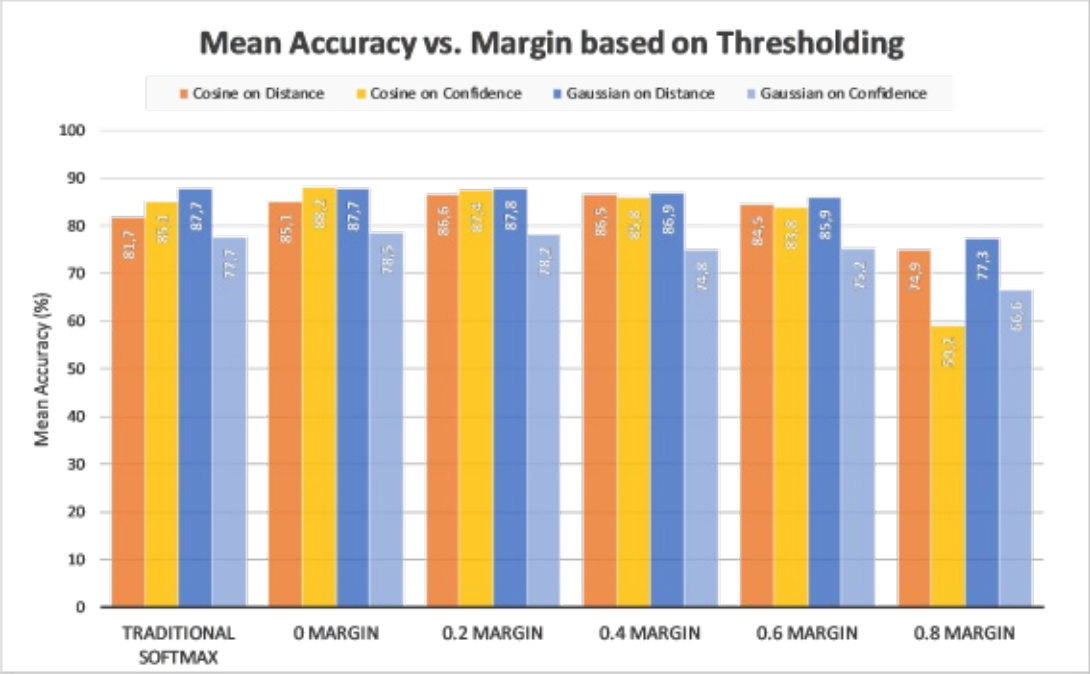
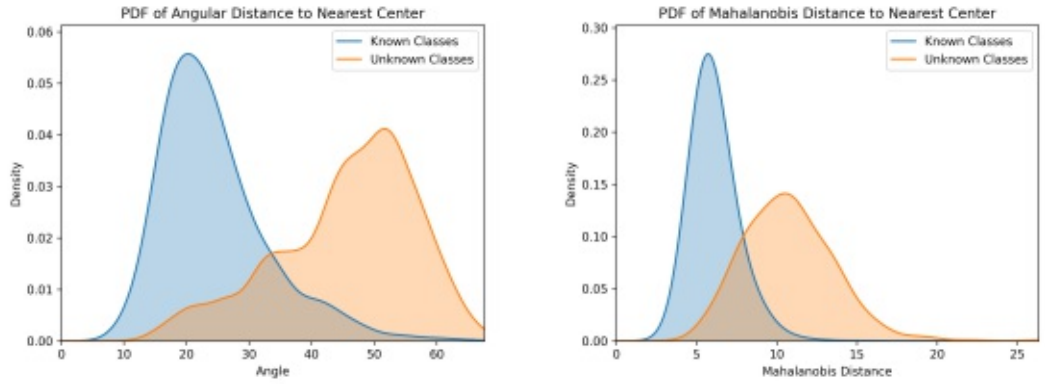


Figure 4.4: Mean accuracy for distinguishing between known and unknown classes based on a threshold value. The thresholding is done either on a distance value or confidence score, for both the Gaussian classifier and the cosine classifier.

For the Gaussian classifier, better results are achieved by the distance threshold, whereas

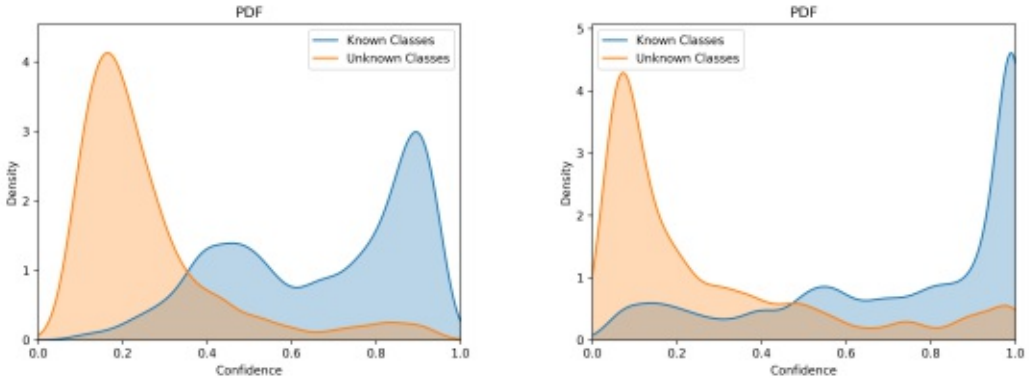
the opposite is the case for the cosine classifier. If only the peak of each threshold configuration is considered, all configurations, except for the Gaussian classifier which uses the confidence threshold, perform similarly well. This worse performance can be explained by the relative nature of the confidence threshold, which can falsely boost a large minimal distance if the distances to the other known classes are even greater, as discussed in Section 3.3.1. This is not such a big problem for the cosine classifier because the maximum angular distance is limited to 180 degrees. In contrast, the Gaussian does not have a theoretical upper bound, as it stretches to infinity in all directions.
For a specific margin configuration, a more detailed view on the underlying distributions on which thresholding is performed can be obtained by plotting the respective PDFs for the known and unknown classes, as in Figures 4.5, 4.6 and 4.7. These PDFs are obtained from a limited number of samples using a Gaussian KDE. If mean accuracy is evaluated, the optimal threshold value is located at the intersection point of the known and unknown class distributions. This is because both distributions have an equivalent prior probability in this case. The area in which the two distributions overlap represents the error caused by the thresholding. More precisely, the size of this area relative to the sum of the full areas underneath both curves is equal to the error in percent. Figure 4.5 shows the PDFs for distance thresholding and Figure 4.6 the PDFs for confidence thresholding under optimal scale parameter settings.



(a) Cosine classifier (b) Gaussian classifier

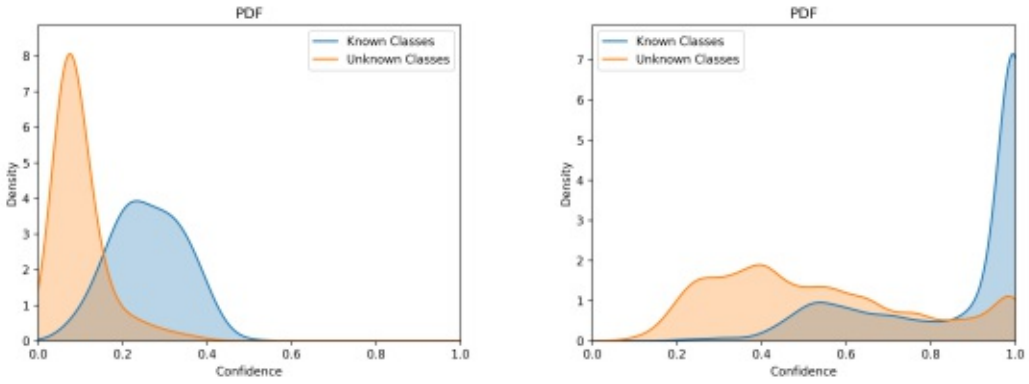
Figure 4.5: The distribution of the smallest distance to any known class plotted based on samples from known and unknown classes.

