University of West Bohemia

Faculty of Applied Sciences

Department of Cybernetics

MASTER THESIS

PILSEN, 2020 JAN BENEŠ

Declaration of Authorship

Acknowledgements

I would like to express my gratitude and appreciation to Ing. Ivan Gruber for his guidance throughout the project.

Likewise, I would like to thank my supervisor Ing. Marek Hrúz, Ph.D. for providing me with invaluable feedback.

My gratitude also goes to MetaCentrum VO for allowing me to use their computational resources.

Finally, I have to thank my family for supporting me in the pursuit of my studies.

Abstract

The goal of this study is to design and implement an end-to-end facial recognition system. The first part is focused on a general overview of modern methods followed by an in-depth description of state-of-the-art research of loss functions. The emphasis is being put on the ArcFace loss as it is the research which forms the basis of the facial recognition system implemented in this thesis. The second part deals with the design and implementation of the system. The end of the text contains a comparison with a commercial algorithm. The performance was evaluated on a dataset which was created from the recordings of evening news on the czech public television broadcast (Česká Televize).

Key words

Machine learning, facial recognition, identification, verification, convolutional neural networks, loss functions, ArcFace

Abstrakt

Cílem této práce je návrh a implementace systému rozpoznávání obličeje. V první části je poskytnut přehled moderních metod, na který navazuje podrobný rozbor výzkumu ztrátových funkcí. Důraz je kladen na ztrátovou funkci ArcFace. Tato funkce byla použita při trénování modelu, jenž tvoří jádro systému implementovaného v rámci této práce. Druhá část práce obsahuje návrh a popis implementace systému. V závěru je systém porovnán s komerčním algoritmem. Vyhodnocení proběhlo na datasetu, jenž byl vytvořen ze záznamu večerních zpráv České Televize.

Klíčová slova

Strojové učení, rozpoznávání obličeje, identifikace, verifikace, konvoluční neuronové sítě, ztrátové funkce, ArcFace

Contents

Chapter 1

Introduction

Facial recognition systems recently exceeded the performance of humans on many real-world benchmarks.

The beginning of the quest to give computers the ability to recognize human faces dates back to the 1960s [1] with Woody Bladson being the first researcher to attempt the feat. The issue with the first systems was the decrease of performance when the faces were not in the optimal position. A big stepping stone towards pose invariance came with the invention of convolutional neural networks 2. These models are the cornerstone of modern computer vision.

In the field of facial recognition, a lot of effort was put into the design of loss functions. These functions are essential in all of the machine learning, for they are used to supervise model fitting. The researcher's goal is to design this function in such a way that mathematical optimization leads to the model with desired properties. In the last few years, as is manifested in the superhuman performance of these models, the research was very successful. An impressive property of these systems is the grace with which they generalize beyond faces present in the training dataset. This makes it much easier to determine that the person is not in the set of known identities. This is crucial for biometric authentication.

The improved accuracy and reliability of facial recognition technology led to a widespread deployment of these systems. These systems found their place in our phones giving them the ability to unlock automatically; they found a place on the servers of big technology companies, like Google who uses the technology to better organize our photos; the technology is also at the heart of modern surveillance systems.

The last-mentioned use case is the reason why the deployment of these algorithms is plagued by controversy. As the saying goes, technology is a double edged sword. Facial recognition is the embodiment of this saying. On the positive side, there were many cases in which the technology helped authorities capture dangerous individuals [2]. However, no matter how good this benefit is, for many, it is not worth the risk of losing the privacy. This is the case in San Francisco, where the legislators voted to ban the use of the technology by law enforcement [3].

1.1 Goals

The goals of this thesis are:

1. to provide an overview of modern facial recognition methods;
2. to implement a state-of-the-art algorithm;
3. to evaluate the system's performance on appropriate benchmark dataset;
4. to compare the results with commercial algorithm (EyeFace SDK 3.2.3).

1.2 Outline

This text consists of seven chapters with introduction being the first one.

The second chapter is about convolutional neural networks and the different types of layers hidden within their architecture.

In the third chapter I provide an overview of the chain of processes constituting typical facial recognition system (pipeline). In this chapter there is also a description of the most important benchmarking datasets and selected commercial systems.

Fourth chapter is a description of losses used in facial recognition tasks and the ideas leading to them. At the end of the chapter, there is a description of ArcFace research. This research plays an essential part in this project because the model used at the core of the system I have implemented was trained under the supervision of ArcFace loss.

In the fifth chapter, I define metrics which are later used for system evaluation.

The sixth chapter contains a description of the algorithm I have implemented. At the end of the chapter, there is a comparison with the commercial algorithm.

The last chapter is a conclusion.

Chapter 2

Convolutional Neural Networks

Convolutional Neural Network (CNN) is a class of Artificial Neural Networks[1], which allows for efficient training on high dimensional data. This is especially useful in the field of *computer vision*[2] as image data is fundamentally high dimensional.

This thesis is dealing with a subset of computer vision called *face recognition*3 in which CNNs achieve state-of-the-art results.

The typical architecture of CNNs contains many layers. Because of that, these models belong to the class of machine learning methods called *deep learning*[3] In the following section 2.1, I will deal with layer types used in CNNs.

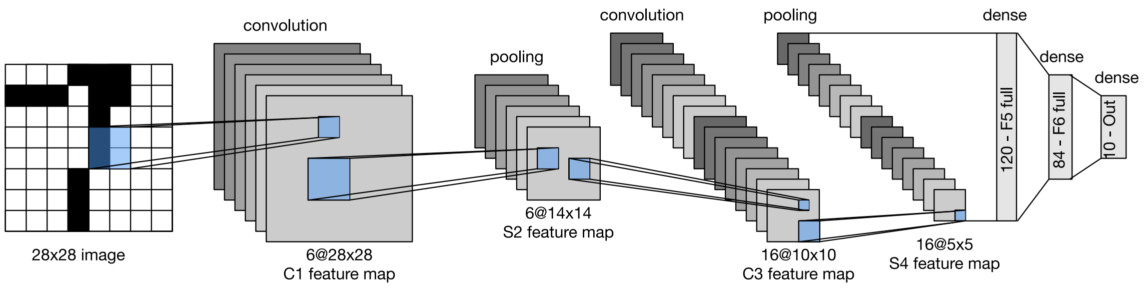


Figure 2.1: Example of CNN model (LeNet 5) [4]

[1]A computational model inspired by animal and human brains.

[2]A scientific field dealing with extraction of high level understanding from imagery using computers.

[3]A set of machine learning models with *credit assignment path (CAP)* higher than 2. The CAP is the chain of transformations from input to output.

2.1 Layer Types

The image 2.1 is an illustration of one of the oldest CNN models, called *LeNet-5*, which was used for handwritten character recognition. There are three layer types in the architecture: *dense* 2.1.1, *convolutional* 2.1.2, and *pooling* 2.1.3 layer. Modern CNNs also usually contain *locally connected layer* 2.1.4.

2.1.1 Dense/Fully Connected

The dense layer is the simplest type of layer present in CNN models. Its output is determined simply by multiplication of its *inputs* (x) with *weight matrix* (denoted W in 2.1) and addition of *bias* (b):

$$h[i,j] = b[i,j] + \sum_{k,l} W[i,j,k,l] \cdot x[i+k, j+l], \tag{2.1}$$

where $x\,[i,j]$ and $h\,[i,j]$ denote pixel location (i,j) in an image and hidden representation, respectively. The indices k, l run over both positive and negative offsets, covering the entire image.

To demonstrate the number of parameters needed, let's imagine that we want to feed a greyscale image which is 256 pixels high and wide to a dense layer. First we flatten the image, which results in a vector with $256 \cdot 256 = 65536$ dimensions. Even if we do aggressive reduction to 1000 hidden dimensions, we end up with approximately 65 million parameters. This makes the dense layer impractical when dealing with imagery. For this reason, a convolutional layer 2.1.2 is usually used as the input layer of computer vision models.

2.1.2 Convolutional

As I hinted in the previous section, the main advantage of CNNs [5] is the low amount of parameters needed. In the convolutional layer, this feat was achieved by the application of two principles: *invariance* 2.1.2 and *locality* 2.1.2.

Invariance Principle

The core of the invariance principle is a reuse of weights. This is achieved by the application of weights on one part of the image, shifting the weights by a predetermined set of pixels (called a stride), and then applying the weights again. Let us examine the equation 2.2 to see how the original equation 2.1 changes.

$$h[i,j] = b + \sum_{k,l} W[k,l] \cdot x[i+k, j+l] \tag{2.2}$$

As is to be expected, bias b and the weight matrix W are no longer dependent upon the image coordinates *(i, j)*. As an example, we can think of an airplane detection algorithm whose goal is to find whether there is an airplane present in any part of the scene. The core principle used during the algorithm design would be sliding one set of weights (kernel) describing the airplane over the image. The algorithm would then classify the scene with high impulse response as containing an airplane.

This type of invariance is called *translational invariance.*

Locality Principle

Another principle used in CNNs is called *locality principle.* This principle suggests, that we do not need to look far away from *(i,j)* to gain valuable information about what is going on in that particular location. This is achieved, mathematically speaking, by limiting k and l to a range Δ.

$$h[i,j] = b + \sum_{k=-\Delta}^{\Delta} \sum_{l=-\Delta}^{\Delta} W[k,l] \cdot x[i+k, j+l] \tag{2.3}$$

It is important to note that the range Δ is also known as the kernel size. Equation 2.3 is the final form describing the convolution layer.

2.1.3 Pooling

The main objective of pooling is to decrease the spatial dimension of the inner representation of the data. During classification (very common application of CNNs) we are not interested in the location of the classified object within the image. The only information that interests us is whether the object is present in the scene or not. With this information in mind, pooling [6] was invented.

The pooling layer is similar to the convolutional layer in a sense that both are "looking" only at a part of the input at once. This point of view (window) is then shifted as I described in section 2.1.2. What makes the pooling layer different from the convolutional one is that there is no kernel present in the operation. The only thing pooling does is that it selects/computes the most representative value from the window. There are two common pooling types: *max pooling* and *average pooling*. The first type selects the maximum value and the second one computes the average.

Stacking pooling layer on top of convolutional layer is a powerful combination as pooling makes the output of CNNs more robust to local translations[7].

2.1.4 Locally Connected

Locally connected layer is similar to convolutional layer but the difference is that every location in the feature map learns different set of filters. This layer type is, for example, used by the DeepFace system 3.2.1.

2.2 Other Concepts

This section is about concepts which are not defined as a layer but form an important piece of modern CNNs nonetheless. To name these concept specifically, they are Batch Normalization 2.2.1 and Rectifier 2.2.3 and Padding 2.2.2.

2.2.1 Batch Normalization

Batch Normalization is a method of speeding up the training process of Artificial Neural Networks (ANNs) [8]. According to the authors, this feat is achieved by a reduction of the negative impact of *internal covariate shift*. *Covariate shift* is a change in the input distribution of learning systems. When this concept is applied not to the whole system, but only to its part, we call this phenomenon an *internal covariate shift*.

During the training the change of input distribution is caused by adjustments of the parameters of the preceding layers. This makes the optimal parameter setting in the hidden layers a moving target which dramatically slows down the convergence.

This issue is alleviated by an incorporation of normalization into the model architecture, i.e., by fixing the means and variances of layer inputs.

The mean and variance is computed over the current batch which is the reason why this method is called Batch Normalization.

