Bernhard D. Feldhofer, BSc

Secured Face Authentication with Time-of-Flight on Android

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof

Institute of Computer Graphics & Vision
www.icg.tugraz.at

Dipl.-Ing Dr.techn. Markus Dielacher
Infineon Technologies Austria

Graz, November 2018

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Abstract

In recent years, Time-of-Flight (ToF) sensors for 3D imaging have become popular for consumer industries. Due to their low power consumption and affordable costs, they can be easily embedded in smartphones and cars. In this Master's thesis, we examine the applicability of ToF sensors for reliable face recognition. Due to the low resolution of the sensor, we will explore possible login distances and also the impact of sunlight. Face recognition is getting more and more important to the human lifestyle. Many techniques were developed to face this problem. For example, Eigenfaces in the early 1990s and Local Binary Pattern Histogramming (LBPH) some years later. State-of-the-art nowadays are neural networks. They can outperform nearly all algorithms which have been developed for different areas. Face recognition with ToF sensors received only a little attention from the research community. With our ToF camera (Picoflexx), we can record more than just depth data, for example, we additionally get the amplitude data (greyscale image) and noise information of the specific pixel. We will show how well a Convolutional Neural Network (CNN) performs on noisy depth and greyscale images on a very low resolution. We developed a preprocessing pipeline to address the challenges with a small set of training data. To meet the company requirements of Infineon, we implemented an Android-based prototype application to demonstrate our results on a consumer smartphone in real-time.

Keywords: Time-of-Flight, Face recognition, Smartphone

Kurzfassung

In den letzten Jahren wurden Time-of-Flight (ToF) Sensoren für die 3D-Bildgebung in der Konsumgüterindustrie populär. Aufgrund des geringen Stromverbrauches und der geringen Kosten sind sie problemlos in Smartphones und Autos integrierbar. In dieser Masterarbeit untersuchen wir die Anwendbarkeit von ToF-Sensoren für eine zuverlässige Gesichtserkennung. Aufgrund der geringen Auflösung des Sensors untersuchen wir die möglichen Arbeitsdistanzen und auch den Einfluss von Sonnenlicht. Die Gesichtserkennung wird für den täglichen Lebensstil immer wichtiger. Viele Techniken wurden entwickelt, um dieses Problem zu lösen. Zum Beispiel Eigenfaces in den frühen 1990er Jahren und Local Binary Pattern Histogramming (LBPH) einige Jahre später. Neuronale Netze können heutzutage nahezu alle Algorithmen übertreffen, die für verschiedene Bereiche entwickelt wurden. Die Gesichtserkennung mit ToF-Sensoren ist ein kaum erforschtes Gebiet. Mit unserer ToF-Kamera (Picoflexx) können wir nicht nur die Tiefendaten aufnehmen, wir erhalten zusätzlich die Amplitudendaten (Graustufenbild) und Rauschinformationen des jeweiligen Pixels. Wir zeigen, wie gut ein Convolutional Neural Network (CNN) bei sehr geringer Auflösung mit verrauschten Tiefen und Graustufenbildern zurecht kommt. Wir haben eine Vorverarbeitungs-Pipeline entwickelt, um der Herausforderung mit nur wenigen Trainingsdaten entgegenzutreten. Um den Anforderungen des Unternehmens Infineon gerecht zu werden, haben wir einen Android-basierten Prototypen implementiert, um unsere Ergebnisse an einem Consumer-Smartphone in Echtzeit zu präsentieren.

Schlüsselwörter: Time-of-Flight, Gesichtserkennung, Smartphone

Contents

List of Figures

List of Tables

Chapter 1

Introduction

To gain access to a system, it is necessary to enter a key or a password. Another option is a chip card which allows access to the system. The disadvantage of these approaches is that a loss of the information could lead to an abuse of the system. Face recognition is a biometric verification method and is more reliable than traditional methods [SNG14]. Classical approaches compare one login image to many other images of a database (Eigenfaces [TP91] or Fisherfaces [Luo+99]). Neural networks can identify an individual with fewer data. No solution is faultless, to increase the security of a biometric system, combinations of different biometric systems (multi-modal approaches) can be made to make the login more secure [SNG14].

Car sharing becomes more and more popular. Due to the active build-in near-infrared light source of our Time-of-Flight (ToF) camera, we can log in in the dark. With face recognition, trusted users can start the car while others can't. The vehicle can adjust the seat to the needs of different users automatically. The suppression of the near-infrared spectrum of the sun makes it possible to take images under strong sunlight. Due to the design of our sensor, we can get depth information and the greyscale image. We hypothesise that the solution with this sensor and the depth information is more secure compared to standard 2D face recognition. At the time of writing this thesis, we could not find any publication which uses the ToF data and neural networks for face recognition entirely. For this, we explore the usage of ToF for reliable face recognition. Additionally, we build a demonstrator based on Android. This is another challenge, because of the limited resources and a lack of power on the embedded device. The number of smartphones rises with each year[1]. Apple introduced with the iPhone X face recognition with 3D data into a smartphone. This work aims to build a reliable face recognition which is suitable for a smartphone with a ToF 3D sensor. Due to the rising sales of smartphones in the coming years, we hypothesise that our solution can lay the foundation stone for ToF and neural networks for smartphones. To face this problem, we searched for relevant and related literature. Much research is done with solutions based on RGB data but not for ToF. For this, we evaluate modified RGB based solutions with our ToF data. Next, we assess the sensor quality. Due to the low resolution of our sensor, we decided to use state-of-the-art verification methods like neural networks. We use TensorFlow as a framework due to the compatibility to Android. To make the neural network training faster, we use a preprocessing pipeline which segments the face from the background. It is necessary

[1] https://de.statista.com/statistik/studie/id/3179/dokument/smartphones-statista-dossier/

to build up a database with ToF data to train the network. No public database is available with ToF data or aligned 2D and 3D ToF data. We used a pre-trained, modified AlexNet [KSH12] with a specified metric as evaluation and evaluated the recorded RGB data of the smartphone front camera from our individuals in the database. We compare the results against our trained network on ToF data. Overall, the ToF trained network clearly outperformed the modified AlexNet. We could not reach the accuracy of trained state-of-the-art RGB networks like FaceNet [SKP15].

This thesis is structured as follows. First, we review the ToF principle, neural networks basics, state-of-the-art neural networks and classical methods for face recognition. In Chapter 3 we evaluate the best sensor parameters and give an overview of our created design. Next, we introduce the implementation of our preprocessing pipeline and CNN's in Chapter 4. Chapter 5 shows the results of the implementation. Finally, Chapter 6 concludes and discusses potential future directions.

Chapter 2

Background

Research is done with face detection based on Microsoft Kinect and CNN's [Bal+18]. Face recognition is not well researched with ToF, *i.e.* we discovered researches with Iterative Closest Point (ICP) [MBO07] and face prints based on depth data [MW09]. No research has been done with ToF and CNN's for face recognition at the moment we write this work.

In this chapter, we explain some basics of ToF, how a digital image is structured and how a convolutional neural network (CNN) works. We summarise the basics of the layers in neural networks and how a network can learn similarities. We summarise important values to monitor a CNN to gain a well-trained network. Last, we explain state-of-the-art CNN's and also methods which are based on ToF data and aligned 2D and 3D data from a laser scanner.

2.1 ToF Basics

In this section, we briefly summarise the working principle of our used ToF camera (Picoflexx). The camera operates with the indirect ToF principle, and this means that the camera emits modulated near-infrared light. The objects reflect the emitted light, and the sensor detects the reflected beams. For this, the camera sends out eight phase shifted signals. Four signals are modulated with 60 MHz and another four with 80 MHz. Both measurements use the phase shifts with 0°, 90°, 180° and 270°. In Figure 2.1 we illustrate the indirect ToF principle which is based on phase difference measurement. The reference signal (0°) is compared with the received modulated optical signal, and this results in a phase difference. This difference is proportional to the distance [Alm18]. In contrast, the direct ToF principle measures the travelled time of the emitted signal back to the sensor. After some post-processing, *i.e.* resolve unambiguous pixel, lower the noise level with neighbourhood filtering and thresholds, the result is a depth image and an amplitude (greyscale) image. With the used framework of the camera, it is possible to obtain more processed data. For example, it is possible to retrieve data for every pixel like noise, x and y coordinate (to the camera), intermediate data during the preprocessing pipeline and raw data. To get intermediate and raw data, it is necessary to register a special key at sensor registration on the host system. In this work, we focus on amplitude, depth and noise data. In Figure 2.2 we illustrate these data.

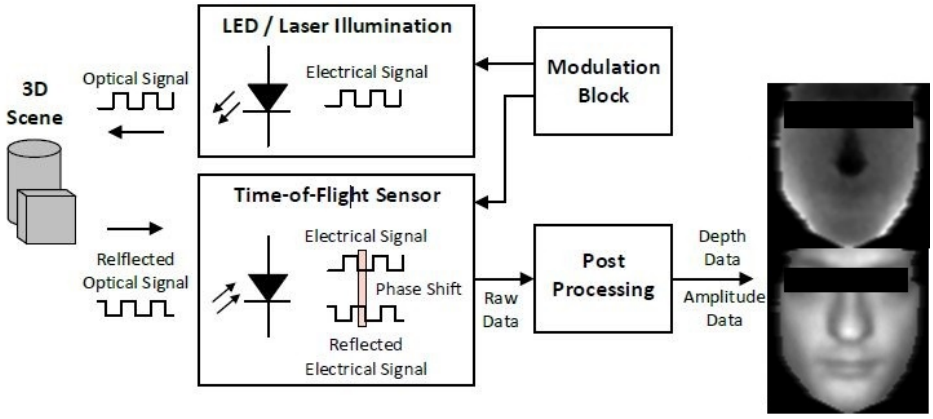


Figure 2.1: The indirect ToF working principle is based on phase difference measurements. The emitted light is reflected from objects in the scene, and this results in a phase difference compared to the reference signal. After some post-processing, the result is a greyscale (bottom) and depth image (top). Darker pixels in the depth image are closer to the sensor. Adapted from [Dru+15].

Figure 2.2: Example of ToF data, on the left side we illustrate the amplitude data, in the middle the depth data and on the right side a typical noise image. Note that bright pixels in the depth image indicate a higher distance. We observed that the noise data is stronger influenced by reflecting backgrounds than amplitude and depth data.

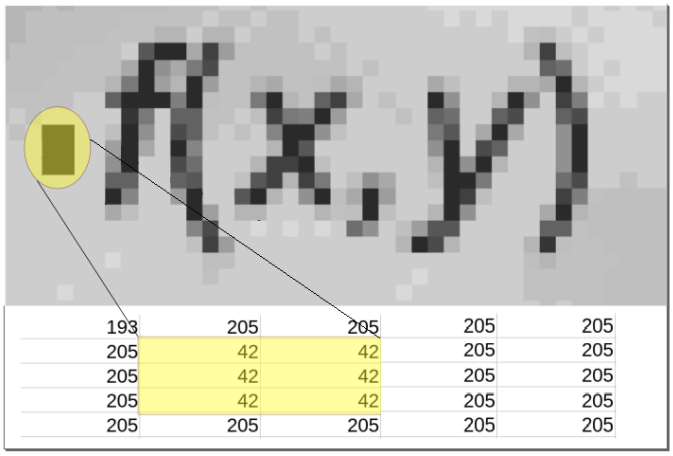


Figure 2.3: Illustration of a 2D digital image. Image adapted with changes from [AS15].

2.2 Basics about a Digital Image, Neural Networks and Face Recognition

In this section, we present the structure of a digital image. Next, we briefly explain how a convolutional neural network works. We give an overview of how convolution operates, max pooling works and present three activation functions. Additionally, we describe details about metric learning. Last, we summarise face recognition methods based on RGB data and an approach which uses classical algorithms for face recognition on ToF depth data.

2.2.1 Digital Image

To manipulate and process images, a digital device needs digital data. The digital image can be presented as

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(0,1) & \dots & f(1,N-1) \\ \vdots & \vdots & \dots & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix},$$

where f(x,y) denotes the intensity function, M the number of rows and N the number of columns [AS15].

In Figure 2.3 we illustrate an example of a simple image. In the second half of the image, the digital values between zero and 255 are presented. Higher values are assigned to the white tone and lower values to black tone. 255 is the highest possible value and represents white, and zero leads to the darkest value black.

An RGB (red, green, blue) image is a stacked image with three channels. In our work, we won't use RGB data directly. Instead, we stack amplitude, depth and other sensor data together to one image.

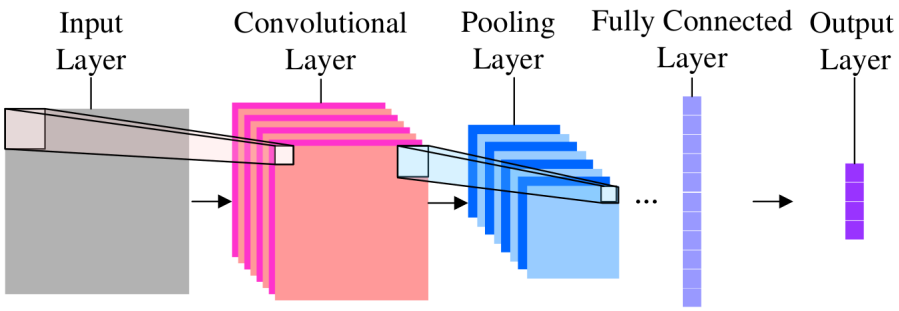


Figure 2.4: Classical CNN structure. The input image is on the left side, followed by convolution and pooling layers. The last two layers represent the fully connected layer and output layer. Figure taken from [Pen+16].

2.2.2 Convolutional Neural Network

In this section, we briefly explain the basics of neural networks and essential values for training. We summarise what convolution means in relation with images, how information reduction is possible and how activation function works. Next, we explain the basics of metric learning and discuss important characteristics to monitor a network for successful training.

2.2.2.1 Convolution, Pooling and Activation Functions

Convolution is a technique to manipulate the input data. The idea behind this technique is an information reduction, and only the most important information should be passed to the next layer of the network. For this, we shift a matrix over the input data and multiply the data with the values of the matrix. We denote this matrix now as a kernel. The kernel for image manipulation is initialised with different techniques. Initialising the kernel with random values is possible. In the end, this can lead to dead neurons due to unsuitable values. Another possibility to avoid this problem is an initialisation with the XAVIER initialiser [GB10]. The authors summarise that this initialisation leads to a significantly lower test error, which is the result of a well-proceeded training. The neural network learns this kernels for image manipulation.

In Figure 2.4 we illustrate a classical CNN structure. The input image is represented on the left side. The convolutional layer is the result of the image manipulation with the kernel onto the input image. More kernels lead to more manipulated images. We summarise now in more detail the steps in a classical neural network.

We define kernels with a certain size and place these kernels onto the input image. Next, we multiply the kernel value with the input image value. This step is necessary for each channel, for this, we have two kernels for two channels. The result of this multiplication is a new, modified image. Last, we sum up the pixels of the modified images. The result of the summation is the new value in the new image.

On the left side in Figure 2.5 we illustrate the input image (input layer). Followed by 5x5 kernels. Each kernel consists of 16x16 pixels. These kernels are learned from the neural network. The last illustration is the result after shifting the 25 kernels with con-

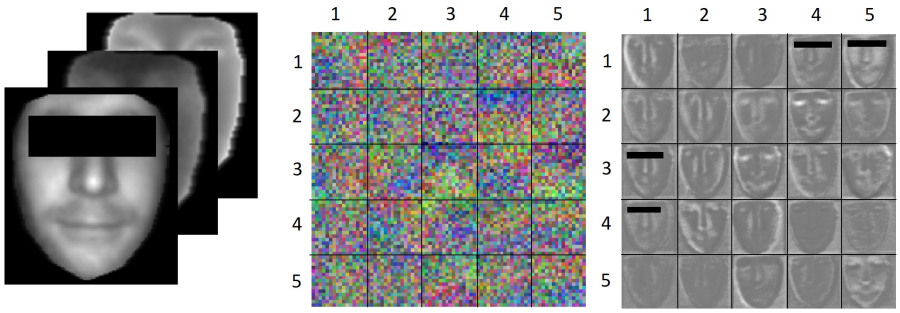
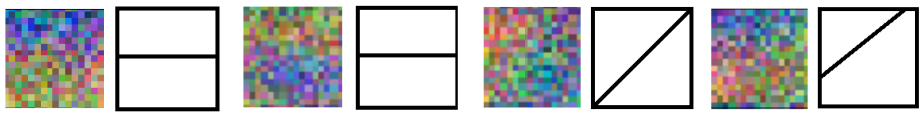


Figure 2.5: Cropped input image on the left side from our ToF camera. Next illustration reflects 25 learned kernels from a neural network in the early state. The last image is the result of the kernel shifting over the input image.



(a) Horizontal edge detector. (b) Horizontal edge detector. (c) Diagonal edge detector. (d) Diagonal edge detector.

Figure 2.6: Illustration of different kernels with the visible colour flow. These gradients in these examples represent learned edge detectors. On the left side of a pair the learned kernel is illustrated. Red represents amplitude data, green depth data and blue noise data. A colour flow from top to bottom illustrates a horizontal edge detector. On the right side of a pair we visualise a schematic illustration of the learned kernel. Best viewed digitally.

volution over the input image. For example, the first kernel (1,1) leads to the first image (1,1) in the last illustration. Networks can learn edge detectors, and they can also learn to use the channels in a specific way. In Figure 2.6 we illustrate some learned detectors. First coloured kernel (a) represents a horizontal learned edge detector. Red colour represents the amplitude data, green the depth data and blue the noise data. A gradient from the top (blue) to bottom (green) is visible. Next kernel (b) illustrates another learned horizontal edge detector. The colour flow is inverted to the first kernel. In illustration (c) the gradient from top left to bottom right is recognisable. The colour changes from red to blue, in mid there is a horizontal green line. This detector represents a diagonal edge detection. Last illustration (d) represents another diagonal edge detection. The colour flow on top left corner starts with blue (noise) and changes to red (amplitude) and green (depth).

Pooling is an operation in neural networks to reach an invariance against small changes, *e.g* if input data is noisy [Ras17]. We can distinguish between two well-known methods. The first method is called max pooling. For this, we take the highest value inside a specified kernel. This highest value is the new entry in the output. Figure 2.7 illustrates this. With mean pooling a mean calculation over the kernel is necessary. The resulting mean value is the new entry in the output.

A neuron can receive many input data (with different weighted strength) and sums

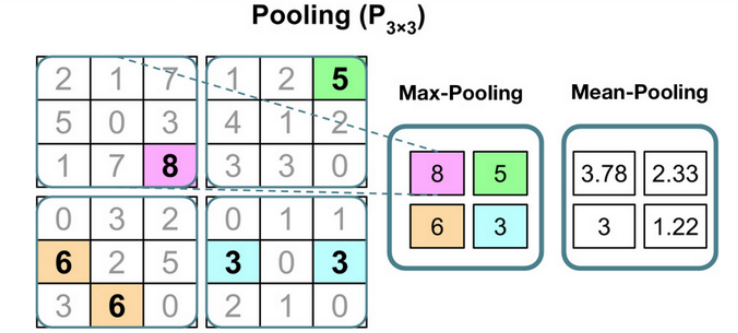


Figure 2.7: Mean and max pooling. Kernel size is 3x3. Max value in the first kernel is eight. For average or also called mean pooling the resulting value is 3.78. Figure adapted from [Ras17].

them up. The result is an output (activation), with a certain strength. To control the firing neurons, it is necessary to use activation functions. The network sends each pixel value through the activation function. In Figure 2.8 we summarise some widely used activation functions [GBC16].

Rectified Linear unit activation function (ReLu)

This function is widely used because the calculation is simple, *i.e.* all values less than zero obtain the value zero, and all positive values are unchanged. The side effect is that many zero values can occur in the network weights because of the elimination of negative values to zero. Dead neurons can be the result. An advantage of ReLu is, that no oversaturation of positive values occur, because the values can pass as they are. We illustrate this function in Equation (2.1). The output can be calculated as

$$f(x) = \begin{cases} x, & \text{if } x \geq 0. \\ 0, & \text{if } x < 0. \end{cases} \tag{2.1}$$

where x is the input value to the function [GBC16].

Sigmoid activation function

The logistic sigmoid thresholds high and low values as the threshold function, it is possible to obtain values between the lower and upper limit. The output can be calculated as

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.2}$$

where x is the input value to the function [GBC16].

Leaky Rectified Linear unit activation function (ReLu)

This function is currently prevalent because it won't eliminate negative values. Instead, it shrinks down the values drastically to small values. Dead neurons should not occur as in the ReLu function. Leaky ReLu leads to higher performances in several networks [MHN13]. We illustrate this function in Equation (2.3). The output can be calculated as

$$f(x) = \begin{cases} x, & \text{if } x \geq 0. \\ \beta \cdot x, & \text{if } x < 0. \end{cases} \tag{2.3}$$

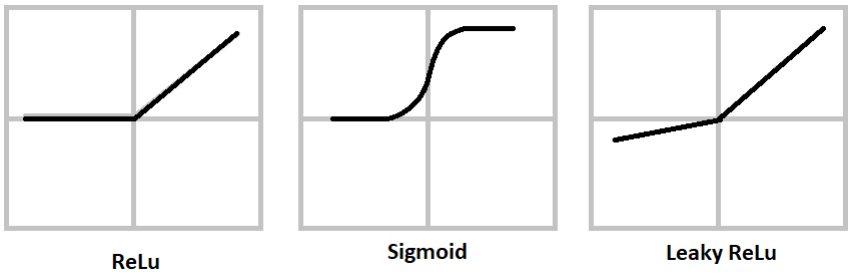


Figure 2.8: An illustration of different activation functions in CNN layers. Generally, neural networks use non-linear functions. ReLu set all negative values to zero, and all positive values can pass as they are. The logistic sigmoid function set high values to a particular value. Values between can pass with a modification. Last activation function is the leaky ReLu. This activation function is widely used because it won't lead to dead neurons as the ReLu function. Negative values can pass with less strength, and positive values can pass as they are. Illustration based on [GBC16].

where x is the input value and β a certain multiplication value for the negative values [GBC16].

2.2.2.2 Metric Learning

We need metrics to compare two results in a specific way together. For this, we evaluate three different metrics which can be used for face recognition. The cosine similarity is simple and additionally widely used for document classification [LPS16]. Triplet Loss is suitable for face recognition to train a network to distinguish between entirely unseen individuals [SKP15]. Last, we summarise weighted χ^2 and Siamese network distance which is used in DeepFace [Tai+14].

Cosine similarity

The cosine similarity measures the cosine angle between the two vectors. A cosine of zero degrees results in one, that means two identical vectors have one as similarity. The cosine similarity is defined as

$$Cos(\alpha) = \frac{AxB}{|A|x|B|}, \tag{2.4}$$

where A and B are the two vectors for comparison. Two completely different vectors result in -1 [LPS16].

Triplet Loss and SiameseTripletLoss

The basic idea of Triplet Loss is to learn embeddings, where similar examples are close and different are further apart. For this, a CNN must calculate a facial feature vector for each image. With Triplet Loss, we compare two vectors together. For this, we subtract both vectors from each other and calculate the square over the resulting

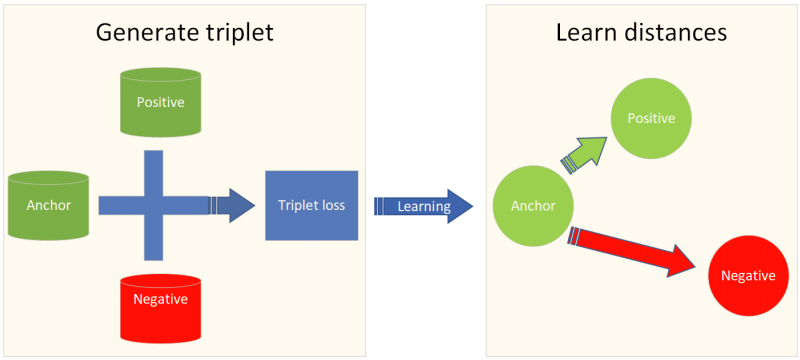


Figure 2.9: On the left side we illustrate the positive, negative and anchor feature vector. These three feature vectors build one triplet. After the Triplet Loss training, the network should be able to separate images from the same individual and from different individuals (more distance to the reference image). Figure based on [SKP15].

vector. With a summation over the resulting vector, we receive a threshold for the similarity between the two images (facial feature vectors). In Equation (2.5) and (2.6) we illustrate this in a more mathematical way. The distance between two vectors can be calculated as

$$\vec{D} = \begin{pmatrix} a_0 \\ \cdot \\ \cdot \\ a_n \end{pmatrix}_A - \begin{pmatrix} l_0 \\ \cdot \\ \cdot \\ l_n \end{pmatrix}_L , \tag{2.5}$$

where $\left(\cdot\right)_A$ is the anchor (reference feature vector) and $\left(\cdot\right)_L$ the login vector to the system. After the square operation the result is the distance in a d-dimensional Euclidean space [SKP15].

The similarity can be calculated as

$$SIM = \sum_{i=0}^{N} \vec{D}_i^2, \tag{2.6}$$

over the distance vector from before, where N is the dimension. With the resulting similarity and a certain threshold we can distinguish between the same individual or if it is another one.

In Figure 2.9 we illustrate the triplet generation on the left side. For this, the CNN calculate the feature vectors of the images. We use two feature vectors of the same individual (anchor = reference image, positive image = another image of the same individual) and a negative one (different individual).

As described in Equation (2.5), we calculate the distance between two facial feature vectors with help of the distance calculation. The positive distance between anchor

vector (reference image) and positive vector (another image of the same individual) can be calcualted as

$$\vec{D}_p = \begin{pmatrix} a_0 \\ \cdot \\ \cdot \\ a_n \end{pmatrix}_A - \begin{pmatrix} p_0 \\ \cdot \\ \cdot \\ p_n \end{pmatrix}_P , \tag{2.7}$$

where $\left(\cdot\right)_A$ is the feature vector of the reference image and $\left(\cdot\right)_P$ the feature vector of the same individual, but another image.

The negative distance between anchor vector (reference image) and negative vector (image of another individual) can be calcualted as

$$\vec{D}_n = \begin{pmatrix} a_0 \\ \cdot \\ \cdot \\ a_n \end{pmatrix}_A - \begin{pmatrix} n_0 \\ \cdot \\ \cdot \\ n_n \end{pmatrix}_N , \tag{2.8}$$

where $\left(\cdot\right)_A$ is the feature vector of the reference image and $\left(\cdot\right)_N$ the feature vector of a different individual.

We select this positive and negative image randomly. In FaceNet [SKP15], a generation of triplets is used which lead to faster convergence. Due to the restrictions to a standard system, we can't generate triplets as in FaceNet. The loss for our optimiser is defined as

$$T_L = \sum max(0, \|\vec{D_p}\|_2^2 - \|\vec{D_n}\|_2^2 + \alpha), \tag{2.9}$$

where α is the additional margin between the positive and negative distances [SKP15]. This additional separation makes the learning harder and the verification better. For example, FaceNet uses an alpha value of 0.2.

Another potential improvement of Triplet Loss could be the "improved Triplet Loss". Zhang et al. [Zha+16] created a modified Triplet Loss. Additionally, we need the distance between the positive and negative feature vector as in Equation (2.10). We define the distance between these two vectors as

$$\vec{D}_{pn} = \begin{pmatrix} p_0 \\ \cdot \\ \cdot \\ p_n \end{pmatrix}_P - \begin{pmatrix} n_0 \\ \cdot \\ \cdot \\ n_n \end{pmatrix}_N . \tag{2.10}$$

This modified Triplet Loss is denoted as SymTriplet Loss. The SymTriplet Loss for one triplet is defined as

$$SymT_L = max(0, \|\vec{D}_p\|_2^2 - \frac{1}{2} \cdot (\|\vec{D}_n\|_2^2 + \|\vec{D}_{pn}\|_2^2) + \alpha). \tag{2.11}$$

The improvement is around half a per cent, and this experiment is based on face detection [Zha+16]. The authors used an alpha value of 1.0.

Weighted χ^2 and Siamese network distance

The DeepFace [Tai+14] network use weighted χ^2 distance and Siamese network distance. The DeepFace network outputs normalised vectors between zero and one. Additionally, the vector is very sparse. The weighted χ^2 similarity is defined as

$$\chi^2(f_1, f_2) = \sum_i \frac{w_i(f_1[i] - f_2[i])^2}{f_1[i] + f_2[i]}. \tag{2.12}$$

where f_1 and f_2 are the output vectors of the DeepFace network. The weight parameters (w) are learned with the help of a linear Support Vector Machine (SVM) [Hea+98].

Another technique is the Siamese network distance. This is an end-to-end metric learning approach. Two networks run with shared parameters on two images. To calculate the Siamese network distance it is necessary to take the absolute distance between two vectors, then map the features with a fully connected layer into a single logistic unit (same and not the same individual) [Tai+14]. The Siamese networks distance is defined as

$$d(f_1, f_2) = \sum_i \alpha_i \cdot |f_1[i] - f_2[i]|, \tag{2.13}$$

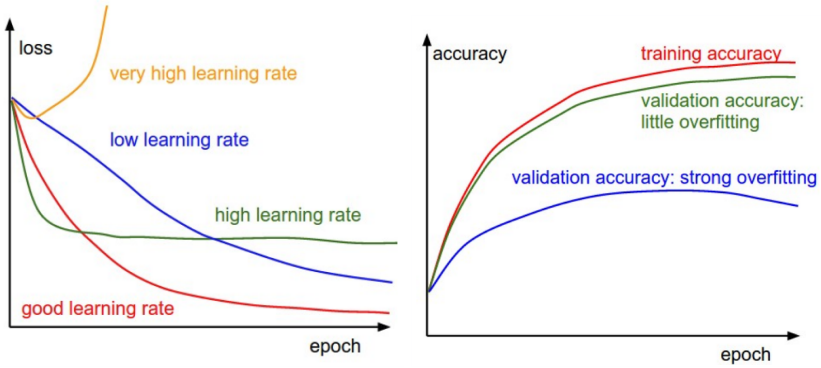
where f_1 and f_2 are the feature vectors of the network and α a learned parameter from the network. DeepFace use standard cross entropy loss for back propagation [Tai+14].

2.2.2.3 Characteristics for Neural Network Training

To monitor the training of a neural network, it is necessary to take care of some parameters. The network training needs an optimiser. For this, we send the training data filled in batches to an optimiser. This optimiser makes a gradient update and changes the weights of the network.

The loss is one of the most important values to observe. It reflects how well the network learns and also if the learning rate is too high or low. Figure 2.10 (a) gives an overview of how different learning rates can influence the loss function. A converged loss function means that the network learned well. No convergence after many epochs means that the network could not learn well and the learning rate is too high. The loss is based in most cases on a batch. We split the train set into batches and send this batches to the optimiser. This batch-wise evaluation during the forward pass can lead to wiggle in the loss function. If a batch contains the whole data set, the wiggle is minimal because the optimiser computes a gradient update on this one batch which includes the entire data set [Kar18]. A too high learning rate can lead to an exploding loss (orange curve). The weights get extreme values, and it is time to stop the training. The blue curve illustrates a too low learning rate. With a high learning rate the loss can't converge, the green curve represents this. A well-chosen learning rate leads to a loss function as the red curve illustrates [Kar18].

Overfitting means that the network learned the training data quite well, but it can't perform as well on the validation data. The network will only receive unknown data, and



(a) Visualisation of different learning rates. Figure taken from [Kar18].

(b) Visualisation of overfitting. Figure taken from [Kar18].

Figure 2.10: Monitor learning and overfitting of a neural network. On the left side, the orange curve illustrates a too high learning rate for the optimiser, red a good learning rate. On the right side, the red curve illustrates the training accuracy and the green curve the validation accuracy. This represents slightly overfitting. The blue curve represents strong overfitting [Kar18].

the aim is that the network performs with nearly the same performance and accuracy on unknown data as on the known training data. Figure 2.10 (b) illustrate samples how we can monitor if a network overfits. If the validation accuracy is almost as good as the training accuracy, then the network learned well. The red and green curve illustrates this. If the validation curve looks like the blue curve, then the training is not successful because the network learned the training data nearly perfect but can't handle unknown data with this performance [Kar18].

Figure 2.11 illustrates how one of our created networks looks like on the last convolutional layer (left side) after training with a too high learning rate. Bright areas of activations indicate that the neural network has not learned features. The kernels (in the middle with size 16x16 pixels each) have no structure. The loss exploded up to 5,000,000 (right side).

Figure 2.12 illustrates on the left side the kernels of the first layer from a network. The kernels are very noisy. The background is in most cases a too high learning rate which leads to unconverged networks. Another symptom is very low weight regularisation penalty [Kar18]. The kernel looks noisy, and no clear edge detection is visible. On the right side, we illustrate the kernels from AlexNet [KSH12]. It is one of the most popular networks. The AlexNet contains smooth and structured kernels, and this is the result of a well-trained network.

2.3 Face Recognition

Face recognition systems focused in general on two-dimensional approaches [MW09]. Security cameras and smartphones have usually built in 2D sensors. According to Hill

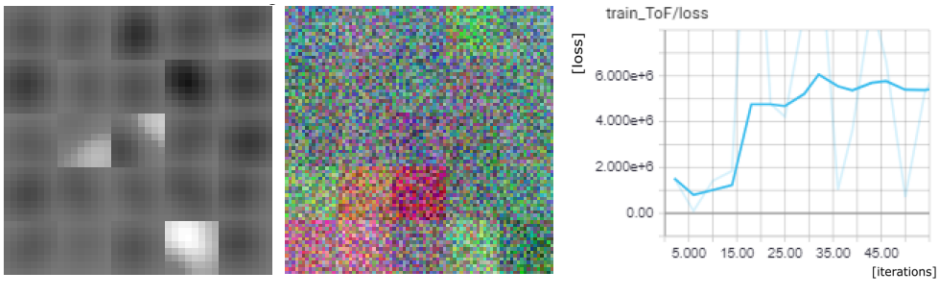


Figure 2.11: The network learned no features due to a too high learning rate. Bright white areas of activations are visible (left side). The middle image illustrates the kernels (five in each row) in the first layer. The kernels are very noisy, and no structure is visible. Right side illustrates the loss over 50 iterations, where one iteration is based on one batch. The loss exploded to values over five million.

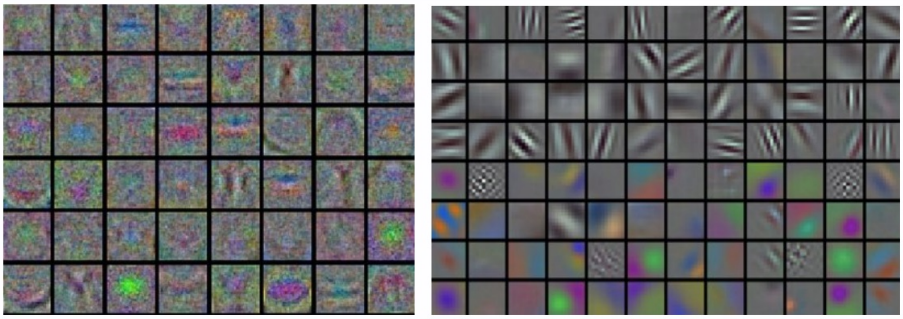


Figure 2.12: On the left side, we illustrate noisy kernels. Wrong learning rate and no convergence can lead to this result. Also, very low weight regularisation penalty can be one symptom. Right side illustrates a network which learned from well-chosen hyperparameters. The kernels are clear, smooth and edge detections are visible. This illustration is based on AlexNet. Figure adapted from [Kar18].

et al. [HSA97], human brains can have difficulties with face recognition from 2D data only. With one 2D image, it is not possible to accurately determine physical dimensions, orientation and location of the face relative to the sensor [MW09]. Facial expressions and make up is another potential problem. Face recognition can be categorised into three major approaches [MBO07]:

Holistic approach
Holistic approaches match the face as a whole against other faces. Examples for this method are Eigenfaces [TP91], Fisherfaces [BHK97], Independent Component Analysis [Luo+99], SVM [Hea+98] and neural networks [MBO07]. Cartoux et al. [WPW03] researched face profile matching in the 1980s with laser scanners. One disadvantage of laser scanner systems is that the subject must retain without motion until the scanning process is finished.