If the scale parameter is not configured properly, the confidence distributions get distorted. The further away the value of the scale parameter is from its optimum, the worse this distortion becomes and the distributions of the known and unknown classes start to overlap more and more. This is illustrated in Figure 4.7.



(a) Cosine classifier (b) Gaussian classifier

Figure 4.6: The distribution of the largest confidence score of any known class plotted based on samples from known and unknown classes.



(a) Scale value too small (b) Scale value too large

Figure 4.7: These PDFs show how the confidence distributions for the cosine classifier are affected if the used scale parameter is too small or too large.

4.4 Embedding Space Metrics

4.4.1 Simple Angular Distance Statistics

The MAAD and StdAAD for TRCs and TECs and various margin values are shown in Figure 4.8a and Figure 4.8b, respectively.

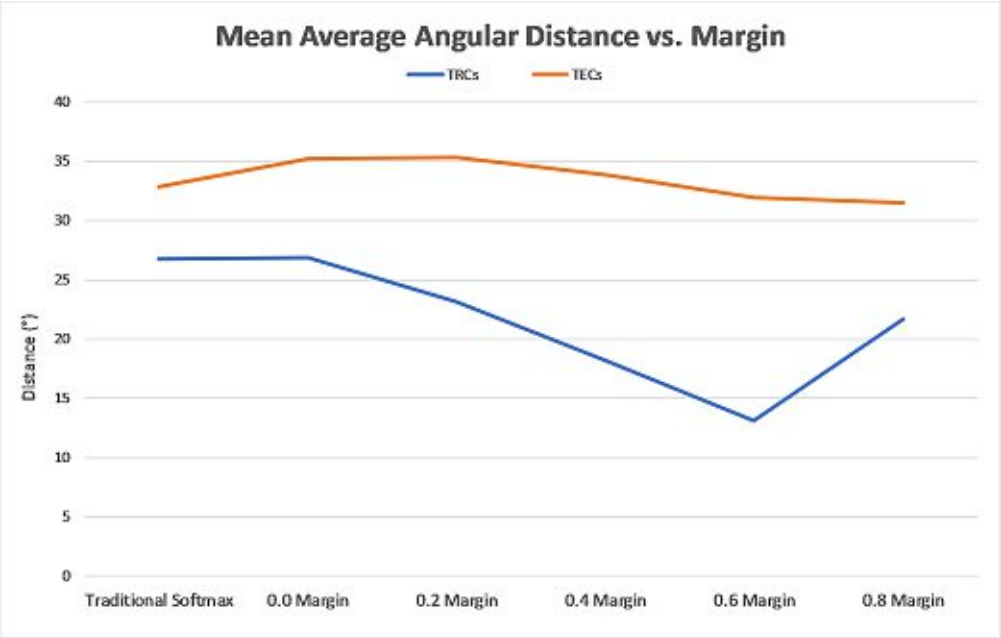
For the TRCs, the MAAD decreases for an increasing margin up to a value of 0.6. This shows that larger angular margins make the TRCs' embedding space distributions more compact, as expected. For the TECs, the MAAD also decreases for larger margins, which is desirable. However, Figure 4.8a illustrates that the MAAD is significantly larger for TECs than for TRCs, independently of the configuration. Ideally, the MAAD should be homogeneous for both class groups, as this would mean that the features of TRCs and TECs are equally well represented in the embedding space. The larger MAAD values for the TECs indicate that the uncertainty for recognizing TECs in greater than for TRCs. Figure 4.8b shows that the StdAAD drops for the TRCs for an increasing margin. In this context, a small StdAAD means that the distributions of different TRCs have similar variance. On the other hand, the StdAAD grows almost linearly for an increasing margin for the TECs. These comparably large StdAAD values imply that both TECs with small and large AADs exist. Features of TECs with a small AAD are represented better in the embedding space than features of TECs with a large AAD. This suggests that some LPTs of the TECs are more similar to the LPTs covered by the TRCs than others, which is the case. The gap between the StdAAD of the TRCs and TECs grows for an increasing margin. This means that the network starts to overfit to the TRC features as the margin gets larger.

Overall, Figure 4.8 suggests that the underlying features of the TECs have not been learned well enough by training with the TRCs. In other words, the network has not been able to fully generalize to features which are similar but not equivalent to those given by the TRCs. This can easily happen if the learning task is not sufficiently rich in complexity or shortcuts to a solution are exploited, which a human would not take. This is discussed with more detail in Section 4.5. Both problems can be mitigated by training with more data of more distinct classes.

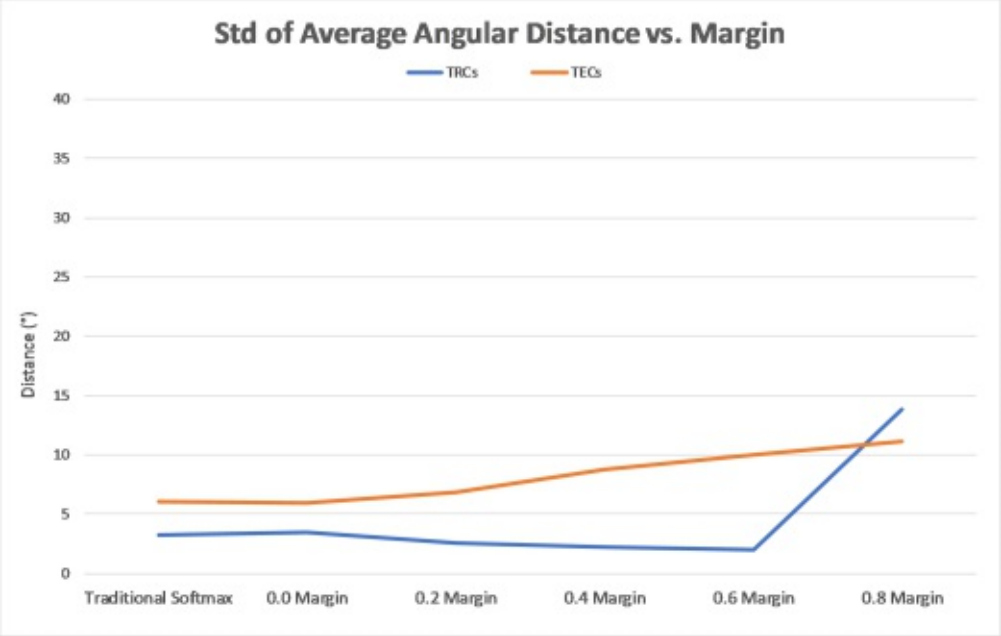
4.4.2 PCA

Figure 4.9 illustrates the average relationships between the principal components of the TRC's and TEC's distributions. Assuming that the TRCs and TECs are distributed like multivariate Gaussians, Figure 4.9a and Figure 4.9b can be interpreted as showing the mean relative standard deviations along the Gaussian's major axes, respectively.

Figure 4.9 illustrates that the ratios between the principal components are quite similar for TRCs and TECs. Furthermore, it shows that on average TRCs are rounder than TECs for margin values smaller than ~ 0.4. The opposite is true for margins larger than this. In general, the distributions become less round for an increasing margin and do not have hyper-spherical shapes.