However, new research has shown that the *internal covariate shift* might not be the main reason for the improved speed of convergence [9]. The researches reintroduced the covariate shift into the training process by an injection of i.i.d. noise sampled from a non-zero mean and non-unit variance distribution into the outputs of the Batch Normalization layer. Surprisingly, the change in the speed of convergence was negligible. According to the research paper, the profound effects of Batch Normalization are caused by the reparametrization of the underlying optimization problem into a form, which is more stable and smooth. These two traits result in a training process with gradients that have higher predictive power. This enables faster and more effective optimization.

The output y_i of the normalization step is computed using the following formula:

$$y_i = \gamma \bar{x}_i + \beta, \tag{2.4}$$

where $\bar{x}_i$ is the average of the current batch. Parameters γ and β are adjusting scale and shift, respectively. These parameters are learned during training. They were incorporated into the formula in order to restore the representative power of the network.

2.2.2 Padding

To avoid loss of information at the edges of the image we usually use a method called *padding*. This technique increases the image dimension by pixel addition around the original image. There are few padding variants which are differentiated by the value of the new pixels. The most common ones are *zero padding* and *reflective padding*. The first-mentioned type, as the name implies, sets the new pixels to zero. The second one is more sophisticated and consists of mirroring of the neighboring pixels.

2.2.3 Rectifier

Another important innovation of modern CNNs is rectifier activation function[4] defined as:

$$\text{ReLU}(x) = max(0, x), \tag{2.5}$$

where x is the scalar input.

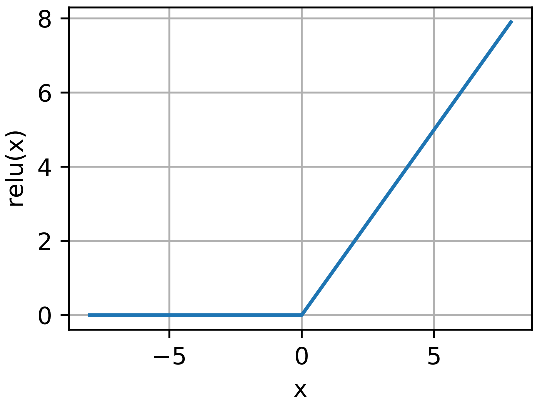


Figure 2.2: Illustration of ReLU activation function [10]

A unit implementing this activation function is called a rectifier linear unit (ReLU) 2.5. In CNNs ReLUs are usually positioned at the output of convolutional layer 2.1.2.

The advantage of ReLU is its efficacy during the training and the reduced likelihood of a vanishing gradient[5].

[4]The activation function of a node defines the output of that node given an input or set of inputs.

[5]A situation where a deep neural network is unable to propagate useful gradient information from the output end of the model back to the layers near the input end of the model.

2.3 Modern Models

This section contains an overview of selected modern models used extensively in facial recognition tasks.

2.3.1 InceptionNet

InceptionNets are a class of models in which there are multiple kernel sizes operating at the same level. This is desirable because the right kernel size is dependent on how globally the information is distributed. A large kernel is preferred when the information is distributed globally and, vice versa.

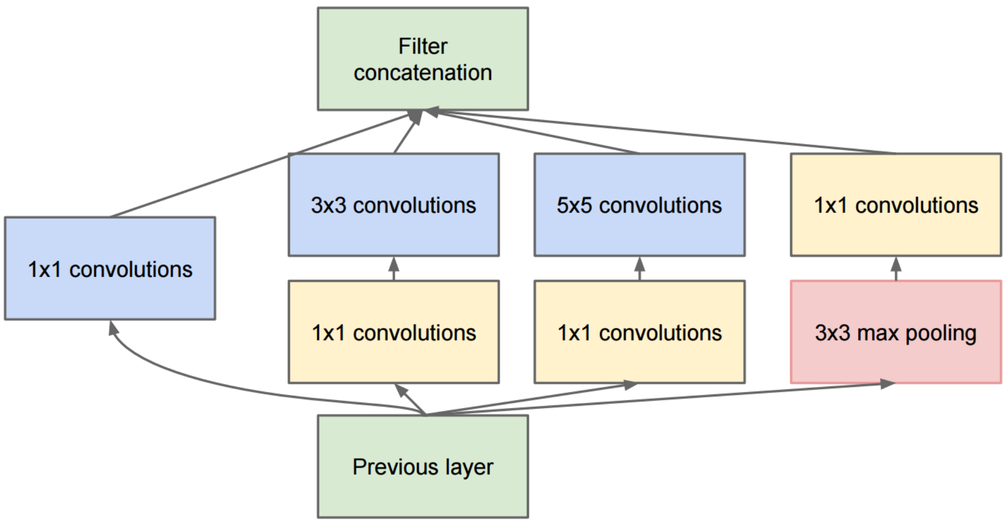


Figure 2.3: InceptionNet block [11]

The inception of the inception block (figure 2.3) took place in 2014 in the paper [11] published by Google Inc. Models using inception blocks kept on improving and at the time of writing, there is the fourth version in use.

2.3.2 ResNet

A residual neural network (ResNet) [12] is an ANN which allows for the training of very deep neural networks containing tens of layers. Training of ANNs this deep had been practically impossible before the invention of ResNet due to the problem of *vanishing gradient* and *degradation problem*. The first problem is exposed as a lack of convergence and the second one as a high training error.

Both of these problems have been avoided by implementation of *skip connections* which are illustrated in the figure 2.4.

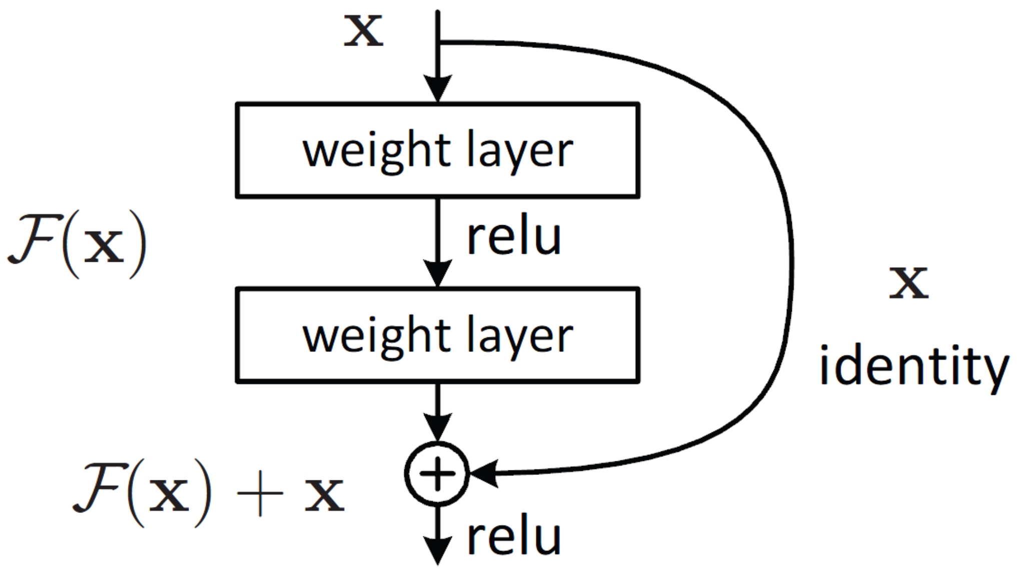


Figure 2.4: Residual learning [12]

The skip connections are implemented as elementwise addition and they let the in-between layers fit a residual mapping.

ResNets are widely used in the field of facial recognition as well as in the field of computer vision as a whole.

In the experimental part, I use 18-layer version called ResNet-18 (see section 6).

Architecture of ResNet-18

In the architecture, there are 17 convolutional layers and 1 dense layer. As is usual, the dense layer is placed at the output. Each of these layers is followed by batch normalization. The dimensionality of the first layer's feature map is reduced by max pooling. ReLu is used extensively within the whole architecture.

2.3.3 DenseNet

DenseNet [13] is a type of CNN 2 which was introduced in 2017. The difference from regular CNNs is that every layer is connected to all the subsequent layers. Traditional CNN with L layers has exactly L connections. On the other hand DenseNet with the same amount of layers contains $\frac{L(L+1)}{2}$ connections. The block of layers is illustrated in the figure 2.5.

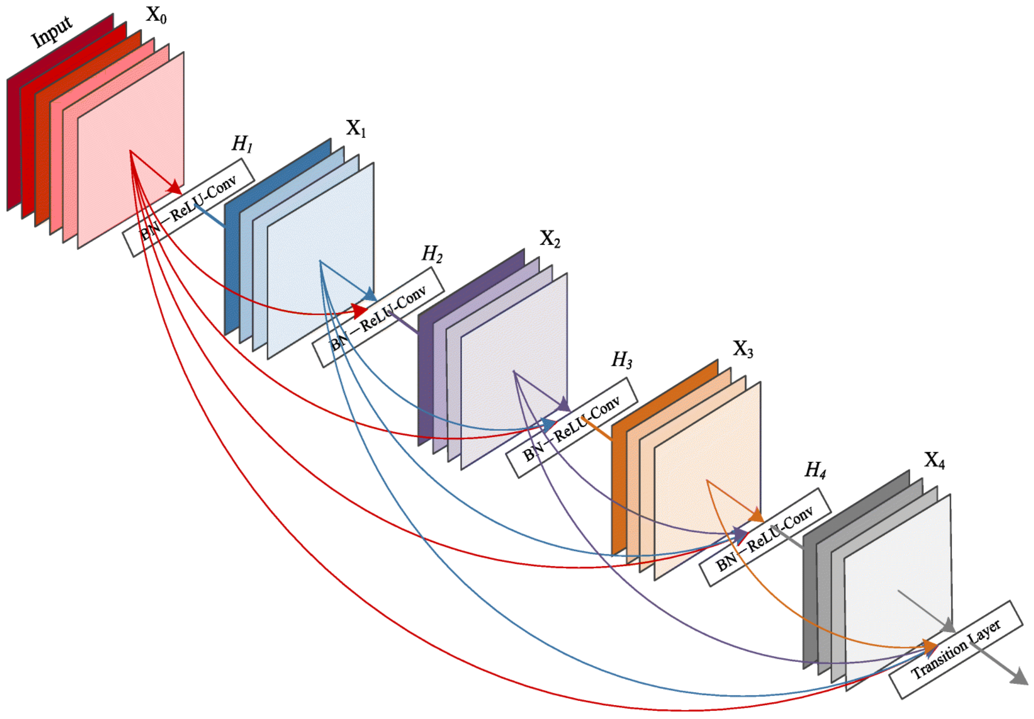


Figure 2.5: A 5-layer dense block [13]

DenseNets implement the connections with preceding layers differently than ResNets. While ResNets use elementwise addition, DenseNets concatenate the feature maps. The big size of the feature map is not an issue because it is fed as an input into a convolutional layer, and consequently, the training does not require training more parameters.

DenseNets have several compelling advantages [13]. They alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

Chapter 3

Facial Recognition

Facial recognition is a task of verifying or identifying a person from digital image/video.

As I mentioned in the definition, there are two main subtasks [14]:

1. **Verification** deals with verifying whether the person in the image is who he claims he is. A typical modern use case of verification is smartphone unlocking with face. An example of such system is Face ID developed by Apple Inc.

2. **Identification** is a task of matching a person to an identity. To formulate it in another way, the goal of identification is to give us an answer to the question of who the person in the image is.

3.1 Pipeline

Facial recognition pipeline[1] usually has the following four steps:

1. The first one is **face detection**. As the name implies, it deals with the determination of face location within the image. Usually the output of the algorithm is face coordinates and facial landmarks. The landmarks are a set of coordinates marking important points of the face (eyebrows, nose, mouth, . . .). The knowledge of these points is necessary for the following step. An example of face detection system is described in section 3.3.

[1]A chain of processing elements, arranged so that the output of each element is the input of the next.