Region based approach
Feature or region based approaches match regions like eyes, noses and mouth against each other. One advantage of region-based approaches is that it can be more powerful in the case of variations in illumination and expression [MBO07].

Hybrid matching approach
The hybrid approach use aligned 2D and 3D data for the face recognition [MBO07]. This approach can score with higher performance than the 2D or 3D methods without the combination. Mian et al. [MBO07] claim that 3D face recognition is more sensitive to facial expressions than 2D data. The verification rate can drop with emotions around 15–22 per cent [MBO07].

In this section, we illustrate methods to verify a face with ToF data and also state-of-the-art solutions with RGB data. First, we summarise the NIRFaceNet [Pen+16] and the FaceNet [SKP15]. The NIRFaceNet is trained on near-infrared images. This network uses a classification algorithm. This means the network is only able to classify a new input image within existing and known individuals which occur during training. We want a face recognition which can distinguish two completely unknown individuals. To face this problem, we use for training Triplet Loss. Less research is done with ToF and face recognition. For this, we evaluate in this section a depth based face recognition with ToF data only. We summarise a paper where the authors use aligned 2D and 3D data. We use the same principle because of the aligned ToF data. The authors process the aligned data with a preprocessing pipeline to verify an individual with classical methods. We use some approaches form this paper.

2.3.1 Near Infrared FaceNet

In this section, we briefly summarise the Near Infrared FaceNet (NIRFaceNet). The CNN is a modified version of the GoogLeNet [Sze+15]. The GoogLeNet is build up with 27 layers, the NIRFaceNet with eight. The main idea behind both networks is the inception layer structure which can extract features from images. In Figure 2.13 we visualise an inception module. The design of the module includes four paths. On the left side, the one-by-one convolution takes place. This one-by-one convolution plays two roles in the

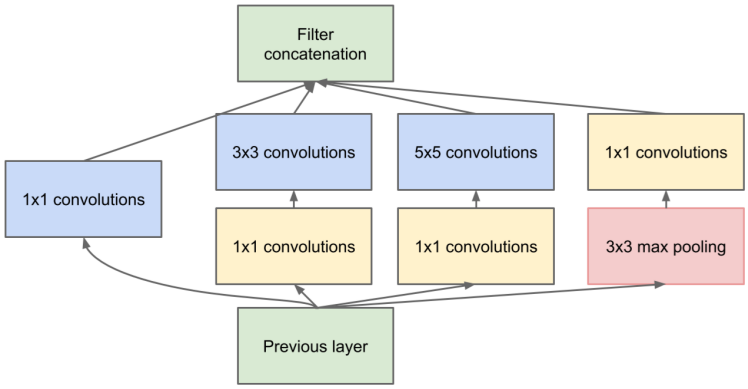


Figure 2.13: Inception module for feature extraction from GoogLeNet [Sze+15]. Figure taken from [Sze+15].

feature extraction. First, this one-by-one convolution increase the non-linearity of the network and the wealth of information is preserved from the upper layer. Second, it can reduce the calculation load before the multi-scale convolution extract the upper features [Pen+16]. The path with max pooling and the one-by-one convolution with a stride of one (shift the kernel by just one pixel), maintain the resolution from the path before and extract more texture details. The authors of NIRFaceNet removed the five by five path because it can dimish the small features of the image and it takes more time to train the network [SKP15; Pen+16]. The authors claim that a five by five convolution needs 2.78 times more computation time than a three by three convolution. The authors use MatLab 2015 and Caffe for the implementation. The paper is published around one year before we started with this work. The training itself took around 30 hours on a DELL PRECISION T3600. The used CPU is a Xeon E5-1620, 64 gigabyte RAM and a Nvidia Quadro 600. The used batch size is 35, and the dropout rate is set at 50%. We couldn't figure out which optimiser is used by the authors. We believe that the authors used AdaGrad [DHS11] or ADAM [KB15] optimiser because the Google FaceNet is published one year before and they use AdaGrad. In the end, the network runs for 350.000 iterations. In Figure 2.14 we illustrate three examples of the Chinese Academy of Sciences Institute of Automation (CASIA) database. The database contains 3940 images from 197 different individuals with a resolution of 640x480 for each image. The authors don't mention the distance to the camera, and it seems that the range is always the same and also that the background is nearly entirely black. We believe that this setup makes the recognition and training more accessible. Additionally, the authors removed the images in which people wear glasses. The training set is split as follows: Three images of each candidate were taken, this results in 597 train images with 197 different individuals. The authors use data augmentation to generate more training images out of these 597. We explain data augmentation in more detail at the end of this section. The test set contains 2739 images. The authors created nine different test sets, for example, one set without expressions and one with added Gaussian noise. The design of the NIRFaceNet contains a Softmax classifier, and this restricts the network to classify each new incoming image to 197 classes (different individuals). We can't use the Softmax classifier for our work.

Figure 2.14: Three examples of a candidate from the Chinese Academy of Sciences Institute of Automation (CASIA) Database. The database contains different emotions and viewpoints of an individual. Figure taken from [Pen+16].

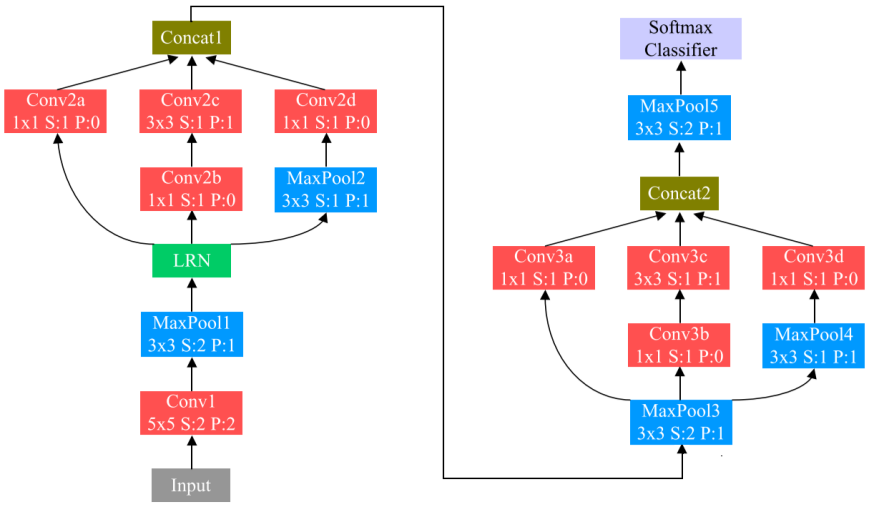


Figure 2.15: Illustration of NIRFaceNet. Figure based on [Pen+16].

Instead, we use another technique as in FaceNet. We explain FaceNet in the next section in more detail. We illustrate the complete NIRFaceNet in Figure 2.15. The network contains a prefiltering at the beginning with a five-by-five convolution and a stride of two to reduce the whole image with edge detectors. With the next layer (max pooling with stride two and size of three by three) another reduction of the image takes place, *i.e.* only the most reliable information can survive. Additionally, an invariance against small changes and noise are the result of max pooling. The next layers are the inception modules without the five-by-five convolution paths from GoogLeNet. In the end, another max pooling layer is placed before the Softmax classifier. The authors don't use fully connected layer due to performance reasons. In Table 2.1 we summarise the network from Figure 2.15. On the left side, we illustrate the layer, next the type is illustrated, followed by the properties (parameters for the layers) and last the resulting output of each layer.

Table 2.1: NIRFaceNet architecture. On the left side, we illustrate the layer number. Next, we illustrate the type of the layer, followed by the properties (Z means kernel size, LR means leaky ReLu, S means stride, K the amount of kernels). Last we summarise the output size of the layer.

Layer	**Type**	**Properties**	**output size**
—	Input	112x112	—
Layer 1	Conv1	Z: 5x5, LR, S: 2x2, K: 64	56x56
Layer 2	Maxpool	Z: 3x3, S: 2x2	28x28
—	Local response normalisation	—	28x28
Layer 4	Conv2dA	Z: 1x1, LR, S: 1x1, K: 64	28x28
Layer 3	Conv2dB	Z: 1x1, LR, S: 1x1, K: 64	28x28
Layer 4	Conv2dC	Z: 3x3, LR, S: 1x1, K: 128	28x28
Layer 3	Maxpool	Z: 3x3, S: 1x1	28x28
Layer 4	Conv2dD	Z: 1x1, LR, S: 1x1, K: 64	28x28
—	Concat	Conv2dA, Conv2dC, Conv2dD	—
Layer 5	Maxpool	Z: 3x3, S: 2x2	14x14
Layer 7	Conv2dA	Z: 1x1, LR, S: 1x1, K: 128	14x14
Layer 6	Conv2dB	Z: 1x1, LR, S: 1x1, K: 128	14x14
Layer 7	Conv2dC	Z: 3x3, LR, S: 1x1, K: 192	14x14
Layer 6	Maxpool	Z: 3x3, S: 1x1	14x14
Layer 7	Conv2dD	Z: 1x1, LR, S: 1x1, K: 128	14x14
—	Concat	Conv2dA, Conv2dC, Conv2dD	—
Layer 8	Maxpool	Z: 3x3, S: 2x2	7x7
—	Softmax	—	197

2.3.1.1 Data Augmentation

For a neural network, an essential part for training is data augmentation. We can prevent overfitting with purposeful data augmentation. For example, a network could learn that one type of a car always directs in the same direction, or one car still has the same colour. To prevent this, we can flip our data or change the brightness for example. Another advantage of data augmentation is that we can generate more training data. For example, flipping the images doubles our database of training images. In this section, we present some useful data augmentation steps. As mentioned, horizontal flipping the images double the database. We believe that horizontal flipping is not profitable for face recognition because the user won't log in to a smartphone if the rotation is around 180 degrees. To overcome this issue, it is also possible to read out the gyro sensors from the smartphone and rotate the image. Vertical flipping is also not a good choice because we believe that a face is generally not completely symmetric and with vertical flipping, we would learn the network to be invariant against symmetry and maybe lose valuable information. The horizontal and vertical scale is helpful because the face also has variable parts like the mouth. We believe that stretching an image could counteract this issue. Another essential augmentation technique for our face recognition is to rotate an image. Theoretically, we should never have a rotated image as input to our CNN due to our preprocessing pipeline. It is possible that the landmark detection fails, and the preprocessing pipeline outputs incorrect rotated images. With rotation, we can generate more train images, and we can also verify the input image correctly because small rotations of some degree won't influence our system. Contrast and brightness augmentation are in our case useful because due to our active illumination unit it is possible that the teeth reflect much more light than the face itself, *i.e.* that the illumination unit lowers the illumination time. With brightness and contrast augmentation we can simulate a behaviour like this.

In Figure 2.16 we illustrate an individual with data augmentation from the NIRFaceNet paper. The first image (a) represents the image from the database, the second image with added motion blur, next two images with different Gaussian blur in different strength. First two images in the second row are the result of added salt and pepper noise with different intensities. Last two images are the result of added Gaussian noise with varying intensity.

We obtain blurred images from our sensor, for this, we won't use additional blurring effects. The authors claim that the addition of Gaussian noise with a density of 0.001 leads to the best results (the successful login rate to the system differs around one per cent with a noise density of 0.01 instead of 0.001).

2.3.1.2 Dropout as Regularisation Technique

A technique to improve the generalisation (performance on unknown data) is to set a dropout rate in the fully connected layer. The idea is to disable neurons in the training phase, and the network must handle the data with this handicap. With each run neurons are deactivated during training. After training, the network gets back all the neurons and should perform better, because it can handle the data with fewer neurons too. For example, NIRFaceNet uses 50% dropout and GoogLeNet from 40 to 70% [Sze+15]. Dropout generally performs better on deep networks [Sze+15]. Due to a higher login rate with our data and our CNN's we are still using dropout combined with data augmentation. Addi-

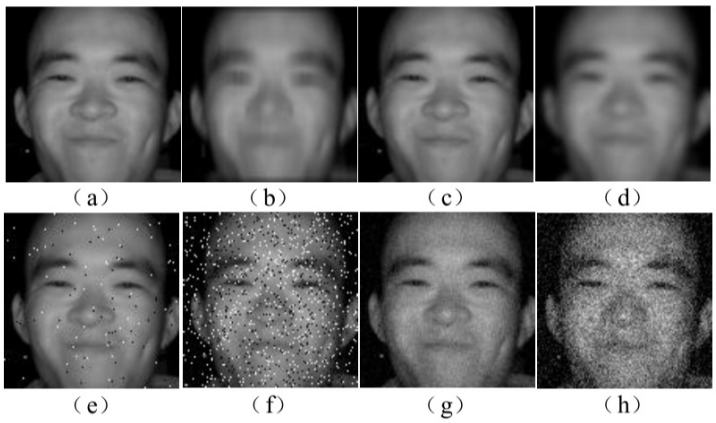


Figure 2.16: Data augmentation. The first image illustrates the input image, and the next image is prepared with motion blur. Following two images are prepared with Gaussian blur in a different strength. First two images in the second row are manipulated with salt and pepper noise in different intensity. Last two images are prepared with Gaussian noise in different intensity. Figure taken from [Pen+16].

tionally, adding noise is a strategy to construct new input images and make the network more resistant to noise [GBC16].

2.3.2 FaceNet

FaceNet is trained on a private database with around 260 million images. DeepFace [Tai+14] in contrast, is trained on 4,4 million images from 4030 different individuals. FaceNet uses two different architectures, a Zeiler&Fergus [ZF14] based and an inception based architecture. The Zeiler&Fergus architecture generally represents the classical network architecture. It consists of convolution, non-linear activation functions like ReLu or leaky ReLu, LRN and max pooling. This construct is repeated to go deeper. In FaceNet a one-by-one convolution is additionally added. Second architecture is based on the inception architecture. Both networks need the same amount of Floating Point Operations Per Second (FLOPS), the inception based variant has fewer parameters. The results are similar. FLOPS are important for our mobile platform. The Zeiler&Fergus network is 22 layers deep and has 140 million parameters with 1.6 billion FLOPS per image. The inception based approach has around 7 million parameters and 1.6 billion FLOPS. FaceNet learns facial features with triplets (two positive images (same individual) and a negative one (not same individual) for training). For comparison, Softmax is a classifier and can only sort the new login image into a category of N individuals which occur during training. DeepFace uses a weighted χ^2 distance and Siamese network distance. Recall again Section 2.2.2.2 for more details. FaceNets weights are initialised randomly and use Stochastic Gradient Descent (SGD) for backpropagation with AdaGrad [DHS11]. For the non-linear activation functions, ReLu is used. The embedding dimension of the facial feature vector is 128 and the input size 220 by 220 pixels with RGB data.

The full batch with triplets is delivered to the CNN architecture and the optimiser.

Table 2.2: FaceNet architecture. On the left side, we illustrate the type of the layer, followed by the properties (Z means kernel size, S means stride, K represents the amount of kernels, LRN means Local response normalisation). Last we summarise the output size of the layer.

Type	**Properties**	**output size**
Input	220x220x3	—
Conv2d	Z:7x7, S:2, K: 64	112x112 x64
Maxpool	Z: 3x3, S: 2x2	56x56x64
LRN	—	56x56x64
Inception(2)	Z: 3x3, K: 192	56x56x192
LRN	—	56x56x192
Maxpool	Z: 3x3, S: 2x2	28x28x192
Inception(3a)	Z: 1x1, K: 64; Z: 3x3, K: 128; Z:5x5 , K: 32	28x28x256
Inception(3b)	Z: 1x1, K: 64; Z: 5x5, K: 128; Z:5x5 , K: 64	28x28x320
Inception(3c)	Z: 5x5, K: 256; Z:5x5 , K: 64	14x14x640
Inception(4a)	Z: 1x1, K: 256; Z: 3x3, K: 192; Z:5x5 , K: 64	14x14x640
Inception(4b)	Z: 1x1, K: 224; Z: 5x5, K: 224; Z:5x5 , K: 64	14x14x640
Inception(4c)	Z: 1x1, K: 192; Z: 5x5, K: 256; Z:5x5 , K: 64	14x14x640
Inception(4d)	Z: 1x1, K: 160; Z: 5x5, K: 288; Z:5x5 , K: 64	14x14x640
Inception(4e)	Z: 5x5, K: 256; Z:5x5 , K: 128	7x7x1024
Inception(5a)	Z: 1x1, K: 384; Z: 3x3, K: 384; Z:5x5 , K: 128	7x7x1024
Inception(5b)	Z: 1x1, K: 384; Z: 5x5, K: 384; Z:5x5 , K: 128	7x7x1024
Average pooling	—	1x1x1024
Fully-connected	—	1x1x128
L2 normalisation	—	1x1x128

FaceNet uses an online triplet generation, *i.e.* during runtime with a particular combination of triplets. It means they search for a certain amount of positive samples of the same individual in a batch, mixed with random negative samples. Next FaceNet uses L2 normalisation on the embedding vector. The result is a normalised 128-dimensional face feature vector. The network learns a direct mapping from face images to a Euclidean subspace, and this distances directly correspond to a measure of the face similarities [SKP15]. The verification needs feature vectors from the network. With two feature vectors from two images, we can calculate the similarity between these two vectors. We illustrate the deepest inception based version of FaceNet in Figure 2.2. This version is 24 layers deep, and the complete training took around 2000 hours on a server cluster with thousands of CPUs.

In Figure 2.17 we illustrate 14 examples of the same individual from the FaceNet test set. We recognised that two images in the last row are identical, we believe this is a mistake in the paper. The private database contains images with different illumination and viewpoints of the individuals. With our sensor, we are less independent of the illumination due to our active illumination unit in the camera.

Figure 2.17: FaceNet test set examples. Figure adapted from [SKP15].

2.3.3 Spherical Intersection Profiles (Face Print)

In this section, we summarise Spherical Intersection Profiles [MW09]. This method is a face recognition approach which uses 3D data. The first step is to detect the nose with the help of 2D and 3D data. Therefore the depth data are filtered for distances which are interesting for face recognition, *i.e.* distances which are far away from the camera or extremely close are filtered and won't appear in the test set for verification. Horizontal and vertical boundaries of the region of interest are determined by detecting the top of the head. With the help of anthropometric data (*i.e.* knowings body-specific dimensions) and the depth map, the final head position can be determined. With the help of amplitude and depth data, the nose is detectable by searching for specific brightness (for example brightest point in a frontal face due to the closer distance to the camera) and geometric characteristics. After segmenting the essential regions of the image, the verification step is done with spherical intersections. Figure 2.18 illustrates the spherical intersection technique for face cropping. It starts with the nose tip and recognises data points in the 3D point cloud which relates to the radius from the nose tip. This radius changes and takes different layers from the 3D point cloud. It is not necessary to rotate the face due to the radius principle from the nose tip. Compared to ICP, this technique is faster and won't lead to convergence problems. Compared to other technologies, like Extended Gaussian Images [Hor84], it also retains spatial information [MW09]. To increase the quality of the depth data (*i.e.* lower noise) the authors use supersampling on the ToF data. After finding the faceprints (spherical intersection set), the authors normalise the set by a rotation transformation that means that the data directs to a specified direction. Next important step is to do symmetric optimisation since the human face is highly symmetric, *i.e.* the faceprint can be optimised by detecting a plane of bipartite symmetry. We don't argue with the argument that the face is highly symmetric and won't use this idea. If we mirror the half of the face from one side to the other side than the face is not entirely the same. In Figure 2.19 we illustrate how a face print looks like with marked symmetry points. On the right side, there are three face prints located from different humans. It is visible that the face prints are not looking the same. An optimisation allows the changing of the reference face print with each correct login into the system. Login data with lower noise level reach higher importance and can change the existing model more than images with high noise level [MW09]. To compare face prints, the average Euclidean distance between two corresponding points is calculated. The authors used a SwissRanger SR3000 ToF camera with a resolution of 176x144 pixels. The database

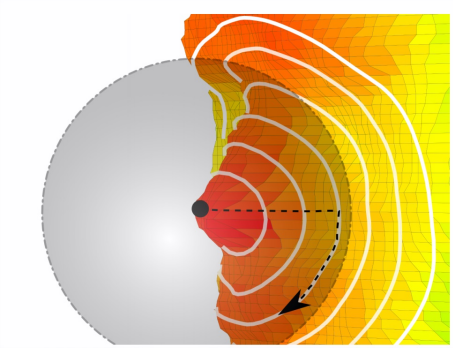


Figure 2.18: Spherical intersections started with a certain radius from nose tip. Figure taken from [MW09].

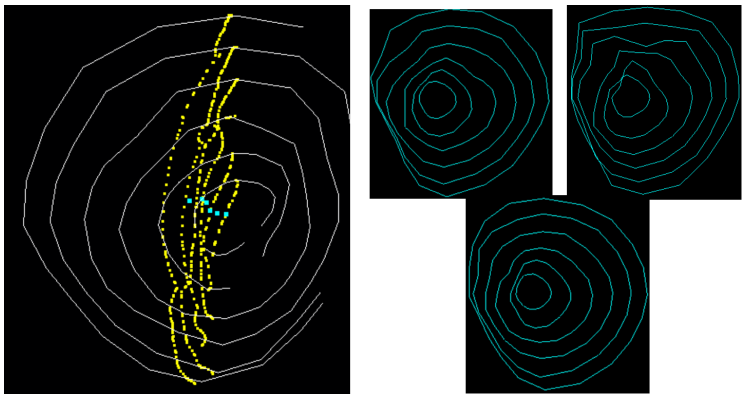


Figure 2.19: The left image illustrates a face print (white lines) with symmetry points in yellow. Midpoints of spheres are marked in cyan. On the right side, the three smaller images illustrate three face prints of three different humans. The faceprints are different. Figure adapted from [MW09].

consists of ten humans at a fixed distance of one meter to the camera and with images at various angles. The preprocessing without capturing takes around 32 ms for one frame (on an Intel Centrino 1,8 GHz). This solution is suitable for real-time applications. No false positive or true positive statistics are mentioned. The release of the paper is from the year 2009. At the time we write this thesis, not much research has been done in the area of face recognition with ToF. This solution is entirely based on ToF with an older camera than our model and with less resolution. We believe that a rejection classifier in the early stages of the preprocessing pipeline could be an improvement. Another improvement could be the optimisation of the reference model with each login. We can summarise improvements like a fast rejection classifier in the early state with spherical face prints, fast face detection with the help of the nose detection on ToF data and the optimisation of the reference image with each successful login to the system.

Figure 2.20: Aligned 2D and 3D data. Example from FRGC v2.0. Figure taken from [MBO07].

2.3.4 Multimodal (2D and 3D) Hybrid Approach for Face Recognition

Mian et al. [MBO07] created a multimodal approach where they use 2D and 3D data for face recognition. They use the FRGC v2.0[1] database which contains laser scanned individuals from a Minolta Vivid laser scanner. The resolution is 480x640, and the distance varies. The individuals are recorded frontal (with small variations) and from shoulder level up. The database contains the x, y, z, depth confidence data and the coloured image. No individual wears glasses. Exposure and emotions vary slightly. The coloured images are registered to the 3D data. The authors mentioned that they found some not correct registered faces in the database. The authors use a test set with 4,007 images based on 466 subjects. In Figure 2.20 an image of the FRGC v2.0 with variation is illustrated. The authors picked out an image where the 3D data has some holes (right side). First, the created preprocessing pipeline detects the nose with a coarse to fine approach. In Figure 2.21 (a) a horizontal slice is illustrated. The point with the highest altitude is selected with triangles. With spherical cropping (Figure 2.21 (b)) from the nose tip it is possible to segment the inner face. The radius for the sphere is selected with 80 mm. Second, hole filling and spike removing with the neighbourhood is computed. For hole filling, cubic interpolation is used and for z-axis filtering a median filter. The third step contains a uniform resampling of 2D and 3D data on a one mm square grid. The authors claim that this step is significant because all faces end up in the same resolution.

The authors use 3D data for pose correction. Due to the registration of 2D and 3D data, it is possible to rotate the 2D data in the same way as the 3D data. They use for this purpose the Hotelling transformation which is related to the Principal Component Analysis (PCA). They calculate the covariance matrix based on 3D data. After performing Hotelling transformation on the covariance matrix, the output is a matrix with eigenvectors and a diagonal matrix of eigenvalues. If the covariance matrix converges to the identity matrix, then the pose correction process is finished. As a low-cost rejection classifier in early steps they use holistic (complete face) 3D Spherical Face Representation (SFR) combined with Scale-invariant feature transformation (SIFT) [Low04] to reject a face fast in early stages. The last step is the recognition of the remaining faces with a modified ICP. They prepare the data and crop the eyes, forehead and the nose because these parts of the face are relatively less sensitive to expressions. Figure 2.22 illustrates the outcome

[1] https://www.nist.gov/programs-projects/face-recognition-grand-challenge-frgc

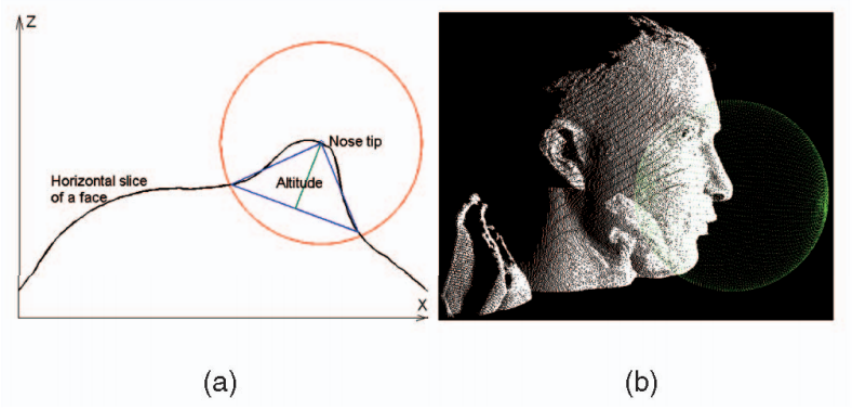


Figure 2.21: Visualisation of the nose detection. Best viewed digitally. Figure taken from [MBO07].

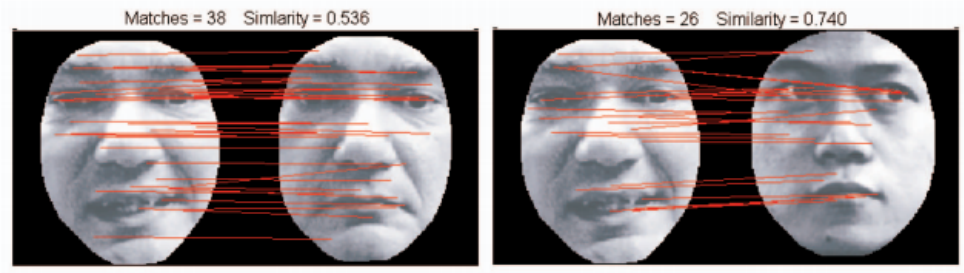


Figure 2.22: Illustration of the outcome of face segmentation of the preprocessing pipeline. Faces are compared with a modified ICP algorithm. The result is on the top side of the image pairs. Note that a lower score reflects a higher similarity. The left comparison illustrates the same humans and right side two different humans. Figure taken from [MBO07].

of the face segmentation with pose correction and modified ICP matching scores (top side of images).

In Figure 2.23 we visualise the complete pipeline. On the top half is the processed reference image and on bottom half the incoming data from the login image. First, a quick rejection is calculated with SIFT and SFR as mentioned before. If the login image passes this rejection classifier, then a comparison of the image takes place with the modified ICP algorithm. The authors created three versions of the pipeline, the first includes the whole pipeline as described in Figure 2.23, the next only contains the 3D matching of nose and forehead (Figure 2.24 illustrates this) and the last version fuses 3D matching engine with SIFT and SFR matching. The first version is the most efficient and the last the most accurate. The authors claim very high login rates (around 99 per cent) to the system. Improvements are better nose detection, skin detection, more robust illumination normalisation [MBO07].

We can summarise some ideas for our project: We hypothesise that the crop of the face helps the CNN for learning. Generally, a neural network can learn this by itself, but we can improve the network with the segmentation to make the learning independent of

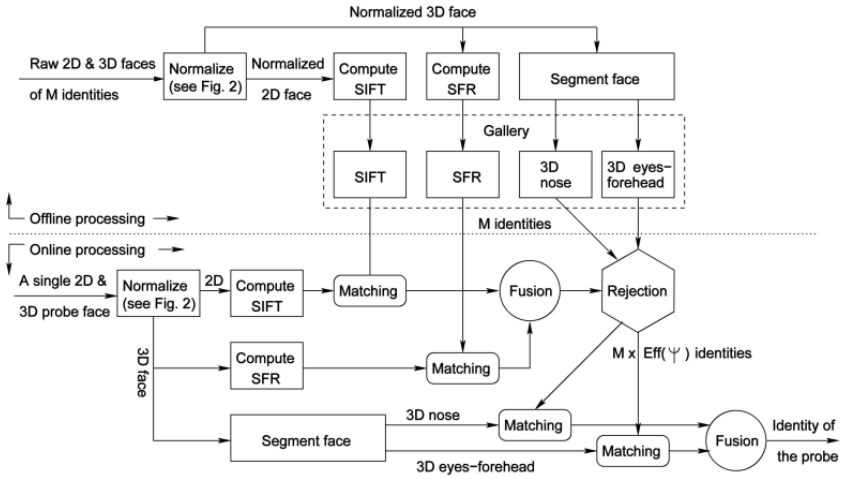


Figure 2.23: Complete processing pipeline of the multimodal approach. Figure taken from [MBO07].

Figure 2.24: Figure (a) illustrates the whole face, (b) the segmented forehead and (c) the segmented nose. Figure taken from [MBO07].

the background. The idea of the aligned texture (2D) and 3D (depth) data is also usable for our approach. It is possible to use the results of 2D algorithms (landmark detection for example) on 3D data. We can conclude, for example, the fast rejection classifier, hole filling and the 3D nose tip detection to remove the slow face detection in our approach.

Chapter 3

Prototype

Our work aims to show that face recognition is possible with the Picoflexx ToF camera on Android platform. The solution must be reliable, and additionally, an acceptable login rate is targeted. Due to the low resolution and noisy data, it is a hard challenge to realise this project. RGB-based approaches like FaceNet [SKP15], NIRFaceNet [Pen+16] and AlexNet [KSH12] reach high success rates in face recognition and object classification. Our restriction on the low resolution of the ToF camera, the interdependency to the smartphone and no existing training, validation or test set makes it more difficult. It is not possible to build deep networks like FaceNet on an Android device due to the computational limitations. We focus in our work on a network with fewer layers like the NIRFaceNet and take the verification part from FaceNet due to the restriction to Softmax in the NIRFaceNet. Additionally, the design of the NIRFaceNet is based on near-infrared data. We research the best parameters to fine tune the ToF camera to achieve higher quality. Next, we build a preprocessing pipeline which helps the network to learn faster, *i.e.* the pipeline segments unwanted information and prepare the data for the network to reach better learning. In the end, a neural network can learn this segmentation by itself. To reach faster learning with fewer data and to reach better results, we decided to segment the faces from the background. We structured this chapter as follows: First, we summarise our setup. Second, we evaluate the best exposure of the sensor for face recognition and tune the parameter to reach higher quality. Last, we give a design overview of the created prototype.

3.1 Requirements

Our setup consists of an Android smartphone with a mounted Picoflexx ToF camera. In Figure 3.1 we summarise how we realised the prototype. We use the same sensor which is built in Google Tango[1] phones. Our complete setup is illustrated in Figure 7.1 (appendix). Due to the active light source, we believe in an advantage in our verification compared to solutions without active illumination unit. A potential problem is the low resolution of the camera. We decided to segment the background from our data due to the fast computation and to make the learning and verification independent of the environment.

The prototype requires an Android device with operating system version 6.0 or later. We use the equipment as illustrated in Table 3.1.

[1] https://www.lenovo.com/at/de/tango/

(a) Front view of our mobile setup.

(b) Back view of our mobile setup.

Figure 3.1: Samsung Galaxy S7 with mounted Picoflexx ToF camera.

Table 3.1: Implementation requirements.

Program/Device	**Version**
Android-Studio	2.3.3
CUDA	9.2
CMAKE	3.6.4111459
LLDB Debugger (LLDB)	2.3
Linux Fedora	27 64-Bit
Matlab	2016b
Native Development Kit (NDK)	15.1.4119039
OpenCV[2]	3.2
Photonic Mixing Device (PMD) ToF sensor	Picoflexx
PMD Royale framework	3.5.0
Python	3.6
PyntCloud[3]	0.0.1
Samsung S7	Android 7.0
Tensorbord logging[4]	—
TensorFlow	1.8
Windows 10	64-Bit

[3] `https://pyntcloud.readthedocs.io/en/latest/`
[2] `https://opencv.org`
[4] `https://gist.github.com/gyglim/1f8dfb1b5c82627ae3efcfbbadb9f514`

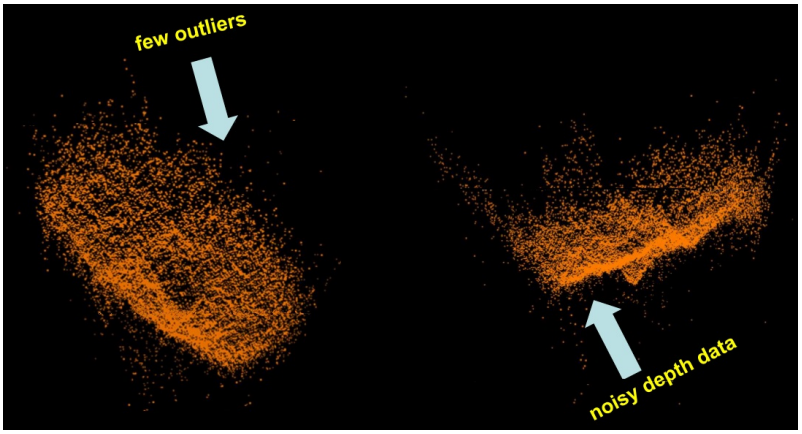


Figure 3.2: Exposure time of 50 μs. Note, that many details like the flow from chin to mouth isn't visible anymore. Less outliers occur.

3.2 Quality Evaluation of Sensor Data

Due to the default sensor settings, which are suitable for general applications, it is necessary to fine tune the sensor parameters for face recognition. First, we evaluate the best exposure, because different exposures influence the quality of the data. Higher exposure leads to less noise but also more outliers (called flying pixels). Second, we evaluate the best parameter settings of the Royale framework. For this, it is necessary to register a code to the framework to reach a higher access level to the sensor.