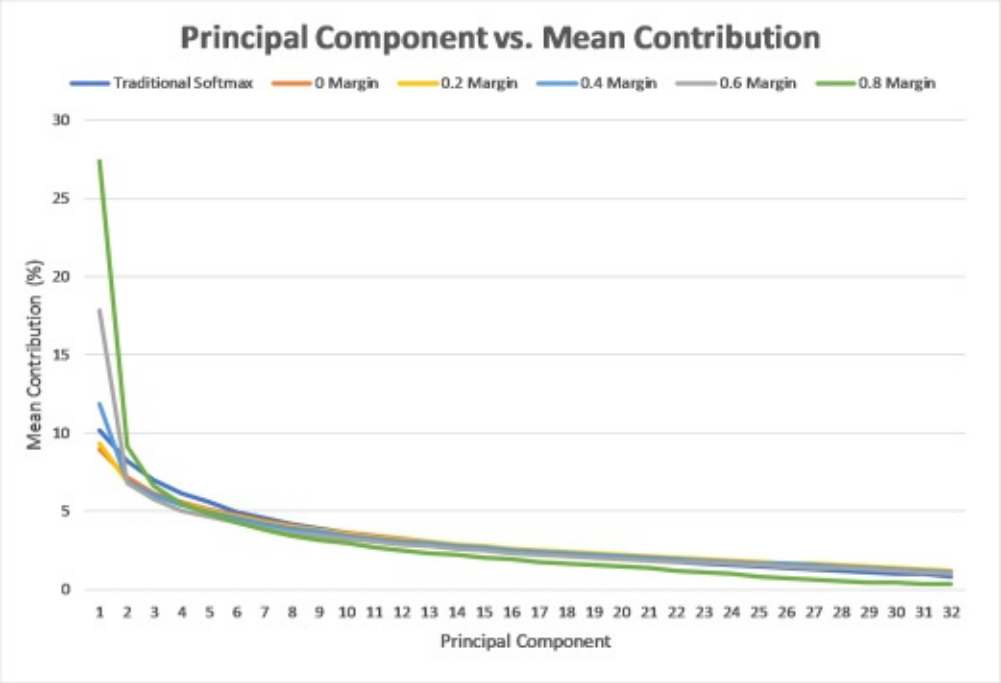


(a) The MAAD drops for an increasing margin and is significantlly smaller for TRCs.

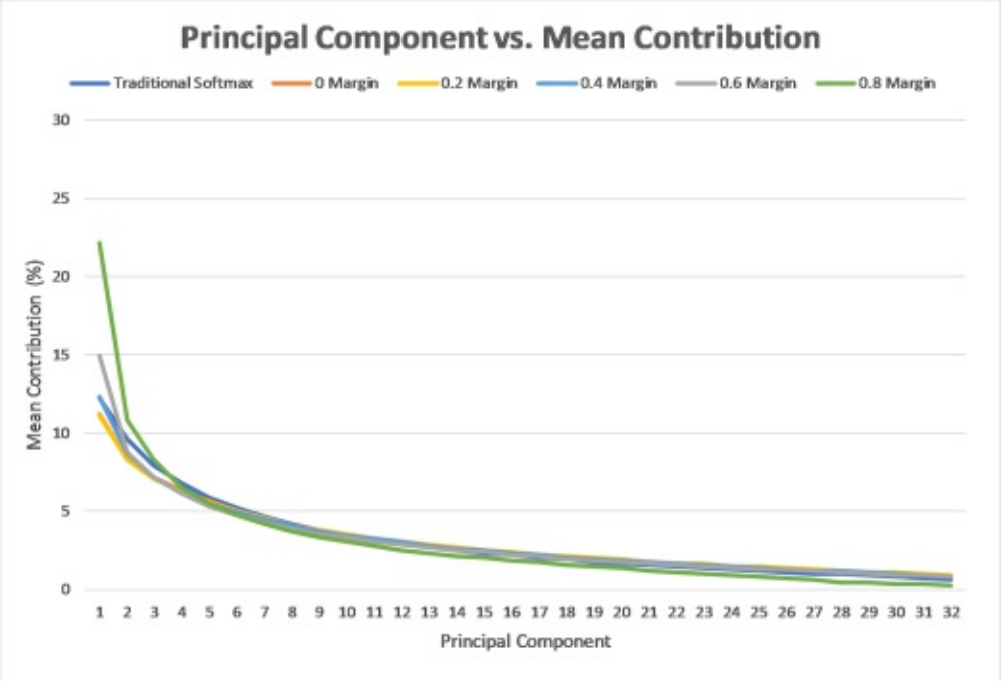


(b) The StdAAD of TECs rises rapidly for larger margin values and increases the gap between the StdAAD values for TRCs and TECs.

Figure 4.8: Simple angular distance statistics for different margin values for both TRCs and TECs .



(a) TRCs



(b) TECs

Figure 4.9: The principal components' amounts of contribution to the total standard deviation in percent. Both plots show the average contribution over all respective classes. Values are shown in percent rather than absolute values, such that averaging over distributions which have standard deviations at different scales is possible. A perfectly round distribution would be visualized as a straight line in this plot, where each principal component would have a contribution of $\frac{1}{32}$.

4.4.3 Separability of Distinct Classes

The GRR is computed for only TRCs, only TECs and all classes and is shown for various margins in Figure 4.10. The calculated values are based on 99.9% confidence SDHEs. This value has been chosen because it is similar to the confidence interval given by the 6σ rule, which is commonly used for univariate Gaussian distributions.

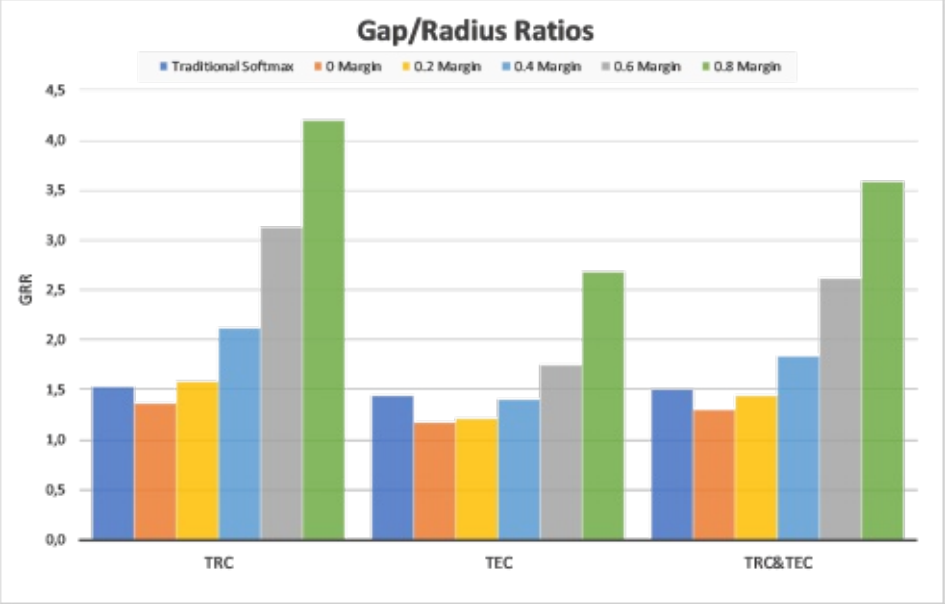


Figure 4.10: The GRR for different margin values and class subsets. As the margin increases, the relative gaps between distinct classes on average grow larger.

