FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Lernen von Fingerabdrucksdetails für eine biometrische Authentifizierung mit Hilfe von tiefen Netzwerken

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Thomas Pinetz, BSc
Matrikelnummer 1227026

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Univ.Prof. DI. Dr. techn. Robert Sablatnig

Wien, 1. Jänner 2001

Thomas Pinetz	Robert Sablatnig

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Learning Fingerprint Minutiae for Biometric Authentication via Deep Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media Informatics and Visual Computing

by

Thomas Pinetz, BSc
Registration Number 1227026

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. DI. Dr. techn. Robert Sablatnig

Vienna, 1st January, 2001

Thomas Pinetz

Robert Sablatnig

Erklärung zur Verfassung der Arbeit

Thomas Pinetz, BSc
A7132 - Frauenkirchen Maria Weitner-Platz 7

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

Wien, 1. Jänner 2001

Thomas Pinetz

Danksagung

Ich danke dem Austrian Institute of Technology für die Möglichkeit diese Arbeit unter ihrer Aufsicht verfassen zu dürfen. Dabei möchte ich besonders Daniel Soukup und Reinhold Huber-Mörk danken, welche mich am AIT betreut haben und sich mit meinen Problemen auseindandergesetzt haben.

Auch möchte ich meinem Betreuer Robert Sablatnig danken. Ich möchte mich darüber hinaus bei Peter Wild für die Hilfe beim verfassen der Arbeit bedanken. Zuletzt möchte ich mich noch bei Thomas Pock für seine hielfreichen Bemerkungen zu adversarial Training bedanken.

Acknowledgements

I want to thank the AIT for providing me the opportunity to write this thesis under their supervision. I am especially thankful to my advisors at the AIT, namely Daniel Soukup and Reinhold Huber-Mörk, who tirelessly worked with me to make the best thesis possible.

Additionally I would like to thank my advisor Robert Sablatnig for all the time he spent with me working on this thesis.

I thank Peter Wild for helping me write the thesis and for all those insightful conversations we had on biometrics. Last but not least I want to thank Thomas Pock for his insightful remarks to the adversarial training part of this thesis.

Kurzfassung

Biometrische Authentifizierung wird eingesetzt um wichtige Infrastrukturpunkte abzusichern. Die meist benutzte Form der biometrischen Authentifizierung ist das Vergleichen von Fingerabdrücken. Derzeitige Methoden benutzen spezielle Details, genannt „Minutiae“, in Fingerabdrücken für robuste Ergebnisse. Dazu braucht es einen robusten Algorithmus um diese Fingerabdruckdetails zu extrahieren. Der derzeitige State of the Art benutzt zeitraubende Bildverbesserungsprozeduren und qualitativ hochwertige Fingerabdrucksbilder um eine stabile Erkennung von Minutiae gewährleisten zu können. Mit dem Anstieg an Verfahren in verwandten Gebieten, welche auf tiefen neuronalen Netzwerken basieren, und der freien Verfügbarkeit von synthetischen Fingerabdrucksgeneratoren, ist es vielversprechend diese Technologie hier anzuwenden. Die Idee hierbei ist ein gleichwertiges binäres Segmentierungsproblem zu lösen, welches für jeden Pixel die Wahrscheinlichkeit ermittelt, dass dieses zu einem Minutiaepunkt gehört. Dieses Problem wird dann mit einem U-shaped Fully Convolutional Neural Network (U-net FCNN) gelöst. Mit dieser Information wird dann eine Liste an Minutiae extrahiert und mit existierenden Verfahren auf Basis der Vergleichswerte verglichen. Hierbei werden ähnliche Werte erzielt, wie derzeitige state-of-the-art Verfahren.

Auf neuronalen Netzwerken basierende Verfahren haben eine starke Abhängigkeit zu den Daten, welche benutzt werden um diese zu lernen. Der frei verfügbare Fingerabdrucksgenerator Anguli wird benutzt um den Grundstock an Daten zu generieren. Hierbei ist für jeden Fingerabdruck auch ein sauberes Bild des dazugehörigen Fingerabdrucksmuster erstellt worden. Die Ground Truth dazu wird vom kommerziellen Minutiaeextraktor Verifinger auf den Fingerabdrucksmusterbildern erstellt. In dieser Arbeit wird davon ausgegangen, dass der derzeitige State-of-the-Art auf diesen Bildern keine Fehler macht. Um die Unterschiede zwischen den Trainingsbildern und echten Bildern zu minimieren wird ein neuer Ansatz, basierend auf adversarial training angewendet. Dieser Ansatz wird verwendet um Veränderung zu lernen, welche die generierten Fingerabdrucksbilder ununterscheidbar von echten Bildern machen. Dieser Algorithmus kann während dem Training angewendet werden um die Trainingsbilder zu verändern und verbessern die Vergleichswerte des Verfahrens über alle getesteten Datenbanken.