3.2.1 Best Exposure for Face Recognition

An exposure of 50 μs leads to fewer outliers and less environmental influence. The side effect is a low surface quality. In Figure 3.2 we illustrate a face from two different views with an exposure of 50 μs. In Figure 3.3 we illustrate a face from the side view with 2000 μs exposure time. Many outliers are visible, and the face contour is more detailed as with 50 μs.

In Figure 3.4 (left) we illustrate a measurement with only one single shot. In the middle, we visualise the calculated image over three shots with the median. The last image is processed with the mean of all three images. We decided to capture three images because with ten frames per second and a certain verification time it is not possible to take more images in a reasonable time for the login process. We hypothesise that the calculated median version is the best one. The colour illustrate the scaling, the median variant has a smoother surface than the single shot version, and the range of data is similar. The mean calculated version changed the data more significantly.

In Figure 3.5 we illustrate the standard deviation over three images. The largest error is 3,9 mm. This indicates that the noise level is severe. In the case of face recognition, high quality of the data is desired and needed.

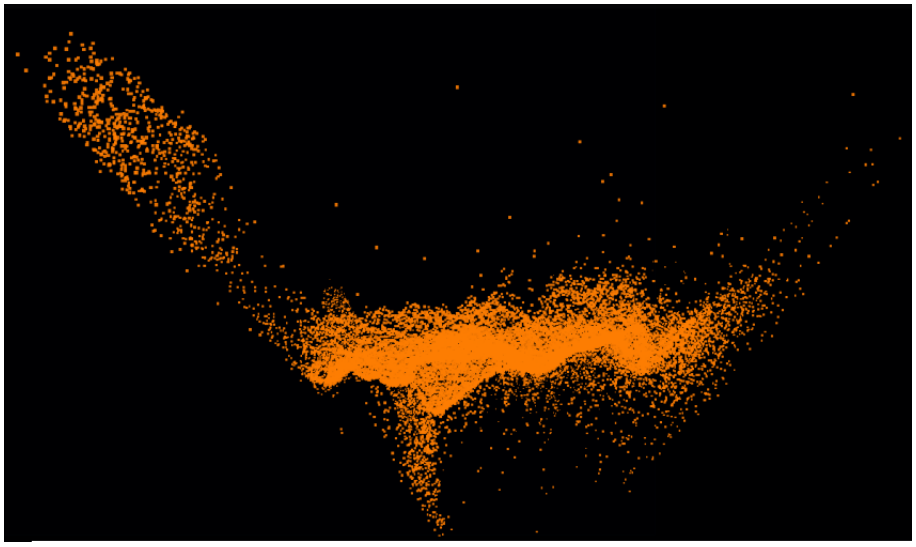


Figure 3.3: Exposure time of 2000 μs. Face from side view. Good quality of face surface, many outliers (flying pixels) occur around the face.

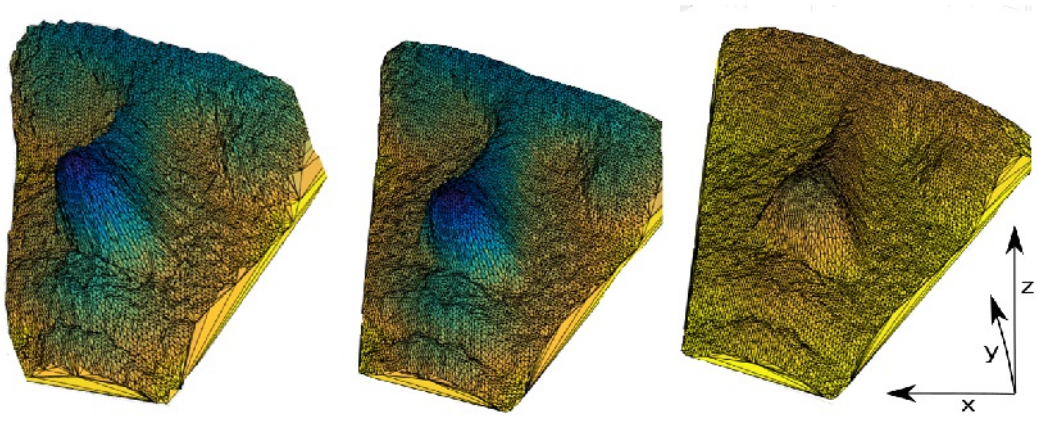


Figure 3.4: Cropped faces of one individual. On the left side, we illustrate a single shot from the framework. In the middle, we demonstrate the calculated image with the median over three login images. On the right side, we illustrate three shots combined with mean calculation.

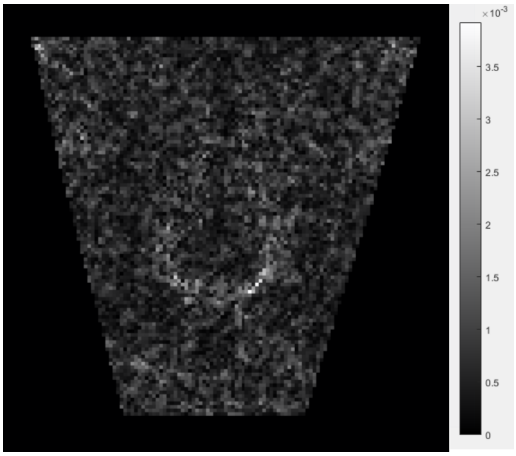


Figure 3.5: Standard deviation over three continuous captured images. The largest error is 3,9 mm. We recorded this image from a frontal view, and in the middle, we visualise the nose. The strongest standard deviation occurred around the nose.

3.2.2 Sensor Parameter Tuning

We can fine-tune our sensor with different parameters with the help of the framework. We briefly explain our changes and the impact on the data. In Figure 3.6 we illustrate the recorded face with default parameters. The nose is sharp, and longer then the real subject had (this effect often occurred). In Table 3.2 we compare the possible changeable values of our ToF framework with the default parameters and our tuned parameters. Parts of the description can be found in the Royale documentation. After these changes, the nose artefacts disappeared. Additionally, one flag (UseValidateImage) sets the pixel value to zero if the confidence is low, and this can lead to holes in the face.

In Figure 3.7 we illustrate a single shot without parameter tuning. A noisy surface is recognisable. Additionally, an unwanted peak occurs on the nose tip. In Figure 3.8 we illustrate another single shot of the same individual with fine-tuned parameters. A small side peak on the nose occurred. Generally, the surface is more detailed and less noisy. We captured these measurements with five frames per second and with auto exposure. To build a database, we use these fine-tuned parameters, but we changed the frame rate to ten frames per second due to a faster recording time. This step is invented due to time reasons of the recorded individuals. Additionally, a higher frame rate is necessary for a login to the system in a shorter time. In the end, the quality drops slightly due to the higher frame rate because the highest possible exposure with ten frames per second is 1000 μs. Additionally, we recorded the measurement from Figure 3.7 and 3.8 without background reflections. With background reflections, the auto exposure lowers the illumination time, and a very noisy surface is the result. In our dataset we discovered combined results, *i.e.* measurements with high and low quality. We hypothesise that this is a good combination because the network can learn from images with high and low noise level. Additionally, the impact of the sun leads to noisy images and with this quality changing in our datasets we hypothesise that a login in the sun is possible.

Table 3.2: Default and tuned parameters for our face recognition system.

Flags	Def.	New	Description
RemoveFlyingPixel	True	True	Removes flying pixel on final depth. Compared with the left/right neighbour or top/bottom neighbour. The threshold depends on FlyingPixelsF0 and FlyingPixelsF1.
FlyingPixelsF0	0.018	0.01	Scaling factor for lower depth value normalisation. Value: distance
FlyingPixelsF1	0.14	0.03	Scaling factor for upper depth value normalisation. Value: distance
FlyingPixelsFarDist	4.5	2.0	Upper normalised threshold value for flying pixel detection. Value: distance
FlyingPixelsNearDist	1.0	0.1	Lower normalised threshold value for flying pixel detection. Value: distance
MPIFlagAverage	True	True	To increase quality a 3x3 median filter is placed on depth and amplitude data if flag UseMPIFlagAmpBool and/or UseMPIFlagDistBool is set. Makes thresholding more reliable. MPI means Multi-Path-Interference -> systematic error.
MPIFlagAmp	True	True	Activates MPI flag for amplitude data.
MPIFlagDist	True	True	Activates MPI flag for depth data.
MPIAmpThreshold	0.3	0.5	Threshold for amplitude discrepancy. Value: amplitude.
MPIDistThreshold	0.1	0.1	Threshold for distance discrepancy. Value: distance.
MPINoiseDistance	3.0	2.0	Soft scaling factor. Avoid noise triggered misinterpretation. Value: distance. This factor is multiplied as a margin with the two values above.
NoiseThreshold	0.07	0.2	Upper limit to generate distance out of noisy raw data. If the 3D image has a strange behaviour in near range, use this value for optimisation. Max: 0.2, Value: distance.
LowerSaturationTh	400	200	Lower limit for raw data. Value: amplitude.
UpperSaturationTh	3750	4090	Upper limit for raw data. Value: amplitude.
AdaptNoiseFilter	True	True	Activated flag reduces distance noise with spatial filters, based on calculated distance noise.
AdaptNoiseFilter	2	2	Kernel size based on value. One means a 3x3 kernel with noise reduction factor of around 2.5, value two uses a kernel with 5x5 and reduces noise by a factor of around 3.5. Max: 2.
UseValidateImage	True	False	Activates output image validation. True sets invalid pixels to zero. We deactivated this flag due to many holes in near face images.
UseFilter2Freq	True	True	Activates two frequency filtering. Only avaliable on some use cases. Uses eight phase images to determine real distance.

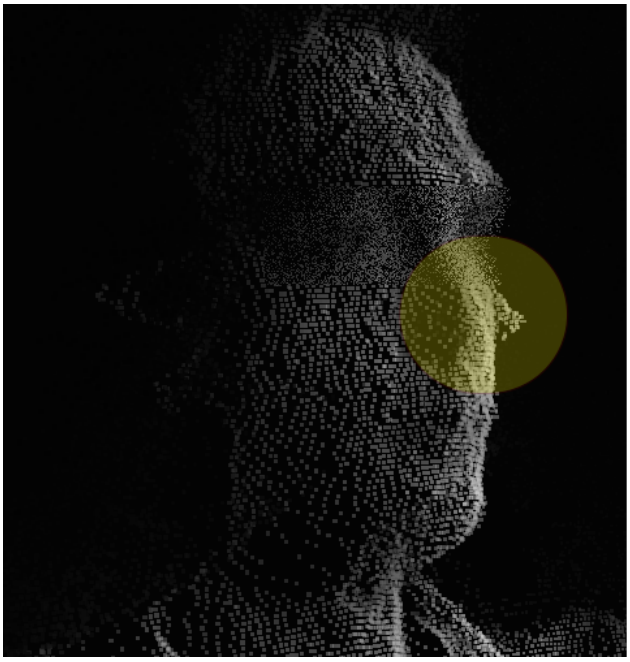


Figure 3.6: The nose of the subject has an unwanted peak. Wrong parameters can lead to this phenomenon.

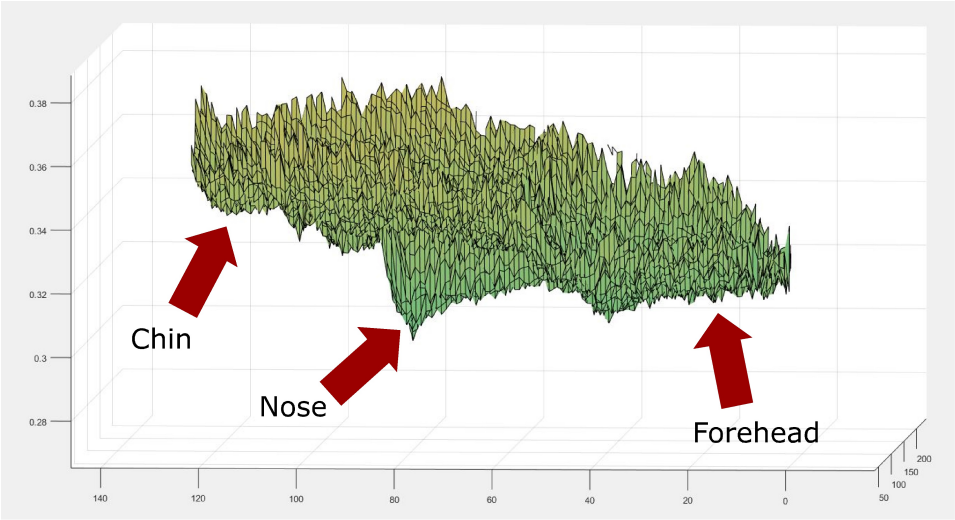
3.3 Design Overview

In this section, we give an overview of how we designed our system to reach high performance. We use Linux and Windows for our project with TensorFlow [Aba+15], due to the compatibility to Android. First, we illustrate our build prototype and describe the functionality. Second, we summarise how our prototype records the tuned data from the Royale framework. Third, we describe how Java and C++ collaborate with TensorFlow. Last, we describe the capturing process of the individuals. In Figure 3.9 we illustrate the first prototype of our application which can capture and preprocess the data. We describe in more detail the created GUI and the functionality of our prototype.

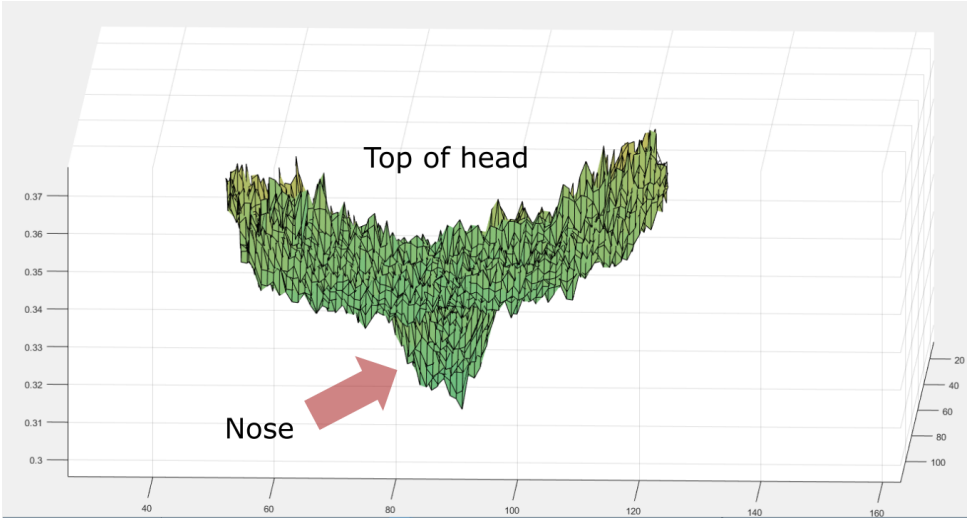
JavaCameraView
With the JavaCameraView we visualise the captured front camera image. On the right side of the JavaCameraView, we display the ToF camera data.

Start and stop camera
It is possible to start the camera without the capturing process, to prepare the user for the measurement. The individual can see himself on display. If problems occur, the candidate can stop the camera.

Take pictures, start pipeline and verification
With these buttons, the candidate can start the capturing process for training, the pipeline for preprocessing and the verification itself. With this construction, it is possible to change the filename with the GUI and analyse different data.

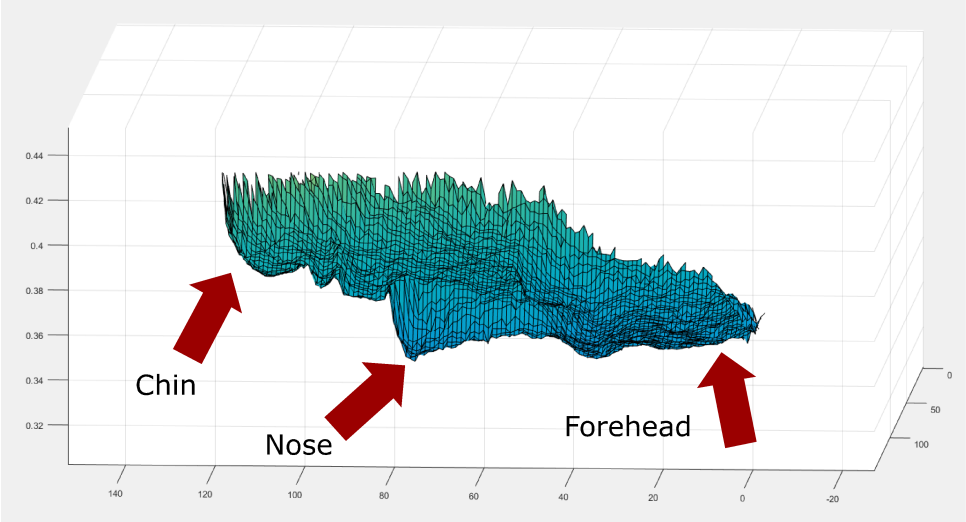


(a) Illustration of a cropped face from the side view. A noisy surface is recognisable. Note that a peak on the nose occurred.

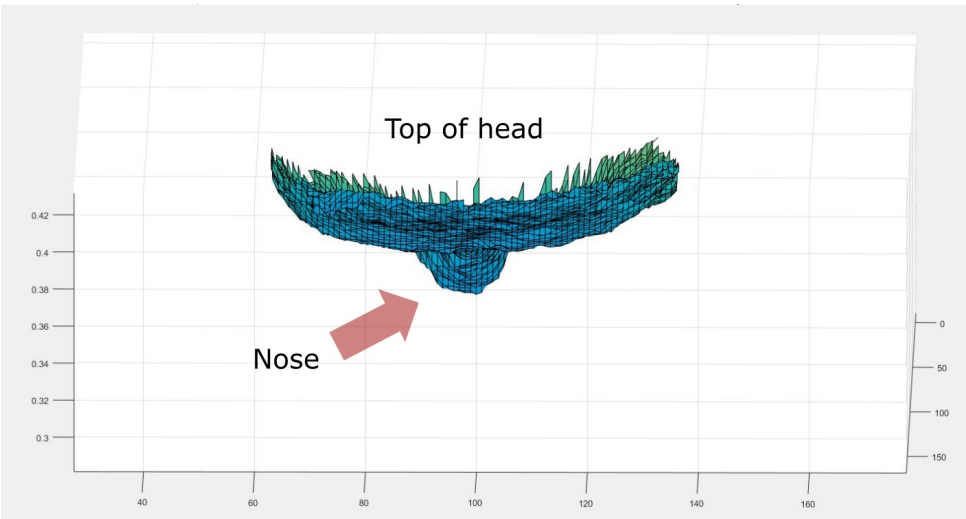


(b) Illustration of a cropped face from the bird's view. A noisy surface is recognisable. Note that a peak on the nose occurred.

Figure 3.7: Visualisation of the face from a distance of around 32 cm and different views. The default parameters lead to a noisy surface with outliers in the area of the nose. We recorded this measurement with five frames per second and auto exposure. Additionally, we recorded this measurement in an environment without strong background reflection.



(a) Illustration of a cropped face from the side view. The surface is structured, and the sharp peak on the nose disappeared.



(b) Illustration of a cropped face from the birds view. The surface is structured, and the sharp peak on the nose disappeared.

Figure 3.8: Visualisation of the face from a distance of around 35 cm and different views. The tuned parameters lead to a more structured face compared to the default parameters. We recorded this measurement with five frames per second and auto exposure. Additionally, we recorded this measurement in an environment without strong background reflection. Note that the peak on the nose is tiny, compared to the default parameters.

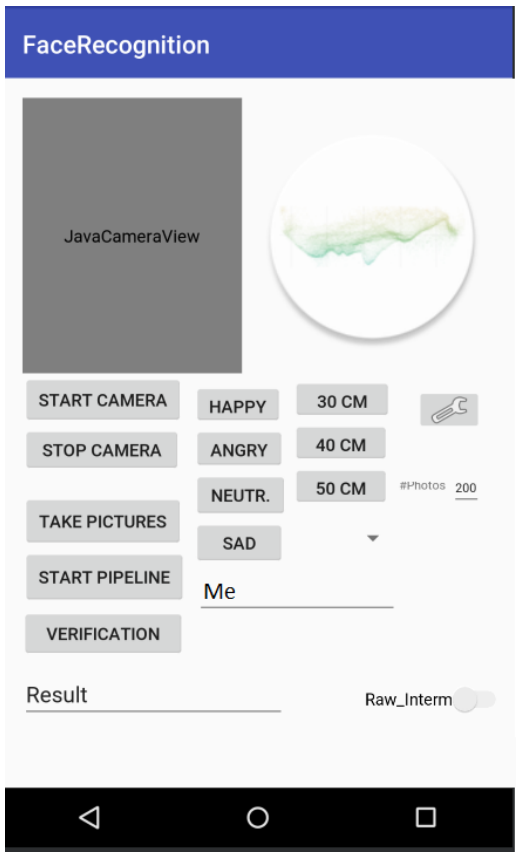


Figure 3.9: Prototype for recording individuals and preprocess the data.

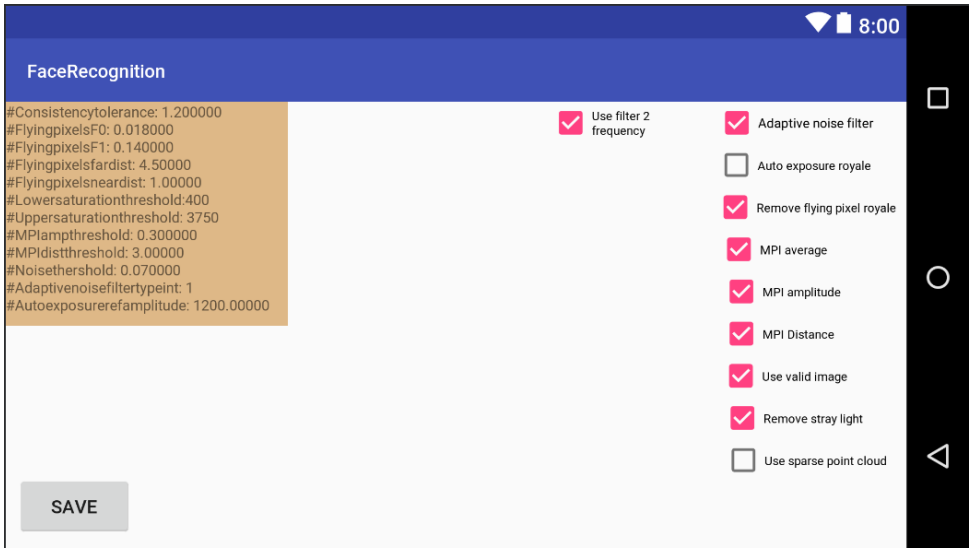


Figure 3.10: Changeable sensor parameters with the GUI. A higher access level to the camera is necessary.

Emotions, different distances and other elements

It is possible to change the emotion within four categories and three different ranges. This allows us to capture the candidate with 12 different variations. Additionally, it is necessary to fill in a name to label the data. Last, the number of shots is changeable.

For the sensor fine-tuning we implemented an additional Android Activity [Kün16] to change the parameters from the sensor itself. For this, it is necessary to use the higher level code for the sensor which is accessible for employees of Infineon and PMD. It is possible to change the same values as in Section 3.2.2.

We save each recorded image in binary format. On binary file has precisely the size of 299 kilobytes. A recording with 250 images took around two minutes and generates around 1600 MB of data (299 kilobytes · 250 · images (amplitude, depth, noise, x, y, depth confidence (six times intermediate data and nine times raw data))). In the end, we decided to capture the amplitude data, depth, noise, x, y, and depth confidence. These data are the most promising data to train a neural network. The recording is around one minute faster per candidate. With different emotions and distances, this results in a 1900 MB (average) sized folder of data per individual. Additionally, it is possible to record all mentioned data (intermediate and raw) with a flag in the GUI.

In Figure 3.11 we illustrate the internal capturing process. The connected ToF camera capture the data within the C++ part on Android. We split this data and save them individually to examine the data after the recording.

In Figure 3.12 we illustrate the major parts of our application. With Java, we control the GUI interaction to the user. Additionally, we manage the application rights, *i.e.* control the front camera, save data to the storage and control the USB port to the ToF camera. We build a connection to TensorFlow for Android to deliver the preprocessed data to the neural network. After receiving the feature vectors from the network it is possible to

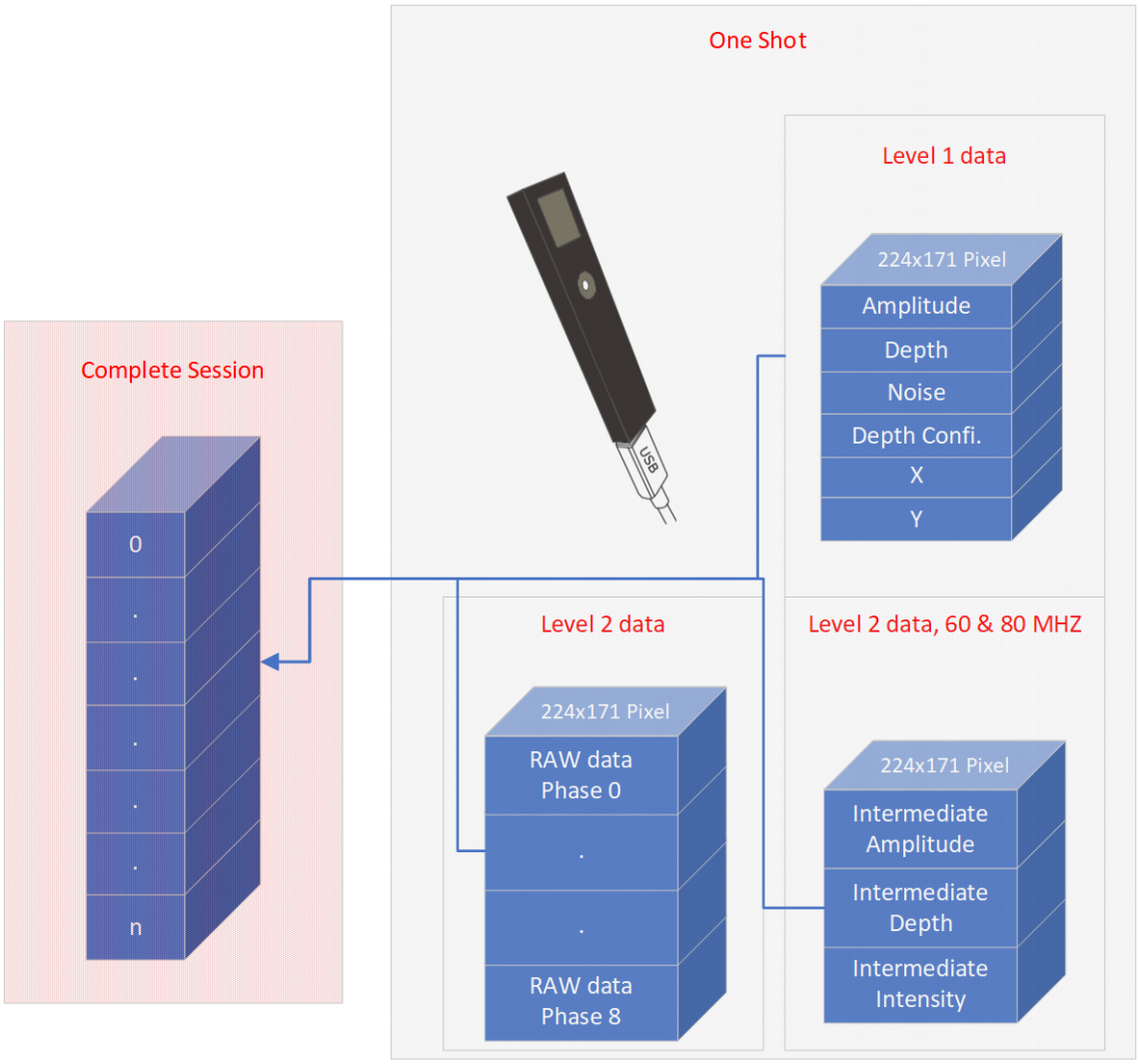


Figure 3.11: Visualisation of the capturing process on Android. The ToF camera delivers the standard level one data (amplitude, depth, noise, depth confidence, x and y). Level two access is needed to store the intermediate and the raw data. Several single shot instances result in a complete session.

calculate the similarity between two individuals. In C++ we configure the sensor with the tuned parameters. Additionally, we implemented the whole preprocessing pipeline in C++ and stored the captured and the preprocessed data to the storage.

The Java part includes the following:

GUI

With the help of the GUI, it is possible to change the sensor parameters and to start the capturing process with a certain amount of images. This amount is adjustable with the GUI. Additionally, information of the folder to save, emotions and the distance can be delivered to C++ with the help of the GUI. Java sends this information with Java Native Interface (JNI) calls to C++. After the C++ part finished with the sensor configuration, it sends back the current image from the sensor. The GUI can display this image in the JavaCameraView.

Manage application rights

With Java, we control the rights of our created application. For example, different Android versions need different storage management rights. Additional we control the front camera to capture the RGB images of the individuals. This step is necessary to test an RGB based network onto the data.

Control neural network

We control our created neural network with the help of the TensorFlow wrapper functions. The wrapper function calls a C++ function. Due to maintainability, we don't use the C++ API directly.

Calculate similarity between individuals

We use the designed CNN to extract facial features. These features are represented in a high dimensional space. Our application calculates the similarity between two input images with the help of a particular metric.

Control front camera

With the help of OpenCV, we can control the front camera. Our application saves the RGB image to the storage in the PNG format.

The C++ part includes the following:

Configure and communicate with the sensor

As described in Section 3.2.2, with the help of C++ and the unique employee access code to the camera we can change the sensor parameters. Additionally, we capture incoming frames as in Figure 3.13 described. Our application alternate the capturing between ToF camera and front camera with the help of JNI calls. This capturing process should guaranty that the ToF image is similar to the front camera image.

Capture and store data

We capture the data in a format as visualised in Figure 3.11. After the complete capturing of a set with a certain amount of images, the application saves the captured session to the storage. For this, each image is saved separately to the storage as a binary file. This method is not that efficient than saving all data into one file,

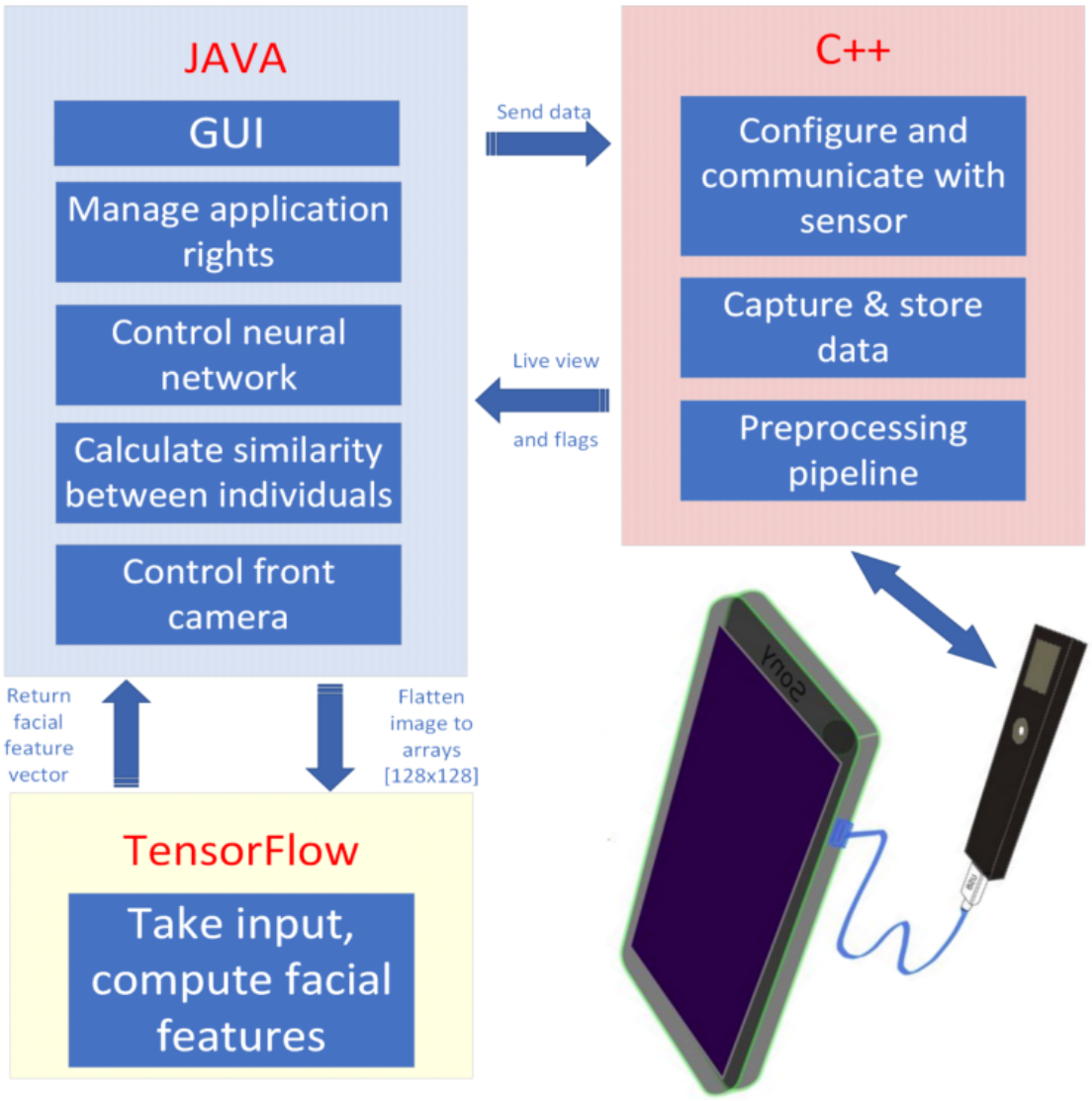


Figure 3.12: Collaboration of Java, C++ and TensorFlow.

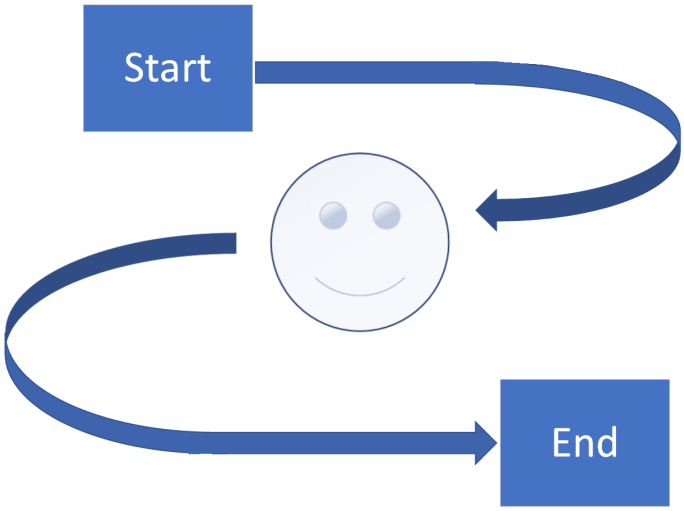


Figure 3.13: Illustration of capturing process. Each candidate is asked to move the smartphone around their face and take images as in this illustration. The continuous capturing from left to right should cover all login positions of a user. The candidates were asked to take this curves from different distances.

but with this separation, we can evaluate each image independently with the help of our created MatLab and Python scripts.