At first glance, it may seem like a contradiction that the GRR grows for larger margin values while the performance for the cosine classifier decreases, as illustrated in Figure 4.1a. However, the GRR does not consider the true distributions of the classes. It simply approximates the classes as hyper-spheres. This assumption is almost equivalent to the prior used by the cosine classifier, except that the latter additionally assumes that the hyper-sphere approximations of all classes have the same radius. Figure 4.9 shows that the true distributions become less spherical for larger margin values, which explains why the performance of the cosine classifier gets worse. The GRR on the other hand increases because the distributions become more compact, as displayed in Figure 4.8a.
To summarize, although the gaps between distinct classes grow larger for an increasing margin, the distributions become less spherical. This in turn leads to worse performance of the cosine classifier due to stronger violations of the prior. In contrast, the Gaussian classifier demonstrates that larger GRR values can also have a positive effect on the performance if the true underlying distributions are estimated well enough. This is visualized in Figure 4.1b, where the performance increases up to a certain margin. However, as mentioned in Section 4.3.1, the estimated Gaussians do not describe the true

distributions of TECs accurately, which suggests further performance improvements for better estimates of the true distributions.
These results show that analyzing the quality of an embedding space solely based on the GRR can be misleading because the true shapes of the distributions are neglected to a large degree. For this reason, we propose to use the GRR in combination with PCA, as these two metrics complement each other well. PCA evaluates the variances of the distributions while the GRR computes how densely distinct distributions are packed together.
Experiments for computing a lower bound for the GRR instead of an approximation have also been carried out. To retrieve a lower bound each class is also approximated by a hyper-sphere, but the radius of a specific hyper-sphere is based on the length of the respective class's largest principal component. In contrast, the approximated GRR shown in Figure 4.10 is computed using the length of the principal component with median length. The computed lower bounds were quite poor and yielded values < 1 for the GRR because there were more overlaps than gaps between distinct classes. This is because the longest major axis is about $3 - 10$ times longer than each of the major axes in the lower half of the distribution shown in Figure 4.9. As a consequence, the exaggeration for the radius yields many overlaps which do not actually exist.

4.4.4 Mean Distance PDF

The mean angular distance PDFs for the TRCs' and TECs' test sets are shown in Figure 4.11. On one hand, the image shows the distribution of the distances which are relevant for the cosine classifier's decisions. At the same time, it illustrates the final distribution of the classes' angular distances after Arcface training.
The TRC PDF looks similar to a univariate Gaussian. However, its right tail, which corresponds to larger angles, stretches out much further than its left tail. This is because variance is not constant in all directions, as shown in Figure 4.9a. The TEC distribution is shifted to the right compared to the TRC PDF, i.e. its distances on average are larger. Figure 4.11 also reveals the full distributions from which the MAAD and StdAAD values shown in Figure 4.8 have been computed for a margin of 0.
Figure 4.12 visualizes the mean Mahalanobis distance PDF for the TRCs and TECs. Additionally, it shows the distribution both of them are expected to match. Notice that the expected distribution is known because the visualized TRC and TEC distributions are based on their respective Gaussian estimates. The similarities to the expected distribution are discussed in Section 4.4.5.
Finally, let us consider the mean Euclidean PDFs for the TRCs and TECs, which are illustrated in Figure 4.13. The overall shape of both PDFs looks very similar to the shape of the mean angular distance PDF of the TRCs shown in Figure 4.11. This is expected, as the angular distances are measured based on the projection of the true distribution onto the manifold of a unit hyper-sphere. In contrast, the Euclidean distances and Mahalanobis distances are measured using the true distribution.
Figure 4.13 also shows that both PDFs are very similar, which seems to contradict

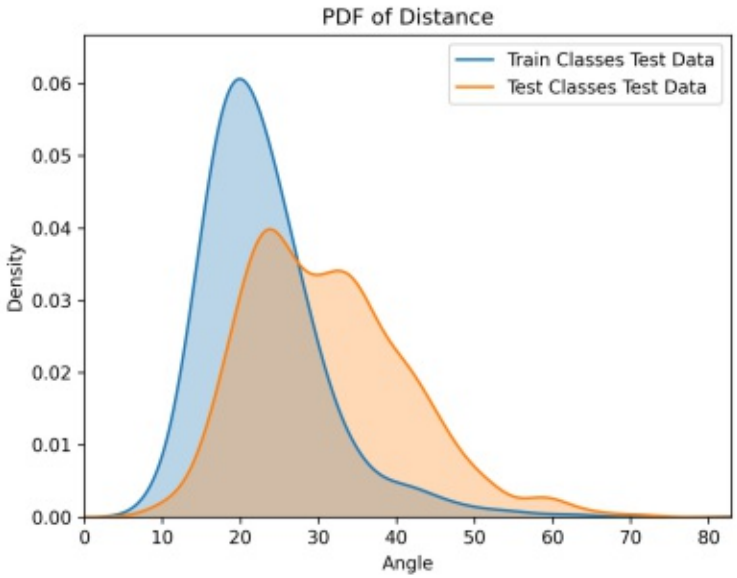


Figure 4.11: The mean angular distance distributions obtained from the test sets of the TRCs and TECs. Both distributions are plotted for the cosine classifier's optimal margin configuration of 0.

the insights obtained from Figure 4.11 at first sight. Although the true distributions of the TRCs and TECs have a very similar mean Euclidean distance distribution, the TEC's mean angular distance distribution on average has larger values than the TRC's respective distribution.
To explain this, a different view point has to be considered. As stated above, the angular distances are based on a projection, which effectively drops one dimension of information which represents the radius. The further away a true class distribution is from the embedding space's origin, the smaller the resulting projection on the unit hyper-sphere's manifold will be. To understand this intuitively, imagine the sun was closer to the earth, which would lead to more sunlight hitting the earth. This is due to the larger projection of the sun onto the earth's surface. Knowing this, the mean angular distance distribution mismatch between TRCs and TECs could be explained if the TECs' true distributions on average were closer to the origin of the embedding space. This is because the TEC distributions resulting from the projection would be larger than those of the projected TRC distributions. In fact, this is the case and is visualized in Figure 4.14.

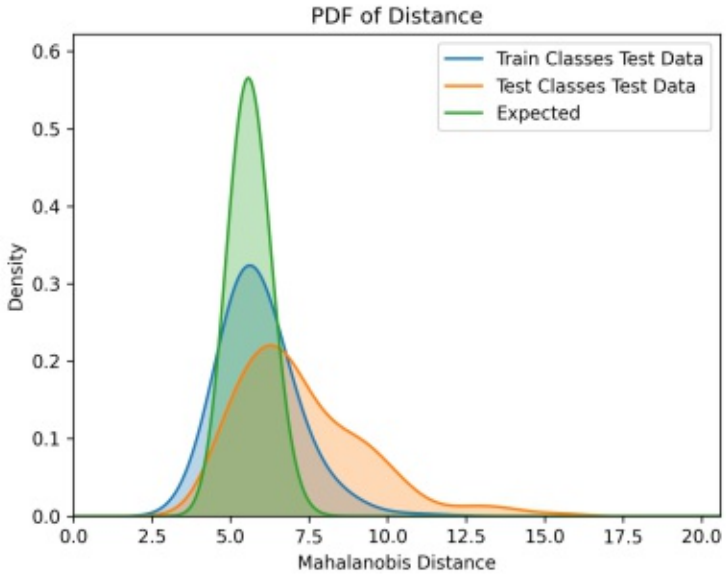


Figure 4.12: The expected mean Mahalanobis distance distribution and the actual PDFs estimated from the TRC's and TEC's test sets. Both are based on a training using the best margin of 0.4 for a Gaussian classifier and do not match the expected curve perfectly.