Abstract

Biometric authentication is used to secure vital infrastructure like airports. The most common form thereof is fingerprint matching. Current matcher use information about special landmarks in fingerprints, called minutiae for robust matching results. For this reason a stable fingerprint minutiae extractor is needed. Current solutions use time-consuming image enhancements routines and rely on specific quality measurements to allow reliable extraction of minutiae landmarks. With the rise of deep learning in similar areas and the free availability to synthetic fingerprint generators it is promising to try this technology on the minutiae extraction problem. The idea is to solve an equivalent binary segmentation problem, where for every pixel it is determined, if this pixel belongs to a minutiae region. This problem is then solved using an U-net FCNN. Using this information, a minutiae list is extracted and is then compared to existing solutions based on the matching score. Thereby results similar to other state-of-the-art methods are reported.

Learning based approach depend on their training data. The freely available synthetic fingerprint generator Anguli is used to generate the basic dataset for training. Anguli also provides clean images of the ridge pattern to every fingerprint. The commercial minutiae extractor Verifinger is then used on those clean images to add a suitable ground truth. In this thesis it is proven, that it is possible to learn a minutiae extractor, which is better than the minutiae extractor used to generate the ground truth. To minimize the difference between synthetic images and real fingerprint images a novel approach based on adversarial learning is proposed, where suitable refinements are learned instead of hand crafted. This enables to generate synthetic fingerprint images indistinguishable from real fingerprints and improve the matching score across all tested databases.

Contents

CHAPTER 1

Introduction

Biometric Authentication is a reliable way to secure vital infrastructure and is therefore used by major law enforcement departments to ensure national security [MMJP09]. Even apart from national security, biometric authentication is on the rise due to a variety of cheap sensing technology and use cases, such as mobile phones, where fast and reliable authentication is needed [MMJP09]. Currently, the most widely used and accepted form of biometric authentication technique is fingerprint matching [MMJP09].

Fingerprints are a reliable way to determine the identity of an individual, because of their uniqueness and their temporal stability [HWJ98]. The characteristic information in fingerprints are special landmarks named minutiae [MMJP09]. These landmarks are irregularities in ridge pattern. Two out of 150 [MMJP09] such patterns are termination and bifurication as shown in Figure 1.1. Termination symbolizes an ending of a ridge, while the fusion of two ridges is called bifurcation. While 150 different irregularity patterns have been identified [Moe71], all can be reduced to some form of ridge termination or bifurcation [MMJP09]. While the amount of minutiae on a fingerprint varies from finger to finger, there are approximately one hundred of them in a regular fingerprint [MMJP09]. Still, only 10 - 15 minutiae are required to reliably identify an individual [MMJP09].

Current matching technology [WGT+04] relies on minutiae points. Therefore a robust minutiae extractor is needed. Extraction of minutiae relies on fingerprint images with a dpi of at least 500. Additionally, to work under unstable lighting conditions and noise, image enhancement algorithms are used [HWJ98, CLJ14]. However, the interplay of feature extraction with enhancement in general and single enhancement routines in particular is undocumented, unclear and a holistic form of processing [Ver10]. Therefore, reliable minutiae extraction on arbitrary fingerprint images is an open problem [TGF16], due to the large amount of changes in fingerprint images as is shown in Figure 1.3. Figure 1.3(a,b) show the same fingerprint acquired with different sensing technology.

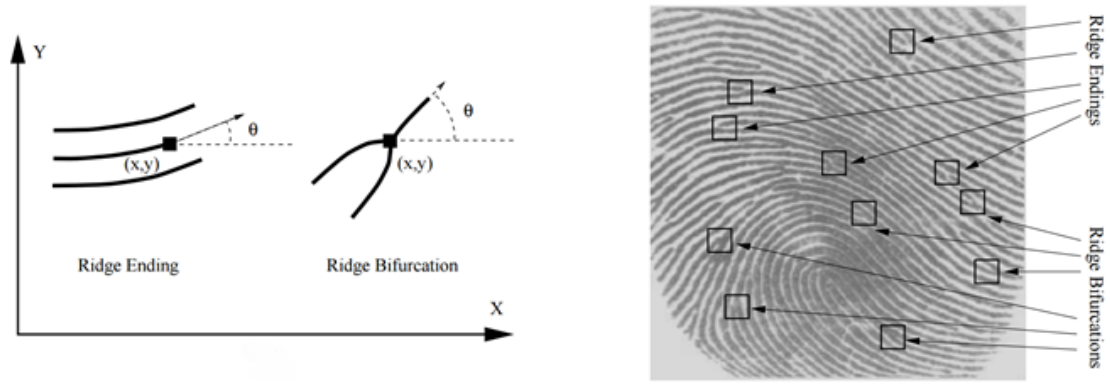


Figure 1.1: Example ridge endings and bifurcations of a fingerprint. This image is taken from [MMJP09].