The individuals started the capturing process from top left of their face and moved the smartphone with the mounted ToF camera to the right side. If the individuals reached a position as in Figure 3.13, then they should move to the mid of their face and stop at the end position as illustrated. The smartphone captures around 250 images. If the candidates finished the s-curve than they can move the smartphone free over the whole face. Additionally, to the ToF data, we capture the RGB data from the front camera from the same view.

Preprocessing pipeline

Inspired by [MBO07] and [SKP15] we decided to create a preprocessing pipeline to separate the face from unwanted information. We hypothesise that we need less training samples because we removed potential problem sources. In the end, the network has two options: Learn the high-frequency border of the face after the cropping or the facial features itself. We break the high-frequency parts with our created pipeline and the used data augmentation (recall data augmentation in Section 2.3.1.1). In the end, the network must learn facial features. Generally, a network can learn the facial features without help, but with this segmentation, we help the network during the learning process, and we potentially need less training samples.

TensorFlow is used for:

Compute facial features

After the preprocessing pipeline finished the computation, the application sends

the processed data to the CNN. We build an interface to receive the data from Java. After the transmission of the data, we activate the CNN and fetch back the feature vector. With a specific metric, we can compare the two output vectors from two images together. Two images of the same individual should lead to a close range and two different individuals to a high distance.

Chapter 4

Verification Pipeline

In this chapter, we discuss our designed preprocessing pipeline. Additionally, we discuss the created CNN architectures, the neural network implementation on our Windows computer and data augmentation.

4.1 Preprocessing Pipeline

The aim of our work is a face recognition system which is suitable on an Android smartphone. For this, it is necessary to build most parts in C++ because we need high performance. Java is an interpreted language and not suitable for this requirement on our system. Our complete preprocessing pipeline is developed in C++ and also usable on our Linux system. Another problem is that not every function is ported from standard C++ to the Native Development Kit (NDK) which allows us to build C++ applications on Android. Due to this restriction, we started our whole implementation on Android and to not lose the focus on the performance of the smartphone. Another challenge is that the measurement should not be susceptible to errors because a stable demonstrator is the desired result.

In Figure 4.1 we illustrate our final preprocessing pipeline. We briefly explain the steps through the pipeline. After receiving the whole data vector, we iterate over all image sets. If an error occurs then the preprocessing pipeline drops this image set and iterates over the next. If the image set with amplitude, depth, noise, x, y and depth confidence passes the loading step, then we also check if the images contain information, *i.e.* the maximum value of the image must be higher than zero. With this check, we can prevent corrupted images. If the image is damaged, then the pipeline also drop the current image set. We use OpenCV for face detection with Haar-Cascades due to the high performance. We use for facial landmark detection an modified C++ implementation from GitHub[1]. If the face detection fails then also the landmark detection fails. If more than eight missing pixels in the inner face occur then the pipeline discard the image set. This step is necessary to prevent a potential rejection from the CNN in early stages because a rejection in our preprocessing pipeline is faster. If all rejection steps are passed, then our pipeline removes unwanted pixels in background and foreground which can't belong to the face. With our detected landmarks in amplitude data, we can take the depth data at the same pixel position and measure important points in the face. For example,

[1] https://github.com/memory/face-alignment

the nose, chin or head foreground. These areas are potentially the nearest points to the camera. After a first segmentation of the face from the background and some outliers, we shift the nearest point to zero. The idea is to stretch the whole face between zero and 255. Next, we rotate the whole face to zero degrees. In the next step, we use our noise image to create a binary mask dynamically for the precise segmentation of the face. This step is necessary because a neuronal network appreciates well-scaled data. If we remove all strong outliers, then we can scale the data with more precision from zero to 255. We prepare different borders for our neural network training because we want to evaluate the impact of different backgrounds and also to break the edge from the face to the background. We use geometric transformations for upscaling (cubic) and shifting the face in the middle of the 128x128 image. Due to our aligned 2D and 3D data, we can use the same methods for all data. Only the z-axis filtering is processed on depth data with a bilateral filter with a kernel size of 5 by 5. In the end, we save our preprocessed data on the storage. We don't send the data directly to Java because we can monitor the output of our preprocessing pipeline on the file system for each image.

Face and Landmark Detection

First, we load a Haar-Cascade from OpenCV for face detection. OpenCV face detector detects the inner face. For the landmark detection algorithm, we need the outer face. For this purpose, we use a scaling factor on the resulted bounding box (which includes the inner face, *i.e.* the eyes, nose and mouth) of around 30% to get the outer face (hairline, chin and ears) [Wan+10]. Next, we load the trained model file. With this file, we can build up the random forest. We use histogram equalisation on amplitude data to get better face and landmark detection results. Generally, the landmark detection works as expected, but also on RGB data problems occurred. In Figure 4.2 on the left side we illustrate how glasses influence the landmark detection on ToF data. We discovered the same effects with RGB images. The eyes and nose are detected imprecisely. We highlight this with arrows. On the right side of the figure, we illustrate a slightly right shifted dot mask due to wrong landmark detection. If it is not possible to recognise a face or landmarks, then the pipeline discard the image set and iterate over the next. Additionally, we measure the amount of over-saturated pixels in inner face. If more then eight pixels are missing then we discard the dataset. This value was determined empirically due to many observations in real images.

Remove Back and Foreground and Shift Face to Origin

We remove in an early state unwanted information, based on our face and landmark detection. Due to the 2D and 3D alignment of our data we can use the detected 2D landmarks and use them in our 3D image. We read out the nose tip, chin and forehead and remove everything in front of them, *i.e.* we set the specific pixel value to zero for all data (amplitude, depth, noise, x, y and depth confidence). Due to the imprecise landmarks we set a five per cent margin (from origin to the closest point in the face), *i.e.* we remove everything in front of the nearest location with a margin. With this technique, we shouldn't get holes in the face due to wrong

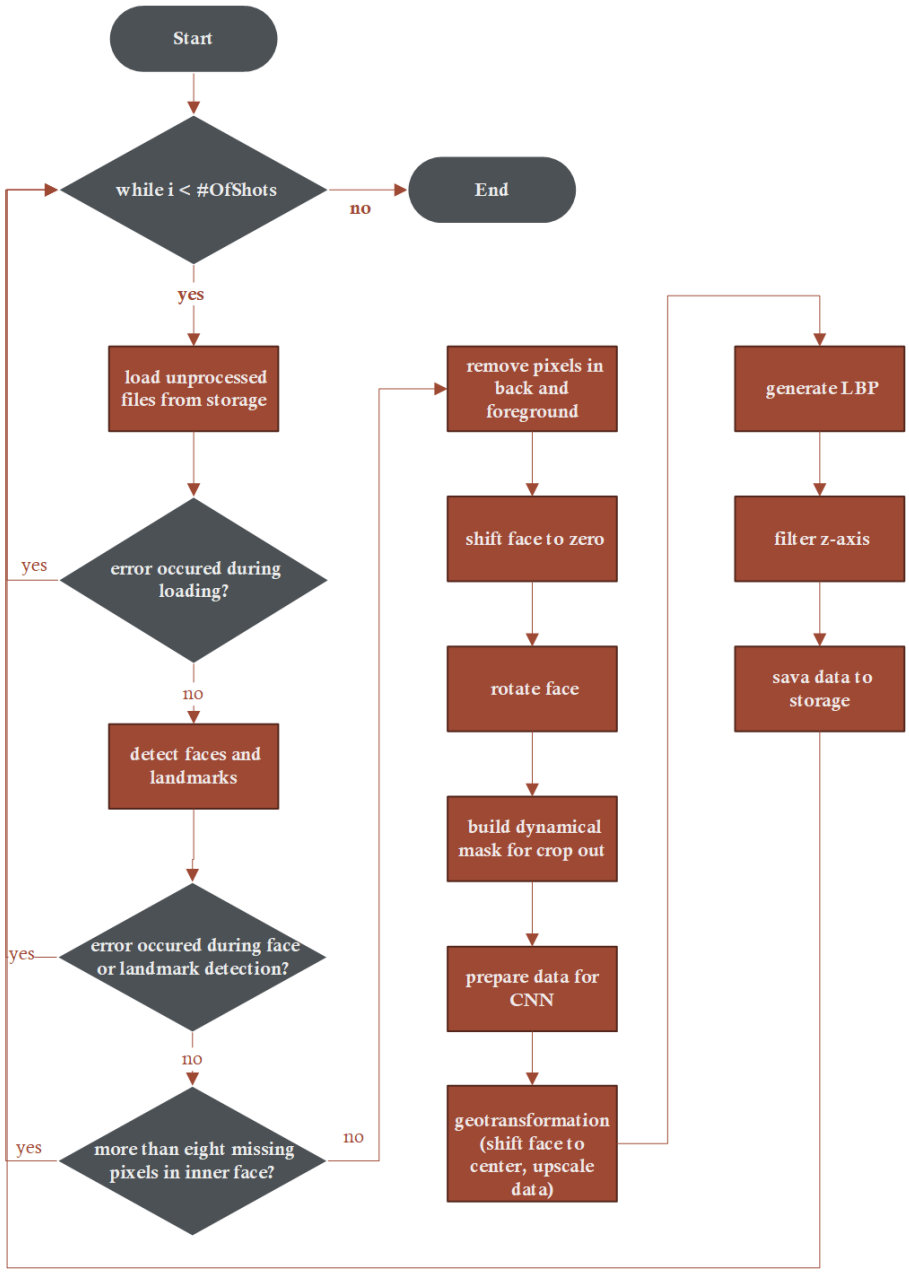


Figure 4.1: Complete preprocessing pipeline.

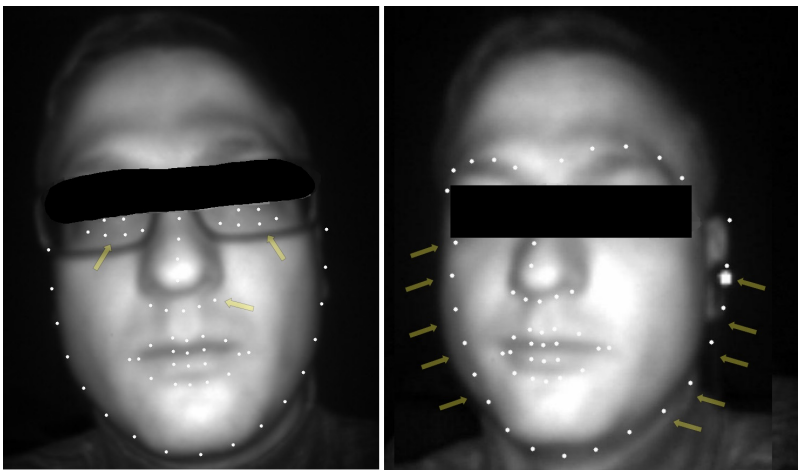


Figure 4.2: Problems with glasses for the landmark detection. Additionally, wrong detected landmarks are possible due to the blurred greyscale images from the camera.

detected landmarks. Additionally, we shift the whole face with the closest point to zero. This step is necessary to reach better scaling. In Figure 4.3 we illustrate the fore and background removing with followed zero shift of the face.

Rotate Face

To rotate the data, we need the processed landmarks. Other approaches ([MW09; MBO07]) find the nose and the face, for example, with 3D data (recall Section 2.3.3 and 2.3.4). In Equation (4.1) we calculate the distance between two eyes. We take the outer eye corners and measure the distance

$$\Delta d_x = l_{e_x} - r_{e_x}, \tag{4.1}$$

between two eyes, where l_{e_x} is the x coordinate of the left eye and r_{e_x} the x coordinate of the right eye. In Equation (4.2) we measure the height difference

$$\Delta d_y = l_{e_y} - r_{e_y}, \tag{4.2}$$

between two eyes, where l_{e_y} is the y coordinate of the left eye and r_{e_y} the y coordinate of the right eye. In Equation (4.3) we calculate the angle between the two eye corners. The angle is calculated as

$$\theta_{rad} = tan^{-1}\left(\frac{\Delta d_y}{\Delta d_x}\right). \tag{4.3}$$

This results in a radiant value. For the rotation matrix (Equation (4.4)) we need degrees. For this, a conversion from radiant to degree is necessary. The rotatation matrix to rotate an image is defined as

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}, \tag{4.4}$$

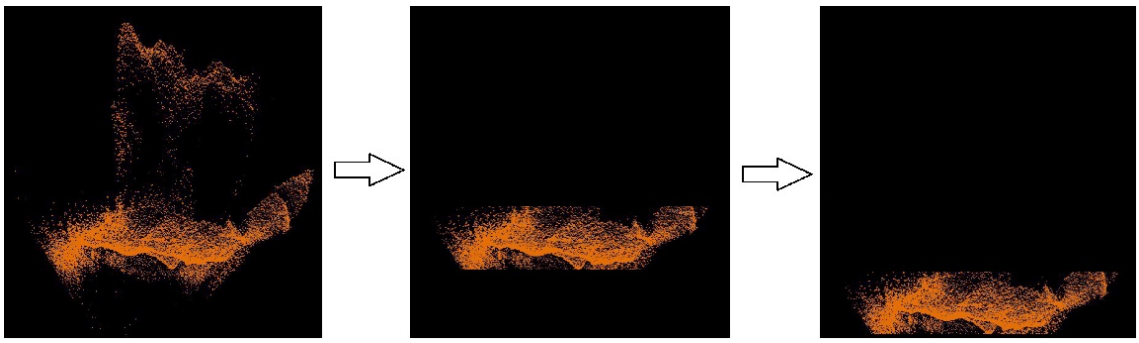


Figure 4.3: Remove outliers and unwanted information in front of the nose and background with a five per cent margin. This step is necessary to remove strong background reflections which we can't remove with the binary mask in the next step. Next, we shift the closest point (nose) and the whole face with a five per cent margin to nearly zero.

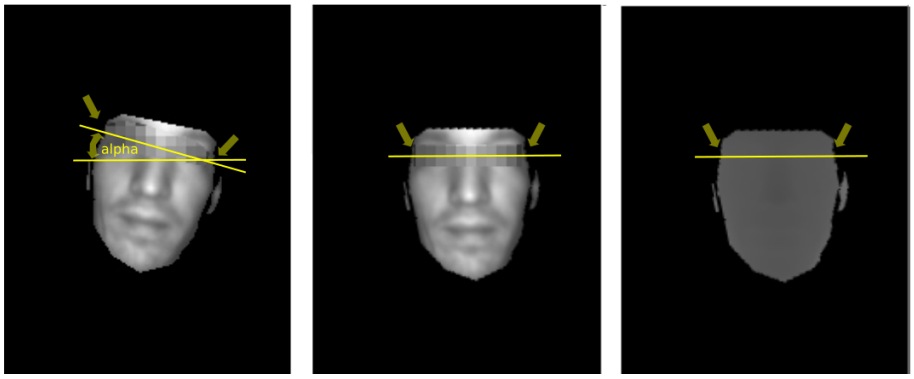


Figure 4.4: Rotation of the data with facial landmarks. Distance measurement between eye corners in height and width. With the resulting angle, a rotation of all data (due to the alignment of 2D and 3D data) is possible.

where θ denotes the angle in degree. With *warpAffine* from OpenCV and the rotation matrix we rotate all data.

In Figure 4.4 we visualise on the left side the cropped image without rotation. We illustrate the result after rotation in the middle with amplitude data and the last image with depth data.

Dynamic Bitmask Creation

In Figure 4.5 we illustrate with Pyntcloud[2] and 3D data from the camera how outliers in front of a glass wall influence the data. The image contains two humans. Outliers occur if the sensor gets multiple depth information in one pixel and the distance is not entirely distinguishable. Nothing is around these two humans, and strong outliers occur. Strong background reflections can lead to this effect. Our demonstrator should be able to handle all day situations. Due to this requirements,

[2] https://pyntcloud.readthedocs.io/en/latest/

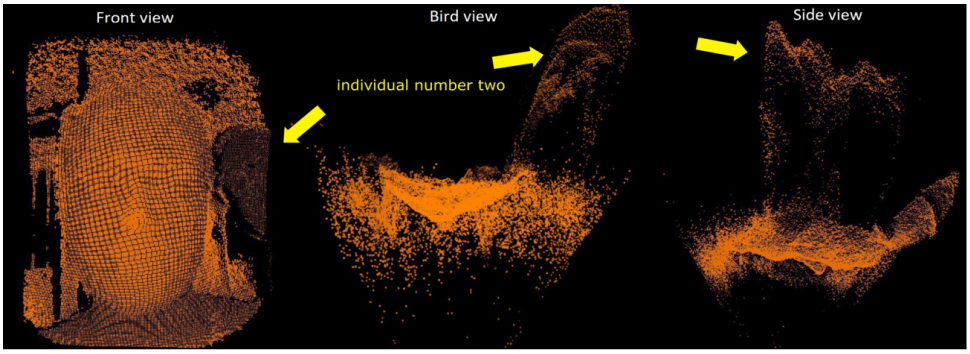


Figure 4.5: Different views of a face to demonstrate the outliers which occurred with two humans in front of a glass wall. From left to right: frontal view of two people, bird view and the view from the side.

we decided to segment the face with our noise image.

To segment the face reliably from additional unwanted data (*i.e.* noise and outliers) we use our noise image provided by the Royale framework. In Figure 4.6 we visualise the noise image from around 20 cm distance to the camera. The whole face has almost the same noise level, the outliers (right side with zoomed-in window) have higher noise values. For this, we segment high values with thresholding. We crop the face as in Figure 4.7. On the left side, we visualise the created coarse binary mask with the help of the detected landmarks. Next image illustrates the cropped noise image. We determine the highest noise level in the inner face (third image). Next, all values above the threshold are removed from the outer face (last image). In Equation (4.5) we illustrate our thresholding. The noise image pixel value can be calculated as

$$N(x,y) = \begin{cases} 0, & \text{if } N(x,y) > \tau \\ N(x,y), & \text{otherwise} \end{cases}, \tag{4.5}$$

where τ is our determined threshold (highest noise value) from the inner face. The pixel value of the noise image gets the value zero if the value is higher than the determined threshold. Next, we create a binary mask with the segmented noise image. In Equation (4.6) we summarise how we generate the binary mask. The current mask value at the same position as the noise value is determined as

$$B(x,y) = \begin{cases} 1, & \text{if } N(x,y) \neq 0 \\ 0, & \text{otherwise} \end{cases}. \tag{4.6}$$

To remove some additionally, unwanted artefacts (like parts of the ears), we use the erode operation. In Figure 4.8 on the left side we illustrate the created binary mask, on the right side, we illustrate the binary mask after the erode operation. Removed ears and a slightly smaller face is the result. This smaller face is desired because at the edges of the face occur the strongest outliers, with erode operation we can remove them. With the created binary mask we crop out all data, *i.e.* amplitude, depth, noise, x, y and depth confidence.

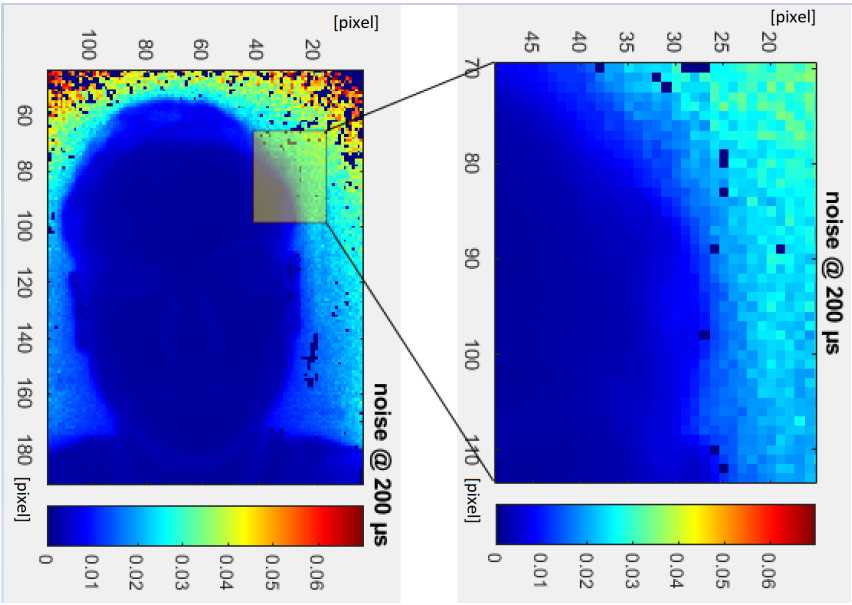


Figure 4.6: Noise data of a face image. We used pseudocolour for the visualisation. On the left side, we illustrate the face from around 20 cm distance. Right side a zoomed-in window. Note, that the whole face has nearly the same noise level. The background has a strong difference in the noise data.

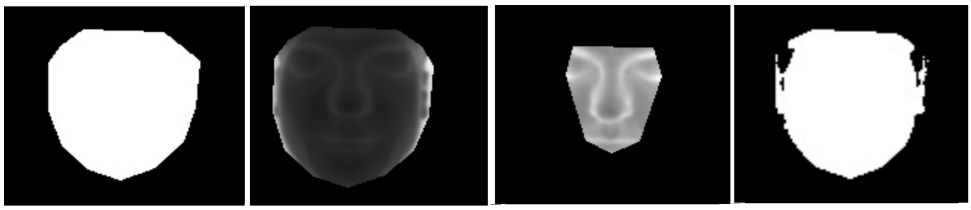
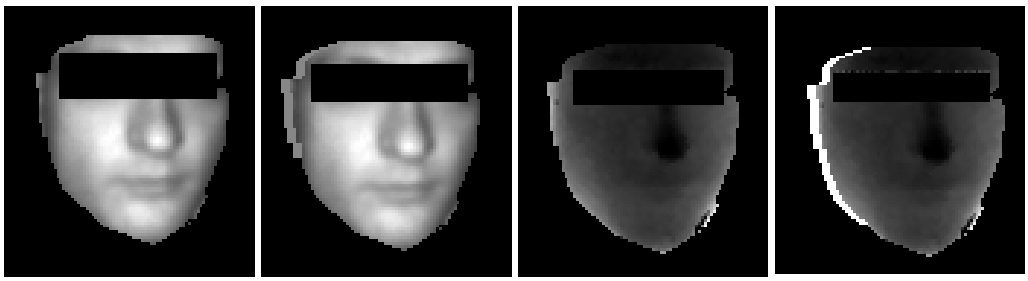


Figure 4.7: Steps for the dynamical bitmask creation. On the left side, we illustrate the binary mask with the help of the facial landmarks. Next image visualises the cropped noise image with this binary mask. The third image represents the inner face. We evaluate the highest noise value in this inner face and threshold the whole face on this value. The last image represents the resulting bitmask after thresholding.

Figure 4.8: On the left side, we visualise the coarse binary mask. On the right side, we illustrate the binary mask after the erode operation. The result is a smaller binary mask with less disturbing outliers.



(a) Hard edes after cropping.

(b) Smoothed edges on the greyscale image.

(c) Hard edes after cropping (based on the depth data).

(d) Smoothed edges on the depth image.

Figure 4.9: Breaking the hard edges around the face to prevent the edge learning of the network. Note that we changed the colour of the border for the visualisation.

Prepare Different Border and Smooth Edges

In this section, we summarise our approach for breaking the hard edges around the face. The CNN should not be able to focus on the edges of the face because this could lead to many false positive logins. In the end, the network should learn the facial features and not the edges around the face.

In Figure 4.9 we illustrate on the left side (a) the cropped face after the dynamical bitmask segmentation. Image (b) visualise with a border how our algorithm in the preprocessing pipeline move along the sharp edges around the face. The pipeline smooth the edges with a margin of three pixels, for better visualisation we doubled the width in image (b and d). Image (c) illustrates the depth image after cropping and image (d) the smoothed edges.

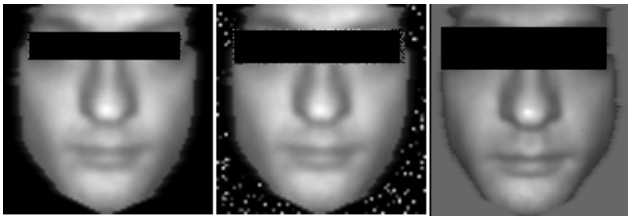


Figure 4.10: Visualisation of three different backgrounds.

We smooth the edges of the current pixel value $I(x,y)$ as

$$I(x,y) = \begin{cases} \frac{I(x,y)+I(x+1,y)+I(x-1,y)}{3}, & \text{if } I(x+1,y) \vee I(x-1,y) == 0 \\ I(x,y), & \text{otherwise} \end{cases}. \tag{4.7}$$

Initially, we constructed three different backgrounds. In Figure 4.10 we show smoothed edges and removed background, random noise on the background and average border. For the average border, the mean value over the whole face is calculated. We hypothesise, it is easier to break the high-frequency parts with a specially prepared background due to the strong difference between background and face. We have not prepared the local binary pattern (LBP) because further research is possible with the LBP images and face recognition as in [MS11].

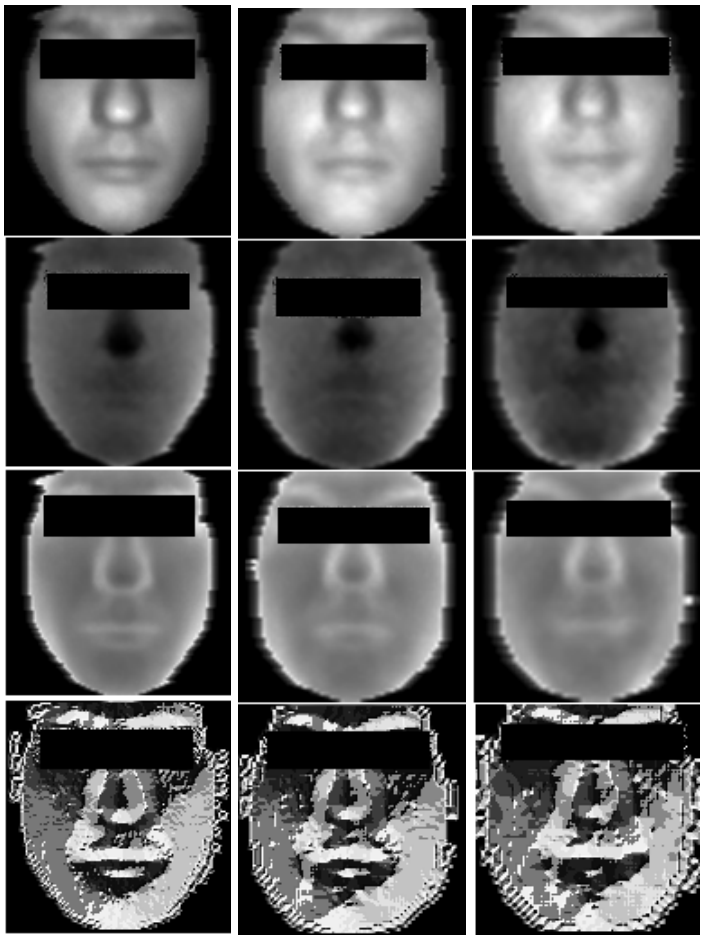
Geometrical transformation

Upscaling won't make our data better, but we hypothesise that our verification will be better because a neural network needs centred data for better learning. We shift the face into the middle of the image, and we upscale the data to 128x128 pixels. In Figure 4.11 we give an overview of how different distances change in quality. In Figure 4.12 we illustrate different distances from the depth images of Figure 4.11. With increasing distance, the quality gets worse.

In Figure 4.13 we illustrate the scalings up to 128x128 of the data at different distances. In the case of the first image, the distance is around 30 cm, and the upscale for this is 139 per cent. Next image is captured from a range of about 40 cm. The upscale to 128x128 pixels is 186 per cent. The third image needs an upscale of 241 per cent from a distance of 50 cm to reach the dimensions of the last image which represents the face with 128x128 pixels.

Local Binary Pattern

The LBP is popular for face recognition [MS11]. It is error prone if the image is noisy. Due to the noisy data of our sensor we want to evaluate if a CNN can handle this LBP with noisy data. Additionally, we hypothesise that more information for training leads to higher verification performance. We evaluate the LBP and combinations with other channels in Chapter 5. In Figure 4.14 we illustrate on the top left the amplitude image. Next, we visualise example values on the bottom left. We threshold with the value in the middle of the kernel the other values. The local

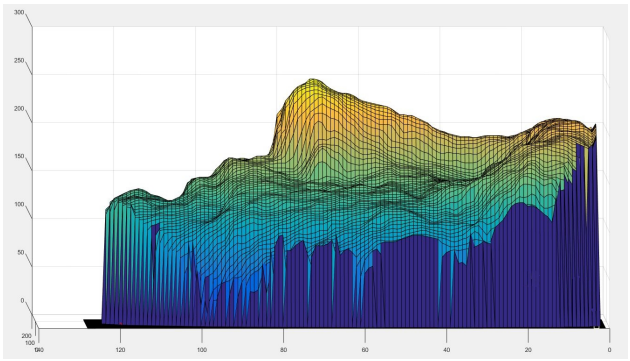


(a) Measurement from a distance of around 30 cm.

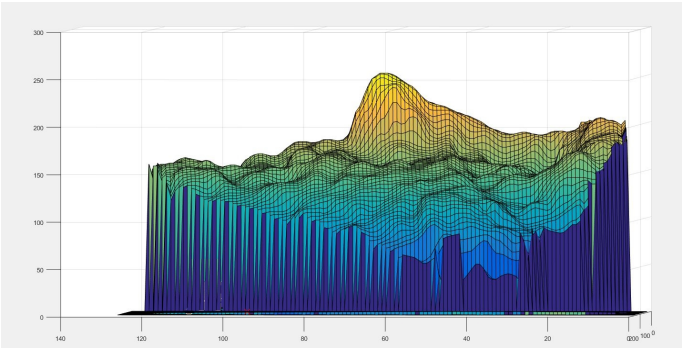
(b) Measurement from a distance of around 40 cm.

(c) Measurement from a distance of around 50 cm.

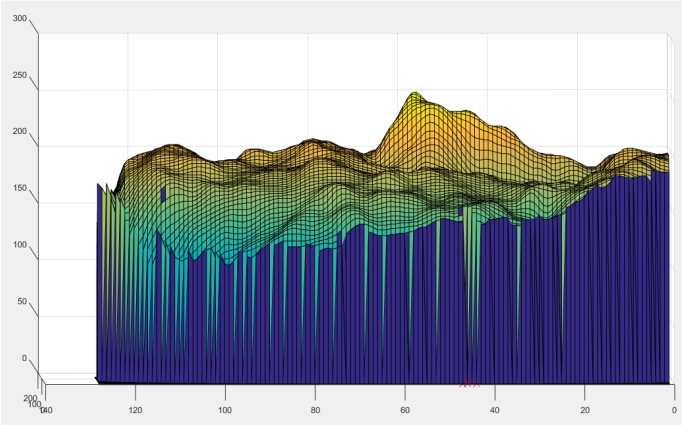
Figure 4.11: We illustrate three different captured distances from a frontal view, with rising distance the quality getting more and more blurred and noisy. We enumerate the images from top to bottom: the first image is our greyscale image, the second is the depth image, the third row illustrates the noise image and the last image is our handcrafted LBP. Image (a) is captured from a distance of around 30 cm. The quality is acceptable. Figure 4.11 (b) illustrates the measurement from a distance of 40 cm, it is possible to see an apparent loss in quality in the depth image (best viewed digitally). In the greyscale image at the top, we can see that the image is more blurred. In the second row, our depth image is noisier and also our noise and LBP image.



(a) Depth image from sideview at a distance from around 30 cm.



(b) Depth image from sideview at a distance from around 40 cm. Note, that the chin is not clearly visible.



(c) Depth image from sideview at a distance from around 50 cm. A quality drop is recognisable.

Figure 4.12: Three different distances, with rising distance the quality drops. In (a) we illustrate the depth image from side view at 30 cm distance. In (b) we illustrate the depth image from 40 cm distance. A quality drop in the surface is recognisable, for example, the chin is not clearly visible. At a distance of 50 cm (image (c)), the contour is not that detailed as at 30 or 40 cm distance. This quality drop can lead to less performance in the verification.

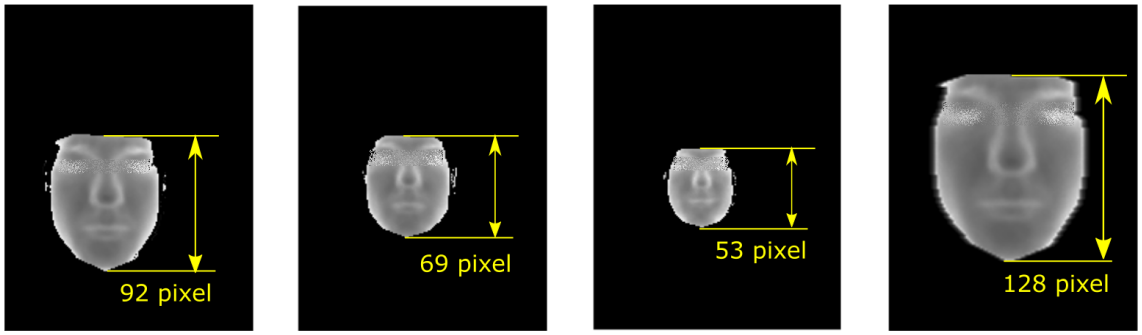


Figure 4.13: Visualisation of the scalings from different distances. The first image is captured from a range of around 30 cm away from the camera. The upscaling factor is 1.39. The second image from left is captured from about 40 cm, and our upscaling factor is around 1.86. The third image from a distance of around 50 cm has the highest scaling factor, upscaling to 128 pixels in height results in the scaling of approximately 2.41.

binary value L can be calculated as

$$L(x,y) = \begin{cases} 1, & \text{if } I(x,y) \geq \tau \\ 0, & \text{otherwise} \end{cases}, \tag{4.8}$$

where τ is the threshold. The resulting binary matrix is illustrated in the middle of the image. The LBP values must be concatenated together. For this, the first value (zero) is the last value after the concatenation [MS11]. The result in decimal is 118. This value represents the value in the LBP image. Our LBP implementation is inspired by OpenCV and a GitHub implementation[3] which relates to the OpenCV implementation.

4.2 Reference Image Selection

To improve the quality of the reference images, we select the best reference images for the verification out of a set. For this, we calculate for all captured reference images the feature vectors with the help of the trained CNN. Next, we calculate the mean value over all reference vectors and sort the reference images according to the distance from the mean reference vector. The best reference image is the image with the closest distance to the mean vector. We hypothesise that most login images score feature vectors near the mean reference vector. We take as reference images the best image, one in the middle and the worst one (highest distance to mean vector) as our reference images. In the next chapter, we evaluate the results for the best reference image selection. In Figure 4.15 we illustrate the sorted images and the selection of the desired images.