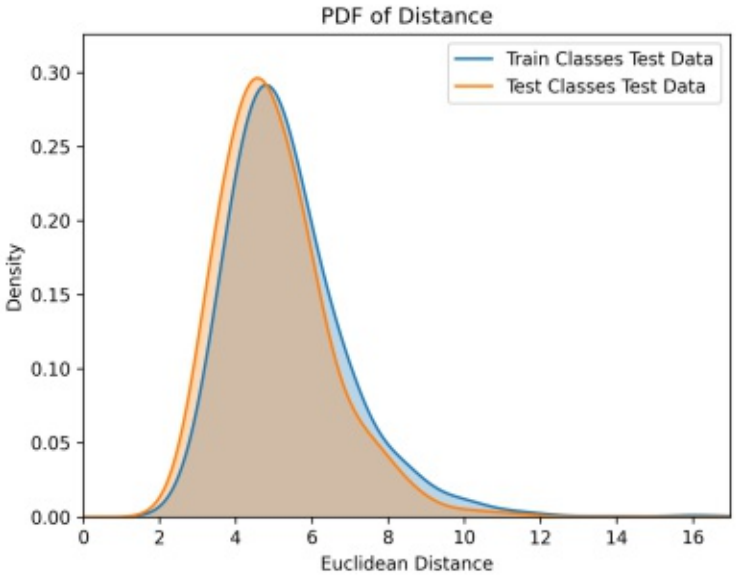
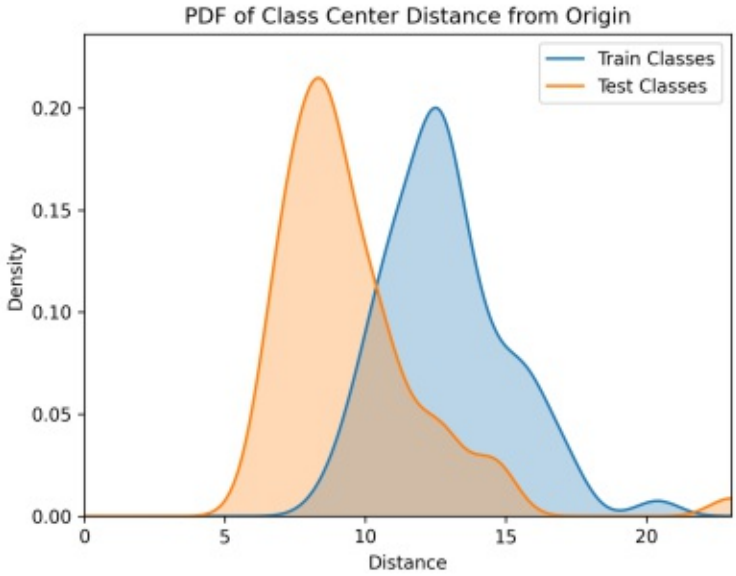
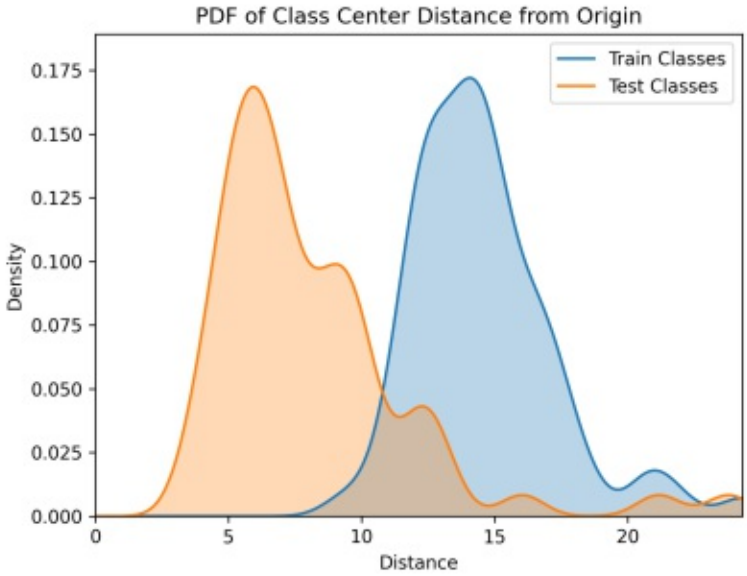


Figure 4.13: The mean Euclidean distance distributions for the test data of the TRCs and TECs. The PDFs are also based on a 0 margin setting to make it comparable to the angular distance distribution in Figure 4.11. Both PDFs look very similar.



(a) Margin 0



(b) Margin 0.4

Figure 4.14: The estimated distributions for the Euclidean distances from the origin of the embedding space to the centers of the TRC and TEC distributions. The TECs on average are closer to the origin. As the margin increases, the difference between the TRC and TEC distributions becomes larger.

4.4.5 Similarity of Estimated Class Distributions to Gaussians

Although the TRC PDF shown in Figure 4.12 visually matches the expected distribution quite well, the difference is statistically significant and not due to random chance. This has been verified using the Kolmogorov-Smirnov [MJ51] goodness of fit test. As null hypothesis, this statistic assumes that the empirical CDF given by the samples matches the CDF of the expected distribution. For the TRC distribution, this hypothesis has been discarded for an alpha of 0.01. The TEC PDF in Figure 4.12 deviates from the expected distribution heavier than the TRC's. For this reason, it is unsurprising that the Kolmogorov-Smirnov test also yields a statistically significant difference between the TEC distribution and the expected Mahalanobis distance distribution for the same value of alpha.
Therefore, it follows that both test set distributions are not perfectly distributed according to their Gaussian estimates. Based on the JSD, the TRC and TEC PDFs have a similarity of 0.72 and 0.48 to the expected distribution, respectively. The low TEC score confirms the hypothesis that the Gaussian estimates for the TECs are not a good fit for the true distributions. This observation explains the poorer performance results of the Gaussian classifier compared to the cosine classifier, which is shown in Figure 4.1. The TRC PDF is not a close match either. However, Figure 4.12 shows that in contrast to the TEC PDF, its deviations are spread more uniformly on both sides of the mode, indicating an overall better match to the estimated Gaussians.

4.5 Interpreting the Saliency Maps

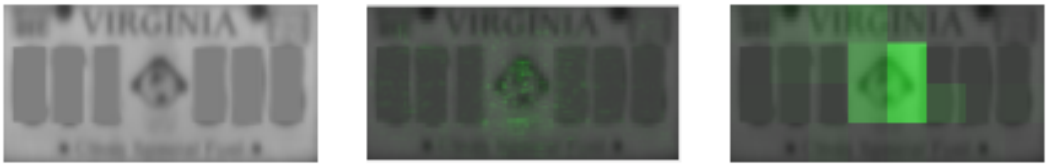
The average vanilla saliency map and the average Grad-CAM saliency map have been calculated for each TRC and TEC. Additionally, the average input image has been computed for every LPT. It must be noted that a saliency map has only been used for averaging if the class represented by the largest logit matched the target class, i.e. $argmax(logit_j) = y$. The saliency maps which failed to fulfill this condition have been discarded. This has been done to avoid using incorrectly classified instances of an LPT when creating its average saliency map.
Most results regarding which features the network reacts to when making a classification decision appear reasonable and convincing. Sadly, there is no metric by which the integrity of the network can be measured, which makes this statement fundamentally subjective. Such a metric does not exist because there is no single right choice of features the network should use to make its decisions for a given LPT. This is because different humans may consider distinct aspects of an LPT when identifying it. Figure 4.15 visualizes some convincing TRC saliency map examples and shows different types of features the network reacts to. The greener the overlay at a pixel is, the more important the pixel is for the network.
In Figure 4.15, the vanilla saliency maps are a lot noisier than the respective Grad-CAM saliency maps. However, both techniques largely agree on the highlighted features. After