2. **Face alignment** is a task of changing the face position in such a way that it resembles the position of faces on which the feature extraction model was trained. In most of the instances, this step improves the accuracy.

3. **Feature extraction** is a process of computing a feature vector[2] from the face. Architectures of models used for the feature extraction were described in the previous chapter 2.

4. **Feature matching** uses the feature vector from the previous task to classify a person in the image. The algorithm uses a database of pre-computed feature vectors and compares them to the newly extracted one.

   If we are dealing with the **closed-set problem** [3], the identity associated with the feature vector which has the smallest distance from the extracted one is considered to be the identity of the person in the image.

   For the **open-set problem** [4], the process is similar with the difference being an addition of a threshold. This thresholds states the maximum distance from the closest pre-computed feature vector for the newly extracted vector to still be classified as that identity. In case this condition is not met for any of the identities in the database, we establish the vector as a new identity.

It is important to note that the second step is not always present and it is deemed unnecessary by some [15].

[2]A feature vector is a vector that contains information describing an object's important characteristics.

[3]Identifying samples which were present in the training dataset.

[4]Identifying samples which were not present in the training dataset.

3.2 Face Recognition Systems

There are two main approaches of training CNNs for face recognition.

The first one is to train a multi-class classifier which can separate identities directly. An example of such system is DeepFace 3.2.1.

The second approach is to learn embedding using the triplet loss 4.2 function or similar. FaceNet 3.2.2 is an example of a system being trained using the second approach.

3.2.1 DeepFace

DeepFace [7] is a system developed by FaceBook Inc. in 2014.

The research is notable for its use of advanced alignment technique which consists of three steps:

1. **2D Alignment**

   In this step, the image is aligned in such a way that the fiducial points/landmarks are in a similar position to predetermined reference positions. To carry out this process it is first necessary to detect the 6 fiducial points/landmarks. These points and the reference positions are then used to find the parameters of an affine transformation. Applying the transformation to the original image gives us the desired result.

2. **3D Alignment**

   In the second step, the image is warped onto a generic 3D shape model. This is achieved by localization of 67 fiducial points in the image and then fitting an affine camera[5] P using the generalized least squares solution and the reference position x_{3d} of points on the 3D shape model.

3. **Frontalization**

   This is the final step and it consist of a computation and application of a piece-wise affine transformation T from x_{2d} source to $\tilde{x_{3d}}$ target. The target $\tilde{x_{3d}}$ is

[5]linear mathematical model to approximate the perspective projection followed by an ideal pinhole camera.

a list of positions of reference fiducial points from the previous step enriched with residuals r. These residuals were added to the reference positions $\tilde{x_{3d}}$ to account for non-rigid deformations which are not modeled by the affine camera P. Without these residuals, all faces would be warped into the same shape losing important discriminative factors.

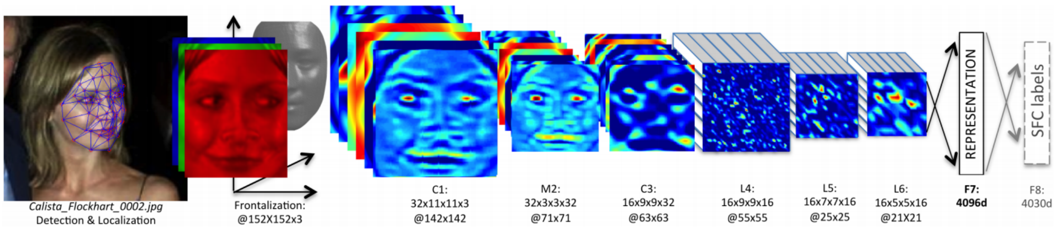


Figure 3.1: Outline of DeepFace architecture [7]

There are 9 layers in the model with over 120 million parameters. The process of classification is visualized in the picture 3.1. The model was trained on more than 4 million images and as the name of the research paper [7] implies, the results (**97.35%** on LFW dataset 3.4.1) almost matched the results of humans (**97.53%** on LFW dataset).

3.2.2 FaceNet

FaceNet [15] is a system developed by researchers at Google Inc. in 2015.

An interesting innovation of FaceNet is the format of its output. The output of the network is a vector representing a position in an euclidean space (so called embeddings) instead of a number representing an identity. This approach allows for straight-forward implementation of *verification* and *identification* 3. The implementation of verification involves thresholding the distance between the reference and the newly obtained embedding; and identification becomes k-NN classification problem.

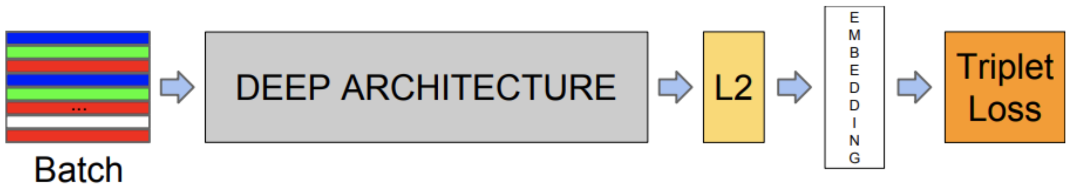


Figure 3.2: Outline of FaceNet architecture [15]

The loss function used to train the model is called *triplet loss* 4.2. Researches at Google came up with a new online method[6] which ensures that the difficulty of triplets is rising as the network trains.

The advantages of the model are its accuracy and the compactness of the face representation. The accuracy exceeded that of human with **99.63%** on LFW dataset 3.4.1 and the euclidean space has only 128 dimensions.

Another advantage is how well the model handles faces which are not in ideal position. This removed the need for complex preprocessing and face frontalization. To use proper terms the system is *pose-invariant*.

3.2.3 EyeFace SDK

EyeFace SDK is a library providing face detection, face recognition, etc. developed by Eyedea Recognition s. r. o.. One of the goals of this thesis is to exceed the performance of this commercial algorithm. The SDK is closed source; therefore, the implementation details are not known.

This library is mentioned in this thesis because it is used at the core of the facial recognition system developed by the Department of Cybernetics. The system is a search engine, where the input is a name and the output is a list of occurrences of the corresponding identity within the czech television broadcast. One of the goals of this thesis is to exceed the accuracy of this system within the same context (Czech News dataset 3.4.5).

[6]Training samples are selected during training.

3.3 Face Detection

As I mentioned in the description of a facial recognition pipeline, the goal of face detection is to find the location of the face within the image. This is challenging in unconstrained environments due to various poses, illuminations and occlusions.

To not digress too much from the main topic, I will describe only the system which was employed in the experimental part of this thesis. The detection algorithm is called MTCNN 3.3.2.

Before going through the process of face detection, it is necessary to describe a method called Non-maximum Suppression.

3.3.1 Non-maximum Suppression

Non-maximum Suppression (NMS) is a filtering algorithm of overlapping bounding boxes[7]. NMS consists of five simple steps:

1. Create a list of proposal bounding boxes ordered by the confidence score.
2. Select the bounding box with highest confidence score and add it to the filtered list of boxes.
3. Compute IoU 6.2 between the selected bounding box and all the remaining ones.
4. Remove all the boxes whose IoU is higher than some predetermined threshold.
5. Go to 1 and repeat the process until there are no remaining bounding boxes within the original list.

Having NMS defined we can proceed with actual face detection.

3.3.2 MTCNN

MTCNN [16] stands for Multi-task Cascaded Convolutional Networks. This model consists of three stages.

[7]A rectangle describing face position.

Stage 1

The first stage is called **Proposal Network** (P-Net) and its role is to find the candidate windows and their bounding box regression vectors. P-Net is fully convolutional neural network.

Before passing the image to P-Net we resize the image to many different sizes. By doing so we make the model scale-invariant.

Now we feed the images to the net.

The net produces many bounding boxes with a varying confidence. We parse the output and delete the boxes with low confidence score.

Now we standardize the coordinates by converting the boxes from the coordinate systems of the resized images to that of the unscaled one.

At this point we run NMS 3.3.1 once for every scaled image. Then we put all the survivors into one list and run NMS once more.

Before passing the boxes to stage 2 we make the boxes square by elongating the shorter sides.

Stage 2

The name of the second stage is **Refined Network** (R-Net). The purpose of this stage is to filter out a large number of false positives and to calibrate the boxes.

Initially, we take the boxes from the previous stage and copy the pixel values to separate arrays. In case the box is out of bounds we fill the ”empty space” with zeros.

Now we resize all the arrays to have the size of 24×24 pixels. We also normalize the pixel values to $< -1; 1 >$.

At this point, we feed the images to R-Net and collect the outputs.

The outputs are similar to that of P-Net. They also include the coordinates and the confidence levels. The difference is that the new coordinates are more accurate.

In the last few steps of the stage 2, we remove the boxes with a lower confidence and perform NMS to remove the redundant ones.

Now we standardize the coordinates and reshape the bounding boxes to a square.

Stage 3

In the last stage, we take the boxes from the stage 2 and copy the pixel values to separate arrays. If there are any boxes which cross the image bounds, we deal with them in the same way as in the previous stage, i.e., we fill the empty space with zeros.

Now we resize the images to be of 48×48 pixels and feed them into a neural network called **The Output Network** (O-Net).

O-Net is split into three layers at the top and as a result of this architectural choice produces three outputs: the coordinates of the bounding box, the coordinates of the facial landmarks and the confidence level of each box.

In the final processing step of the whole MTCNN algorithm, we get rid of the boxes with a low confidence score, we standardize the coordinates of both the landmarks and the boxes, and we filter the boxes with NMS.

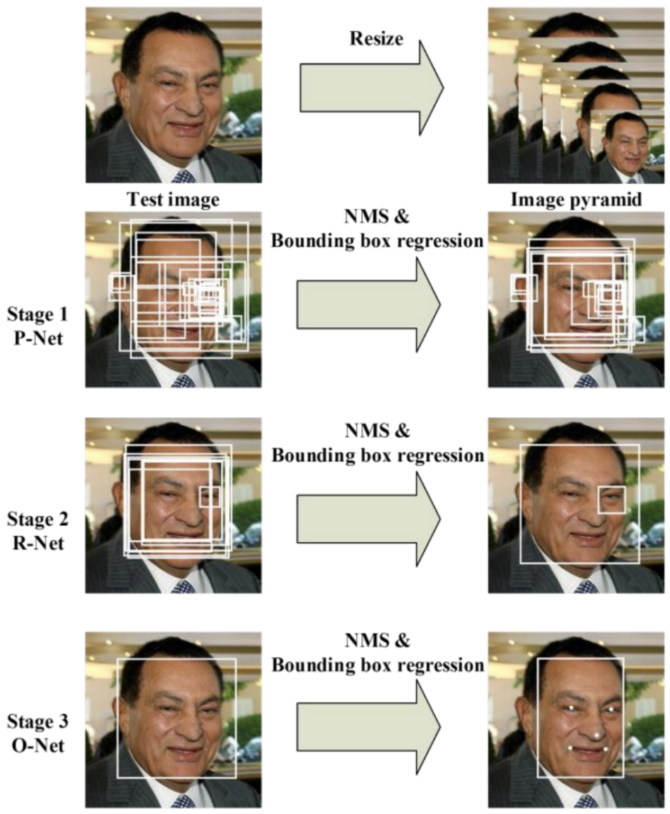


Figure 3.3: MTCNN face detection pipeline [16].

Figure 3.3 is a visualization of this three-stage process.

3.4 Datasets

In this section, I will briefly describe datasets used for training and evaluation of facial recognition models. There are too many different datasets used in practice. For this reason, I will focus only on those mentioned in this text.

3.4.1 LFW

LFW is an acronym for Labeled Faces in the Wild [17]. The dataset contains 13,000 images and 1680 identities. Every identity is represented by at least two samples. The faces were detected by Viola-Jones face detector[8].