[3] https://github.com/bytefish/opencv

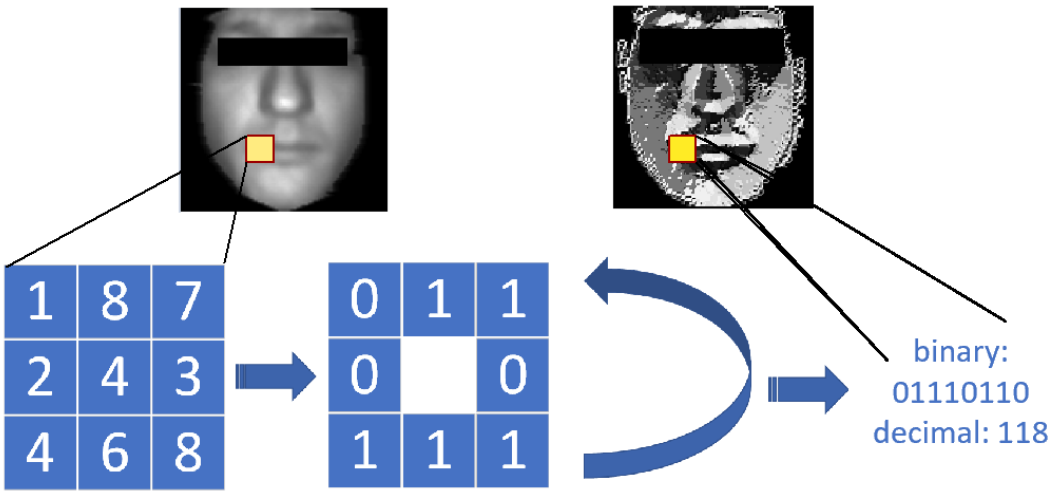


Figure 4.14: Generation of LBP. Figure based on [MS11].

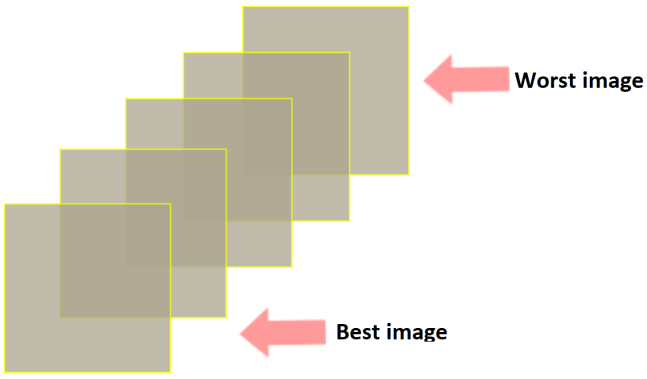


Figure 4.15: Illustration of referenece image selection. We select the images with following distances to the mean feature vector: take the best image (nearest distance to mean), the worst one and one between them.

Table 4.1: Neural network with a standard structure. We denote this version as CNNV1. Large kernels in the first layer can learn local features.

Layer	Type	Properties	output size
—	Input	128x128	—
—	BN	input: 128x128	128x128
Layer 1	Conv2d	Z: 16x16, S: 1x1, K: 25	128x128
Layer 2	Maxpool	Z: 2x2, S : 2x2	64x64
Layer 3	Conv2d	Z: 16x16, S: 1x1, K: 25	64x64
Layer 4	Maxpool	Z: 2x2, S: 2x2	32x32
Layer 5	Conv2d	Z: 16x16, S: 1x1, K: 25	32x32
Layer 6	Maxpool	Z: 2x2, S: 2x2	16x16
Layer 7	Conv2d	Z: 16x16, S: 1x1, K: 25	16x16
Layer 8	Maxpool	Z: 2x2, S: 2x2	8x8
Layer 9	Fully connected	1000	1000
Layer 10	Fully connected	128	128

4.3 CNN Architectures

In this section, we summarise the created networks. We created different CNN architectures with different backgrounds. At the beginning of this section, we summarise the used techniques for training, followed by the network architectures itself.

For all our experiments we use Batch Normalisation (BN) because in literature we discovered that a faster convergence during training and higher verification performance is the result [TTC18; XJT17; Lau+16; IS15]. Additionally, the BN in the first layers of the CNN can improve the performance of the computation between 30–50% [Dua+18]. Each layer is dependent on the layer before. Deep networks bear under the internal covariate shift, a lower learning rate during training and careful initialisation of the parameters is necessary [IS15]. The idea behind BN is that it normalises each training batch and face the problem of the covariate shift. We use in all experiments for the convolutional layers the leaky ReLu activation function and XAVIER initialiser [GB10]. Additionally, we use the ADAM [KB15] optimiser for the weight manipulations. We shorten for the next tables the kernel size with Z, stride with S and the number of kernels with K.

In Table 4.1 we illustrate the CNNV1. The kernels in the first layers are large with a size of 16x16 pixels. We hypothesise that a feature extraction with this large kernels is possible. In the last layers, we placed two fully-connected layers (inspired by FaceNet [SKP15]). In Table 4.2 we illustrate the CNNV2. The kernels in the first layers are small with a size of 5x5 pixels. With these small kernels, it is possible to learn edge detectors. In the last layers, we placed three fully-connected layers. In Table 4.3 we illustrate the CNNV3. The structure is identical to CNNV2, expect in the last layers, we placed two fully-connected layers. The CNNV2 version has one fully-connected layer more, and we evaluate the impact of this additional layer in the next chapter. Inspired

Table 4.2: Neural network with a standard structure. We denote this version as CNNV2. Small kernels in the first layer can learn edge detectors.

Layer	Type	Properties	output size
—	Input	128x128	—
—	BN	input: 128x128	128x128
Layer 1	Conv2d	Z: 5x5, S: 1x1, K: 25	128x128
Layer 2	Maxpool	Z: 2x2, S: 2x2	64x64
Layer 3	Conv2d	Z: 9x9, S: 1x1, K: 25	64x64
Layer 4	Maxpool	Z: 2x2, S: 2x2	32x32
Layer 5	Conv2d	Z: 16x16, S: 1x1, K: 25	32x32
Layer 6	Maxpool	Z: 2x2, S: 2x2	16x16
Layer 7	Conv2d	Z: 16x16, S: 1x1, K: 25	16x16
Layer 8	Maxpool	Z: 2x2, S: 2x2	8x8
Layer 9	Fully connected	1000	1000
Layer 10	Fully connected	1000	1000
Layer 11	Fully connected	128	128

Table 4.3: Neural network with a standard structure. We denote this version as CNNV3. Small kernels in the first layer can learn edge detectors.

Layer	Type	Properties	output size
—	Input	128x128	—
—	BN	input: 128x128	128x128
Layer 1	Conv2d	Z: 5x5, S: 1x1, K: 25	128x128
Layer 2	Maxpool	Z: 2x2, S: 2x2	64x64
Layer 3	Conv2d	Z: 16x16, S: 1x1, K: 25	64x64
Layer 4	Maxpool	Z: 2x2, S: 2x2	32x32
Layer 5	Conv2d	Z: 16x16, S: 1x1, K: 25	32x32
Layer 6	Maxpool	Z: 2x2, S: 2x2	16x16
Layer 7	Conv2d	Z: 16x16, S: 1x1, K: 25	16x16
Layer 8	Maxpool	Z: 2x2, S: 2x2	8x8
Layer 9	Fully connected	512	512
Layer 10	Fully connected	128	128

Table 4.4: Neural network with a standard structure. We denote this version as CNNV4. Large kernels in the first layer can learn local features. FaceNet inspires the fully-connected layers.

Layer	Type	Properties	output size
—	Input	128x128	—
—	BN	input: 128x128	128x128
Layer 1	Conv2d	Z: 16x16, S: 1x1, K: 25	128x128
Layer 2	Maxpool	Z: 2x2, S: 2x2	64x64
Layer 3	Conv2d	Z: 16x16, S: 1x1, K: 25	64x64
Layer 4	Maxpool	Z: 2x2, S: 2x2	32x32
Layer 5	Conv2d	Z: 16x16, S: 1x1, K: 25	32x32
Layer 6	Maxpool	Z: 2x2, S: 2x2	16x16
Layer 7	Conv2d	Z: 16x16, S: 1x1, K: 25	16x16
Layer 8	Maxpool	Z: 2x2, S: 2x2	8x8
Layer 9	Fully connected	256	256
Layer 10	Fully connected	128	128

by the Zeiler&Fergus [ZF14] architecture in FaceNet, we placed a 256 and 128 dimensional fully-connected layer as the last layers (Table 4.4). The structure before is identical to CNNV1. In Table 4.5 we illustrate the CNNV5. The kernels in the first layers are sized as in CNNV1 and CNNV4 with a size of 16x16 pixels. We placed only one fully-connected layer because the NIRFaceNet [Pen+16] has no fully-connected layer, but we need one for the 128 feature embedding. The performance improvement is higher because of the removed fully-connected layer. In Table 4.6 we illustrate our CNNV6. We paced large kernels for feature detection in the first layer. In the last layer, we placed a fully-connected layer with 1000 entries and one with only a face embedding dimension of 64. We hypothesise that a smaller embedding can improve the performance on our small network. In contrast, FaceNet uses 128 on a deep network with a big database for training. In Table 4.7 we illustrate our NIRFaceNetV1 with fewer kernels than the original NIRFaceNet [Pen+16] implementation. The aim is to reach high performance with fewer parameters to improve the calculation time on a smartphone. We use two fully-connected layers as FaceNet with the Zeiler&Fergus architecture. In Table 4.8 we illustrate our NIRFaceNetV2 with fewer kernels than the original implementation with the same aim as in NIRFaceNetV1. For this architecture, we use one fully connected layer as in CNNV5 to improve the computation performance. In Table 4.9 we illustrate our NIRFaceNetV3, which represents the original NIRFaceNet with removed SoftMax layer. We use one fully-connected layer as in NIRFaceNetV2 to improve the performance and to realise the Triplet Loss training. In the end, we use in V3 more kernels but only one fully-connected layer.

Table 4.5: Neural network with a standard structure. We denote this version as CNNV5. Large kernels in the first layer can learn local features. NIR-FaceNet inspires the single fully-connected layer.

Layer	Type	Properties	output size
—	Input	128x128	—
—	BN	input: 128x128	128x128
Layer 1	Conv2d	Z: 16x16, S: 1x1, K: 25	128x128
Layer 2	Maxpool	Z: 2x2, S: 2x2	64x64
Layer 3	Conv2d	Z: 16x16, S: 1x1, K: 25	64x64
Layer 4	Maxpool	Z: 2x2, S: 2x2	32x32
Layer 5	Conv2d	Z: 16x16, S: 1x1, K: 25	32x32
Layer 6	Maxpool	Z: 2x2, S: 2x2	16x16
Layer 7	Conv2d	Z: 16x16, S: 1x1, K: 25	16x16
Layer 8	Maxpool	Z: 2x2, S: 2x2	8x8
Layer 9	Fully connected	128	128

Table 4.6: Neural network with a standard structure. We denote this version as CNNV6. Large kernels in the first layer can learn local features.

Layer	Type	Properties	output size
—	Input	128x128	—
—	BN	input: 128x128	128x128
Layer 1	Conv2d	Z: 16x16, S: 1x1, K: 25	128x128
Layer 2	Maxpool	Z: 2x2, S: 2x2	64x64
Layer 3	Conv2d	Z: 16x16, S: 1x1, K: 25	64x64
Layer 4	Maxpool	Z: 2x2, S: 2x2	32x32
Layer 5	Conv2d	Z: 16x16, S: 1x1, K: 25	32x32
Layer 6	Maxpool	Z: 2x2, S: 2x2	16x16
Layer 7	Conv2d	Z: 16x16, S: 1x1, K: 25	16x16
Layer 8	Maxpool	Z: 2x2, S: 2x2	8x8
Layer 9	Fully connected	1000	1000
Layer 10	Fully connected	64	64

Table 4.7: Inception based neural network. We used NIRFaceNet and modified the network (two fully connected layers, fewer kernels, different stride, no max pooling between inception modules and removed Softmax). We denote this version as NIRFaceNetV1.

Layer	Type	Properties	output size
—	Input	128x128	—
—	BN	input: 128x128	128x128
Layer 1	Conv2d	Z: 5x5, S: 1x1, K: 16	128x128
Layer 2	Maxpool	Z: 3x3, S: 1x1	128x128
—	Local response normalisation	—	128x128
Layer 3	Conv2dA	Z: 1x1, S: 1x1, K: 16	128x128
Layer 3	Conv2dB	Z: 1x1, S: 1x1, K: 16	128x128
Layer 3	Conv2dC	Z: 3x3, S: 1x1, K: 32	128x128
Layer 4	Maxpool	Z: 3x3, S: 1x1	128x128
Layer 4	Conv2dD	Z: 1x1, S: 1x1, K: 16	128x128
—	Concat	Conv2dA, Conv2dC, Conv2dD	—
Layer 5	Conv2dA	Z: 1x1, S: 1x1, K: 16	128x128
Layer 5	Conv2dB	Z: 1x1, S: 1x1, K: 16	128x128
Layer 5	Conv2dC	Z: 3x3, S: 1x1, K: 32	128x128
Layer 6	Maxpool	Z: 3x3, S: 1x1	128x128
Layer 6	Conv2dD	Z: 1x1, S: 1x1, K: 16	128x128
—	Concat	Conv2dA, Conv2dC, Conv2dD	—
Layer 7	Maxpool	Z: 3x3, S: 2x2	64x64
Layer 8	Fully connected	256	256
Layer 9	Fully connected	128	128

Table 4.8: Inception based neural network. We used NIRFaceNet and modified the network slightly (one fully connected layer, fewer kernels, no max pooling between inception modules and removed Softmax). We denote this version as NIRFaceNetV2.

Layer	**Type**	**Properties**	**output size**
—	Input	128x128	—
—	BN	input: 128x128	128x128
Layer 1	Conv2d	Z: 5x5, LR, S: 1x1, K: 64	128x128
Layer 2	Maxpool	Z: 3x3, S: 2x2	64x64
—	Local response normalisation	—	64x64
Layer 3	Conv2dA	Z: 1x1, S: 1x1, K: 16	64x64
Layer 3	Conv2dB	Z: 1x1, S: 1x1, K: 16	64x64
Layer 3	Conv2dC	Z: 3x3, S: 1x1, K: 16	64x64
Layer 4	Maxpool	Z: 3x3, S: 1x1	64x64
Layer 4	Conv2dD	Z: 1x1, S: 1x1, K: 16	64x64
—	Concat	Conv2dA, Conv2dC, Conv2dD	—
Layer 5	Conv2dA	Z: 1x1, S: 1x1, K: 16	64x64
Layer 5	Conv2dB	Z: 1x1, S: 1x1, K: 16	64x64
Layer 5	Conv2dC	Z: 3x3, S: 1x1, K: 32	64x64
Layer 7	Maxpool	Z: 3x3, S: 1x1	64x64
Layer 7	Conv2dD	Z: 1x1, S: 1x1, K: 16	64x64
—	Concat	Conv2dA, Conv2dC, Conv2dD	—
Layer 8	Maxpool	kernel: 3x3, S: 2x2	32x32
Layer 9	Fully connected	128	128

Table 4.9: Inception based neural network. This represents the original version of NIRFaceNet with removed Softmax.

Layer	Type	Properties	output size
—	Input	128x128	—
—	BN	input: 128x128	128x128
Layer 1	Conv2d	Z: 5x5, LR, S: 2x2, K: 64	64x64
Layer 2	Maxpool	Z: 3x3, S: 2x2	32x32
—	Local response normalisation	—	32x32
Layer 3	Conv2dA	Z: 1x1, S: 1x1, K: 64	32x32
Layer 3	Conv2dB	Z: 1x1, S: 1x1, K: 64	32x32
Layer 3	Conv2dC	Z: 3x3, S: 1x1, K: 128	32x32
Layer 4	Maxpool	Z: 3x3, S: 1x1	32x32
Layer 4	Conv2dD	Z: 1x1, S: 1x1, K: 64	32x32
—	Concat	Conv2dA, Conv2dC, Conv2dD	—
Layer 5	Maxpool	Z: 3x3, S: 2x2	16x16
Layer 5	Conv2dA	Z: 1x1, S: 1x1, K: 128	16x16
Layer 5	Conv2dB	Z: 1x1, S: 1x1, K: 128	16x16
Layer 5	Conv2dC	Z: 3x3, S: 1x1, K: 192	16x16
Layer 6	Maxpool	Z: 3x3, S: 1x1	16x16
Layer 6	Conv2dD	Z: 1x1, S: 1x1, K: 128	16x16
—	Concat	Conv2dA, Conv2dC, Conv2dD	—
Layer 7	Maxpool	Z: 3x3, S: 2x2	8x8
Layer 8	Fully connected	128	128

4.4 Approach

NIRFaceNet is trained on 597 near-infrared images. To increase the size of the training database, data augmentation is used. FaceNet is trained on an extensive database with RGB data, the authors from NIRFaceNet claim that they get better results with a shallow structure network on a smaller database compared to FaceNet. Additionally, they use optimised parameters for standard systems (batch size of 35 instead of 1500 like FaceNet). Last, NIRFaceNet is suitable for the smartphone. We modify the NIRFaceNet to identify unknown individuals, for this, we removed the Softmax layer and integrated another metric learning approach like Triplet Loss from FaceNet. We use BN for faster and better training, and we use max pooling from NIRFaceNet instead of L2 pooling from FaceNet. We found two reasons for this: first, NIRFaceNet is created for less training data and score a slightly better performance than deep networks like GoogLeNet [Sze+15]. Second, the network is trained on near-infrared data. Additionally, the network has feature extraction modules (inception modules) with removed five by five layer, and this makes the training faster. In the end, the NIRFaceNet is a newer version of a CNN as FaceNet and GoogLeNet or DeepFace [Tai+14]. Due to this reasons, we align our ToF CNN much more to the NIRFaceNet.

4.5 TensorFlow

We use TensorFlow [Aba+15] as the framework for neural networks in our project due to the compatibility to Android. In Figure 4.16 we illustrate how we implemented our TensorFlow solution for training. First, we load the images from the storage. Next, we stack the data together like an RGB image. We generate a stacked image for the reference image, a stacked image for the positive image (same individual) and a negative image (different individual). We build up three CNN structures with shared parameters, and this means we need less memory but the computation time of three networks. The network outputs a feature vector with the facial features for the reference (anchor), the positive and negative sample. We calculate the loss with Triplet Loss and backpropagate the gradients with the ADAM optimiser.

After the training, we freeze the model to use it on Android. For this, we created a Python program which loads the model, removes all unnecessary weights and optimise it for Android. Additionally, we created the API to send the data from Android to the network model. The API takes flattened images with the specified amount of data (amplitude, depth, noise, LBP). It is necessary to flatten the whole image to 16384 pixels (128x128).

Inspired by NIRFaceNet, we use data augmentation to generate additional images for the training of the network. Another effect is to prevent overfitting because a changing of the data with random rotation, noise and brightness can face the problem. Another critical point to use data augmentation is to reach a resistance against small rotations if the landmark detection fails or if the noise level rises outdoors. In Figure 4.17 we illustrate four images with data augmentation during a training run. The first image is based on amplitude data, the second image on depth data. Next image represents the noise image, and the last image is the generated LBP.

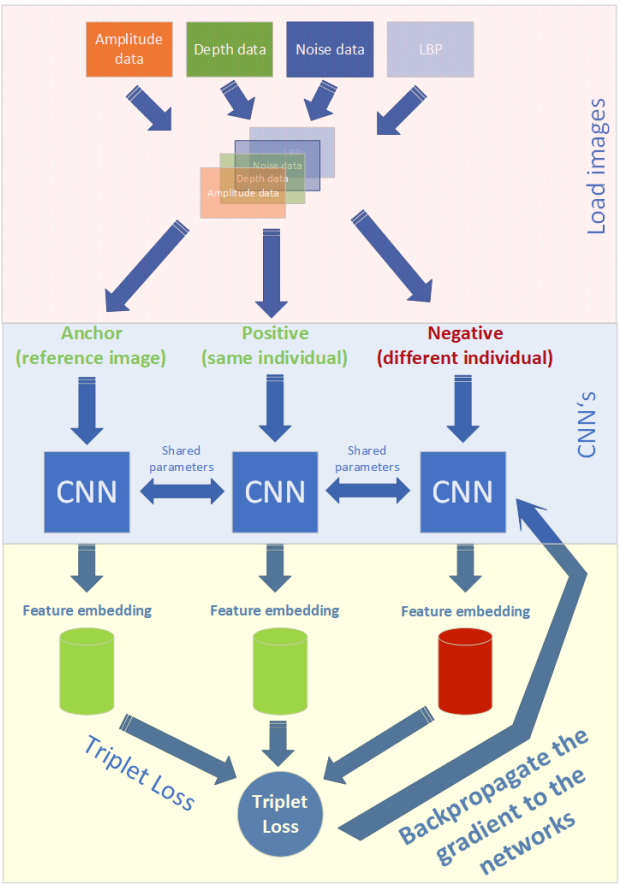


Figure 4.16: Visualisation of the code we created in Python and TensorFlow. First, we load the images and stack them together to one image. Next, we generate anchor images (reference image), positive images (same individual) and negative samples (different individual). We send this data to the constructed networks and receive the feature vectors for each image. Note, that we use shared parameters, this means, we build up three networks which share the parameters between them. This leads to less memory consumption. We calculate the loss with the three feature vectors, and the optimiser backpropagates the gradients of the CNN's. Illustration design is inspired by OpenFace from GitHub[a]. Visited on 17.11.2018

[a] http://bamos.github.io/2016/01/19/openface-0.2.0/

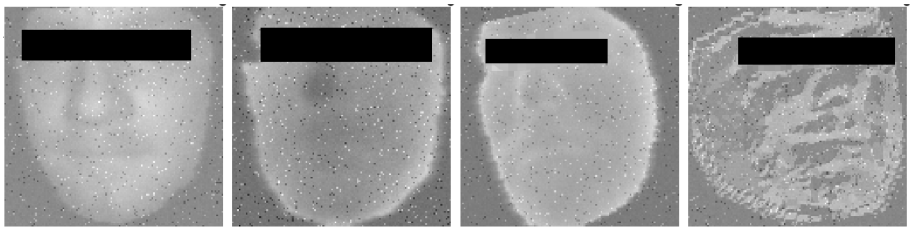


Figure 4.17: Single channels with data augmentation, from left to right: amplitude, depth, noise data and LBP.

Table 4.10: We augment our data with a rotation of ±6°, little brightness and contrast adjustments and add Gaussian noise to prevent overfitting. The random contrast adjustment factor is between the lower bound (0.5) and the upper bound (1.5). The random brightness adjustment factor is between -0.1 and 0.1.

Description	**Range**
Rotation	±6°
Brightness	max delta: 0.1
Contrast	lower = 0.5, upper = 1.5
Gaussian noise	standard deviation: 0.005

We use for data augmentation the parameters as illustrated in Table 4.10. The image rotation is around ±6°. We use little brightness and contrast adjustment because strong reflections on objects like the teeth or the eyes can lead to a dark image due to the auto exposure. Additionally, we add Gaussian noise to break high-frequency parts in the image and additionally prevent the network from learning the edges around the face.

Chapter 5

Evaluation

In this chapter, we evaluate our created database, the preprocessing pipeline, weaknesses in the face and landmark detection and different network architectures. Additionally, we assess the best hyperparameters, the final implementation on Android and we evaluate the impact of the sun. Last, we try to fool our solution with printed images. We tested AlexNet [KSH12] on the captured and cropped front camera images to compare the RGB based network, which is trained on objects, against our solution. This trivial test reflects the impact of some aspects like image quality and resolution on the network. A commonly used technique in neural networks is transfer learning. With this technique, it is possible to use pre-trained network weights and modify or retrain the network with these parameters. We use a pre-trained AlexNet[1] for the front camera benchmark, but we didn't use transfer learning for our ToF based solution because we could not find networks which are trained on these data. Due to the restrictions to our Android smartphone, it is not possible to build deep networks like FaceNet [SKP15] and guarantee a login in a user-friendly time. The preprocessing pipeline should be fast enough that login in an accurate time is possible. Due to time reasons, we won't optimise our solution to reach the maximum performance, but we always keep in mind the performance problem. At the end of this chapter, we summarise some performance statistics.

5.1 Dataset

We have to build a prototype as in Chapter 3 described to capture the individuals as desired. We record amplitude, depth, noise, depth confidence, x and y data. We captured 47 different individuals with different emotions, *i.e.* neutral, happy, sad and angry. This lead to 30,845 ToF images which passed the restrictions and sanity checks in our preprocessing pipeline (recall Section 4.1). Most of the rejections caused by oversaturated pixels and undetected faces in the image due to a too strong side view.

Our data set is split into a training set, a validation set and a test set. We divided the captured 47 individuals as follows: 33 into the training set, 6 into the validation and 8 to the test set. This results in a split of 80% for training, 11% for validation and 9% for testing (based on the resulted images). The training set consists of 24773 images. With our validation set, we have tuned the hyperparameters of the networks. The validation set includes 3418 images. Additionally, we monitored with the training and validation

[1] http://www.cs.toronto.edu/~guerzhoy/tf_alexnet/

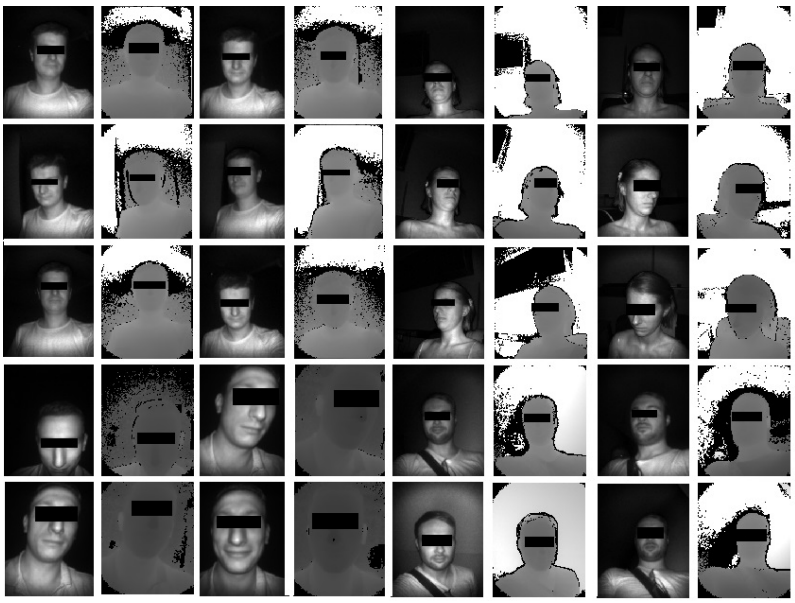


Figure 5.1: Visualisation of four individuals in the training set. Different distances, pose and emotions are recognisable. Odd columns represent the greyscale images, even columns the depth images. Note that bright pixels in the depth image indicate more distance.

set overfitting, *i.e.* if the login rate rises in training but falls on the validation set. We sample each 20th image from the training set to monitor the training. The last set, the test set, should reflect the real world and the generalisation ability of the network to completely unseen data. This set contains 2654 images.

In Figure 5.1 we visualise a small part of the training data of four different individuals. We always visualise a pair of data, *i.e.* amplitude and depth data. We didn't visualise the noise data, because most images are incredibly dark and not useful for this visualisation. Different pose and different facial expressions are recognisable in the training set. Depending on the distance and background, bright pixels indicate more distance and dark pixels a closer distance to the camera. In Figure 5.2 we visualise four individuals of the validation set. Again we visualise a pair of amplitude and depth data. Different pose and distances are recognisable. Illumination changes, as in image five in the first row, are caused by the strong background reflection. Due to this effects, we use data augmentation to overcome this illumination changing. In Figure 5.3 we visualise four individuals of the test set. We build this dataset with the same settings as the training and validation set. We captured different pose, emotions and distances. Our recorded RGB dataset consists of around 43,000 images (33.496 images for training, 4951 for validation and 4522 for testing). In Figure 5.4 we illustrate samples of the RGB test set. Three individuals didn't want to capture the RGB images. We replaced these three candidates with three individuals from the training set.

Due to the rejections in the preprocessing pipeline, we lost many ToF measurements. Additional, we lost images due to problems during the recording, *i.e.* the participants forgot to move the smartphone with mounted ToF camera correctly, which led to back-

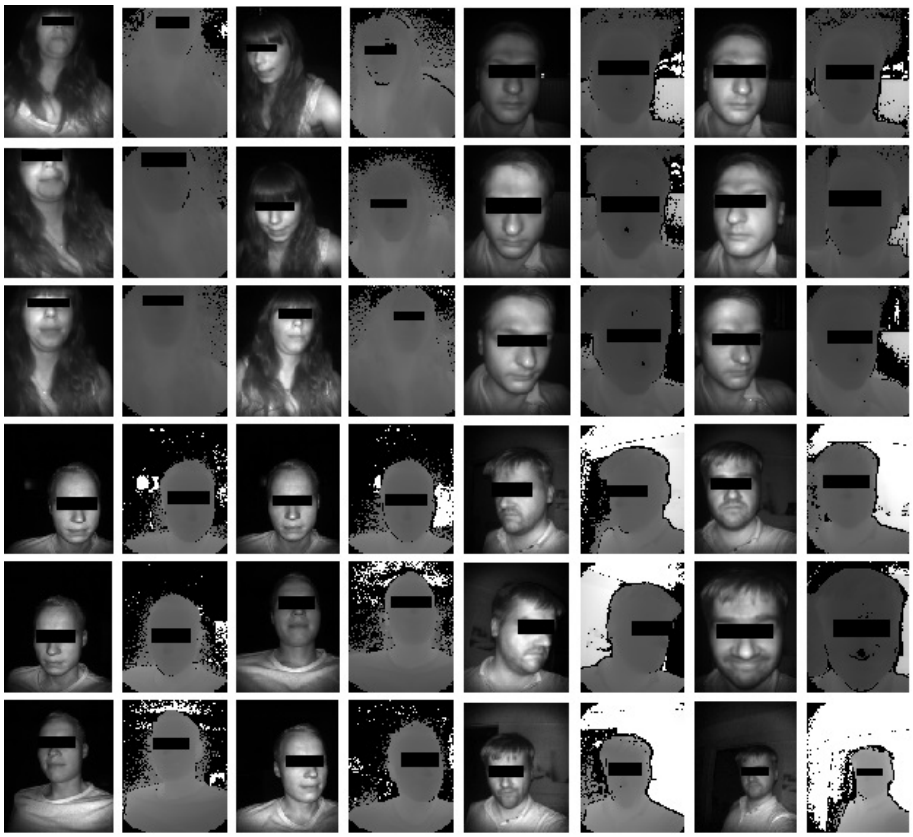


Figure 5.2: Examples of the validation set. Different distances, pose and emotions are recognisable. Odd columns represent the greyscale images, even columns the depth images. Note that bright pixels in the depth image indicate more distance.

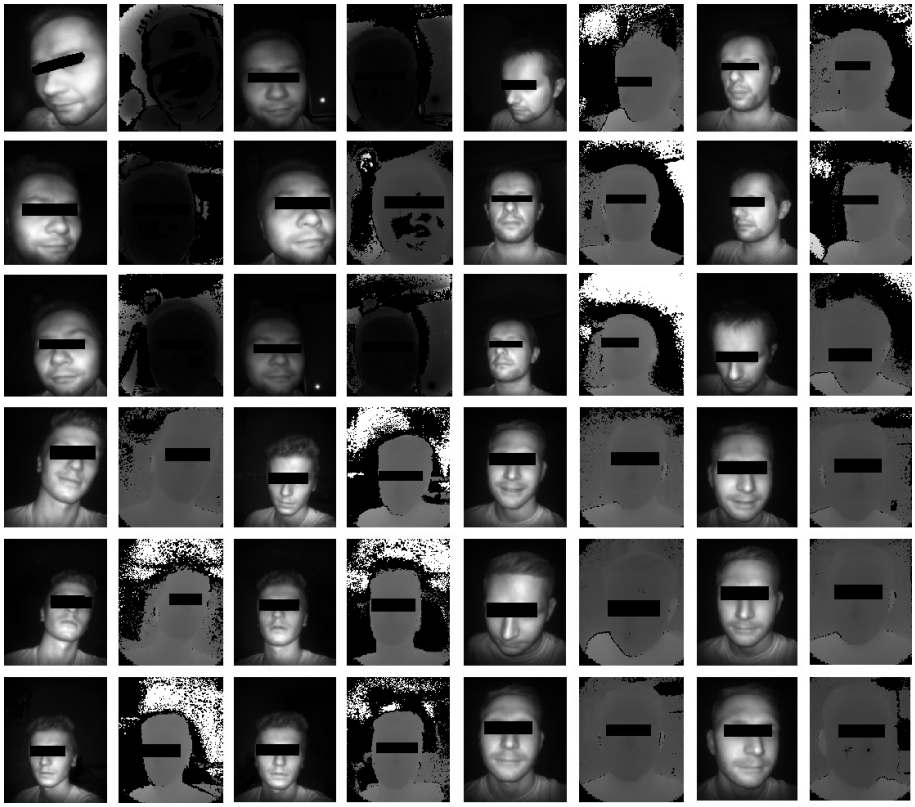


Figure 5.3: Samples of the test set. Different distances, pose and emotions are recognisable. Odd columns represent the greyscale images, even columns the depth images. Note that bright pixels in the depth image indicate more distance.

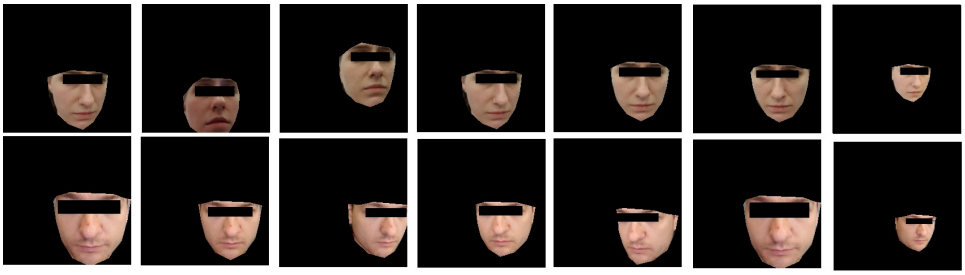


Figure 5.4: Visualisation of two individuals from the recorded RGB images. Different distances and poses are recognisable.

ground images without a face on the recording. Many of these images are rejected from the pipeline. Additionally, strong movements blurred the images and holes in the image occurred due to pixel oversaturation. The OpenCV face detection could not find faces from a strong side view.

5.2 Preprocessing Pipeline

In this section, we evaluate our created preprocessing pipeline. For this, we briefly summarise the steps of the preprocessing pipeline and additionally, we demonstrate some problems of the pipeline and last, and we illustrate correct processed outputs.

If the captured image passes all the sanity checks, then our pipeline starts with the preprocessing. This step speeds up the whole computation of the pipeline because we discard unreliable data in very early steps. One sanity check includes an inspection if the face or landmark detection went wrong. We use for the landmark detection an implementation from Github[2] which is based on [Ren+14] and a model which is trained with RGB data on images from the LFW Database [Hua+07]. We expect more errors on the low-quality ToF data because on the one hand it is not trained on this data and on the other hand the resolution is much lower than the dataset from LFW. In Figure 5.5 we illustrate some examples for failed face and landmark detections to give a better understanding. It is possible that OpenCV mistakenly recognises a face, *i.e.* the resulting bounding box won't include a face. Instead, a neck or one eye is the result. Figure 5.6 illustrates a failed landmark detection from frontal view. The green line indicates the correct position of the eyes and the yellow line the detected position. In the validation set we counted around 6% wrong detected faces and landmarks, *i.e.* if the face or landmarks are detected wrong as in Figure 5.5 or 5.6 illustrated. The recorded test set contained around 5% wrong detected results. Generally, most problems occurred with wrong detected landmarks. It is possible that the neural network rejects samples with wrong detected faces and landmarks. To face this issue, we augmented the training data with random rotation between ±6°. This value is determined empirically and is based on the validation set. If the landmark detection results in wrong captured landmarks, then this leads to a wrong rotated face as in Figure 5.6.