careful visual inspection of the average saliency maps, it appears that altogether special symbols like in Figure 4.15a and textures as in Figure 4.15b are the most attractive features for the network to learn. This seems reasonable, as both feature types typically are very discriminative for different LPTs. Slightly less popular feature types are text, as in Figure 4.15c and the positioning of the identifying characters, like in Figure 4.15d. The reason why they are not used as frequently is that several LPTs share the same text or the same character positioning in many cases. For example, "VIRGINIA" is written in the same font and on the same spot for the two LPTs in Figures 4.15a and 4.15e. Therefore, this feature alone cannot be used by the network to determine the correct LPT. Consequently, the network either has to learn an additional indicator or a completely different feature in the first place. In this concrete example, the network seems to have chosen the latter option. Another example is the star in the top left corner of Figure 4.15b, which is not regarded by the network, as it also appears in other LPTs of Texas. In some cases, the network uses the absence of any specific feature, i.e. a static background color, as strong indicator for choosing a class. Examples of this are the highlighted black spot near the bottom left corner of the Grad-CAM saliency map in Figure 4.15d and the gap between the identifying characters in Figure 4.15e. It is attractive for the network to learn this type of feature as indicator for a certain TRC if there is no other TRC which has a static background color at the exact same spot. This is because a static background color is the most primitive feature.

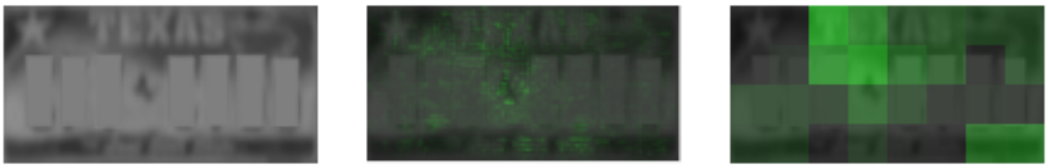
The network should not even be blamed for learning such features as shortcuts. After all, it solves the task it is trained for, which is distinguishing between TRCs, perfectly well. However, in the context of classification on open sets this sort of behavior is undesirable and problematic because it generalizes poorly to TECs. Imagine a scenario where a network has learned to use a spot of a TRC LPT showing a static background as the only indicator for identifying the respective class. This scenario is similar to the saliency maps shown in Figure 4.15e. Now, if one of the TECs happens to have the same static background at the exact same spot, the network will not be able to distinguish between these two LPTs.

To solve this problem, many more LPTs must be used for training. The more diversity these LPTs have the better. More TRCs decrease the likelihood of the network learning such undesired behavior, as shortcuts which exploit static background are less likely to exist. In general, it can be said that more feature rich TRCs force the network to learn more specific features. This is very desirable, as more complex features also improve the generalization capabilities to correctly classify TECs.

The average saliency maps of correctly classified TEC images appear equally convincing as the ones of the TRCs. The TEC saliency maps of misclassified images sometimes reveal further insight about why certain confusions occur.

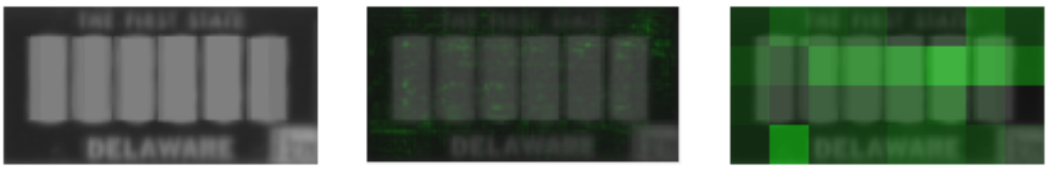


(a) Symbol



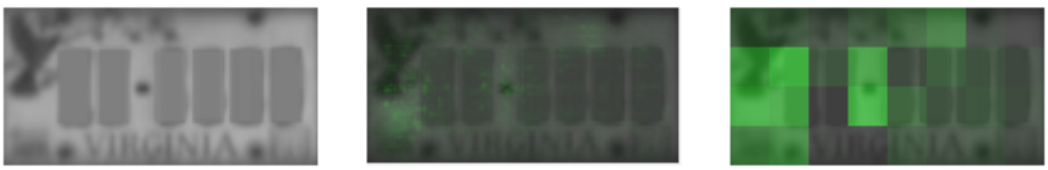
(b) Texture

(c) Text



(d) Character positioning

(e) Gap

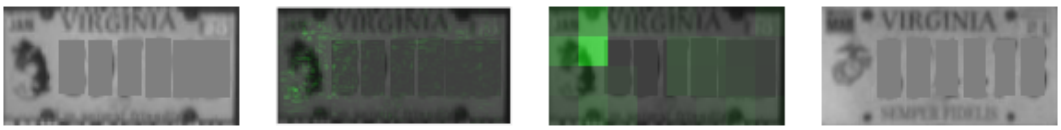


(f) Mixed

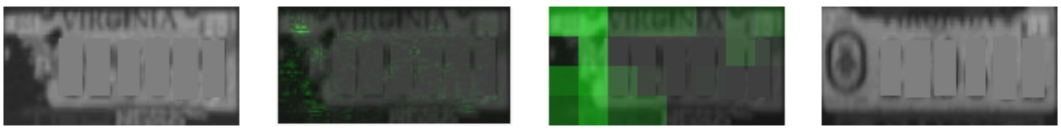
Figure 4.15: Some examples for different types of indicators which the network uses to make a classification decision. The caption of each sub-figure specifies the most important indicator for the respective LPT. The columns from left to right show the average input image, the average vanilla saliency map and the average Grad-CAM saliency map for each LPT.

A few examples of misclassified images are shown in Figure 4.16. The respective saliency maps of each image highlight why the network believes that the false prediction it makes should be correct.
Figure 4.16a illustrates the problem of mostly using static background color and character positions as features for identifying a certain LPT. The image regions highlighted by the Grad-CAM saliency map are identical for the predicted LPT and the target LPT. In order to correctly distinguish between these two LPTs, the text at the top would have to be considered more heavily by the network. Figure 4.16c shows a confusion which happens because the special symbol identifying the target LPT is misinterpreted. This suggests that the network has not learned to generalize features of TEC symbols very well. In this particular case, the bear on the target LPT is easily distinguishable from the coat of arm on the predicted LPT for a human. A similar example caused by a symbol confusion is demonstrated in Figure 4.16b. Figure 4.16d shows a confusion between two LPTs where the only difference is the spatial location of the "GEORGIA" text on the top.

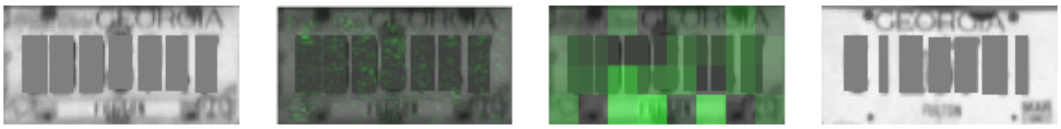
(a) Background and character positioning



(b) Symbol misinterpretation



(c) Another symbol misinterpretation



(d) Undesired position invariance

Figure 4.16: Some examples of misclassified LPT images. The columns from left to right show the input image, the vanilla saliency map, the Grad-CAM saliency map and a representative image of the falsely predicted class, for each LPT.