There are now four publicly used versions of the dataset. These versions are differentiated by the type of preprocessing (different methods of alignment) applied to the images.

3.4.2 YTF

YTF stands for YouTube Faces [18]. The data set contains 3425 videos and 1,595 unique identities. The average length of the video clip is 181.3 frames and there are on average 2.15 videos for each subject.

3.4.3 MS-Celeb-1M

MS-Celeb-1M is a dataset constructed by Microsoft Research [19]. There are 10 million face images with nearly 100,000 individuals. The data were harvested from the Internet.

Due to the method with which the images were collected, there are many mislabellings in the dataset. For this reason, there are different versions available on the Internet containing refined data (like MS1MV2).

As the name implies, the dataset contains images of celebrities. In this context, celebrity is assumed to be anyone with frequent online presence. This became a controversial issue, and as a result, Microsoft pulled the dataset off the Internet.

[8]Real-time object detection framework.

3.4.4 CASIA WebFace

CASIA WebFace [20] is a dataset used for scientific research of unconstrained face recognition. There are approximately 500,000 images and more than 10,000 identities in the database. The images were crawled [9] from the Internet by Institute of Automation, Chinese Academy of Sciences.

3.4.5 Czech News

Czech News is a dataset which was created from the recordings of evening news on the czech public television broadcast (Česká Televize). The dataset consists of 15 videos and the same amount of annotation files in *json*[10] format.

In every annotation file (visualized in figure 3.4), there is a dictionary object, where key is a name, and value is a list of detections. Every detection contains information about the frame (frame number) and the position of the face (bounding box). This information is later used to process the dataset (section 6.1). The reference face position was selected by a human.

In the dataset there are 1238 identities and 732607 detections.

As I mentioned in the section about EyeFace SDK 3.2.3, this dataset is used to evaluate my implementation of the face recognition algorithm (chapter 6).

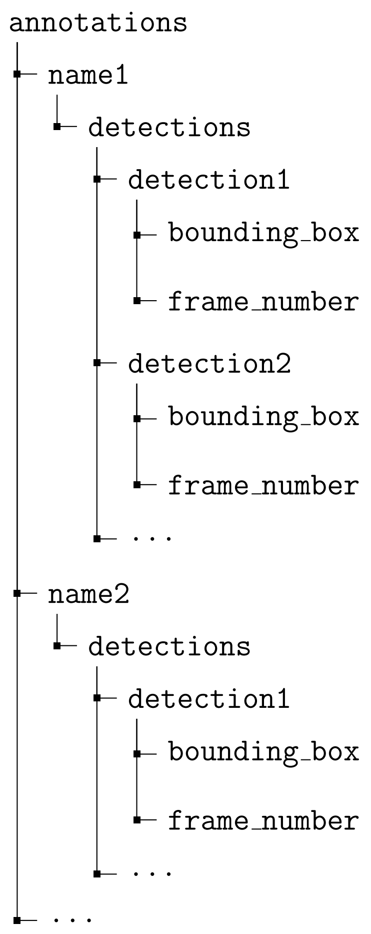


Figure 3.4: Format of annotations

[9]Automatically located and downloaded from the Internet.

[10]An open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types.

Chapter 4

Loss Functions

The main objective of the research of loss functions is to design a function which leads to a model with superior discriminative abilities. How well the model discriminates can be measured by the compactness of clusters (intra-class variance) and the distance between them (inter-class variance). The goal is for the clusters to be as compact as possible while maximizing the distance in-between them.

This chapter focuses on the research of loss functions used in facial recognition tasks.

4.1 Softmax Loss

Softmax [21] is the most widely used loss function in general classification tasks. The definition of the loss is as follows:

$$\mathcal{L}_S = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j-1}^{n} e^{W_j^T x_i + b_j}}, \tag{4.1}$$

where $x_i \in \mathbb{R}^d$ denotes the feature vector of the i-th sample belonging to the y_i-th class. $W_j \in \mathbb{R}^d$ is the j-th column of the weight matrix $W \in \mathbb{R}^{d\times n}$ and b is the corresponding bias term. N is the batch size and n is the class number.

There is one major drawback of softmax loss. It doesn't encourage cluster compactness. In other words, it fails to guarantee similarity among samples within a category. This makes the learned features not discriminative enough for the open-set face recognition problem.

Another issue is the dimension of the output weight matrix which grows linearly with the number of identities in the training set. This makes softmax loss impractical for a large scale deployment.

4.2 Triplet Loss

Triplet loss [22] is a loss function which can be optimized by minimizing the distance between *anchor* and *positive* point while maximizing the distance between *anchor* and *negative* point. These points are represented by vectors in the feature space. The learning process is illustrated in the figure 4.1.

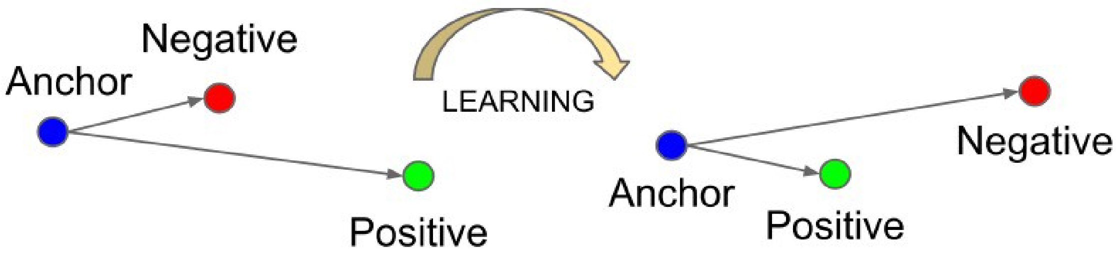


Figure 4.1: Illustration of the triplet loss [22]

Anchor and positive points belong to the same class, whereas negative point belongs to another one. In the context of facial recognition one class is one identity. This makes the anchor and the positive point a vectorised representation of two images of one face.

In mathematical terms the loss can be described using the Euclidean distance function as follows:

$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p))\|_2^2 - \|f(x_i^a) - f(x_i^n))\|_2^2 + \alpha \right], \tag{4.2}$$

where $f(x_i^a)$, $f(x_i^p)$ and $f(x_i^n)$ are the feature vectors of the anchor and the positive and the negative point. The index i denotes the triplet. N is the number of triplets in the dataset.

The drawbacks of *triplet loss* are the demands entailed by the construction of the triplets. The number of those is subject to a combinatorial explosion and inevitably results in a slow convergence and instability. This is a serious issue especially for large datasets.

4.3 Center Loss

The main goal of this research [23] was to design a loss function which solves the main drawbacks of the two previously mentioned losses (*sofmax loss* 4.1 and *triplet loss* 4.2).

The discriminative power of the learned features is enhanced by incorporation of the distance of features to the corresponding class centers. In other words, the network is penalized whenever the features are too far from the class center. In the course of training the the centers are updated and the distances of features from centers are minimized. The model is trained under a joint supervision of softmax loss and center loss. The effect of these two losses is balanced by a hyperparameter[1] λ. This joint supervision results in a loss function which combines the best of both worlds: inter-class discriminative power of *softmax loss* and intra-class distance minimization of *center loss*.

The center loss function is intuitively defined by the following equation:

$$\mathcal{L}_C = \frac{1}{2}\sum_{i=1}^{m} \|x_i - c_{y_i}\|_2^2 , \tag{4.3}$$

where $c_{y_i} \in \mathbb{R}^d$ denotes the y_i-th class center of the features. However, recomputing the centers by taking the average of the features over the whole training set is too inefficient and impractical.

To address this problem, the authors proposed updating the centers based on a mini-batch instead of the whole training set. To avoid large perturbations caused by mislabeled samples, there is a parameter λ with which we control the learning rate of the centers.

[1]A parameter which is not a subject to the learning process

The equation 4.5 defines the center update:

$$\frac{\partial \mathcal{L}_C}{\partial x_i} = x_i - c_{y_i} \tag{4.4}$$

$$\Delta c_j = \frac{\sum_{i=1}^{m} \delta(y_i = j) \cdot (c_j - x_i)}{1 + \sum_{i=1}^{m} \delta(y_i = j)}, \tag{4.5}$$

where $\delta(condition)$ is 1 if the condition is satisfied and 0 otherwise.

The final loss is described by the following formula:

$$\mathcal{L} = \mathcal{L}_S + \mathcal{L}_C = -\sum_{i=1}^{m} \log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j-1}^{n} e^{W_j^T x_i + b_j}} + \frac{\lambda}{2} \sum_{i=1}^{m} \|x_i - c_{y_i}\|_2^2\,, \tag{4.6}$$

Figure 4.2 is a great visualization of the effect of hyperparameter λ upon the cluster compactness. As is to be expected, higher values of the parameter result in clusters which are more compact.

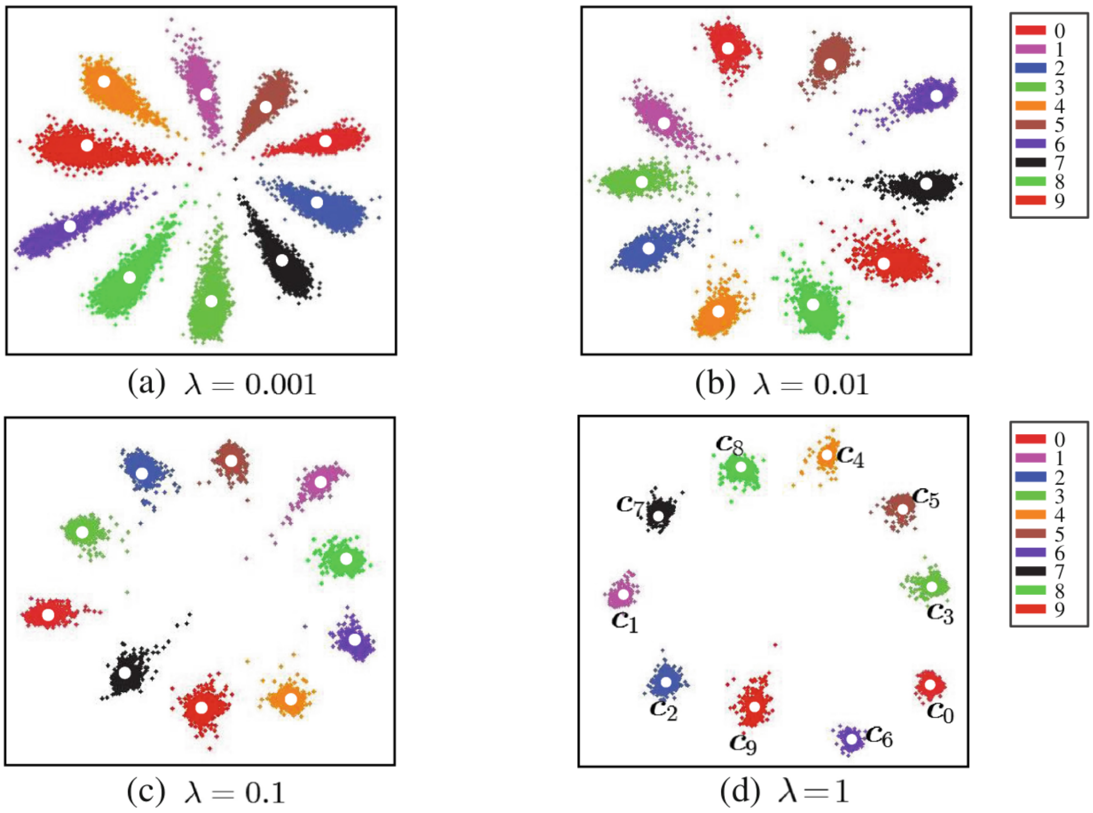


Figure 4.2: Distribution of the features for different values of hyperparameter λ [23]. The white dots $(c_0, c_1, \ldots, c_9)$ denote 10 class centers.