[2] `https://github.com/memoiry/face-alignment`

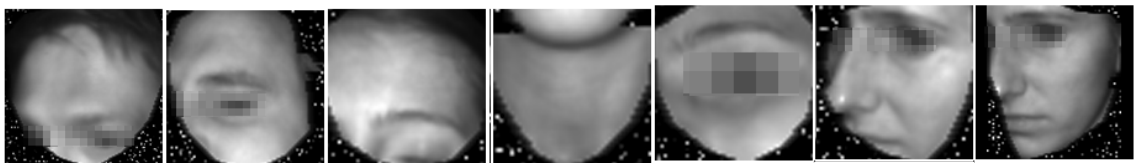


Figure 5.5: First five images illustrate failed face detections, and the last two images illustrate failed landmarks.

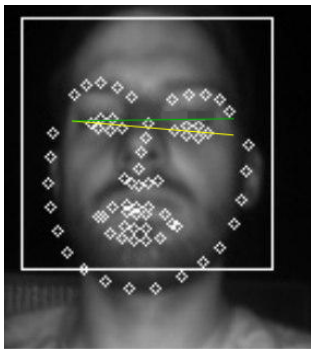


Figure 5.6: Wrong detected landmarks. The nose, mouth and one eye are incorrect detected. The green line illustrates the expected result, yellow line the real results

If the face and landmark detection provide a face and landmarks, then the preprocessing pipeline compute the input for the CNN. After a coarse fore and background removal, the pipeline rotates the face with the landmarks. In the next step, a dynamical mask with the noise data is created to segment the face with more precision. If the face has more than eight missing pixels in the inner face, then the pipeline discard these one dataset due to reliability reasons. Missing pixels can occur due to oversaturation in the pixel itself. If the illumination unit sends out too strong light beams, then this oversaturation occurs. After the created dynamical mask we use the erode operation to reach a more precise and smaller bit mask, *i.e.* we shrink the created dynamical mask slightly. If eight pixels are missing at the same position in the captured face (for example at the nose tip or around the eyes), then the erode operation resize this hole. In Figure 5.7 we illustrate some problems with the preprocessing pipeline. The first row visualises the amplitude data, second row the according depth data. In the first two images, a hole in the area of the eyes occurred, due to the dynamic mask we crop out the same hole in the depth data for example. The last image illustrates a hole in the area of the nose. Due to these problems, it is possible that the captured face getting rejected from the neural network. This effect happens if the auto exposure can't regulate the illumination unit fast enough to an exposure which is reliable. For example, a fast movement from 50 cm to 30 cm can lead to this problem. In Figure 5.8 we illustrate 21 images of one individual of the validation set. The faces are rotated and cropped correctly.

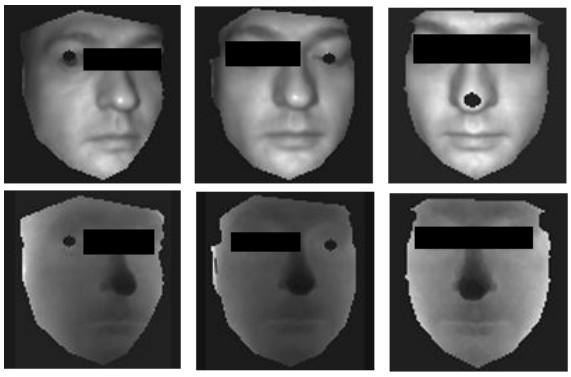


Figure 5.7: Illustration of oversaturated pixels which lead to a hole in some regions of the face. First two images illustrate an oversaturation in the area of the eyes and the last image at the nose tip. This oversaturation leads to some holes in the face. Less than eight oversaturated pixels can pass the sanity checks of the pipeline. If all oversaturated pixels are located at the same place, then the erode operation resizes the hole and lead to these images.

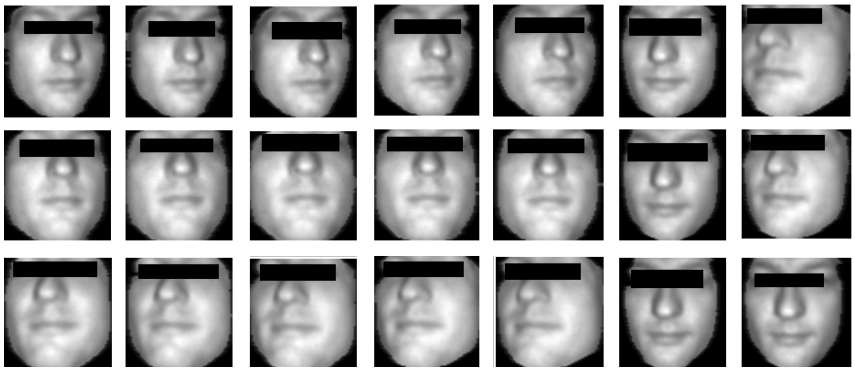


Figure 5.8: Correct processed images of our created preprocessing pipeline.

5.3 Comparison of FaceNet and NIRFaceNet

Important values to evaluate verification results are the Recall (R), False Discovery Rate (F), Precision (P), Accuracy (A), Negative Predictive Value (N) and the F_1 measure (F_1) [Mos17]. For this evaluation we need the True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) values. TP represents the images which should log in to the system, TN represents the correct rejected images, FP represents the individuals who logged in to the system, but they should not be able to log in. An FP prevention is the aim of a face recognition system. The FN value represents the rejected images which should log in to the system, but they are not able to log in. We briefly summarise some values to compare face recognition systems together [DG06; SR15; Mos17]. The Recall represents the successful login rate to the system. The Recall is defined as

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{5.1}$$

The False Discovery Rate for face recognition is typically 0.001. With this value the performance of different networks can be measured. The False Discovery Rate can be calculated as

$$F = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}. \tag{5.2}$$

The Precision reflects the real login candidates compared to all logins to the system. A high Precision is important for face recognition because no FP should enter the system. The Precision is calculated as

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{5.3}$$

The Accuracy reflects the correct classification rate of the network. The Accuracy is calcualted as

$$A = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FN} + \mathrm{TN} + \mathrm{FP}}. \tag{5.4}$$

The Negative Predictive Value represents the correct rejections in relation to all rejections (TN = correct rejection, FN = should log in but get rejected). The Negative Predictive Value can be calculated as

$$N = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}}. \tag{5.5}$$

F_1 is the harmonic mean of Recall and Precision and can be calculated as

$$F_1 = \frac{2 * P * R}{P + R}. \tag{5.6}$$

NIRFaceNet

NIRFaceNet [Pen+16] is built as a network for near-infrared face recognition. This network uses Softmax for the classification. The network scored on the Chinese Academy of Sciences Institute of Automation (CASIA) database identification rates

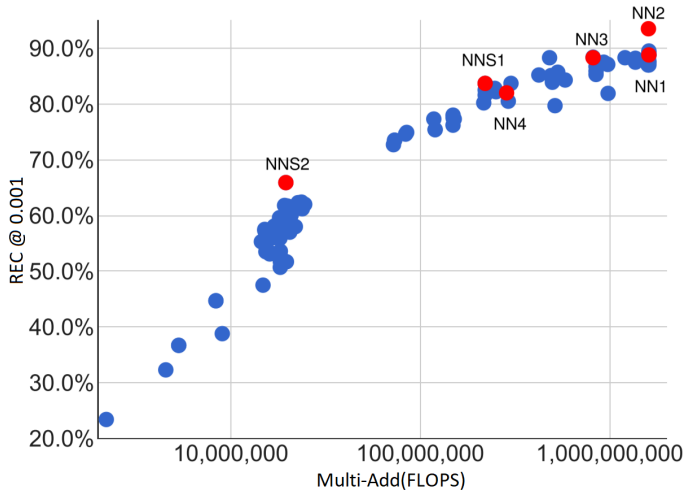


Figure 5.9: Recall values for the different FaceNet networks on LFW database. Version NNS2 is usable for the smartphone, the Recall is less than 70 per cent. Adapted from [SKP15].

around 95–98 per cent. This is three to five per cent higher than the trained GoogLeNet on the same data. The authors summarise that the shallow structure network outperformed the deeper networks. The network needed 30 hours of training with a DELL PRECISION T3600. The used CPU is a Xeon E5-1620, 64 gigabyte RAM and a Nvidia Quadro 600. Recall Section 2.3.1 for more details.

FaceNet

FaceNet [SKP15] is designed for RGB data. The inception based version is 24 layers deep. The network is trained on a private database with around 260 million images. With 2.6 million images and 700 hours of training on a server cluster, the network scored a Recall of 76.3% on the test set. With 26 million images in the training set, the network reaches a Recall of 85.1%. An amount of 260 million images increases the Recall by just 1.1%. A wrong labelled image can lead to problems with smaller False Discovery Rates than 0.0001. Overall, the training of the network took around 2000 hours [SKP15].

In Figure 5.9 we illustrate the performance of FaceNet on the database of LFW. NN2 is the deepest network with the most FLOPS and NNS2 the version for smartphones with fewer FLOPS per image. NN2 score with more than 90% and the smartphone version scored less than 70%.

5.4 Experiments with Neural Networks

In this section, we illustrate methods to visualise and monitor the training of a neural network with TensorFlow [Aba+15]. Additionally, we evaluate the impact of more training data, the networks from Chapter 4 and the best hyperparameters for these networks. To monitor the training of the network with TensorFlow, we sample every 20th image of the training set. We didn't use these images for training itself. With these images and

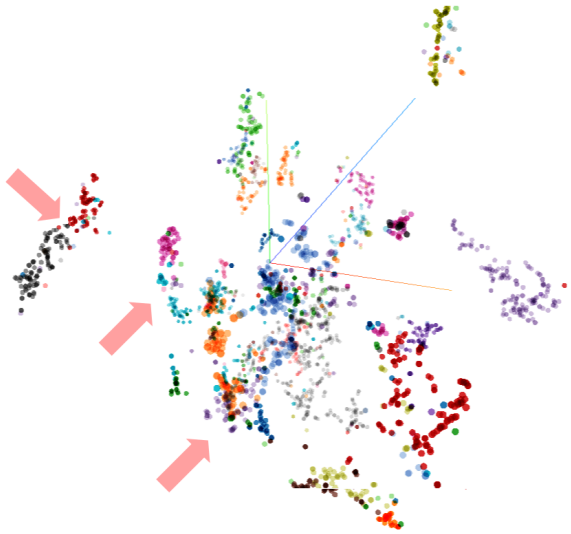


Figure 5.10: 3D T-SNE visualisation of the individuals in the training set. One individual is represented by one colour. T-SNE projects high dimensional space onto low dimensional subspace for visual inspection. Separation problems between two individuals are marked with red arrows.

the validation set, we can prevent overfitting due to early stopping. Additionally, to determine values as the Recall or Precision, we compare all images against the reference images. This should reflect the real login situation with one reference image and many login images. We used for all our experiments the well known ADAM [KB15] optimiser and the XAVIER [GB10] initialiser. We experimented with SGD [RM51] like FaceNet but the convergence was slower, and hyperparameter tuning with additional parameters is necessary [Li18].

5.4.1 Monitor the Training of Neural Networks

Monitoring the training of a network is essential. Wrong hyperparameters like the alpha value for the Triplet Loss, the embedding dimension with the resulting face features or the learning rate can influence the results of the trained network. T-SNE [Rog+17] can visualise the high dimensional space of the embeddings in 2D or 3D space. With the help of this visualisation, we can monitor the learning progress because different individuals should be separated from each other. In Figure 5.10 we visualise problems in an early stage of the training (marked with red arrows). Points with the same colour belong to the same individual, and one point reflects one tested image in our training set. If two colours are close together, then the network has problems with the separation of the individuals. With more iterations over the training set the network learns a more efficient separation as Figure 5.11 illustrates. Due to 3D visualisation, it is possible to rotate the separation results and evaluate them coarsely. In our experiments, an intersection-free visualisation is not possible with the 2D or 3D T-SNE.

The learning rate is another essential parameter for the training of a neural network. The optimiser manipulates the weights of a network. The learning rate influences this

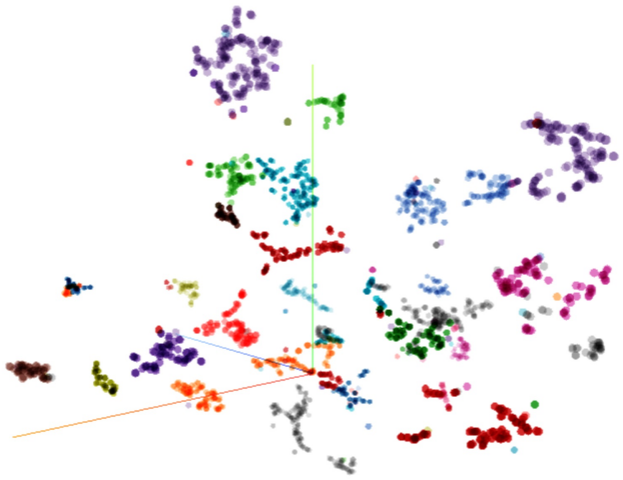


Figure 5.11: 3D T-SNE visualisation of our individuals in training set after around 200,000 iterations. The separation of the individuals is visible. Due to 3D visualisation, it was not possible to illustrate these separations without intersections.

optimisation. We set the learning rate to values where the loss converges to zero (recall Section 2.2.2.3 or [Kar18]). We configured our red channel with amplitude data, green with depth and blue channel with noise data. In Figure 5.12 we illustrate the effect of a wrong learning rate. Learning rates higher than 0.0003 in our experiments led to unstable networks. The first image illustrates the learned kernels of the network. Recall Section 2.2.2.3 or [Kar18] for better understanding how a well-trained network should look like. The kernel matrix (5x5) has nearly homogeneous coloured kernels. Next three images visualise the output of the layers in the network, where each layer consists of convolution, leaky ReLu and max pooling. Generally, a neural network should reduce the important information to a minimum and remove the unwanted information with each layer. In this case, the network is not able to reduce the information. The last layer should separate the essential information, *i.e.* brighter values indicate stronger activations in the network. Typically small areas should be firing. In our example, many neurons are firing. The loss exploded for this network to values up to 6.000.000. In Figure 5.13 we illustrate how a trained network should look like. The first image illustrates the input image to the network with three channels. The second image visualises the five by five matrix with the learned kernels. Each kernel has 16 x 16 pixels with three channels. Red represents the amplitude data, green the depth and blue the noise data. Structured kernels are visible. With the visualisation of the kernels, it is possible to determine the used and combined channels from the network in each kernel. The third image is the result of the input image with the manipulation of the learned kernels. It represents the output after the first layer which contains convolution with the learned kernels, the activation function (leaky ReLu) and max pooling. Next three images till the next input image represent the output after the next layers. In the last layer, some activations occurred. The network tries to focus on the nose, the chin and eyes for example. For better understanding, we visualise a second input image with the output of the layers. To highlight the focused face features,

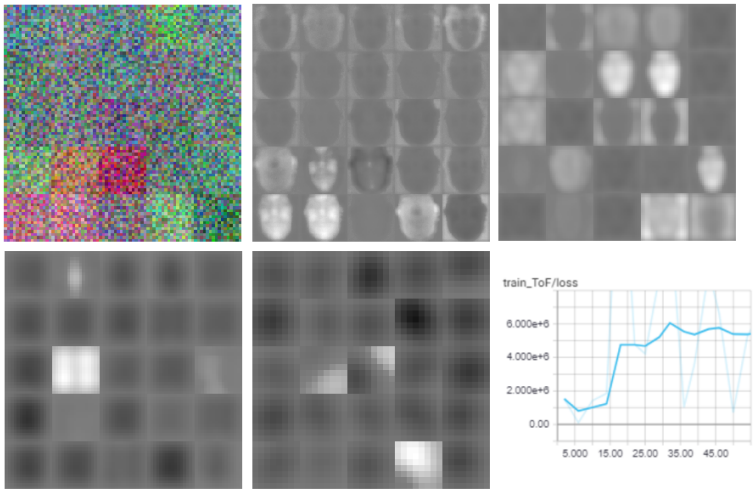


Figure 5.12: Exploded loss function because the learning rate is too high. Kernels have the same structure. No edge detectors are visible. Whole areas of activation functions are firing in the last layer. Typically small areas are firing.

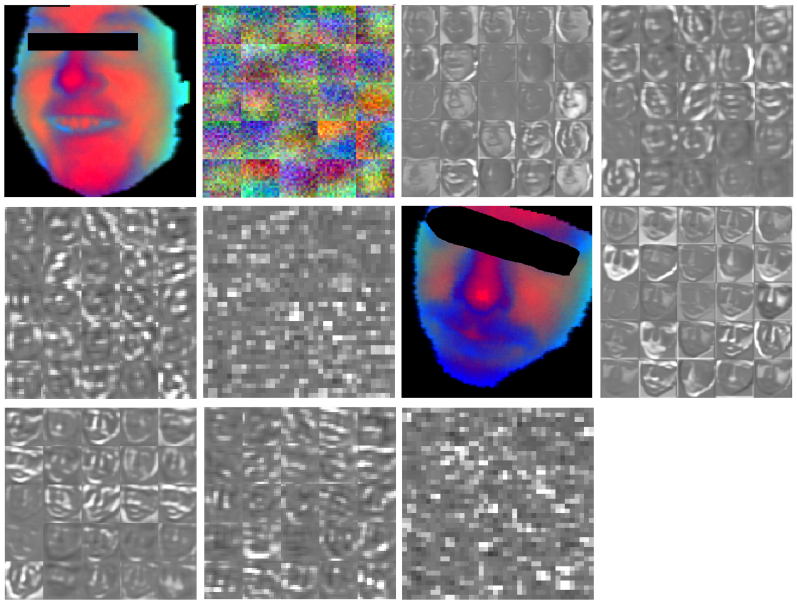


Figure 5.13: The first image represents the input image. The second image illustrates the learned kernels with three channels (red = amplitude, green = depth and blue = noise data). Placing the kernels on the reference image with convolution technique followed by max pooling and leaky ReLu leads to the third image. Next three images illustrate the deeper layers of the network with convolution. Note that the kernels are structured.

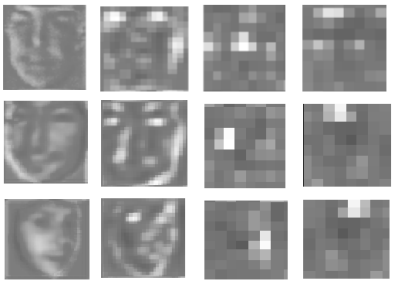


Figure 5.14: Illustration of the data in the neural network. The first column illustrates the images after the first layer of the network with three different individuals. Next column represents the second layer where the network detected the eyes. Next column indicates a detected nose in the last layer of the network. The last column visualises a forehead detector in the last layer.

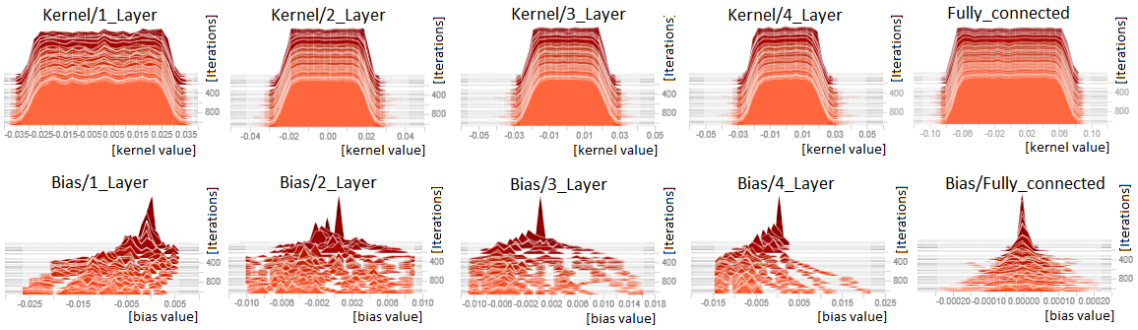


Figure 5.15: At the beginning of the training, a uniformly initialised kernel with the help of Xavier initialiser is visible. The kernels change during training, and this indicates that learning happened in the network. The bias in the second row should change to values nearly to the width of the kernels.

we visualise in Figure 5.14 the results in the layers of a neural network. The network concentrated on the eyes, nose and forehead of the individuals. The activations illustrate this focused points of interest. We illustrate the output image of the first convolutional layer of three different individuals in the first column. Next column represents the eyes detector after the second layer. The third row illustrates a nose detector after the last layer. The last row visualises a forehead detector.

The change in the kernels itself is important for neural network training. In Figure 5.15 we illustrate this with histograms and TensorFlow [Aba+15]. The kernels are uniformly initialised. With rising iterations and training, the histograms change to a normal distribution with high values in the centre. The second row illustrates the changing in the bias weights. This bias is only an offset which is added to manipulate the data. In the last stages of the training phase, the bias should be in the same width as the convolution kernels from the first row. No change in the histograms is the result if no learning takes place. After 105.000 iterations the histograms changed their shape as Figure 5.16 illustrates. The original shape is illustrated in Figure 5.15, after 105.000 iterations it changed the shape to a normal distribution. This is an indicator that learning happened.

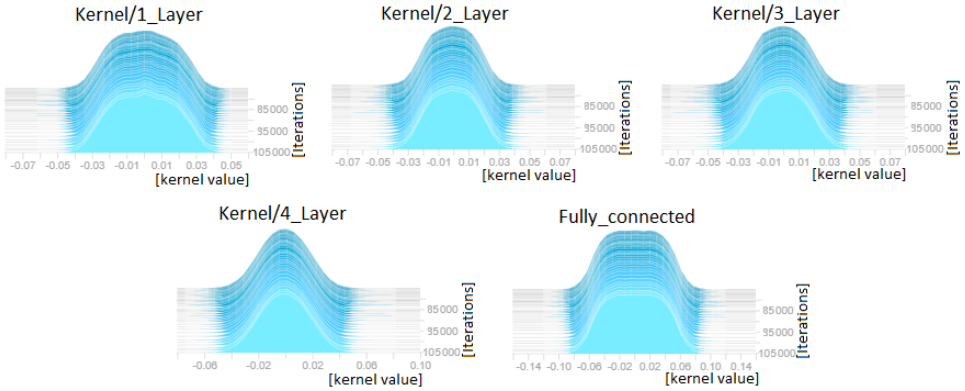


Figure 5.16: Histograms of kernels after 105.000 iterations. At iteration zero, the histograms are distributed uniformly. With rising iterations, the kernels changed to a normal distribution. These changes indicate that the network changes the kernels. Note that a restart of the training changes the colour of the visualisation in TensorFlow.

TensorFlow visualises relevant information which indicates the ongoing training of the neural network. For this experiment, we set the learning rate to 0.0001 and use the CNNV1. In Figure 5.17 the orange curve visualises the start of the training, a restart of the training is represented by the red curve. We restart the network from the last training step with lowered learning rate if the loss can't converge to zero. The loss function in the first image falls as expected (orange curve). The mean negative value indicates the mean over the negative distances to the reference image. With rising iterations, the network learns a better separation and the negative values rise. The mean positive value is built over the positive distances and the reference image. During training, this positive mean distance should be lower than the negative mean value. The first illustration in the second row represents the triplets with resulting loss per iteration. Last two images visualise the false positive rate coupled with the success rate of the system. We used in early steps a False Discovery Rate of 0.1 to evaluate a threshold for the false positive and success rate calculation. If a network looks promising, then we set the False Discovery Rate to 0.001 as in literature is used to compare them.

5.4.2 Data Dependency

For this experiment, we evaluate the impact of the number of training images in the database. We started the first training run with 28 different individuals. We use the amplitude, depth and noise data and use the CNNV1. Generally, the used network and channels are not that important for this experiment because we expect the same behaviour for different network architectures. In Figure 5.18 we summarise how a different amount of individuals in the training set influence the training progress and the results on the validation set. The orange curve illustrates the training set of 28 individuals and the blue curve illustrates the training set with 33 individuals. We can summarise that the training set with more individuals (blue curve) leads to a higher Precision of around

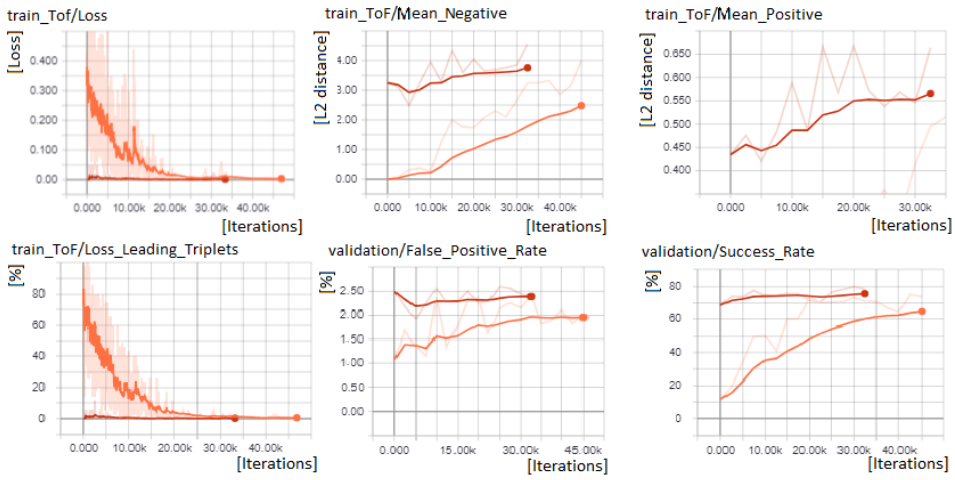


Figure 5.17: The orange curve illustrates the start of our network from zero, after 47.500 iterations we restarted the training from the last training step to get new triplets and change the learning rate. We used the CNNV1 for this demonstration. Mean negative and positive value differs strong, this means the network learned facial features to separate different individuals. The loss-leading triplets counter indicates the triplets which cause a loss. The false positive rate is based on the validation set and a False Discovery Rate of 0.1. The last image illustrates the success rate based on the calculated threshold.

1.5%. The false positive rate is 0.87% lower with a training set with five individuals more. In the original FaceNet paper [SKP15], the authors summarise that more training images lead to significantly better results. We can conclude the same result that more individuals in the training set lead to better results.

5.4.3 Impact of Initialisation

In this section, we evaluate how three starts with entirely the same setup influence the result. Our used network is the CNNV1 with three channels (amplitude, depth, noise). We trained the network for 30,000 iterations with a learning rate of 0.0001. For all runs, the loss flow is nearly identical. We can summarise that the initialisation is important for the results of the network based on the Recall. In Table 5.1 we summarise values to compare the three runs reliably. For this, we use a False Discovery Rate of 0.001 for comparison. The Precision, which includes the FP, is one of the most important values. The Precision is on par, and the Accuracy and Negative Predictive value differ slightly. To differentiate the values with more precision, it is necessary to complete even more training runs. Additionally, we hypothesise that the networks need more training iterations to reach stable values and better convergence. Due to time reasons, it is not possible to evaluate all networks with more iterations. The triplet selection can also influence the results [WB18]. We believe that the calculated threshold from the False Discovery Rate can also influence the Recall stronger due to the separation of the TP and FN with less training runs. In our experiments the triplet selection is randomly chosen, this can lead

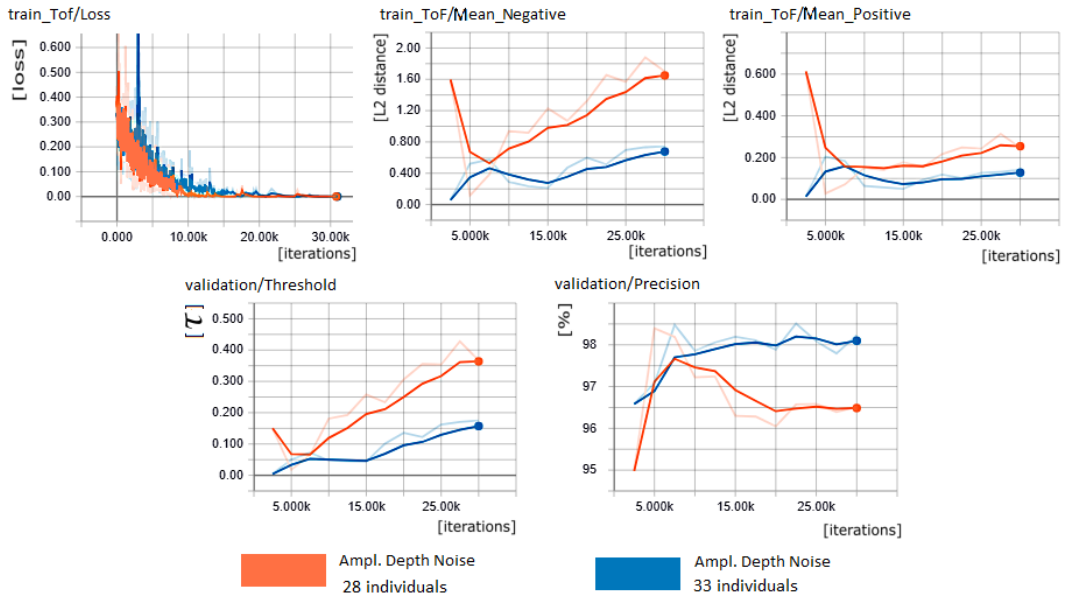


Figure 5.18: Illustration how more individuals influence the training of the network. The visualisations are based on the training and validation set. The loss converges with nearly the same behaviour. The mean negative and positive value is much higher with fewer individuals. Note that five more individuals lead to a 1.5% higher Precision and a 1% lower false positive rate.

to zero loss and convergence, but for the comparison in our tests against one reference image to all login images, it can influence the result stronger. We must remark, that this experiment is done after the evaluations of Section 5.5.1.2 due to the unstable results.

5.5 Impact of Different Architectures

In this section, we evaluate different architectures. First, we evaluate the standard architecture approaches. Second, we evaluate the inception based networks. Additionally, we analyse the impact of standard batch filling to Triplet Loss, the best hyperparameters of the network architectures and compare the final results of the standard and inception based networks against a pre-trained AlexNet on RGB data.

5.5.1 Evaluation of Standard CNN's

In this section, we evaluate different standard architectures with the best hyperparameters. Additionally, we figured out that the standard batch filling is not suitable for Triplet Loss. For this, we create an improved batch filling technique to solve this problem. We evaluate the standard batch filling version without data augmentation at the beginning. Next, we evaluate the improved batch filling and use data augmentation.

Table 5.1: Comparison of three identical runs. Our used network is the CNNV1 with three channels (amplitude, depth, noise) and a False Discovery Rate of 0.001. The Precision, which is one of the most important values of the network, scored on par. The result based on the Recall differs up to 18 per cent. Additionally, the Accuracy and Negative Predictive Value are compare able. We trained the network for 30,000 iterations. We discovered unstable behaviour with these few training iterations.

Run Nr.	τ	P	R	F_1	A	N
1	0.61769	99.19%	70.79%	82.63%	95.03%	94.48%
2	0.64849	99.24%	74.78%	85.30%	95.70%	95.20%
3	0.51859	99.06%	56.79%	72.19%	92.71%	92.04%

5.5.1.1 Standard Batchfilling

Based on the CNNV1, we demonstrate how the network performs with standard batch filling. Illustration 5.19 is segmented in two parts: On top, we illustrate the loss function along 11,000 iterations. One iteration consists of a batch size of 50 triplets. For the Triplet Loss, we set an alpha value as a margin between positive and negative distances of 0.4. This value is determined empirically. The second row illustrates the mean negative and positive value. Both values are rising, and the negative value is higher than the positive value, *i.e.* the network separates the individuals with a margin between positive and negative samples. The success rate falls on the training and validation set after 11,000 iterations. The batch filling with zero loss triplets cause this problem, and a falling success rate after thousands of iterations is the result.

In Figure 5.20 we restarted the network training for 20,000 iterations from the last training point. The first image visualises the loss function. Less loss occurs, it looks like the network learned well. The loss is most of the time zero because the batch is filled with random triplets which won't lead to a loss. The loss-leading triplets illustration reflects the triplets in one batch which cause a loss. We figured out that only around one per cent leads to a loss, this results in the flow of the loss function. The last image illustrates the falling success rate of the network because of the ADAM optimiser and the zero loss triplets. With rising iterations, the success rate falls, and it looks like overfitting. In the end, standard batch filling and the optimiser caused the problem. It is necessary to generate triplets as in FaceNet. In FaceNet, the triplet generation is during runtime (online), *i.e.* searching for triplets which violate Equation (2.9) during training. In the next section, we illustrate the improved batch filling technique. In Figure 5.21 we test all channels with standard batch filling. Overall the performance is not suitable for verification. The success rate is low compared to the false positive rate. The LBP scored with the lowest false positive rate but also the lowest success rate. The combination of all channels (amplitude, depth, noise and LBP) scored with the highest success rate. We hypothesise that more channels lead to more loss at the beginning with standard batch filling. The network can learn better with more channels in a certain time.

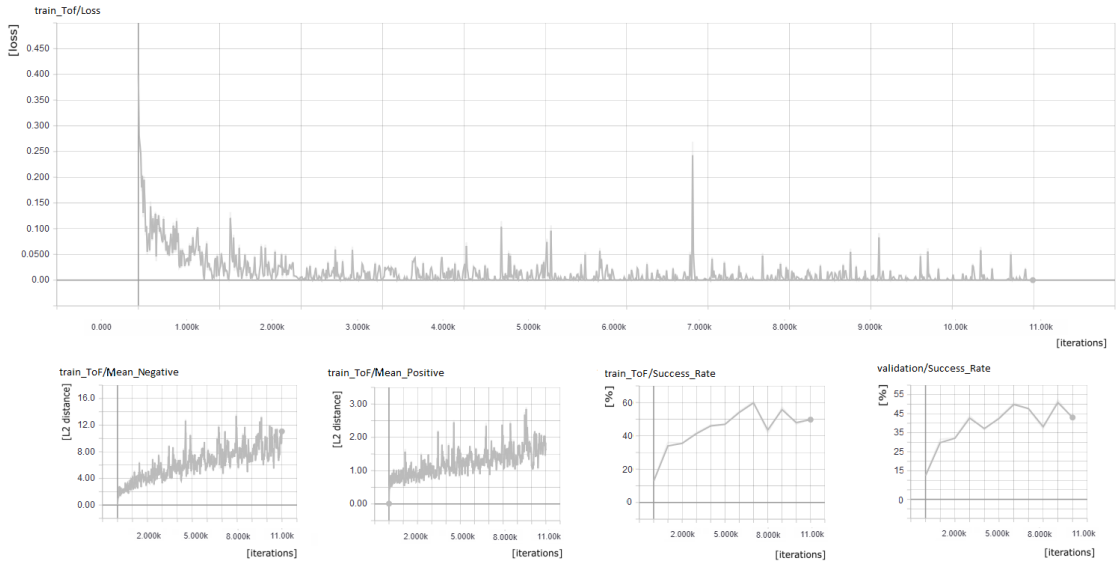


Figure 5.19: CNNV1 with falling loss function after 11,000 iterations on top, bottom line f.l.t.r: The first image illustrates the rising negative mean distance between negative and reference image. The second illustration visualises the rising positive mean value between positive and the reference image. Note that negative distances are higher than positives. The third illustration reflects the success rate on the train set (we sample one image after 20). We are not using these images for training. The last illustration visualise the success rate on the validation set.