4.6 Summary

All insights highlighted within this section refer to the used LPTR dataset introduced in Section 4.1. This dataset only consists of 5740 training images and therefore is considered rather small.
Analyzing the class distributions in the embedding space has shown that increasing the margin hyper-parameter for training decreases the average reach or size of the classes' distributions up to a certain point. This insight can be obtained from the MAAD. Consequently, the gaps between distinct classes grow larger, which is expressed by the GRR. However, at the same time each class distribution also becomes more elongated, which can be observed by conducting PCA. This is similar to the effect of squashing a balloon. In this analogy, the external forces on a class's distribution are applied by the other classes. The StdAAD shows that the variance of the average class size increases with a growing margin. Consequently, the performance of the cosine classifier decreases for an increasing margin, as it does not account for variance in any way. The cosine classifier operates on the embeddings and uses the nearest class center as decision criterion. This type of classifier has been heavily used by recent research, as explained in Section 3.2.
A multivariate Gaussian classifier has been evaluated to test how similar the shapes of the class distributions are to hyper-spheres and hyper-ellipsoids in the embedding space. Gaussians also test if the overall performance can benefit from a more expressive classifier, given that the embedding space is fixed. The results have revealed that for both TRCs and TECs the respective distribution estimates, which are based on the test sets, show statistically significant differences from a true Gaussian distribution. However, the overall shapes of the estimated TRC distributions come close to the shape of a true Gaussian. This is also noticeable in the mean accuracy on the TRC test set, where the performance of the Gaussian classifier is better than the performance of the cosine classifier for a large margin value. For the TECs, the distribution estimates deviate more heavily from a true Gaussian distribution. As a result, the Gaussian classifier for the optimal margin configuration has a worse performance than the optimal cosine classifier for the TECs. However, for large margins, the Gaussian classifier has a better performance than the cosine classifier on TECs. This suggests that the assumption of a Gaussian distribution for the classes is not fundamentally incorrect, but not expressive enough.
The types of features of LPTs which the network considers when making a classification decision have been analyzed using gradient-based saliency maps. The manual inspection of the saliency maps has revealed that especially symbols and textures commonly are strong indicators for a network's decision, as they often are unique to a certain LPT. In some cases, we observed that the network has learned primitive features such as static backgrounds as strong indicators. This is possible in situations where such features allow to distinguish a certain TRC from all other TRCs. These primitive features generalize poorly, as they either capture the absence of meaningful content or a combination of simple reoccurring patterns such as character placements. When the network training exploits such shortcuts towards a good solution, generalization capabilities are reduced.

4.7 Discussion

While Section 4.6 summarized the insights obtained from the experiments, this section is about the quality of the achieved results.
A mean accuracy of 99.8% has been reached on the TRCs. This demonstrates the capabilities and potential of using deep learning for the LPTR task. However, the experiments have also shown that the performance on the TECs is a lot worse than on the TRCs. This implies that the generalization capabilities of the trained model are limited and not optimal. But considering that only 4060 images have been used for training the network, we believe that the achieved mean accuracy of 92.2% on the TECs is a good result. The same argument applies to the reached mean accuracy of 88.2% for the OSR of unknown classes.
In the beginning, we were quite confident that accuracy would benefit from modeling the class distributions in the embedding space as multivariate Gaussians. We were surprised to see that this is not the case. It is not clear to what degree this is due to the limited number of available samples for estimating the Gaussians. For some of the TECs, the Gaussians have been estimated based on 10 to 30 images. Using data augmentation to create several embeddings per image to have more samples for parameter estimation only slightly mitigates the problem. This is because the embeddings which are based on the same original image are not independent from each other. Sadly, we could not test how the performance of the Gaussian classifier would be affected by a significantly larger dataset. Due to the lack of samples the estimated Gaussians are prone to overfitting. Therefore, it is possible that the true class distributions are closer to Gaussians than our results indicate.

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

Learned embedding spaces are commonly used for classification problems on an open set of classes. In this work, the class distributions in such an embedding space have been analyzed. For this purpose, the Gap Radius Ratio (GRR), which measures how filled a given embedding space is, has been introduced. For embedding spaces which are trained based on angular distances, two more metrics have been proposed. Both evaluate the distributions as they are seen by the training procedure for the embedding space. The Mean Average Angular Distance (MAAD) measures the mean intra-class compactness. The Standard Deviation of the Average Angular Distance (StdAAD) quantifies the variance of the intra-class compactness as a complementary measure. Adapting these two metrics to work with different metric spaces is straight forward. Our analysis has revealed that the class distributions in our trained embedding spaces do not follow multivariate Gaussians accurately for the used license plate type dataset.
Another contribution is the analysis on the effects of masking out the identifying characters of a license plate. Empirical experiments have shown that training with only unmasked data encourages the model to learn specific characters as features. Likewise, the model tends towards learning the masks when it is only trained with masked license plates. A solution which solves this problem by randomizing the masking process during training has been proposed. Using this technique, the results on the test show no significant difference independently of whether or not the identifying characters of a license plate are masked out.
On the used license plate type dataset, mean accuracies of 99.7% and 92.2% are achieved on the classes used for training the network and the test classes which are added in retrospective, respectively. For distinguishing between known and unknown license plate types, a mean accuracy of 88.2% is reached.
A method for computing gradient-based saliency maps for test classes has been introduced.

Furthermore, a mean saliency map has been computed for each known license plate type. Overall, we find that our model has learned suitable features for recognizing the license plate types it has been trained with. However, due to the size of the used dataset its generalization capabilities are limited. Our saliency maps also show that features which generalize poorly are used for recognizing a few of the license plate types. This explains the significantly worse performance on the test classes compared to the ones used for embedding space training.

5.2 Future Work

5.2.1 Feature Visualization

One interesting next step would be a more rigorous analysis of what a trained network has learned using feature visualization. This would allow to identify the visual properties of the features the network is looking for more carefully. Notice that the saliency map techniques used in this work only answer which parts of an image are important to the network but not why they are. For this reason, saliency maps and feature visualization complement each other quite well. One possible use case of feature visualization would be an investigation of which features the model has learned to extract from special symbols of LPTs. This in turn would reveal further insights about why the network makes confusions which are caused by misinterpreting such symbols.

5.2.2 Larger Dataset

Being able to make use of a much larger dataset which has both more classes and more samples per class would allow to conduct further interesting experiments. First of all, it could be investigated how the network's generalization capabilities develop depending on the amount of training data used. This can be measured by both traditional benchmarks such as mean accuracy and by some of the introduced embedding space metrics like the MAAD. Again, in the optimal case there should be little to no difference between the results for TRCs and TECs. Furthermore, adding and removing carefully selected classes from the training data would enable to study the network's behavior regarding the learning of shortcuts. For example, if a shortcut which exploits static background is learned for a specific class of a smaller dataset, additional classes could be added to the training data. If they were considered individually, these additional classes would allow to make use of the same exploit. However, as all these classes could not make use of the same exploit if they were considered together, the network would have to learn different features for each one in order to recognize all classes correctly. The experiment would be to investigate how the learned features adapt in such a scenario.
In general, training on a much larger LPTR dataset is a key aspect for the future. This will enhance generalization capabilities and will discourage the network from taking undesired shortcuts during training. It is expected that the performance gap between TECs and TRC will decrease significantly, as the network learns to generalize better and

better by using more training data.

5.2.3 Automatic Inference for the State of Origin

Another idea is to automatically infer the state of origin if an image of an unknown LPT class is given as input. In theory, this should be possible in a lot of cases because many LPTs of a certain state share a lot of similarities with each other. Automatic state inference would be possible by showing that certain features present in an unknown LPT can also be found in known LPTs of some state. One way to achieve this could be to train a network which has additional outputs for predicting the state of origin of an LPT. The total loss of such a network would then be a linear combination of the state and LPT classification losses. Learning to predict the correct state would encourage the network to learn commonly shared features of LPTs with the same state of origin. The hypothesis is that in many cases these learned features will also work reliably for unknown LPTs of the same state.