4.4 Congenerous Cosine Loss

Congenerous Cosine Loss [24] (known as COCO loss) is a research presenting a loss function which was published in 2017.

To define the COCO loss we first have to clarify how the centroid of class k is computed. The centroid[2] equation is the following:

$$\boldsymbol{c}_k = \frac{\sum_{i\in\beta} \delta\left(l_i, k\right) \boldsymbol{f}^{(i)}}{\sum_{i\in\beta} \delta\left(l_i, k\right) + \epsilon} \in \mathbb{R}^{D\times 1}, \tag{4.7}$$

where β is a mini-batch, D is the feature space dimension, δ is an indicator function, ϵ is a trivial number for computation stability, $\boldsymbol{f}^{(i)}$ is the i-th feature vector and l_i is its label.

Having the centroid equation defined, we can proceed with a definition of the term we are trying to maximize during a model fitting:

$$p_{l_i}^{(i)} = \frac{expC(\boldsymbol{f}^{(i)}, \boldsymbol{c}_{l_i})}{\sum_{k\neq l} expC(\boldsymbol{f}^{(i)}, \boldsymbol{c}_k)} \in \mathbb{R}, \tag{4.8}$$

where C is standard cosine similarity.

Now we can move to the final COCO loss function definition:

$$-\sum_{i\in\beta} \log\left(p_{l_i}^{(i)}\right). \tag{4.9}$$

4.5 SphereFace Loss

SphereFace [25] is a loss function which was published in 2018. The research significantly distinguishes itself from the previously mentioned losses by not relying on an euclidean margin. SphereFace uses an angular margin instead. This has proven to be highly effective in face recognition tasks. The name of the loss hints about how the features are transformed during the loss computation. The features are projected onto a hypersphere manifold.

SphereFace originates from the softmax loss 4.1. To derive SphereFace from softmax, we first incorporate the angle into the softmax equation using the dot

[2]Center of cluster.

product definition ($a \cdot b = \|a\| \, \|b\| \cos\theta$):

$$\begin{aligned}
\mathcal{L}_S &= -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j-1}^{n} e^{W_j^T x_i + b_j}} \\
&= -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{\left\|W_{y_i}\right\|\|x_i\|\cos(\theta_{y_i,i}) + b_{y_i}}}{\sum_{j-1}^{n} e^{\|W_j\|\|x_i\|\cos(\theta_{j,i}) + b_j}},
\end{aligned}$$

where $\theta_{j,i}$ is the angle between vector W_j and x_i. The meanings of the remaining symbols are equal to those in the softmax equation 4.1.

Next we normalize $\|W_j\| = 1, \forall j$ and set the bias term to 0.

$$\mathcal{L}_{modified} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{\|x_i\|\cos(\theta_{y_i,i})}}{\sum_{j-1}^{n} e^{\|x_i\|\cos(\theta_{j,i})}} \tag{4.10}$$

While it's possible to learn features with the modified loss the result would not be discriminative enough. To solve this issue, the researches incorporated angular margin:

$$\mathcal{L}_{ang} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{\|x_i\|\cos(m\theta_{y_i,i})}}{e^{\|x_i\|\cos(m\theta_{y_i,i})} + \sum_{j\neq y_i} e^{\|x_i\|\cos(\theta_{j,i})}}, \tag{4.11}$$

where $\theta_{y_i,i}$ lies in $\left[0, \frac{\pi}{m}\right]$.

The decision boundary for a binary case is defined by:

$$\cos m\theta_1 = \cos\theta_2, \tag{4.12}$$

where θ_i is the angle between the feature and weight of class i.

To make the loss 4.11 optimizable for CNNs the definition range of $\cos(\theta_{y_i}, i)$ is expanded. This is achieved by replacing the cosine term with monotonically decreasing angle function $\Psi(\theta_{y_i}, i)$

$$\mathcal{L}_{ang} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{\|x_i\|\Psi(m\theta_{y_i,i})}}{e^{\|x_i\|\Psi(m\theta_{y_i,i})} + \sum_{j\neq y_i} e^{\|x_i\|\Psi(\theta_{j,i})}} \tag{4.13}$$

The angle function has the following definition:

$$\Psi(\theta_{y_i,i}) = (-1)^k\cos(\theta_{y_i,i}) - 2k, \tag{4.14}$$

where $k \in [0, m-1]$. The parameter $m \geq 1$ gives us control over the angular margin size.

4.6 CosFace Loss

CosFace [26] is another loss using margin to improve the discriminative power of softmax 4.1.

The decision boundary for a binary case is defined by the following equation:

$$\cos\theta_1 - m = \cos\theta_2, \tag{4.15}$$

where the symbols have exactly the same meaning as in the equation 4.12 of SphereFace decision boundary. Since the decision boundary of CosFace is not being defined over the angular space, the loss is easier to optimize than SphereFace. Optimization in angular space is more difficult because of the non-monotonicity of the cosine function.

Another innovation over SphereFace is the fact that not only is the weight vector W_j normalized, but the feature vectors x_i as well. This results in a much lower intra-class variability of the learned features as the emphasis is being put on the angle during training. By fixing $\|x\|$ as some predetermined radius s in equation 4.10 we get:

$$\mathcal{L}_{ns} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{s\cos(\theta_{y_i,i})}}{\sum_{j-1}^{n} e^{s\cos(\theta_{j,i})}} \tag{4.16}$$

This loss is called Normalized Softmax Loss (NSL) in the CosFace research paper. NSL emphasizes correct classification but it is not discriminative enough for face recognition tasks. For this reason (as was mentioned in the SphereFace research as well), the margin is incorporated:

$$\mathcal{L}_{lmc} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{s\left(\cos(\theta_{y_i,i})-m\right)}}{e^{s\ \left(\cos(\theta_{y_i,i})-m\right)} + \sum_{j\neq y_i}^{n} e^{s\ \cos(\theta_{j,i})}}, \tag{4.17}$$

subject to

$$W = \frac{W*}{\|W*\|}, \tag{4.18}$$

$$x = \frac{x*}{\|x*\|}, \tag{4.19}$$

$$\cos(\theta_j, i) = W_j^T x_i. \tag{4.20}$$

The meaning of the equation constituents is equivalent to those in the previous sections.

The equation 4.17 is the final form of the CosFace loss. The authors of the research call it the Large Margin Cosine Loss (LMCL).

4.7 ArcFace Loss

ArcFace [21] is a research which became public in 2018 and achieved state-of-the-art results on LFW dataset.

Most of the ideas leading to ArcFace were already described in SphereFace 4.5 and CosFace 4.6 sections. To name these ideas, they are normalization of class weights and feature vectors, and incorporation of margin in the loss function equation. These two ideas improve the intra-class variance in angular space resulting in a model with better discriminative abilities in the area of facial recognition tasks.

Like SphereFace and CosFace, ArcFace originates in the equation of *softmax loss* 4.1 as well. There are five steps separating the original and the improved version. The first four are equivalent to those in CosFace:

1. fix the bias $b_j = 0$;
2. transform the logit using the dot product definition $W_j^T x_i = \|W_j\| \, \|x_j\| \cos\theta_j$ (θ is the angle between the weight W_j and the feature x_i);
3. fix the individual weights $\|W_j\| = 1$ by l_2 normalization;
4. do the same for feature x_i and re-scale it to a predetermined feature scale s.

The two normalization steps make the prediction depend only on the angle θ. The embeddings are distributed on the hypersphere with a radius s.

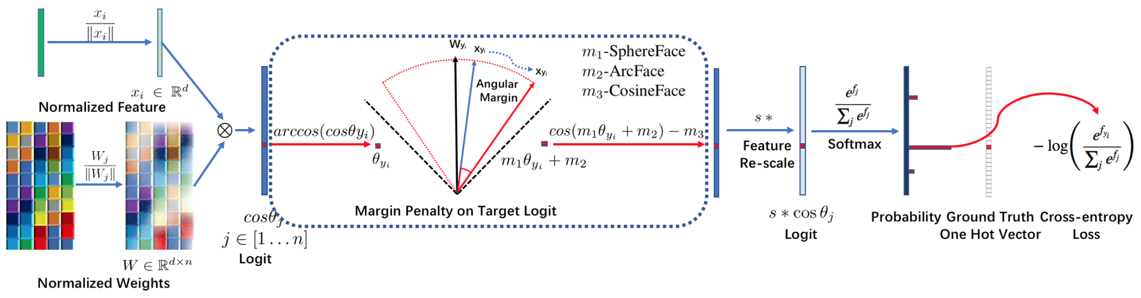


Figure 4.3: Training a CNN for face recognition supervised by the ArcFace loss [21]

At this point the loss function equation is as follows:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cos(\theta_{y_i,i})}}{e^{s\ \cos(\theta_{y_i,i})}+\sum_{j=1,j\neq y_i}^{n}e^{s\ \cos(\theta_{j,i})}}. \tag{4.21}$$

5. In the last step, the additive angular margin penalty m between x_i and W_{y_i} is incorporated. The margin is equal to the geodesic distance[3] on the hypersphere. This is the reason why the method is called *ArcFace*.

Final ArcFace loss function is defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cos(\theta_{y_i,i}+m)}}{e^{s\ \cos(\theta_{y_i,i}+m)}+\sum_{j=1,j\neq y_i}^{n}e^{s\ \cos(\theta_{j,i})}}. \tag{4.22}$$

It has been experimentally determined that the best performance is achieved for $m = 0.5$.

4.7.1 Comparison with Other Losses

By having a look at table 4.1 and figure 4.4 we can do a comparison of geometric differences of different decision margins.

[3]Distance of a curve representing shortest path between two points in a surface.

Loss Functions	Decision Boundaries
Softmax 4.5	$(W_1 - W_2)\, x + b_1 - b_2 = 0$
SphereFace 4.5	$\|x\| \left(\cos m\theta_1 - \cos\theta_2\right) = 0$
CosFace 4.6	$s\left(\cos\theta_1 - m - \cos\theta_2\right) = 0$
ArcFace	$s\left(\cos(\theta_1 + m) - \cos\theta_2\right) = 0$

Table 4.1: Comparison of the decision boundaries under the binary classification case

The advantage of *ArcFace* is its constant linear angular margin throughout the whole interval.

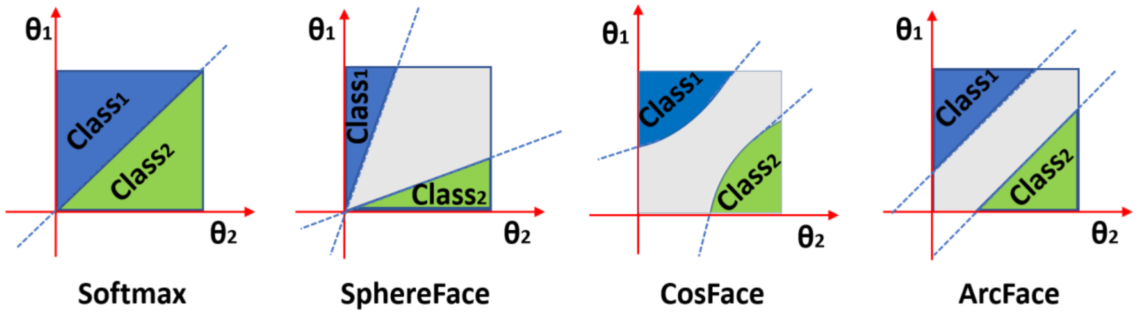


Figure 4.4: Decision margins of different loss functions under binary classification case. [21]

We can see the concrete percentual results in table 4.2. The performance of ResNet100 with ArcFace loss trained on MS1MV2 3.4.3 dataset exceeded other methods mentioned in this thesis.