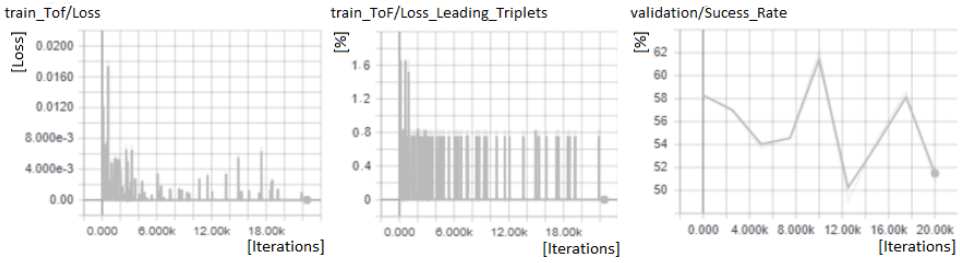


Figure 5.20: Restart of training with CNNV1 and three channels from last training checkpoint. The loss function is close to zero. The loss-leading triplets illustration visualise the inefficient training, *i.e.* the batch is filled with around one per cent loss leading triplets. The success rate begins to fall, and this is a problem with the filled batches and the optimiser.

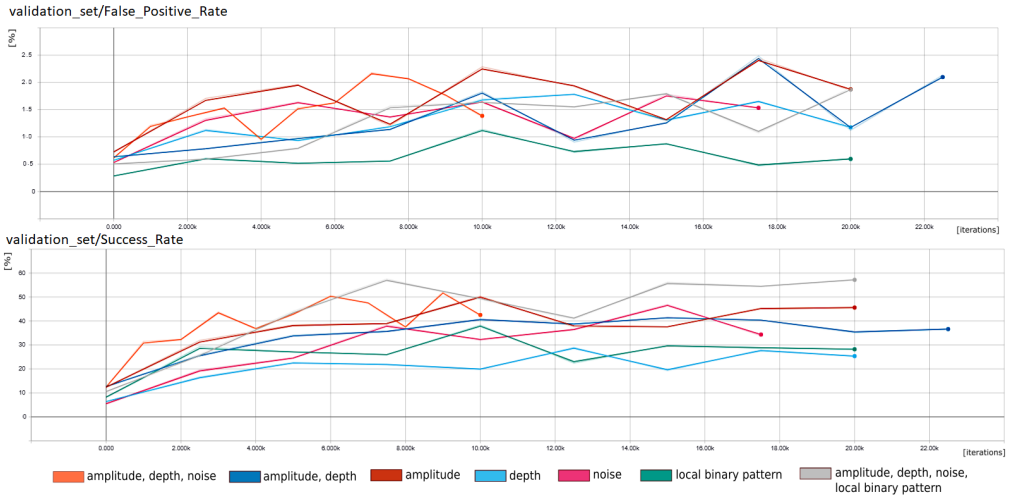


Figure 5.21: Comparison of different channels and combinations with standard batch filling.

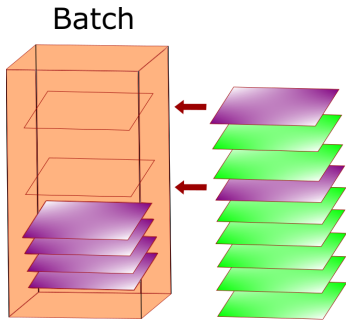


Figure 5.22: Improved batch filling. Only triplets which cause a loss can pass to the optimiser over the full batch. Green indicates a zero loss triplet and violet a triplet which leads to a loss.

5.5.1.2 Improved Batch filling, Data Augmentation and Different Borders

To overcome the problem with the falling success rate, we created an improved batch filling technique. In Figure 5.22 we illustrate the improved batch filling. We stack up non zero loss triplets and then put this to the optimiser. On the left side, we illustrate the batch. The violet rectangles represent the triplets which cause a loss. Green rectangles won't lead to a loss, and they are ignored. In the end, the success rate won't fall with rising iterations because the optimiser gets non zero loss triplets within the whole batch. FaceNet uses an online triplet generator (during runtime). Due to the restrictions to a standard system, we hold on until a batch is full of data. We hypothesise a generation of triplets leads to significant higher CPU load. FaceNet used 1000 CPU cluster with 2000 hours of CPU time for training.

We use data augmentation to build up a bigger training set, *i.e.* we generate more data from the existing training set with image rotation, noise and brightness changing. For further experiments, we use data augmentation.

Table 5.2: We used the network CNNV1 with a learning rate of 0.0003 and trained the network for 30,000 iterations with a training set of 33 individuals. We set the False Discovery Rate to a low value, and we compare the results with different backgrounds. The network training without background images scored the highest Recall. Note that rounding leads to the same Precision for the noise border and no background version.

Background	τ	P	R	F_1	A	F	N
Average border	0.31573	100.00%	32.03%	48.52%	88.67%	0.00	88.03%
Noise border	0.18537	99.91%	33.63%	50.32%	88.93%	6.26e-5	88.28%
No background	0.32011	99.91%	34.17%	50.93%	89.02%	6.22e-5	88.36%
No segmentation	0.58925	99.88%	21.80%	35.93%	86.48%	5.27e-5	85.95%

Additionally, we evaluate different border around the face. After cropping we set the background to zero (black), and a very sharp edge around the face occurs. During training we observed the activation functions and bright areas of activations occur which belong to the background and edges around the face. We hypothesise that a neural network concentrates in early training steps strong on these high-frequency parts. After more extended training, the network concentrates on features in the face like nose and eyes. With adding an average border or noise to the background, the network starts in earlier steps to learn the facial features. To overcome this problem, we evaluated this behaviour with data augmentation.

In Table 5.2 we summarise the results based on the validation set of six individuals. The used network architecture is the CNNV1. We trained the network for 30,000 iterations with data augmentation, a batch size of 50 and an alpha value of 0.5. We prepared all data with the preprocessing pipeline. For the version without background segmentation, we removed the dynamical bitmask creation step in our preprocessing pipeline to obtain the background. Additionally, we figured out that the depth data are less well scaled due to the strong outliers around the face in many measurements. The training without background separation or preparation leads to lowered results. We hypothesise that the network focused on the background which is in all measurements of one individual the same. Additionally, we believe that with more training data, temporal variations between measurements and more iterations the network focuses on the facial features also without background removing. We observed nearly no improvements between the prepared borders. We change the rotation, illumination, contrast and add Gaussian noise to the whole image. Recall Table 4.10 for the used values for the data augmentation. We choose as the best variant the version with no border (black background) because it scored the best results. For further work, we use the black background (no border), data augmentation and the improved batch filling.

5.5.1.3 Performance of Standard CNN Architectures

In this section, we evaluate the created CNN architectures from Chapter 4. We set the False Discovery Rate to a low value. We use an alpha value of 0.5, an embedding di-

Table 5.3: Comparison of the created network architectures. Precision and Accuracy are nearly equal, CNNV1, CNNV2 and CNNV5 performed with the highest Recall and F_1. We used a learning rate of 0.0003 and four channels (amplitude, depth, noise, LBP). The False Discovery Rate is set to low values.

Network	τ	P	R	F_1	A	F	N
CNNV1	0.30513	99.93%	46.00%	63.00%	91.00%	6.35 e-05	90.25%
CNNV2	0.20476	100.00%	44.10%	61.20%	90.68%	0.0	89.94%
CNNV3	0.28969	99.78%	14.41%	25.19%	85.73%	6.35 e-05	85.38%
CNNV4	0.30179	99.88%	25.43%	40.54%	87.57%	6.35 e-05	87.02%
CNNV5	0.66320	99.92%	43.14%	60.27%	90.52%	6.35 e-05	89.79%
CNNV6	0.15118	99.89%	28.35%	44.46%	88.05%	6.35 e-05	87.47%

mension of 128, a batch size of 50 and train the networks for 50,000 iterations with four channels (amplitude, depth, noise and LBP). In Table 5.3 we summarise the results of six different CNN architectures based on the validation set. CNNV1, CNNV2 and CNNV5 perform on par. All variants scored with a low False Discovery Rate and high Precision. We decided to use CNNV1 for further experiments due to the best results.

5.5.1.4 Channel Dependency

In Table 5.4 we compare all channels and combinations. We sort the results based on the Precision and not alphabetically. The highest values in all comparisons score the combination of amplitude, depth and LBP. We believe that this combination scored the highest values due to the high single channel performance of amplitude and LBP. We combine the depth channel with the other channels to get higher security at login. A combination of amplitude and LBP is not a good combination due to the missing depth information. A fooling with a printed image can be the consequence. The single amplitude channel score with the second best performance. The noise and depth data and the combination scores the last places. These channels negatively influence the results based on the Precision and Recall (which reflects the successful login rate to the system). We hypothesise that the network focuses on different channels with different intensity during training. Additionally, the initialisation of the kernels can strongly influence the results. For this, the combination of amplitude, depth and noise data scored with less performance than the amplitude and depth combination.

5.5.1.5 Hyperparameter Tuning

For successful training, it is necessary to evaluate the best hyperparameters. These are parameters which are not changeable by the network itself. We train all our networks for the experiments for 30,000 iterations with a learning rate of 0.0001. To compare the networks, we set the False Discovery Rate to 0.001. This False Discovery Rate is the commonly used value to compare different networks. We choose the three channel variant with amplitude, depth and noise for the evaluation. We found the following reasons for

Table 5.4: Performance comparison of different channels with the CNNV1. The amplitude single channel score with the highest Precision and Recall, followed by the LBP, noise and depth. We used for this experiment a learning rate of 0.0003.

Channel	P	R	F_1	A	N
Amplitude & Depth & LBP	99.13%	64.95%	78.48%	94.06%	93.44%
Amplitude	99.11%	64.29%	77.99%	93.52%	93.33%
Amplitude & Depth	98.98%	55.33%	70.98%	92.46%	91.79%
Amplitude & Depth & Noise & LBP	98.97%	55.04%	70.75%	92.41%	91.74%
Amplitude & Depth & Noise	98.88%	50.25%	66.64%	91.61%	90.94%
LBP	98.82%	45.27%	62.09%	90.79%	90.12%
Noise	98.02%	28.32%	43.94%	87.96%	87.45%
Noise & Depth	97.41%	21.49%	35.21%	86.82%	86.42%
Depth	96.24%	14.63%	25.41%	85.86%	85.40%

Table 5.5: We use for this experiment the CNNV1 with three channels (amplitude, depth and noise). We tune the hyperparameter alpha to get the best results for the Recall and Precision. We use a batch size of 50, a learning rate of 0.0001 and a False Discovery Rate of 0.001. An alpha value of 0.4 scored with best results.

Alpha	τ	P	R	F_1	A	N
0.3	0.254610	98.80%	46.95%	63.65%	91.06%	90.40%
0.4	0.648494	99.24%	74.79%	85.30%	95.70%	95.20%
0.5	0.901617	99.20%	71.24%	82.93%	95.11%	94.55%
0.6	0.746631	98.96%	54.44%	70.24%	92.31%	91.64%

that: to monitor the kernels with TensorFlow, we need one, three or four channels. The combination of four channels needed more computation time and memory on the system. Due to the influence of the kernel initialisation, it is possible that with more runs the combination of amplitude, depth and noise score with a higher Recall than the amplitude and depth combination. Generally, the used channels are not that important for this experiment because we expect the same behaviour for all channels.

In Table 5.5 we compare different alpha values with our CNNV1 and three channels (amplitude, depth and noise). The alpha value sets a margin between the positive and negative distance. We summarise that an alpha value of 0.4 lead to the best results. With the next experiment, we evaluate the best batch size for the network. We use the same set up as before (CNNV1, three channel and an alpha value with 0.4). In Table 5.6 we summarise the results for a batch size of 35 and 50. We figured out that a higher batch size leads to a higher Recall with nearly identical Precision and Accuracy. The next hyperpa-

Table 5.6: Comparison of different batch sizes with CNNV1 and three channels (amplitude, depth and noise). A batch size of 50 scored with best results.

Batch size	τ	P	R	F_1	A	N
35	0.420540	99.12%	60.98%	75.51%	93.40%	92.75%
50	0.648494	99.24%	74.79%	85.30%	95.70%	95.20%

Table 5.7: Evaluation of different embedding sizes with CNNV1 and three channels. The hyperparameter alpha is set to 0.4 and the batch size to 50. An embedding dimension of 128 performs better than an embedding dimension with 64 or 256.

Embedding size	τ	P	R	F_1	A	N
64	0.447732	98.84%	48.60%	65.16%	91.34%	90.67%
128	0.648494	99.24%	74.79%	85.30%	95.70%	95.20%
256	0.560403	99.00%	56.48%	71.92%	92.66%	91.98%

rameter for the further experiments is a batch size with 50. With the next experiment, we evaluate different embedding dimensions. We use an embedding dimension of 64, 128 and 256. In Table 5.7 we summarise the results of the different values. We conclude the same result as in FaceNet, an embedding dimension of 128 scored with the best results.

After these experiments, we figured out that an alpha value with 0.4, a batch size of 50 and an embedding dimension of 128 leads to the best results. For comparison, FaceNet uses an alpha of 0.2 and batch sizes with 1500 triplets and an embedding dimension of 128. We can't use these batch sizes due to the limitation to a standard system.

5.5.2 Evaluation of Inception based CNN's

Due to the (near-infrared based) amplitude and depth data of the sensor we searched for networks which can handle this combination of data. We figured out from the section before that the amplitude data scored with best results. However, it is necessary to combine the amplitude data with the depth data due to security reasons. In the end, this combination lowers the results. The derivation of FaceNet for smartphones is similar to the modified NIRFaceNet, except the networks are trained on RGB data, and instead of max pooling, they use average pooling. We removed the Softmax layer of the original NIRFaceNet and implemented the Triplet Loss from FaceNet. Both network architectures use the inception layer technique from GoogLeNet [Sze+15]. We figured out from the experiments with the standard CNN architectures that the training is unstable due to the kernel initialisation and not enough training runs. Additionally, we figured out, that the inception based networks needed more training iterations to reach zero loss. For this, we train all our inception based networks for 100,000 iterations and a learning rate of 0.0001.

Table 5.8: Evaluation of one channel (amplitude) and two-channel (amplitude and depth) with NIRFaceNetV1. The single channel performance is better based on Recall and F_1. We trained both variants for 100,000 iterations with an alpha of 0.6, embedding dimension of 128 and a batch size of 35.

Channel	τ	P	R	F_1	A	N
Amptlitude	0.26884	98.96%	54.57%	70.35%	92.33%	91.66%
Amptlitude & Depth	0.21983	98.72%	40.52%	57.46%	90.00%	89.35%

First, we evaluate the training of the modified NIRFaceNetV1 with one and two channels. Next, we tune the hyperparameters alpha, dropout, embedding and batch size. Last, we evaluate the best network architecture out of the three created architectures.

5.5.2.1 Performance on Different Channels

After the comparison of the channel combinations with the CNNV1 from the section before, we experiment with one and two-channel combinations with the modified NIRFaceNet. In this section, we evaluate the impact of one channel (amplitude) and two channels (amplitude and depth) on the NIRFaceNetV1. We train the network for 100,000 iterations. We use a learning rate of 0.0001 and a False Discovery Rate of 0.001. In Table 5.8 we summarise the results based on the validation set. The Precision is nearly equal, and the Recall differs around 14%. In the end, the negative influence of the depth data is nearly equal as in the experiment with the CNNV1 (Recall drop of 11%). For security reasons, we use for further experiments the two-channel variant for training. Security is more important for the project than a high login rate to the system.

5.5.2.2 Hyperparameter Tuning

In this section, we evaluate different hyperparameters and their impact on the NIRFaceNetV1 based approach. We evaluate the impact of different batch sizes and alpha. Additionally, we evaluate the impact of dropout because NIRFaceNet and GoogLeNet use high dropout rates (40 to 70%). FaceNet, in contrast, uses no dropout. After hyperparameter search we evaluate three different architectures of the modified NIRFaceNet, *i.e.* with two different fully connected layers and different amount of kernels.

In Table 5.9 we evaluate the impact of different batch sizes on the modified NIRFaceNet. We evaluate these batch sizes with the NIRFaceNetV1 and set the iterations to 100,000. The authors of NIRFaceNet use a batch size of 35. We experimented with a batch size of 35 and 50. Smaller batch sizes are better for training on standard systems. We figured out that a batch size with 35 leads to the best results. The Precision is nearly identical for both versions. The Recall, which is an indicator for the successful login rate to the system, differs between 9–14%. Due to this experiment, we set the batch size to 35 for further tests and the channels to two (amplitude and depth). After the comparison of the batch size, we evaluate the impact of the hyperparameter alpha. This value

Table 5.9: Evaluation of different batch sizes with NIRFaceNetV1. The alpha value is set to 0.6 and the embedding dimension to 128. We evaluate the impact of the batch size on one and two channels. A smaller batch size performed better based on Recall and Precision. Additionally, the single channel performance of amplitude data is higher than the combination with the depth channel. We trained the network for 100,000 iterations with a learning rate of 0.0001.

Channel	**Batch size**	τ	P	R	F_1	A	N
Amplitude	35	0.26884	98.96%	54.57%	70.35%	92.33%	91.66%
Amplitude	50	0.40313	98.44%	34.13%	50.68%	88.93%	88.35%
Amplitude, depth	35	0.21983	98.72%	40.52%	57.46%	90.00%	89.35%
Amplitude, depth	50	0.13895	97.80%	25.46%	40.40%	87.48%	87.01%

is the margin between positive and negative distances. This value hampers the training because the network must separate the individuals with an additional margin. We use an alpha value of 0.2 as suggested in FaceNet. For the experiments, we raised alpha in 0.1 steps till 0.7. We discovered no improvement from step 0.6 to 0.7. We do not expect any increase with higher alpha. To confirm this assumption we set alpha to 1.0. Again, one of the most important values is the Precision. For this, we compare all alpha values based on the Precision and the Recall. Alpha with 0.6 scored with best results. For the further experiments, we set alpha to 0.6. In Table 5.11 we compare the NIRFaceNetV1 with different dropout settings and 100,000 iterations. In the first row, we illustrate the results with 0% dropout. Next row illustrates the result with 10% dropout and last row with 50% dropout as suggested in NIRFaceNet. Based on the Precision we figured out that all variants performed nearly equal. Compared to the Recall, which reflects the successful logins to the system, we choose 10% as the best hyperparameter. The last hyperparameter in the experiment is the embedding dimension. The embedding vector represents the learned facial features. In Table 5.12 we illustrate the results for the NIRFaceNetV1 with 100,000 iterations and a learning rate of 0.0001 for each test. We used an alpha value of 0.6 and 10% dropout. We figured out that the embedding dimension with a size of 64 scored best on the validation set. Compared to the deep FaceNet we believe that the complexity reduction of the modified NIRFaceNet leads to this result. We set the embedding dimension for the last hyperparameter to 64 for further experiments.

5.5.2.3 Performance of Different Inception based CNN Architectures

In Table 5.13 we compare different network architectures with the evaluated hyperparameters based on the validation set. We summarise the hyperparameters: Batch size with 35, alpha value with 0.6, a dropout rate of 10% and the embedding dimension with 64. Recall Section 4.3 for an overview of the created NIRFaceNet derivations.

Table 5.10: Based on the NIRFaceNetV1, we test the impact of different alphas. We trained the network for 100.000 iterations with a learning rate of 0.0001 and an embedding dimension of 128. An alpha value of 0.6 scored with best results.

Alpha	τ	P	R	F_1	A	N
0.2	0.04714	96.58%	14.96%	25.76%	85.72%	85.44%
0.3	0.10826	98.27%	29.69%	45.92%	88.24%	87.70%
0.4	0.11372	98.04%	28.54%	44.21%	88.00%	87.48%
0.5	0.16396	98.18%	28.44%	44.10%	87.99%	87.47%
0.6	0.21983	98.72%	40.52%	57.46%	90.00%	89.35%
0.7	0.17937	97.93%	24.96%	39.78%	87.41%	86.94%
1.0	0.29993	98.11%	27.36%	42.78%	87.80%	87.30%

Table 5.11: In this experiment, we evaluate the best dropout rate with NIRFaceNetV1, 100,000 iterations training and a learning rate of 0.0001. We set the alpha value to 0.6 and the embedding dimension to 128. 10% dropout led to the best results.

Dropout	τ	P	R	F_1	A	N
0%	0.29108	98.50%	35.46%	52.15%	89.15%	88.56%
10%	0.36178	98.64%	41.46%	58.38%	90.15%	89.51%
50%	0.21983	98.72%	40.52%	57.46%	90.00%	89.35%

Table 5.12: Testing of different embedding sizes with NIRFaceNetV1. The embedding dimension with 64 scored in the experiment with the best results.

Embedding	τ	P	R	F_1	A	N
32	0.21594	98.71%	44.00%	60.87%	90.57%	89.92%
64	0.25750	98.91%	49.27%	65.78%	91.45%	90.78%
128	0.36178	98.64%	41.46%	58.38%	90.15%	89.51%

Table 5.13: Comparison of different network architectures of the modified NIRFaceNet. Overall, all versions performed nearly equal, V1 and V3 performed slightly better than V2.

NIRFaceNet	τ	P	R	F_1	A	N
NIRFaceNetV1	0.25750	98.91%	49.27%	65.78%	91.45%	90.78%
NIRFaceNetV2	0.36161	98.80%	47.17%	63.96%	91.10%	90.43%
NIRFaceNetV3	0.32377	98.85%	49.05%	65.56%	91.41%	90.74%

NIRFaceNetV1 has two fully-connected layers and fewer kernels than NIRFaceNetV2 and NIRFaceNetV3. NIRFaceNetV2 has more kernels in the first layer and only one fully-connected layer. NIRFaceNetV3 represents the original NIRFaceNet with removed Softmax. Overall, nearly all networks performed similarly. The NIRFaceNetV1 scored slightly better than the NIRFaceNetV3. We use for further training the NIRFaceNetV3 because the network has only one dense layer but more kernels. We hypothesise that this network is more flexible and with more training more efficient.

We summarise the best setup: alpha value with 0.6, embedding dimension of 64, batch size with 35 and a dropout rate of 10%. Additionally, we use the inception based architecture NIRFaceNetV3 for further training.

5.5.3 Final Results of Networks

In Table 5.14 we illustrate the results of the two-channel (amplitude and depth data) NIRFaceNetV3 after 500,000 iterations of training and the CNNV1 after around 200,000 iterations of training with three channels (amplitude, depth and noise data). The validation set contains six individuals and the test set eight. We started the networks generally with a learning rate of 0.0001. If we discover no changes in the kernels or no improvement in the false positive rate compared to the Recall, then we lower the learning rate. Next used learn rate is set to 0.00005 and for the last iterations at 0.00001. The NIRFaceNetV3 needed around 60 hours of training to reach zero loss. The CNNV1 needed around 21 hours of training. Recall our setup and hardware specifications in Chapter 7.

The CNNV1 and NIRFaceNetV3 perform nearly on par. Due to the inception based structure of NIRFaceNet, the training is more time-consuming. Additionally, we can conclude the influence of the weight initialisation of the kernels. In Table 5.7 the CNNV1 reached more than 70% login rate to the system. With training from scratch again we reached 55% for the CNNV1 on the validation set. To differentiate the deviation with more precision, it is necessary to complete even more training runs. We figured out that the inception based networks are more stable during training. The AlexNet (which is not trained on faces) performed poorly. We use a pre-trained AlexNet for this test with cosine similarity on the FC7 feature vector with 4096 entries. We can conclude that the network training and the quality of the data is an essential aspect for the results. We choose the NIRFaceNetV3 for our Android prototype due to the best performance.

Table 5.14: Comparison of CNNV1 (trained with amplitude, depth and noise data), NIRFaceNetV3 (trained with amplitude and depth data) and AlexNet (trained with RGB data on objects). The validation set consists of six people, the test set of eight.

Network	τ	P	R	F_1	A	N
CNNV1 Valid	0.48348	99.06%	55.62%	71.24%	92.51%	91.84%
CNNV1 Test	0.40206	98.57%	54.71%	70.37%	94.24%	93.92%
NIRFaceNetV3 Valid	0.49192	99.12%	62.52%	76.68%	93.66%	93.02%
NIRFaceNetV3 Test	0.37158	98.71%	60.73%	75.20%	94.99%	94.68%
AlexNet Valid	0.00536	94.49%	08.66%	15.87%	84.69%	84.54%
AlexNet Test	0.00379	86.50%	04.53%	8.62%	87.88%	87.99%

5.6 Android Application

In this section, we evaluate the created Android application. We evaluate the possible login distances and the result of the reference image selection. Additionally, we test the Android application in the real world with nine new individuals (they are not included in the training, validation or test set). Three individuals left the company at the moment we write this work. It is not possible to evaluate the candidates of the test set again with the Android application. Next, we evaluate the possibility to login in strong sunlight. Last, we evaluate the solution against images from the computer monitor and printed images from a high-quality laser printer.

5.6.1 Distance Evaluation and Reference Image Selection

In this section, we evaluate possible login distances. Additionally, we evaluate the reference image selection against randomly selected reference images.

In Table 5.15 we summarise the results of the possible login distances. For this experiment, we used the NIRFaceNetV3 and tested three individuals from different distances. A frontal login with the same facial expression and the same distance as the reference image lead to high login rates. Reference images from 40 cm distance lead to the highest login rates from different distances. A reference image of around 30 cm distance to the camera leads to the worst results if the distance rises with the login. It is nearly impossible to log in to the system at a distance of around 50 cm. We hypothesise that a login image between the boundary distances of 30 and 50 cm can compensate the quality changes with different distances.

We experimented with a Microsoft Surface Pro 5 and discovered that it is not possible to log in (after 40 attempts) with a reference image where the individual wears glasses. The device noticed that many login attempts failed. The Microsoft Surface suggested to capture more reference images to improve the login to the system. We adapted this solution for our system to get higher login rates. For this, we capture more reference images and save them to the database. Inspired by the Microsoft Surface, we experimented with an own creation of a reference image selection. The successful login strongly depends

Table 5.15: Successful login attempts at different distances and frontal measurements with the same facial expressions as the reference image. The results are based on three individuals. We captured one reference image and three login images.

Reference distance [cm]	Login distance 30 cm	Login distance 40 cm	Login distance 50 cm
30	5/5, 4/5, 5/5	3/5, 5/5, 3/5	0/5, 1/5, 0/5
40	5/5, 4/4, 4/5	5/5, 5/5, 5/5	5/5, 4/5, 5/5
50	3/5, 3/5, 4/5	4/5, 4/5, 4/5	5/5, 5/5, 5/5

Table 5.16: Comparison of reference image selection (three images out of five) against a random reference image for 59 login attempts. The reference image selection score with around 19 per cent higher login rate.

Method	Attempts	Rejected attempts	Login rate
Random image	59	20	66.10%
Best reference selection	59	9	84.75%

on the reference image. The recognition is harder if the reference image differs strongly from the login images. If the individual winks with the eyes or moves the smartphone faster during the image capturing, we figured out that some images are not usable for a login to the system. We capture five images and take three images out of this five images. The best one, the worst one and on in the middle (Recall Section 4.2). We hypothesise that reference images in different qualities can improve the successful login to the system. Additionally, the computation time increase due to the comparison with more reference images. We compare each login image with the three reference images of the same individual. We captured 59 login attempts with a random reference image and 59 attempts with the reference image selection (three out of five). We summarise in Table 5.16 that the login rate to the system increase by around 19% with the reference image selection.

5.6.2 Real-World Tests

In this section, we evaluate the Android application with indoor tests, outdoor tests and try to fool our solution with printed images and images from the computer monitor. Additionally, we evaluate different thresholds and the impact on the test set. With different thresholds, it is possible to login with different facial expressions and from side view.

5.6.2.1 Indoor Tests

In this section, we test the application indoor with nine candidates who don't belong to the training, validation or test set. In Table 5.17 we summarise the experiment with the

Table 5.17: Login results at different distances with individuals who don't belong to the training, validation or test set. The average login rate is about 70% with frontal images and without facial expressions.

Subject	Reference distance [cm]	Login distance [cm]	Gender	Successful logins
1	40	30-50	m	10/13
2	30-40	30-50	f	4/5
3	30-50	30-50	f	7/10
4	30-50	30-50	m	6/10
5	30-50	30-50	m	8/10
6	30-50	30-50	m	6/10
7	30-50	30-50	m	6/9
8	30-50	30-50	m	3/5
9	30-40	30-50	m	8/10

test candidates. The average login success rate is about 70%.

We tested different individuals with our application. It was not possible for anyone to log into the system as a false positive one. With the low threshold, it is possible that login attempts from the individual which should be able to login to the system are refused.

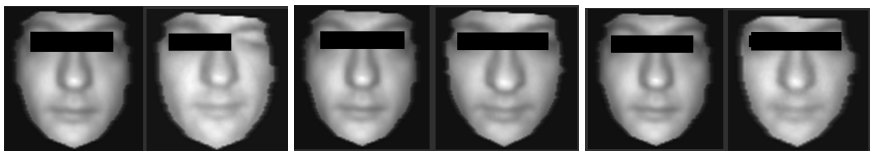
5.6.2.2 Strong Sunlight

Due to influences of the sun to the system, we experimented with the solution in the sunlight. We captured the reference images indoor with the reference image selection. We illustrate six login attempts in Figure 5.23. In Figure 5.23 (a) the login image is very noisy, one eye is closed, and the image is rejected. Next five images (b – d) are accepted. The distance varies between 30 and 50 cm. The reference image is on the left side of an image pair. In Figure 5.24 we illustrate the depth, noise and LBP image. The first row represents the indoor reference image and last row the depth, noise and LBP image from Figure 5.23 (f).

In Figure 5.25 we illustrate the outdoor depth image from Figure 5.24 from side view. The login is possible, and the confidence level is low, *i.e.* the resulted similarity value is close to the threshold. We believe this login image is accepted because the training set included reflecting surfaces in the background of the individuals, *i.e.* the auto exposure lowered the illumination time and noisy images are the result. The contour is visible (lips, chin, nose and forehead) and potentially the reason why the network could identify the individual correctly.

5.6.2.3 Printed Image Fooling

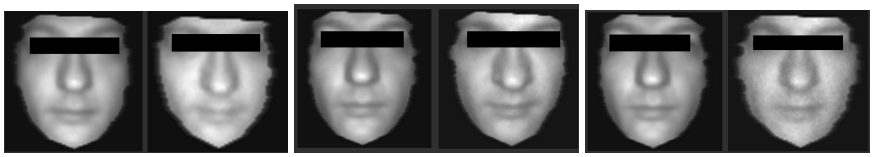
We printed out four images for this test with a high-quality laser printer from Ricoh (Model: MP C3003). We experimented with 32 login attempts with one printed RGB, and 48 login attempts with three printed ToF (greyscale) images. We printed out im-



(a) Rejected image in sunlight at a distance of around 50 cm.

(b) Accepted image in sunlight at a distance of around 40 cm distance.

(c) Accepted image in sunlight at a distance of around 50 cm.



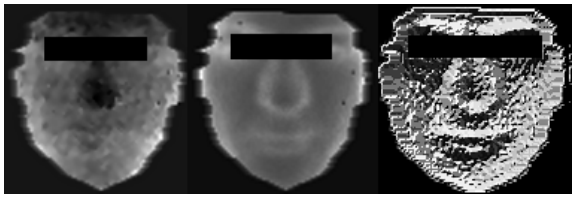
(d) Accepted image in sunlight at a distance of around 50 cm.

(e) Accepted image in strong sunlight at a distance of around 30 cm.

(f) Accepted image in strong sunlight at a distance of around 50 cm.

Figure 5.23: Verification in sunlight. Time: 10:30 AM. We captured the reference images indoor (left side of an image pair). The first image getting rejected, one eye is closed, and the image has visibly bad quality. The five other login attempts are accepted. Sunlight leads to more noise, and the quality of the data is lowered drastically.

(a) Indoor captured reference images. On the left side, we visualise the depth image, in the middle the noise image and last the LBP.



(b) Depth, noise and LBP image captured outdoor.

Figure 5.24: The indoor images are less noisy than the outdoor images. The quality drop is visible in each image. Additionally, holes in the noise image occured.

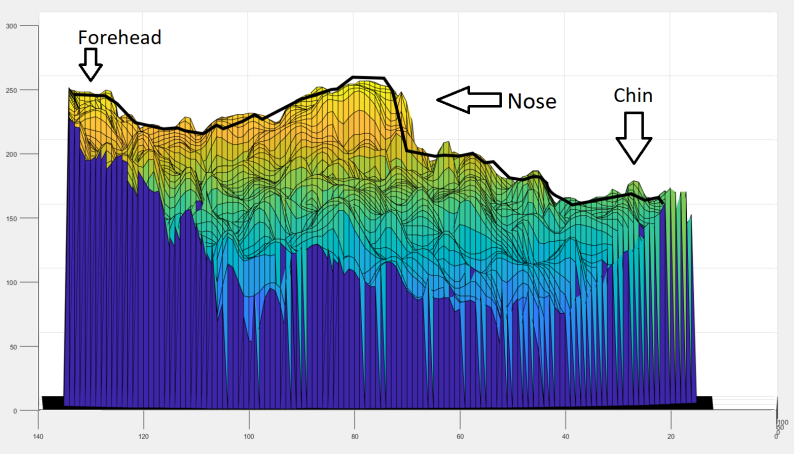


Figure 5.25: Accepted depth image under strong sunlight. Note that it is harder to log in with strong sunlight due to the impact of the sun on the quality of the data. The silhouette is generated from the indoor image.

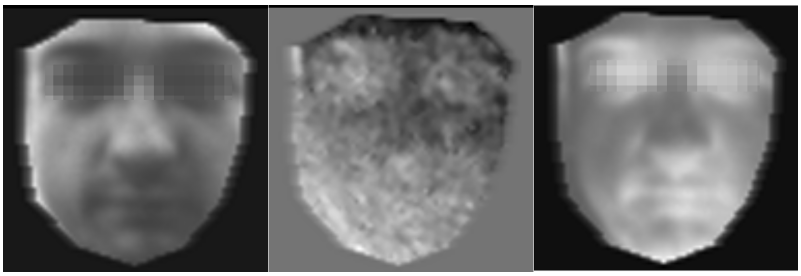


Figure 5.26: Test with a printed image from the front camera. The application rejected all (32) login attempts.

ages which the network accepted within login attempts. In Figure 5.26 we illustrate the outcome from the preprocessing pipeline of the printed RGB image which belongs to an accepted ToF image. The reflection influence the result, *i.e.* the depth image is not a plane. Instead, it has some contours which look like a deformed face. The distances between the reference and printed login image differed between 1.19 and 4.07 with an average value of 2.14. In Figure 5.27 we illustrate four attempts with the three printed ToF greyscale images which the network initially accepted during a login attempt with the real individual. The amplitude image changes the colour and contour depending on the reflection back to the camera and the viewpoint. The depth image has no reliable face contours. The noise image looks like a real noise image directly from the camera. We experimented with 48 login attempts with these three images. No effort was successful.

We experimented to capture the accepted login image from a computer monitor (Samsung 27", C27F396F). The preprocessing pipeline rejected all attempts because no face could be found due to the strong reflection on the monitor. The active illumination unit caused this issue. In our experiments, it was not possible to login from a computer monitor.