5.2.4 Automatic Incremental Growth

Going even one step further, the system could be configured to add new LPTs to its database automatically after its initial deployment. If very similar embeddings which are unlikely to be instances of any known LPT occurred over and over again, this would suggest that they are instances of the same LPT which was unknown to the system at the time. If enough statistical evidence was collected which supports this hypothesis, a new LPT could be added to the known classes automatically based on the collected data. Additionally or alternatively, all automatically collected data of new LPTs could be validated by a human after some time. Then, it could be used for building a new model together with the original training data. Applying this technique iteratively would allow to significantly improve the system over time, assuming that enough data of unknown classes could be collected.

List of Figures

List of Tables

Acronyms

AAD Average Angular Distance. 47, 67

AP Average Precision. 45

BN Batch Normalization. 34, 35, 86

CDF Cumulative Distribution Function. 4, 51, 75

CNN Convolutional Neural Network. 4, 6, 8, 10, 33, 34, 85

FSL Few-Shot Learning. 6, 7, 16

Grad-CAM Gradient-weighted Class Activation Mapping. 29, 30, 52, 75–78, 88

GRR Gap/Radius Ratio. 52, 70, 71, 79, 81, 87

JSD Jensen-Shannon Distance. 50, 51, 75

KDE Kernel Density Estimation. 47, 65

KLD Kullback-Leibler Divergence. 50, 51

kNN k-Nearest Neighbors. 38

LP License Plate. 1–4, 10, 15, 16, 27, 34, 55–58, 85

LPD License Plate Detection. 15, 16

LPR License Plate Recognition. 1, 2, 15, 16, 56

LPT License Plate Type. 1–5, 7, 9, 10, 16, 52–58, 61, 67, 75–79, 82, 83, 85, 88

LPTR License Plate Type Recognition. 1–7, 9, 10, 15, 16, 27, 33, 34, 36, 38, 44–46, 55–57, 59, 63, 79, 80, 82, 87, 89

MAAD Mean Average Angular Distance. 47, 67, 68, 71, 79, 81, 82

mAP Mean Average Precision. 11, 45, 61

NCC Normalized Cross Correlation. 4

OCR Optical Character Recognition. 2, 3

OSR Open Set Recognition. 7, 16, 45, 46, 64, 80

PCA Principal Component Analysis. 4, 49, 67, 71, 79

PDF Probability Density Function. 4, 40, 42, 43, 47–51, 65, 66, 71, 73, 75, 86, 87

PRC Precision Recall Curve. 45, 61, 63, 87

ReLU Rectified Linear Unit. 29, 30, 34, 35, 86

SDHE Standard Deviational Hyper-Ellipsoid. 51, 70

StdAAD Standard Deviation of the Average Angular Distance. 47, 67, 68, 71, 79, 81

TEC Test Class. 6–9, 11–13, 33, 38, 39, 44–47, 50, 52, 53, 55–61, 67–76, 78–80, 82, 86–88

TRC Train Class. 6–9, 11–13, 39, 44–47, 50, 53, 55–57, 60, 61, 63, 67–76, 79, 80, 82, 86–88

VGG Visual Geometry Group Network. 33–35, 55, 89

YOLO You Only Look Once. 16

Bibliography

[AAP+08] Christos-Nikolaos E Anagnostopoulos, Ioannis E Anagnostopoulos, Ioannis D Psoroulas, Vassili Loumos, and Eleftherios Kayafas. License plate recognition from still images and video sequences: A survey. *IEEE Transactions on intelligent transportation systems*, 9(3):377–391, 2008.

[AB12] Omolola A Adedokun and Wilella D Burgess. Analysis of paired dichotomous data: A gentle introduction to the McNemar test in SPSS. *Journal of MultiDisciplinary Evaluation*, 8(17):125–131, 2012.

[AB20] Apoorv Agnihotri and Nipun Batra. Exploring bayesian optimization. *Distill*, 5(5):e26, 2020.

[AGM+18] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. *Advances in neural information processing systems*, 31:9505–9515, 2018.

[BBM+15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[BH09] Jop Briët and Peter Harremoës. Properties of classical and quantum Jensen-Shannon divergence. *Physical review A*, 79(5):052311, 2009.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[CAS+19] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas. *Distill*, 4(3):e15, 2019.

[Cha07] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.

[DGXZ19] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2019.

[DISB12] Shan Du, Mahmoud Ibrahim, Mohamed Shehata, and Wael Badawy. Automatic license plate recognition (ALPR): A state-of-the-art review. *IEEE Transactions on circuits and systems for video technology*, 23(2):311–325, 2012.

[FEHP11] Catherine Forbes, Merran Evans, Nicholas Hastings, and Brian Peacock. *Statistical distributions*. John Wiley & Sons, 2011.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[GHC20] 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.

[GWK+18] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[GZJ+15] Shenghua Gao, Yuting Zhang, Kui Jia, Jiwen Lu, and Yingying Zhang. Single sample face recognition via learning deep supervised autoencoders. *IEEE transactions on information forensics and security*, 10(10):2108–2118, 2015.

[HBL17] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *print arXiv:1703.07737*, 2017.

[HH15] John H Hubbard and Barbara Burke Hubbard. *Vector calculus, linear algebra, and differential forms: A unified approach*. 2015.

[HZRS16a] 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.

[HZRS16b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[Jør17] Hogne Jørgensen. Automatic license plate recognition using deep learning techniques. Master's thesis, NTNU, 2017.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KSSMB16] Ira Kemelmacher-Shlizerman, Steven M Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4873–4882, 2016.

[Kus20] Michael Kustermann. World license plates. *www.worldlicenseplates.com*, 2020.

[LH02] Anhua Lin and Shih-Ping Han. On the distance between two ellipsoids. *SIAM Journal on Optimization*, 13(1):298–308, 2002.

[LSZ+18] Rayson Laroca, Evair Severo, Luiz A Zanlorensi, Luiz S Oliveira, Gabriel Resende Gonçalves, William Robson Schwartz, and David Menotti. A robust real-time automatic license plate recognition based on the YOLO detector. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE, 2018.

[LWY+17] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 212–220, 2017.

[LZG+19] Rayson Laroca, Luiz A Zanlorensi, Gabriel R Gonçalves, Eduardo Todt, William Robson Schwartz, and David Menotti. An efficient and layout-independent automatic license plate recognition system based on the YOLO detector. *arXiv preprint arXiv:1909.01754*, 2019.

[MD18] Wang Mei and Weihong Deng. Deep face recognition: A survey. *arXiv preprint arXiv:1804.06655*, 1, 2018.

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[MJ51] Frank J Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[MN19] Jan R Magnus and Heinz Neudecker. *Matrix differential calculus with applications in statistics and econometrics*. John Wiley & Sons, 2019.

[OMS17] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[SCD+17] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[SCWT14] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems*, pages 1988–1996, 2014.

[SGK17] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pages 3145–3153. PMLR, 2017.

[SJ17] Sergio Montazzolli Silva and Claudio Rosito Jung. Real-time brazilian license plate detection and recognition using deep convolutional neural networks. In *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 55–62. IEEE, 2017.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[Str16] Gilbert Strang. *Introduction to linear algebra*, volume 5. Wellesley-Cambridge Press, 2016.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pages 3319–3328. PMLR, 2017.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[VLL+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[WSM15] Bin Wang, Wenzhong Shi, and Zelang Miao. Confidence analysis of standard deviational ellipse and its extension into higher dimensional Euclidean space. *PloS one*, 10(3):e0118537, 2015.

[WWZ+18] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5265–5274, 2018.

[WYKN20] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53(3):1–34, 2020.