Method	#Image	LFW 3.4.1	YTF 3.4.2
FaceNet 3.2.2	200M	99.63	95.10
Center Loss 4.3	0.7M	99.28	94.90
SphereFace 4.5	0.5M	99.42	95.00
CosFace 4.6	5M	99.73	97.60
MS1MV2 3.4.3, R100, ArcFace	5.8M	**99.83**	**98.02**

Table 4.2: Verification performance (%) of different methods on LFW and YTF datasets.

Chapter 5

Evaluation Metrics

In this chapter, I will define metrics used for performance evaluation.

Before doing so, it is necessary to describe constituents of the confusion matrix 5.1.

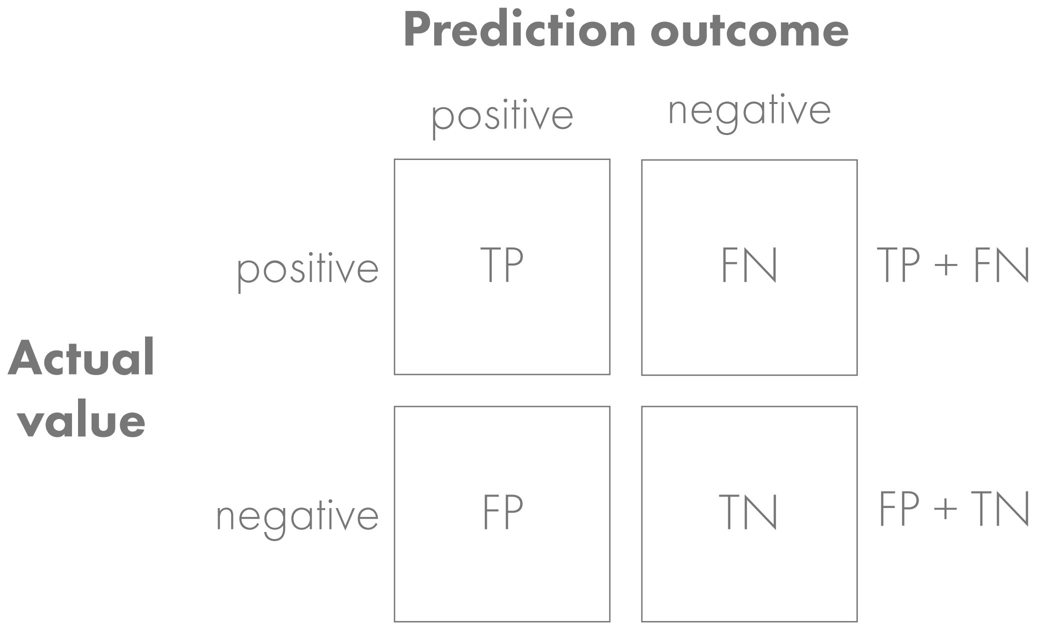


Figure 5.1: Confusion matrix [27]

True Positives (TP) is the number of actual positives being predicted as positive.

False Positives (FP), also known as **type I error**, is the number of actual negatives being predicted as positive.

False Negatives (FN) is the number of actual positives being predicted as negative. FN is called **type II error**.

True Negatives (TP) is the number of actual negatives being predicted as negative.

With these terms introduced, we can proceed with a definition of precision 5.1 and recall 5.2.

5.1 Precision

Precision [28] is the number of relevant instances among the retrieved instances. Mathematically speaking, precision is defined as the following fraction:

$$Precision = \frac{TP}{TP + FP}, \tag{5.1}$$

where TP and FP are the constituents of the confusion matrix 5.1.

A useful feature of precision is that it can be used to detect faulty model and dataset mislabelings. This property was extensively used during system implementation 6.

5.2 Recall

Recall [28] (also called sensitivity) is the fraction of the total amount of relevant instances that were actually retrieved. A definition of recall is the following:

$$Recall = \frac{TP}{TP + FN}, \tag{5.2}$$

TP and FN were described in the beginning of the chapter.

5.3 F_1-score

F_1-score is the harmonic mean of precision and recall. The metric is defined by the following formula:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{5.3}$$

The important property of this metric is that it successfully deals with skewed data. This scenario occurs when there is much more positives than negatives in the data or vice versa. A typical example is a classifier used to diagnose a rare illness. If the model always predicted false, the accuracy[1] would be high even though the system would be essentially useless.

[1]Sum of true positives divided by the number of predictions.

Chapter 6

System Design and Implementation

The system pipeline[1] consists of three separate processes: preprocessing of the Czech News dataset 6.1, feature extraction 6.2, and evaluation 6.3.

The algorithm is implemented in *Python* programming language. Libraries *NumPy* and *PyTorch* [29] are extensively used throughout the whole project.

6.1 Preprocessing of the Czech News Dataset

The goal of preprocessing is to convert the input dataset (Czech News 3.4.5) to the standard format. The standardized dataset consists of directories with each directory representing one identity/label. In these directories, there are images corresponding to the identity. In this case, the images contain faces in different positions.

The desired format is visualized in figure 6.1.

```
dataset
 ├─ name1
 │   ├─ image1_128x128.jpg
 │   ├─ image2_128x128.jpg
 │   └─ image3_128x128.jpg
 ├─ name2
 │   ├─ image1_128x128.jpg
 │   └─ image2_128x128.jpg
 └─ ...
```

Figure 6.1: Standard image dataset format

[1]See footnote on page 13

As I mentioned in the dataset description 3.4.5, the area containing face was selected by human. Because of that, the geometry of the area is not always consistent. This significantly decreases the model performance. For this reason, as is described in the section 6.1.1, this reference position is not used directly in the algorithm.

6.1.1 Preprocessing Algorithm

Before the presentation of the algorithm it is necessary to define "Intersection over Union (IoU)." As the name implies, IoU is computed as a fraction with intersection area in the numerator and union area in the denominator (see figure 6.2).

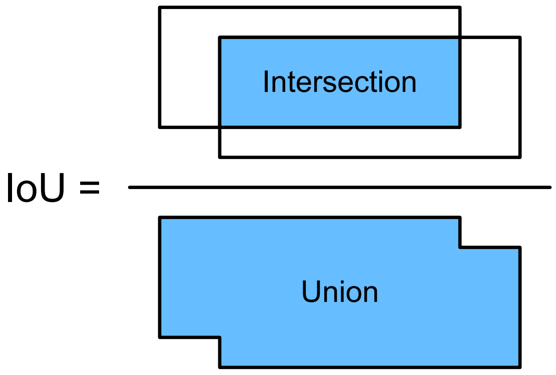


Figure 6.2: IoU visualization[30]

In the context of face detection each area is defined by a corresponding bounding box.

The preprocessing algorithm consists of seven steps:

1. First, I iterate over the annotation files.
2. Then I iterate over the names and detections within the file.
3. In the third step, I fetch the frame out of the video corresponding to the detection.
4. Then I detect all the faces in the frame using MTCNN detector. The detector returns coordinates of the bounding boxes[2] and the facial landmarks[3].
5. I select the detection which meets the following two conditions:

[2]See footnote on page 18
[3]Salient regions of the face.

(a) has the biggest IoU with reference of all the detections;

(b) the IoU value is at least **0.5** (This value has been determined experimentally as is described in the section 6.3.3.).

If these conditions are not met for any of the detections the frame is omitted.

6. At this point I perform face frontalization using the facial landmarks from step 4 to the predetermined reference position. This is achieved by using the least squares method to find an affine transformation[4] between the two sets of coordinates. Applying the transformation results in an image with facial landmarks in the similar position as the reference. To carry out the transformation *OpenCV* library is used. The implementation in this library allows for specification of the output size. I specified the output as 128×128 pixels because it is the input size of the ResNet-18 model.

7. In the last step, I save the image to a location defined by the standardized dataset format (`dataset/name/image_128x128.jpg`).

The processed dataset contains 791 identities and 478529 images. The conditions in step 5 were not met 254078 times. This resulted in the loss of the same amount of images and 447 identities. While the loss seems to be significant, the resulting amount of images is sufficient for system evaluation.

Having the dataset in the desired format we can proceed with feature extraction.

6.2 Feature Extraction

Feature extraction [31] is a process of dimensionality reduction by which an initial set of raw data is reduced to the set of feature vectors. In this study this process is carried out by feeding the input image $x_i \in \mathbb{R}^{128\times128}$ to the CNN model. On the output we retrieve a feature vector $y_i \in \mathbb{R}^{1024}$.

The model used is 18-layer ResNet 2.3.2 which was trained under a supervision of the ArcFace loss on CASIA WebFace dataset 3.4.4. The training process was not

[4]A function between affine spaces which preserves points, straight lines and planes.

executed as part of this study since the trained model is freely available online. The model was downloaded from the ArcFace implementation repository [32].

6.2.1 Feature Extraction Algorithm

The feature extraction algorithm consists of 6 steps:

1. iterate over the images in the dataset;
2. save the image and the flipped version into an array with shape $2\times1\times128\times128$;
3. feed the array to the ResNet-18 model;
4. retrieve the feature vector on the output;
5. save the vector along with the corresponding label into an array;
6. once all the feature vectors are computed, save the resulting array using the *h5py* module.

In the final array there are 478529 feature vectors (the number of images in the processed dataset). The whole file is approximately 2 GB in size.

Having the feature vectors computed, we can carry out the evaluation 6.3.

6.3 Evaluation

The system's performance is evaluated on the verification task. The task consists of computing cosine distance between every two feature vectors and deciding, given some threshold, whether the vectors correspond to the same identity ot not. This computation is carried out for all the threshold values in the specified interval.

The algorithm is described in depth in the following section 6.3.1.

6.3.1 Thresholding Algorithm

First, I would like to analyze the algorithm time complexity and memory demands.

As I previously mentioned, it is necessary to compute the distances between every two feature vectors in the dataset. The number of vector pairs is equal to

$$N_{pairs} = \frac{n\,(n+1)}{2} = \frac{478529\,(478529+1)}{2} \approx 114.50 \cdot 10^{9}, \tag{6.1}$$

where $n = 478529$ is the number of vectors in the dataset. If we computed all the distances using optimized matrix operations at once, we would need at least 460 GB of RAM memory.

Given that the algorithm time complexity is $O(n^2)$ and the big memory demands, it was desirable to split the distance matrix into smaller sub-matrices. Splitting the matrices allows for parallel processing as the sub-matrices can be processed on different CPU cores. This significantly reduced the computation time. My approach made the use of optimized matrix libraries possible while reducing the excessive memory demands.

There are 5 steps in the parallelized algorithm:

1. Initially, I generate all the interval pairs.

2. In the second step, I pass the intervals as an argument to the generator function. This function returns pairs of slices of the feature vector array and corresponding labels.

3. Next, I pass the generator function as an argument to the *imap* method from *multiprocessing* module. This method iterates over the generator function and passes the retrieved values to separate processes. With this step, parallelized processing is achieved.

4. In the fourth step, the matrix of cosine distances is computed between the two feature vector arrays and two label arrays using *cosine_distances* method from *sklearn* library [33]. Computing the matrix for labels allows for simple and efficient comparison of predictions and reference values using matrix operations.

5. Next, I convert the matrices to two long vectors.

6. Now I iterate over the list of threshold values ($[0, 0.05, 0.10, 0.15, \ldots 2]$).

7. I apply the threshold to the distances. This way I retrieve a vector of binary values.

8. I use the methods *sum* and *logical_and* along with elementwise binary value inversion to count the number of *true positives (TP)*, *true negatives (TN)*, *false positives (FP)* and *false negatives (FN)*.