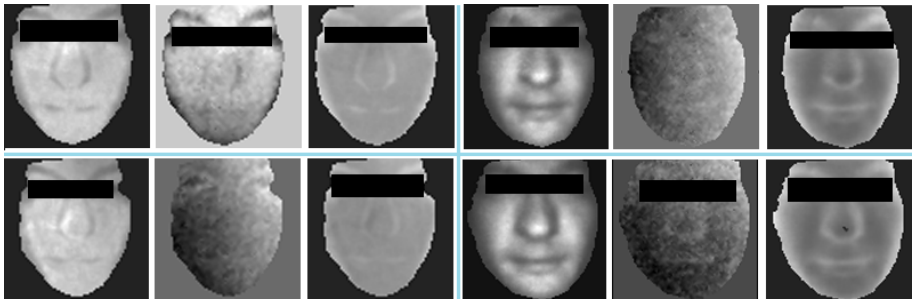


Figure 5.27: Test with printed images from the ToF camera. The application rejected these images.

5.6.2.4 Final Prototype

In this section, we summarise details about the prototype on Android. We use the NIR-FaceNetV3 for the experiments. In Figure 5.28 (a) we visualise the first prototype version on the smartphone. On the top left the saved reference image is illustrated and on the right side the login image. These images are from two different individuals. It is possible to take a reference image (REF.) and a login image (LOGIN). Additionally, the user can change the reference image and start the pipeline (START PIPELINE) and verification (VERIFICATION) independent for more flexibility. The text box visualises the calculated distances between the two individuals. The calculated distances are very high (differs around a value of two or three). Zero means that both images are identical and values from zero to the threshold that it is the same individual. Figure 5.28 (b) illustrates the start screen of the final prototype. The implemented slider (right top) change the threshold as follows:

Slider position: Very Low (τ = 0.44036)

Login images from the side view and with facial expressions are possible. The successful login rate to the system is around 69% with a False Discovery Rate of 0.007 (based on test set). Figure 5.29 (a, b) illustrates two login attempts from different views.

Slider position: Lower (τ = 0.40000)

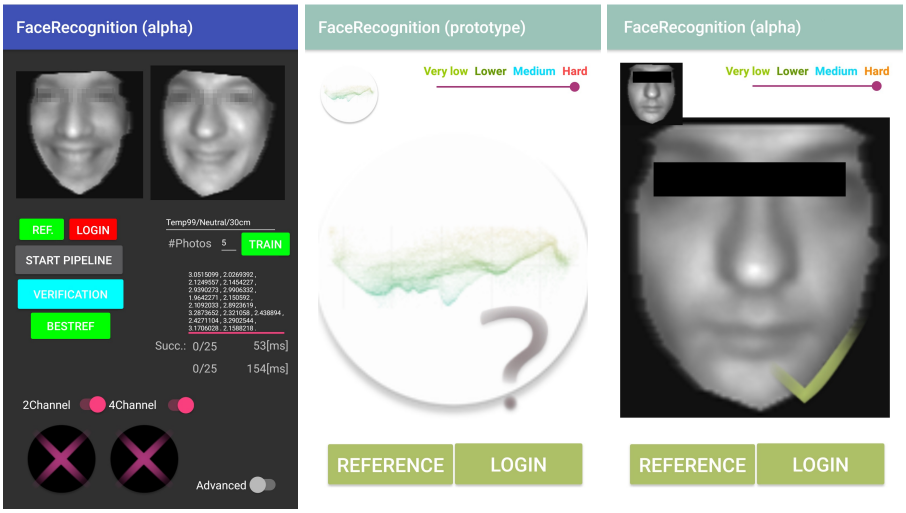
Frontal login images with facial expressions are possible. The successful login rate to the system is around 65% with a False Discovery Rate of 0.003 (based on test set). Figure 5.29 (c) illustrates a login attempt with facial expressions.

Slider position: Medium (τ = 0.37158)

Frontal login images without facial expressions are accepted. The successful login rate to the system is around 61% with a False Discovery Rate of 0.001 (based on test set).

Slider position: Hard (τ = 0.31572)

Very similar frontal login images are possible The successful login rate to the system is around 53% with a False Discovery Rate of 0.0 (based on test set). Figure 5.28 (c) illustrates a login attempt with a similar image as the reference image.



(a) First prototype with two different network architectures (CNNV1 and NIR-FaceNetV3). Best viewed digitally.

(b) Start screen of final prototype.

(c) Correct classified login image with slider position "Hard".

Figure 5.28: Prototype at early stage (left side) and final prototype (in the mid). The right side represents a login attempt with a very similar image as the reference image.

5.6.3 Performance compared with different Devices

In this section, we evaluate the performance with different Android smartphones. In Table 5.18 we compare two smartphones and their performance on the preprocessing pipeline and the neural network. We measured the preprocessing pipeline computation time with the face detection from OpenCV, the landmark detection and the preprocessing time of the amplitude, depth, noise, x, y and LBP data. Additionally, we measured the computation time of the CNN for one feature vector (based on a two-channel image). The real measurement needs more time because the activation of the ToF illumination unit and the capturing process with the ToF camera need around 1–1.5 seconds depending on the system. The data saving process which stores the data to the storage need some milliseconds additionally.

The Samsung Galaxy S9+ computes the six images with the preprocessing pipeline around 156% faster than the Galaxy S7. Additionally, the computation of the feature vectors is 122% faster.

5.7 Additional Insights

In this section, we evaluate the impact of the L2 normalisation on the output feature vector and additionally the impact of Batch Normalisation (BN) in the first layer. In FaceNet, L2 normalisation is used on the output feature vector. In contrast, NIRFaceNet uses max

(a) Login from side view with slider position "Very Low".

(b) Login from side view with slider position "Very Low".

(c) Frontal login with facial expressions and slider position "Lower".

Figure 5.29: The slider position "Very Low" change the threshold and log in from different views is possible. With the slider position "Lower" it is possible to login with facial expressions.

Table 5.18: Performance comparison between different smartphones. We measured the computation time of around 100 images and averaged the value. Additionally, the computation time depends on the system utilisation. The pipeline result is based on six processed images (amplitude, depth, noise, LBP, x, y) with face and landmark detection included. The result of the CNN (NIRFaceNetV3) computation time is based on one feature vector with two channels.

Test phone	Pipeline avg. [ms]	NIRFaceNetV3 avg. [ms]
Samsung S7	255.3	71.56
Samsung S9+	163.5	58.70

Table 5.19: Training with L2 and Batch Normalisation (BN). We used the NIRFaceNetV3 with 100,000 iterations for training. We denote max pooling as M and L2 normalisation as L.

Exp. Nr.	**Additional settings**	τ	P	R	F_1	A	N
1	Standard NIRFaceNetV3	0.32377	98.85%	49.05%	65.56%	91.41%	90.74%
2	With M, BN and L	0.30126	98.82%	45.15%	61.97%	90.77%	90.10%
3	Without M, with BN and L	0.35096	98.72%	44.22%	61.08%	90.61%	89.95%
4	Without M and BN, with L	1.61715	99.22%	72.73%	83.93%	95.36%	94.82%
5	With M, without BN and L	1.02426	99.10%	63.08%	77.09%	93.75%	93.12%

pooling. In Table 5.19 we summarise the experiments with the different settings. For this, we modified the NIRFaceNetV3. We can summarise that the NIRFaceNetV3 with max pooling performs slightly better than the modified version with L2 normalisation. Due to the excellent results in the literature, we are surprised that the removing of BN leads to a higher Recall. We observed the described effect of faster training. The loss function converged faster with BN. In Figure 5.30 we illustrate on the left side the loss function of the version with BN and on the right side without BN.

This effect needs further research to find the dependency. We believe the used ToF data and channel combinations can lead to this effect. We can't find research with CNN's and ToF data for face recognition to explain this effect in more detail. Additionally, we experimented with logins without environmental light or sunlight. We could not find any problems or differences, compared to the login with environmental light indoors.

The last experiment is an experiment with Softmax and ToF data. We trained the NIRFaceNetV3 with Softmax. We observed a significant faster training due to more efficiency of Softmax compared to Triplet Loss. Wojke et al. [WB18] claim that Triplet Loss training is depended on triplet selection and generally more challenging to train a network. We removed the Softmax layer after around 100,000 training iterations and retrained the network with Triplet Loss for around 30,000 iterations. The discovered effect: the results are nearly equal with 130,000 iterations (Softmax and Triplet Loss) and 500,000 iterations (Triplet Loss). Additionally, we discovered more structure in the kernels of the first layer. In the end, we could not reach better results, but the training is faster. In Figure 5.31 we visualise kernels of the NIRFaceNetV3 and CNNV1 with trained Softmax and a restart with Triplet Loss.

Additionally, we experimented with a newer camera (build in 2018) of the same series as our ToF camera (release date 2017). The new version has fewer outliers with the same sensor settings. We believe that a better calibration of the sensor can lead to fewer

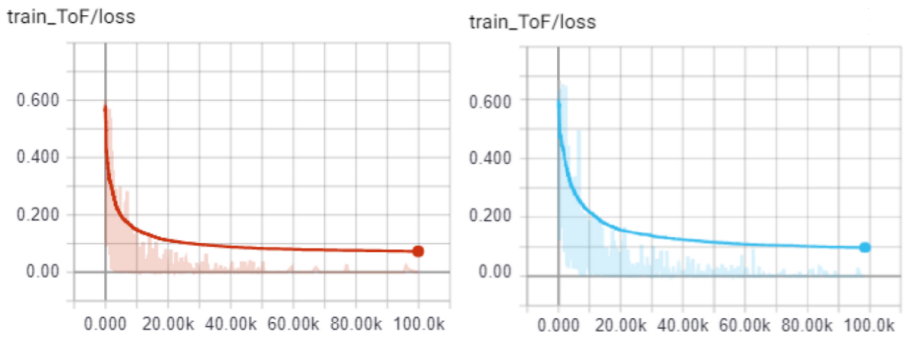


Figure 5.30: On the left side we illustrate the loss of the network from Table 5.19 and experiment number three, on the right side, we visualise the loss of experiment number four. Note that the loss function converged faster with BN.

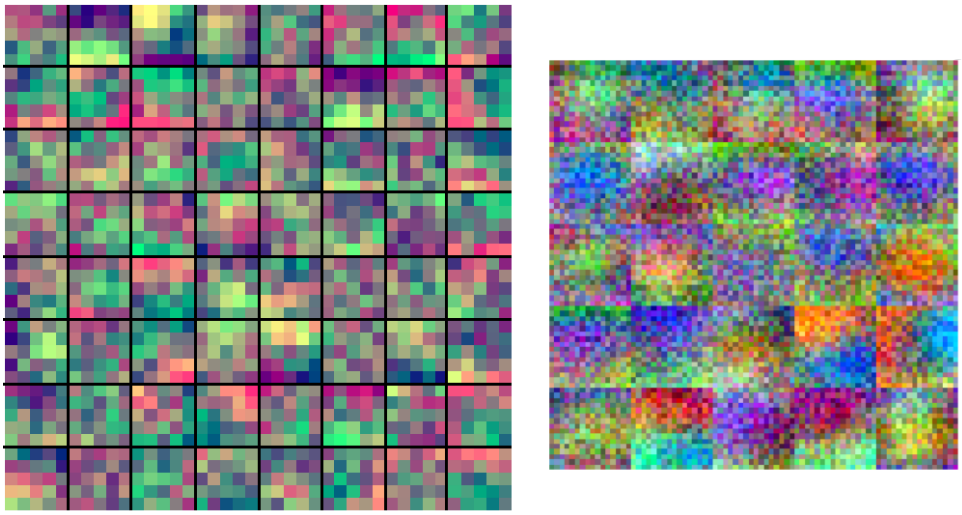


Figure 5.31: Kernel structure of NIRFaceNetV3 (left) and CNNV1 (right) after Softmax and Triplet Loss training. The red channel represents the amplitude data, green the depth data and blue the noise data. For the two-channel version of NIRFaceNetV3, we filled the third channel with zero values to visualise the kernels.

outliers. In Figure 5.32 we illustrate on the left side the captured images from the old camera model and on the right side with the new version of the camera.

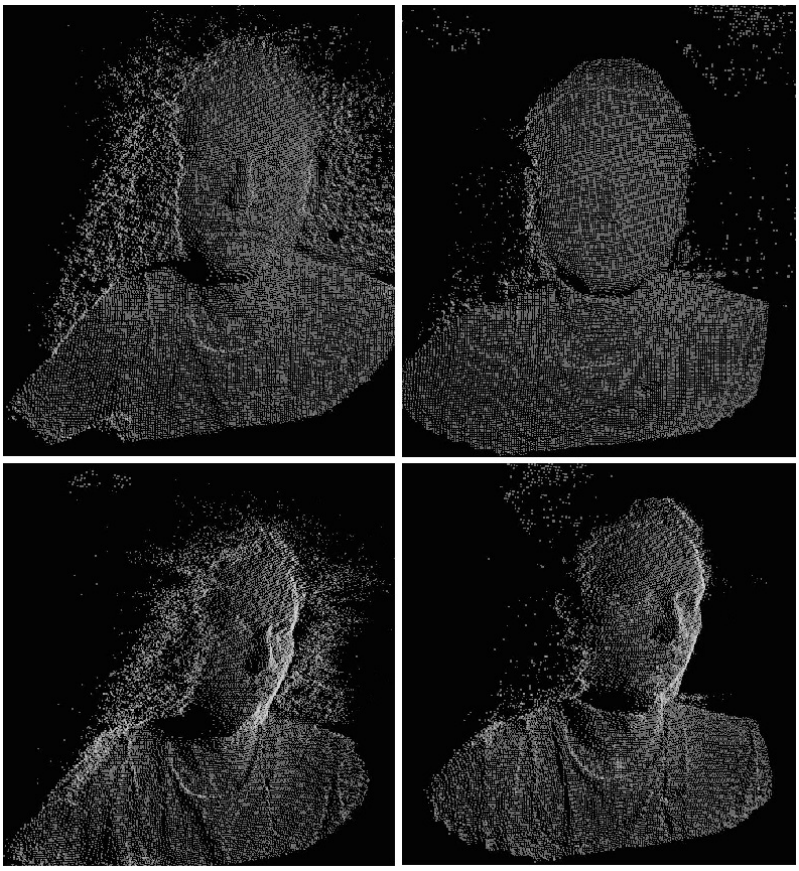


Figure 5.32: Our used camera (year 2017, left) compared to the new camera (year 2018, right). Note that strong outliers occurred around the face with the old model.

Chapter 6

Conclusion

In this chapter, we conclude our work and present ideas for further directives. We summarise the created setup and environment, the best-determined login distances, attempts to fool the solution and the impact of the depth channel on the results. Additionally, we give insights into the new sensor generation and improvements for our face recognition system.

6.1 Summary

In this Master's thesis, our aim was to build a face recognition prototype on Android. The solution should be reliable, easy to use and the False Discovery Rate should be low to prevent a wrong log in to the system. For this, we explored the possibility of ToF for face recognition. With our experiments, we figured out that face recognition with ToF data is reliably possible. Due to the restriction on Android smartphones, we are limited in computational power. It is not possible to use deep networks like FaceNet [SKP15] for verification within a time in which the user accepts the face recognition system.

We build up a database with ToF and RGB data to train and evaluate different networks. For this, we captured 47 individuals. We implemented a preprocessing pipeline that segments the faces and prepares the data for the CNN. We figured out a performance drop with the unsegmented background by around 11% (login rate to the system based on the validation set). Backgrounds with prepared random noise and average borders calculated from the face have no mentionable influence on the result compared to the entirely removed background variant. We hypothesise that the network focuses on the background in earlier training because the environment for the individual is the same in the captured images. We believe that more training data and temporal variations between measurements can solve this problem. In the end, the background segmentation requires more processing power but can improve and speed up the network training. Additionally, we can summarise that the initialisation is essential for the results of the network and differed many per cent with identical starts. In our experiments, we figured out that the inception based derivatives are more stable during training, but they needed more training iterations to reach zero loss. This additional training can also lead to more stable results. Another potential problem could be the triplet selection for the Triplet Loss training which is different for each new run. To differentiate the results with more precision, it is necessary to complete even more training runs.

We figured out that the distance and quality of the reference image are crucial to the results. A login at the same distance as the reference image led to the best results. With a reference image from 40 cm distance to the camera, we reached high login rates from different login distances (30–50 cm). Expressions and side views lowered the results. Additionally, the reference image selection for the comparison itself is important. For our experiments, we used one reference image from the validation or test set and compared them against the login images of the associated set.

We are unable to evaluate the exact effects of different emotions because not all individuals labelled their emotions correctly. Additionally, not all candidates showed the same facial expressions constantly over the entire capturing process. To allow a login with different emotions, we needed a lower threshold. For this, we configured a slider in the Android GUI which changes the threshold of the login system. With a rising threshold, a login with facial expressions and slightly from the side view is possible. The false positive rate increases with rising threshold.

One potential application for verification is car unlock with ToF. Logins in darkness due to the active near-infrared illumination unit and logins in strong sunlight (due to ambient light suppression) are possible. Based on these experiments, we believe that face recognition in cars is possible and could open up new research areas with ToF. For example, face recognition for car sharing and the monitoring of different individuals in a car.

In our experiments, we were unable to fool our face recognition system with printed images. We tried 80 login attempts with different distances, images and viewpoints. We received low similarities. We used images for this experiment which were accepted during login with the real individual. The resulted depth image from the printed image looks like a deformed face. We hypothesise that fooling the system with a printed image is not possible due to the observations of the distances between the reference and printed login image. In our experiments, we were also unable to fool the application with images from the computer monitor due to the strong reflection from the monitor which is caused by the active illumination unit from the camera. Additionally, we figured out that the amplitude data without the depth channel perform with around 14% higher login rates. Mian et al. [MBO07] summarise that 2D images combined with high-quality 3D laser scanned images can improve the login rate to the system. We hypothesise that a less noisy depth channel can improve our solution and additionally improve the security. With the current ToF sensor generation, the depth channel lowered the performance of the system. FaceNet for smartphones reached a Recall between 51.9–67 (private test set with one million images and LFW test set). Our solution performed with a Recall of around 60% on the captured ToF test set with 2654 images.

6.2 Outlook

The preprocessing pipeline processed around 5–6% wrong faces and landmarks. This includes, for example, wrong landmarks from the side view, a neck or one eye which should represent the face. Generally, these examples are rejected and lowered the performance of our system. We hypothesise that a trained face and landmark detection on ToF data can improve our solution (CNN's like [HR17] for example).

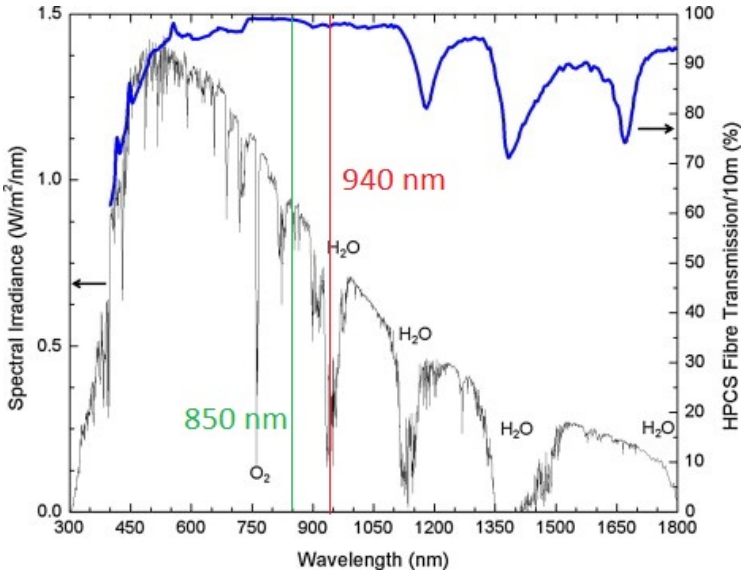


Figure 6.1: Wavelength of sunlight. The sun has less impact at 940 nm wavelength where the new sensor generation operates. Our current ToF camera has a 850 nm illumination unit. Figure based on [Arn+13].

The authors in FaceNet [SKP15] summarise that more training images lead to better performance. In our experiments, we figured out that five more individuals improved the Precision of the system by around 1.5%. We hypothesise that more training images can further improve the performance of the system. Additionally, we believe that rejection classifiers as in [MW09] would accelerate the face recognition process in early stages.

The next sensor generation of our used ToF sensor (which should be released in 2019[1]) operates with a wavelength of 940 nm. In this area, the sun has less influence. We hypothesise that this can improve the login rate outdoors. In Figure 6.1 we illustrate the wavelength of the sun.

We hypothesise that an auto exposure that focuses on the face and not on the whole image can improve the solution further. Strong reflections in the background lead to a lowered auto exposure and noisy images. Additionally, we figured out that a pre-training of the network with Softmax, followed by TripletLoss training can reduce the training time of the network. Wojke et al. [WB18] summaries that the triplet selection is important for the success of the training. We believe that also Triplet Loss can change the results with each training run from scratch due to the random combination of triplets. Developing an effective sampling strategy is complex and additionally, time-consuming. [WB18]. For this, we expect better results with an intelligent triplet selection algorithm.

In [Nah+18] the possibility of pulse measurement with ToF is explored. Another protection mechanism of the system could be the liveness detection. Measuring the pulse with contact and remote photoplethysmography is possible.

An interesting research direction would be the investigation of deeper networks to

[1] https://www.pmdtec.com/html/pdf/press_release/PR_20180109_pmd_IFX_new_imager_final.pdf

ToF. With this thesis, we were limited to shallow networks which can be deployed in our smartphone prototype. State-of-the-art are very deep networks, so we expect significant improvements with deeper architectures.

Chapter 7

Appendix

We illustrate the complete setup of our project in Figure 7.1. It consists of two computers. We use the Windows computer for high-quality graphics with Matlab and also to train the networks. We use Linux Fedora for Android programming and to monitor the output of the smartphone with the Python 3D libraries. To save and archive all the results, we use a one terabyte HDD due to the big size of the recordings. The used CPU for training is an AMD Ryzen 2700x (eight core) and Geforce 1080 GTX graphics card with 8 GB VRAM and CUDA 9.2 libraries.

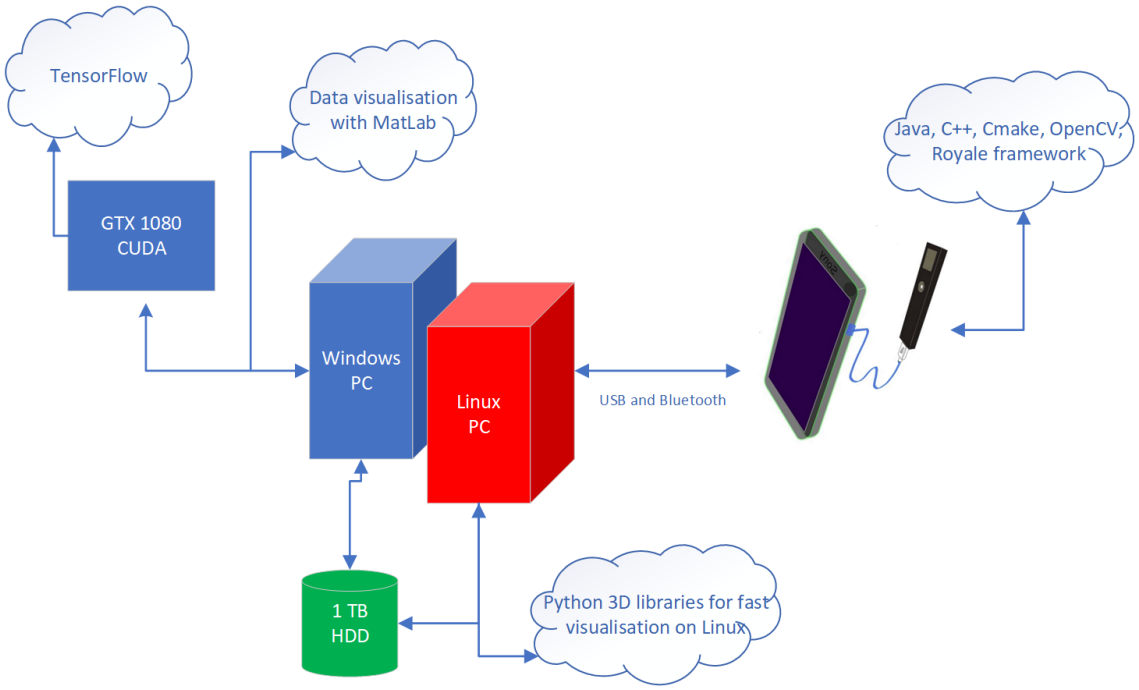


Figure 7.1: Complete setup. Due to fewer troubles we decided to use Windows for Tensorflow [Aba+15] and high-quality visualisation with Matlab. We used Linux for Android programming.

Nomenclature

A ..... Accuracy

BN ..... Batch Normalisation

CNN ..... Convolutional Neural Network

F_1 ..... F_1 measure

F ..... False Discovery Rate

FN ..... False Negative

FP ..... False Positive

ICP ..... Iterative Closest Point

LLDB ..... LLDB Debugger

LBPH ..... Local Binary Pattern Histogramming

NDK ..... Native Development Kit

N ..... Negative Predictive Value

P ..... Precision

PCA ..... Principal Component Analysis

PMD ..... Photonic Mixing Device

R ..... Recall

SIFT ..... Scale-invariant feature transformation

SFR ..... Spherical Face Representation

ToF ..... Time-of-Flight

TP ..... True Positive

TN ..... True Negative

Bibliography

[Aba+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heteeneous Distributed Systems*. 2015. URL: http://download.tensorflow.org/paper/whitepaper2015.pdf (visited on 11/12/2018).

[Alm18] Matthias J. Almer. *Time-Of-Flight 3D Camera Performance Modeling*. Infineon internal. 2018.

[AS15] S. Annadurai and R. Shanmugalakshmi. *Fundamentals of Digital Image Processing*. 1st ed. Pearson India, 2015. ISBN: 9789332501454.

[Arn+13] Georgios E. Arnaoutakis, Jose Marques-Hueso, Tapas K. Mallick, and Bryce S. Richards. "Coupling of sunlight into optical fibres and spectral dependence for solar energy applications". *Solar Energy* 93 (2013), ISSN: 0038-092X. DOI: 10.1016/j.solener.2013.04.008.

[Bal+18] Diego Ballotta, Guido Borghi, Roberto Vezzani, and Rita Cucchiara. "Head Detection with Depth Images in the Wild". In: *Proceedings of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. 2018. DOI: 10.5220/0006541000560063.

[BHK97] Peter N. Belhumeur, João P. Hespanha, and David. J. Kriegman. "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.7 (1997), ISSN: 0162-8828. DOI: 10.1109/34.598228.

[DG06] Jesse Davis and Mark Goadrich. "The Relationship Between Precision-Recall and ROC Curves". In: *Proceedings of the International Conference on Machine Learning*. 2006, DOI: 10.1145/1143844.1143874.

[Dru+15] Norbert Druml, Gerwin Fleischmann, Christoph Heidenreich, Andrea Leitner, Helmut Martin, Thomas Herndl, and Gerald Holweg. "Time-of-Flight 3D imaging for mixed-critical systems". In: *Proceedings of the International Conference on Industrial Informatics*. 2015, DOI: 10.1109/INDIN.2015.7281943.

[Dua+18] Jie Duan, RuiXin Zhang, Jiahu Huang, and Qiuyu Zhu. "The Speed Improvement by Merging Batch Normalization into Previously Linear Layer in CNN". In: *Proceedings of the International Conference on Audio, Language and Image Processing*. 2018, DOI: 10.1109/ICALIP.2018.8455587.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". *Journal of Machine Learning Research* 12 (2011), ISSN: 1532-4435.

[GB10] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics*. 2010,

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN: 978-0262035613.

[Hea+98] Marti A. Hearst, Susan T. Dumais, Edgar Osuna, Jason Platt, and Bernhard Scholkopf. "Support vector machines". *IEEE Intelligent Systems and their Applications* 13.4 (1998), ISSN: 1094-7167. DOI: 10.1109/5254.708428.

[HSA97] Harold. Hill, Philippe G. Schyns, and Shigeru Akamatsu. "Information and viewpoint dependence in face recognition". *Cognition* 62.2 (1997), ISSN: 0010-0277. DOI: 10.1016/S0010-0277(96)00785-8.

[Hor84] Berthold K. P. Horn. "Extended Gaussian images". *Proceedings of the IEEE* 72.12 (Dec. 1984), ISSN: 0018-9219. DOI: 10.1109/PROC.1984.13073.

[HR17] P. Hu and D. Ramanan. "Finding Tiny Faces". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, DOI: 10.1109/CVPR.2017.166.

[Hua+07] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. 07-49. University of Massachusetts, Amherst, Oct. 2007.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the International Conference on Machine Learning*. 2015,

[Kar18] Andrej Karpathy. *Lecture notes on CS231n: Convolutional Neural Networks for Visual Recognition*. Stanford University. July 2018. URL: http://cs231n.github.io/.

[KB15] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the International Conference on Learning Representations*. 2015.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the International Conference on Neural Information Processing Systems*. 2012,

[Kün16] Thomas Künneth. *Android 7: Das Praxisbuch für Entwickler*. 4th ed. Rheinwerk Computing, 2016. ISBN: 9783836242004.

[LPS16] Alfirna R. Lahitani, Adhistya E. Permanasari, and Noor A. Setiawan. "Cosine similarity to determine similarity measure: Study case in online essay assessment". In: *Proceedings of the International Conference on Cyber and IT Service Management*. 2016. DOI: 10.1109/CITSM.2016.7577578.

[Lau+16] Cesar. Laurent, Gabriel Pereyra, Philemon Brakel, Ying Zhang, and Yoshua. Bengio. "Batch normalized recurrent neural networks". In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. 2016. DOI: 10.1109/ICASSP.2016.7472159.

[Li18] X. Li. "Preconditioned Stochastic Gradient Descent". *Transactions on Neural Networks and Learning Systems* 29.5 (2018), ISSN: 2162-237X. DOI: 10.1109/TNNLS.2017.2672978.

[Low04] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". *International Journal of Computer Vision* 60.2 (2004), ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94.

[Luo+99] Jie Luo, Bo Hu, Xie-Ting Ling, and Ruey-Wen Liu. "Principal independent component analysis". *IEEE Transactions on Neural Networks* 10.4 (1999), ISSN: 1045-9227. DOI: 10.1109/72.774259.

[MHN13] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proceedings of the Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[MS11] Kumar Meena and A. Suruliandi. "Local binary patterns and its variants for face recognition". In: *Proceedings of the International Conference on Recent Trends in Information Technology*. 2011, DOI: 10.1109/ICRTIT.2011.5972286.

[MW09] Simon Meers and Koren Ward. "Face Recognition Using a Time-of-Flight Camera". In: *Proceedings of the International Conference on Computer Graphics, Imaging and Visualization*. 2009, DOI: 10.1109/CGIV.2009.44.

[MBO07] Ajmal Mian, Mohammed Bennamoun, and Robyn Owens. "An Efficient Multimodal 2D-3D Hybrid Approach to Automatic Face Recognition". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.11 (2007), ISSN: 0162-8828. DOI: 10.1109/TPAMI.2007.1105.

[Mos17] Ehsan Mostafavi. *An introduction to sensitivity, specificity, positive and negative predictive values*. Pasteur institute of Iran. July 2017. DOI: 10.13140/RG.2.2.13863.96163. URL: https://www.researchgate.net/publication/316476220_An_introduction_to_sensitivity_specificity_positive_and_negative_predictive_values.

[Nah+18] Caterina Nahler, Bernhard Feldhofer, Matthias Ruether, Gerald Holweg, and Norbert Druml. "Exploring the Usage of Time-of-Flight Cameras for Contact and Remote Photoplethysmography". In: *Proceedings of the Conference on Digital System Design*. 2018, DOI: 10.1109/DSD.2018.00079.

[Pen+16] Min Peng, Chongyang Wang, Tong Chen, and Guangyuan Liu. "NIRFaceNet: A Convolutional Neural Network for Near-Infrared Face Identification". *Information* 7.4 (2016), DOI: 10.3390/info7040061.

[Ras17] Sebastian Raschka. *Python Machine Learning*. 1st ed. Packt Publishing, 2017. ISBN: 9781783555130.

[Ren+14] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. "Face Alignment at 3000 FPS via Regressing Local Binary Features". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, DOI: 10.1109/CVPR.2014.218.

[RM51] Herbert Robbins and Sutton Monro. *Annals of Mathematical Statistics. A Stochastic Approximation Method*. Vol. 22. 1951, DOI: 10.1214/aoms/1177729586.

[Rog+17] Nicoleta Rogovschi, Jun Kitazono, Nistor Grozavu, Toshiaki Omori, and Seiichi Ozawa. "t-Distributed stochastic neighbor embedding spectral clustering". In: *Proceedings of the International Joint Conference on Neural Networks*. 2017, DOI: 10.1109/IJCNN.2017.7966046.

[SR15] Takaya Saito and Marc Rehmsmeier. "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets". *PLoS One* 10.3 (2015). DOI: 10.1371/journal.pone.0118432.

[SKP15] Florian Schroff, Dimitry 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*. 2015, DOI: 10.1109/CVPR.2015.7298682.

[SNG14] Adrian. R. S. Siswanto, Anto S. Nugroho, and Maula Galinium. "Implementation of face recognition algorithm for biometrics based time attendance system". In: *Proceedings of the International Conference on ICT For Smart Society*. 2014, DOI: 10.1109/ICTSS.2014.7013165.

[Sze+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015. DOI: 10.1109/CVPR.2015.7298594.

[Tai+14] 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*. 2014, DOI: 10.1109/CVPR.2014.220.

[TTC18] Vignesh Thakkar, Suman Tewary, and Chandan Chakraborty. "Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data". In: *Proceedings of the International Conference on Emerging Applications of Information Technology*. 2018, DOI: 10.1109/EAIT.2018.8470438.

[TP91] Matthew A. Turk and Alex P. Pentland. "Face recognition using eigenfaces". In: *Proceedings of the IEEE Computer Conference on Computer Vision and Pattern Recognition*. 1991, DOI: 10.1109/CVPR.1991.139758.

[Wan+10] Stephen C. Want, Olivier Pascalis, Mike Coleman, and Mark Blades. "Recognizing people from the inner or outer parts of their faces: Developmental data concerning 'unfamiliar' faces". *British Journal of Developmental Psychology* 21.1 (2010), DOI: 10.1348/026151003321164663.

[WB18] Nicolai Wojke and Alex Bewley. "Deep Cosine Metric Learning for Person Re-identification". In: *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*. 2018, DOI: 10.1109/WACV.2018.00087.

[WPW03] Yijun Wu, Gang Pan, and Zhaohui Wu. *Audio- and Video-Based Biometric Person Authentication. Face Authentication Based on Multiple Profiles Extracted from Range Data*. 2003, DOI: 10.1007/3-540-44887-X_61.

[XJT17] Yiliang Xie, Hongyuan Jin, and Eric C. C. Tsang. "Improving the Lenet with batch normalization and online hard example mining for digits recognition". In: *Proceedings of the International Conference on Wavelet Analysis and Pattern Recognition*. 2017, DOI: 10.1109/ICWAPR.2017.8076680.

[ZF14] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014.

[Zha+16] Shun Zhang, Yihong Gong, Jia-Bin Huang, Jongwoo Lim, Jinjun Wang, Narendra Ahuja, and Ming-Hsuan Yang. "Tracking Persons-of-Interest via Adaptive Discriminative Features". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.