9. In the last step, I do elementwise summation of the arrays which were returned from the parallel processes.

The output of the algorithm is an array with 200 rows and 4 columns. The rows correspond to the threshold values; columns to *TP*, *TN*, *FP* and *FN*.

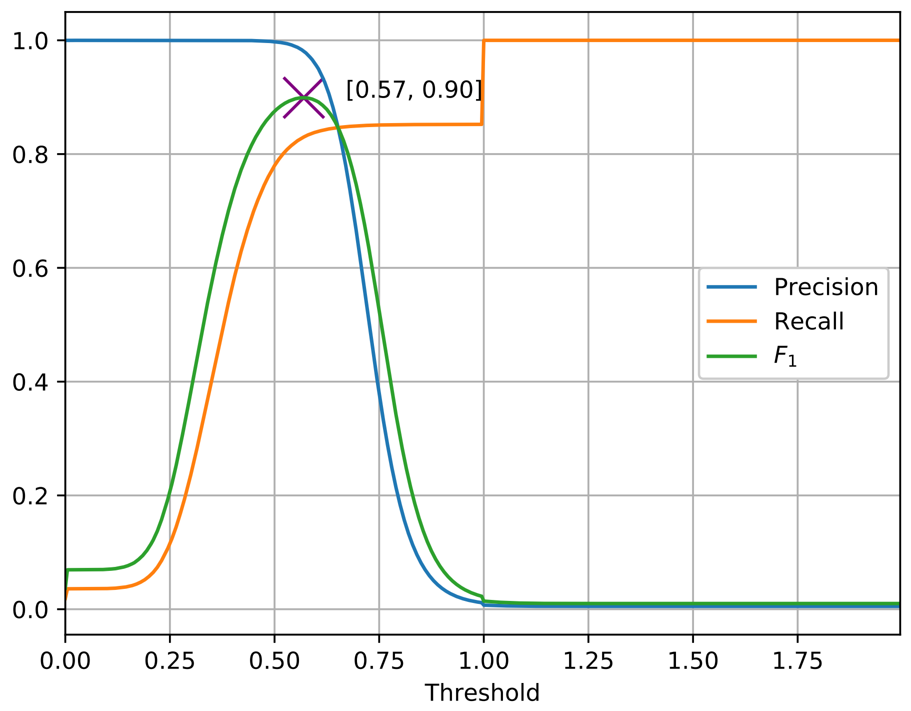
Having all these values accumulated we can proceed with computation of all the relevant metrics.

6.3.2 System Evaluation

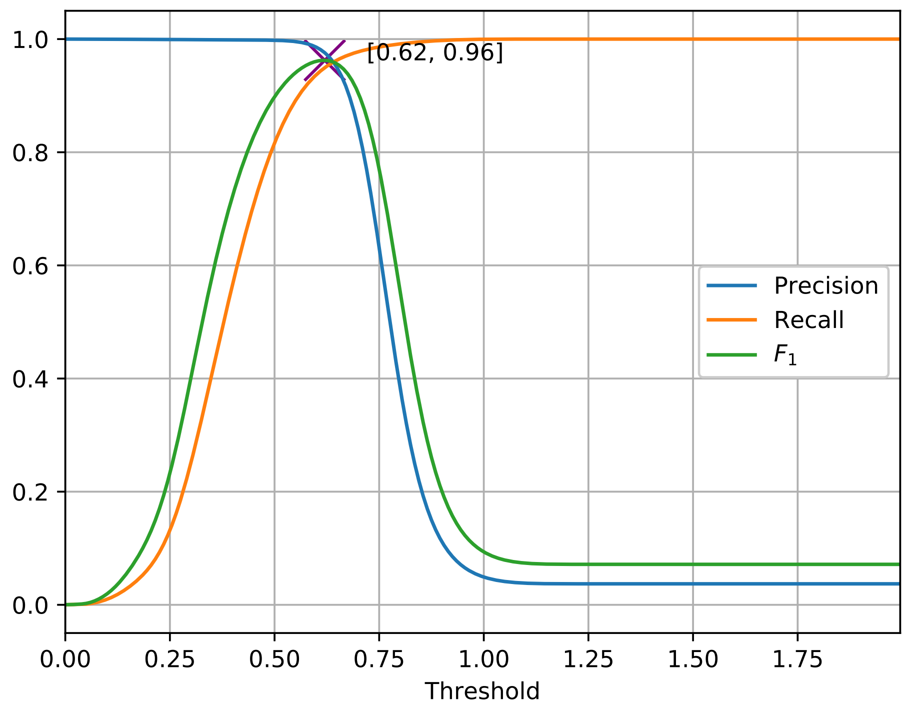
In order to evaluate the performance, I implemented an algorithm which computes precision 5.1, recall 5.2 and F_1 score 5.3 on the data from the previous section. The system is evaluated on LFW 3.4.1 and processed Czech News dataset 3.4.5. In figure 6.3 there is a progression of these metrics.

We can see that in comparison with LFW dataset the system achieved 6 % higher performance on the processed Czech News dataset.

LFW dataset was not aligned using the preprocessing algorithm described in section 6.1, which leads to a slightly different positions of faces than what the model is used to. This is probably the reason for the low F_1 score on the dataset.



(a) LFW dataset 3.4.1



(b) Processed Czech News dataset 3.4.5

Figure 6.3: Progression of precision, recall and F_1 score with increasing threshold values. The metrics were computed on the outputs of my system based on ResNet-18.

Comparison with Commercial System

In figure 6.4 there is a similar chart as the one above. The difference is, that the values were computed on the outputs of EyeFace SDK 3.2.3.

By comparing the chart below with chart 6.3b we can see that the custom implementation surpassed the commercial system by two percents.

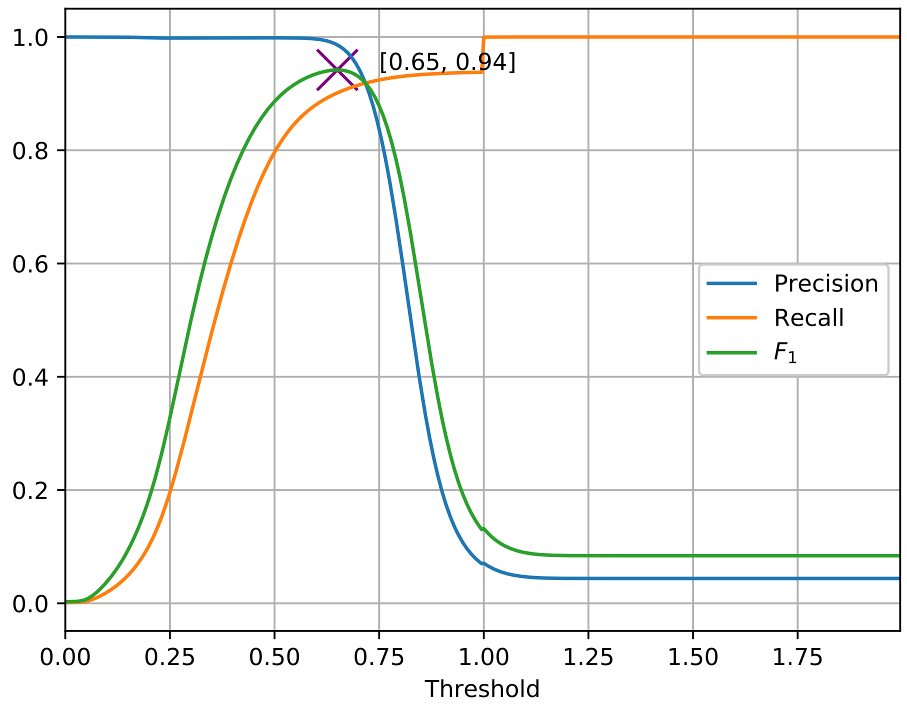


Figure 6.4: Progression of precision, recall and F_1 score with increasing threshold values. The metrics were computed on the outputs of commercial system (EyeFace SDK 3.2.3) from the Czech News dataset 3.4.5.

6.3.3 Detection of Mislabelings

As I mentioned in the section 5.1, the precision curve can be used to detect incorrect labels in the dataset. In figure 6.5 there are three functions dependent on the threshold.

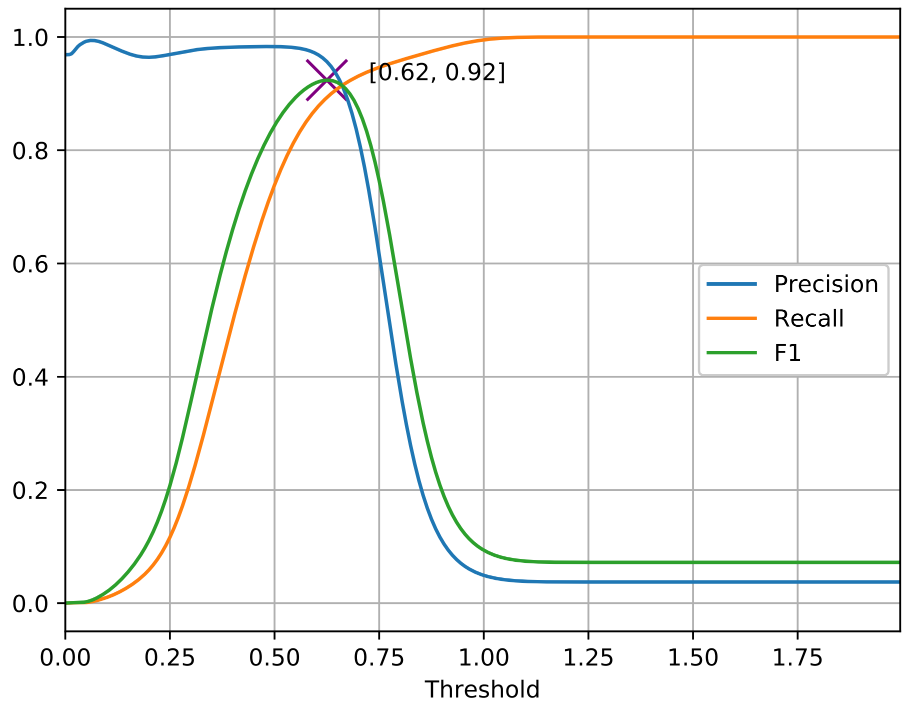


Figure 6.5: An example of precision curve 5.1 computed on a dataset containing errors.

By the definition of precision ($Precision = \frac{TP}{TP+FP}$), there should not be any false positives for threshold 0 because in such a setting, the system is strongly biased towards false negatives. With this in mind, precision value which is not equal to 1 for threshold 0 is a sign of error. This error can either reside in the dataset, or in the model. If the error is in the dataset, it means that there are mislabelings. If it is in the model, the whole system is faulty.

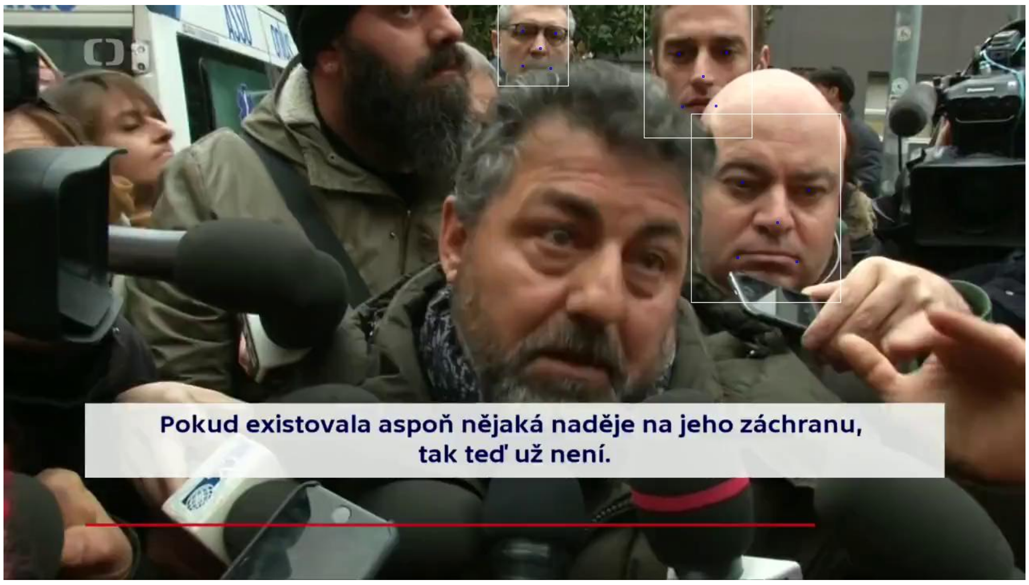


Figure 6.6: An example of a faulty detection.

This property of precision was used extensively during the experimentation. Figure 6.6 is an example of faulty face detection. Because of the white stripe, the man in the foreground was not detected by the MTCNN detector selecting the bald man on the right instead. This led to an error in the dataset. This issue was fixed by putting IoU threshold in place during the data preprocessing 6.1.

6.3.4 Face Detector Evaluation

In order to evaluate the detector, I've implemented an algorithm which computes recall 5.2 on the IoU values between the MTCNN detection and the area selected by human. The process is very similar to the one described in section 6.3.1.

The output is visualized in the image 6.2.

The fact that recall ($\frac{TP}{TP+FN}$) never gets close to 1 is indicative of significant amount of *false negatives*. To say it in other words, there are plenty of cases in which the face is not detected at all.

The value marked by purple cross is the threshold value set in the data processing pipeline 6.1.1. The big drop in recall for thresholds in range $\langle 0.4, 0.5\rangle$ is caused by not accepting the detections of neighboring faces 6.6 as *true positive*. With this in mind, higher recall value doesn't necessarily mean better performance.

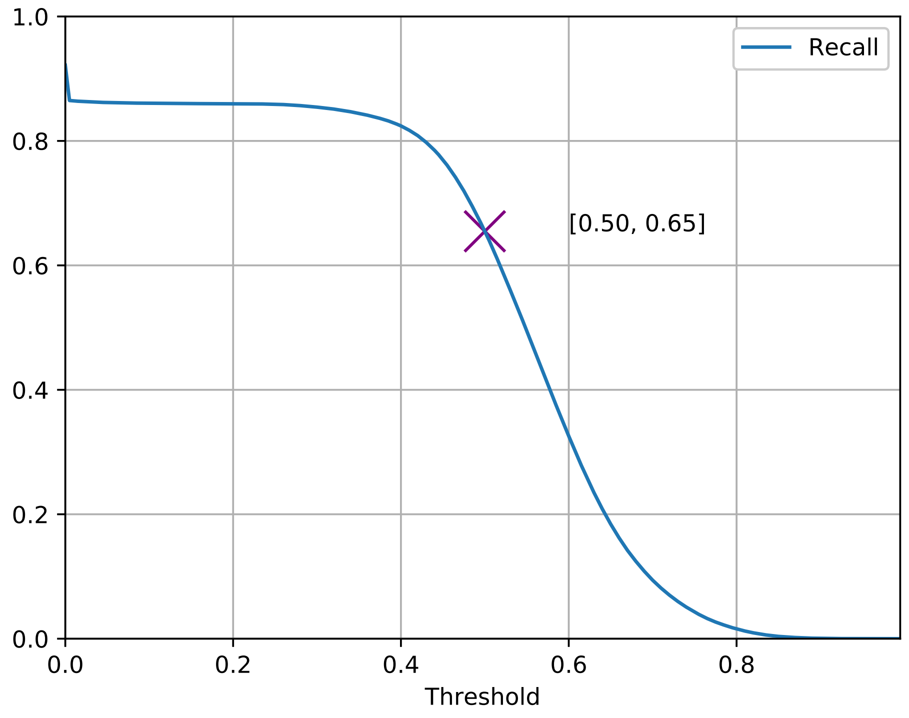


Figure 6.7: Recall computed on IoU 6.2 values.

The selection of neighboring face as *true positive* is exacerbated by the penchant of humans to select larger area as containing the face. To verify this hypothesis, I implemented a script which computes the average ratio of the area selected by human to that of the detector. The resulting ratio is **1.59**. This means that human selects on average an area which is **59 % larger** than the one selected by the detector.

Chapter 7

Conclusion

The main topic of this thesis is automatic face recognition.

In the introduction 1.1 I stated four goals of this study.

The first goal was to provide an overview of modern facial recognition methods. I split this overview into three parts. The first of these is a chapter about convolutional neural networks 2 (CNNs), which are the building block of modern facial recognition systems. The second one is a chapter called Facial Recognition 3. In this part, I describe how the end-to-end facial recognition system works. The last part (Loss Functions4) focuses on the research of loss functions which perform especially well in facial recognition tasks.

The second goal was to implement a state-of-the-art algorithm. The description of the algorithm is in the chapter 6. The implementation utilizes the ArcFace research 4.7.

The third and the fourth goal was to evaluate my algorithm's performance and compare the results with a commercial one (EyeFace SDK 3.2.3). My algorithm exceeded the commercial system by two percents 6.3.2.

To sum up the thesis, the assumption that open-source and freely available research and algorithms exceed their commercial counterparts has been proven correct. Even though the results are good, there is still room for improvement (see section 7.1).

7.1 Possible Improvements

I have two ideas which could lead to the improvements in system's performance:

- *Fitting the model on relevant identities.* The model used at the core of the system (ResNet-18) was fitted on identities which are not relevant in the context of the system's application. While this is not an issue as the model works well on open-set[1] problems, fine-tuning[2] the model again is one way to improve the system.

- *Better face detection engine.* An area where the system is lacking is the face detection engine. The detector available in the EyeFace SDK 3.2.3 seems to be significantly more robust than the one used in this work. This manifested itself when looking at the images of faulty MTCNN detection (see 6.6). EyeFace detector is better at handling edge cases (e.g. white stripe over part of the face). Such edge cases lead to MTCNN producing false negatives.

[1]See footnote on page 14

[2]A process of taking the weights of a trained neural network and using them as an initialization for a new model being trained on a data from the same domain.

Bibliography

[1] *History of facial recognition technology.* URL: https://en.wikipedia.org/wiki/Facial_recognition_system#History_of_facial_recognition_technology (visited on 11/07/2019).

[2] *How facial recognition became a routine policing tool in America.* URL: https://www.nbcnews.com/news/us-news/how-facial-recognition-became-routine-policing-tool-america-n1004251 (visited on 11/07/2019).

[3] *San Francisco is first US city to ban facial recognition.* URL: https://www.bbc.com/news/technology-48276660 (visited on 11/07/2019).

[4] *Data flow in LeNet 5.* URL: https://www.d2l.ai/_images/lenet.svg (visited on 09/30/2019).

[5] *From Dense Layers to Convolutions.* URL: https://www.d2l.ai/chapter_convolutional-neural-networks/why-conv.html (visited on 10/01/2019).

[6] *Pooling.* URL: https://www.d2l.ai/chapter_convolutional-neural-networks/pooling.html (visited on 10/01/2019).

[7] Y. Taigman et al. "DeepFace: Closing the Gap to Human-Level Performance in Face Verification". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition.* June 2014, pp. 1701–1708. DOI: 10.1109/CVPR.2014.220.

[8] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* 2015. arXiv: 1502.03167 [cs.LG].

[9] Shibani Santurkar et al. *How Does Batch Normalization Help Optimization?* 2018. arXiv: 1805.11604 [stat.ML].

[10] *ReLU activation function.* URL: https://d2l.ai/_images/output_mlp_699d0d_3_0.svg (visited on 10/01/2019).

[11] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[12] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[13] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[14] Ivan Gruber. *Face Recognition: Review of State-of-the-Art Methods.* 2018.

[15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 815–823.

[16] Kaipeng Zhang et al. "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks". In: *CoRR* abs/1604.02878 (2016). arXiv: 1604.02878. URL: http://arxiv.org/abs/1604.02878.

[17] Gary B. Huang et al. " Labeled Faces in the Wild: A Database forStudying Face Recognition in Unconstrained Environments ". In: *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition.* Erik Learned-Miller and Andras Ferencz and Fr é d é ric Jurie. Marseille, France, Oct. 2008. URL: https://hal.inria.fr/inria-00321923.

[18] L. Wolf, T. Hassner, and I. Maoz. "Face recognition in unconstrained videos with matched background similarity". In: *CVPR 2011.* June 2011, pp. 529–534. DOI: 10.1109/CVPR.2011.5995566.

[19] Yandong Guo et al. "MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition". In: *CoRR* abs/1607.08221 (2016). arXiv: 1607.08221. URL: http://arxiv.org/abs/1607.08221.

[20] Dong Yi et al. "Learning Face Representation from Scratch". In: *CoRR* abs/1411.7923 (2014). arXiv: 1411.7923. URL: http://arxiv.org/abs/1411.7923.

[21] Jiankang Deng et al. "ArcFace: Additive Angular Margin Loss for Deep Face Recognition". In: *arXiv e-prints*, arXiv:1801.07698 (Jan. 2018), arXiv:1801.07698. arXiv: `1801.07698 [cs.CV]`.

[22] Gal Chechik et al. "Large Scale Online Learning of Image Similarity Through Ranking". In: *J. Mach. Learn. Res.* 11 (Mar. 2010), pp. 1109–1135. ISSN: 1532-4435. URL: `http://dl.acm.org/citation.cfm?id=1756006.1756042`.

[23] Yandong Wen et al. "A discriminative feature learning approach for deep face recognition". In: *European conference on computer vision*. Springer. 2016, pp. 499–515.

[24] Yu Liu, Hongyang Li, and Xiaogang Wang. "Learning Deep Features via Congenerous Cosine Loss for Person Recognition". In: *CoRR* abs/1702.06890 (2017). arXiv: `1702.06890`. URL: `http://arxiv.org/abs/1702.06890`.

[25] Weiyang Liu et al. "SphereFace: Deep Hypersphere Embedding for Face Recognition". In: *CoRR* abs/1704.08063 (2017). arXiv: `1704.08063`. URL: `http://arxiv.org/abs/1704.08063`.

[26] Hao Wang et al. "CosFace: Large Margin Cosine Loss for Deep Face Recognition". In: *CoRR* abs/1801.09414 (2018). arXiv: `1801.09414`. URL: `http://arxiv.org/abs/1801.09414`.

[27] *Confusion Matrix*. URL: `https://static.tildacdn.com/tild6263-3439-4134-b164-626433646133/9.svg` (visited on 11/05/2019).

[28] *Precision and Recall*. URL: `https://en.wikipedia.org/wiki/Precision_and_recall` (visited on 11/05/2019).

[29] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[30] *Intersection over Union*. URL: `https://d2l.ai/_images/iou.svg` (visited on 11/05/2019).

[31] *Feature Extraction*. URL: `https://deepai.org/machine-learning-glossary-and-terms/feature-extraction` (visited on 11/06/2019).

[32] *PyTorch Implementation of ArcFace*. URL: `https://github.com/ronghuaiyang/arcface-pytorch` (visited on 11/06/2019).

[33] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.