Deep learning is used in biomedical image segmentation [DVC+16, RFB15] and fingerprint liveness detection [NdALM16] with promising results. Additionally, with research in synthetic fingerprint generators [Ans11, CMM04], it is possible to generate an arbitrary large fingerprint datasets for training and evaluation. With these tools, it looks promising to apply deep learning for the minutiae extraction problem. The freely available fingerprint generator Anguli [Ans11] is used to generate ridge patterns as in Figure 1.2(a). Anguli also generates multiple, different fingerprint instances for each ridge pattern, called impressions. One such simulated impression of a fingerprint is illustrated in Figure 1.2(b). Because of the difference to real fingerprints as are shown in Figure 1.3 augmentation as shown in Figure 1.2(c) is used to bridge the gap. Here, a novel approach for fingerprint refinement based on Generative Adversarial Networks (GANs) is used to learn refinements indistinguishable from real fingerprints, as shown in Figure 1.2(d).

To be able to use deep learning to extract minutiae from fingerprint images, the problem is reformulated as an equivalent binary segmentation problem. The fingerprint image is segmented in two classes for every pixel determining if this pixel belongs to a minutia or not. To solve this problem, a novel neural network architecture is used in this thesis based on the U-shaped neural network design pioneered in [RFB15].

1.1 Motivation and Contributions

This thesis is motivated by the following research question. Can the current state-of-the-art in fingerprint matching be improved by using deep NNs for the minutiae extraction step? Even though we did not improve the current state-of-the-art in fingerprint matching the following contributions are made as part of this thesis. First, a novel technique to generating realistic, diverse fingerprint is proposed, which are better suited to training a NN for the minutiae extraction task than existing solutions. This novel approach is based

(a) (b) (c) (d)

Figure 1.2: The same cropped fingerprint, with the original ridge pattern shown in (a). (b) represents one simulated fingerprint impression generated by Anguli from the ridge pattern, while (c) is an augmented version using a 3×3 Gaussian filter. In (d) the output of the refinement network is shown.

(a) Fingerprint taken from FVC2000 DB 1. (b) Fingerprint taken from FVC2000 DB 2 (c) Fingerprint taken from FVC2000 DB 3

Figure 1.3: Example real fingerprints taken from the dataset used for the fingerprint competition FVC2000 [MMC+02a]. Note the difference in resolution, fingerprint size and illumination. Different sensors were used to generate those fingerprints.

on GANs [GPAM+14]. However, our method also produces a corresponding ground truth to the generated data. Second, the minutiae extraction task is reformulated as a binary segmentation task as described in Chapter 6, which in turn is solved by a NN, trained from the ground up. Similar results to state-of-the-art fingerprint matcher are shown by using this approach. Third the following design choices in NN are evaluated based on their applicability to the minutiae extraction task:

1. Network design: Different networks exist for the purpose of tackling a binary segmentation problem [DVC$^+$16] [RFB15]. In this thesis the state-of-the-art in binary segmentation [DVC$^+$16] is combined with the state-of-the-art in classification [ZK16] to improve the accuracy for the minutiae extraction task.

2. Optimizer: Adam [KB14], RMSProp [TH12] and Stochastic Gradient Descent [Nes83] (SGD) achieved promising results in segmentation tasks [RFB15, DVC$^+$16] and are therefore evaluated for the minutiae extraction network. Adam yielded superior accuracy compared to the other optimizers.

3. Learning Rate Schedule: Three different learning rate scheduling algorithms are evaluated, namely decay [MSM16], step [MSM16] and cyclic learning [Smi17]. Step Learning produced the best performance for the models.

4. Batch size: The experiments in Chapter 4 show a tendency that higher batch size correlates with higher accuracy. The batch size though is limited by the amount of VRAM and therefore the highest possible value for our machine was used, which is 16.

5. Loss function: Dice Loss and Cross Entropy are used in similar domains, with state-of-the-art results [DVC$^+$16]. While the performance of both loss functions is similar, Dice Loss produced binary output without the need for post processing and was therefore selected as a loss function.

6. Dropout: Dropout [SPT$^+$16] has been used to improve performance of similar segmentation NN [DVC$^+$16]. Those results were not reproduced and Dropout was therefore omitted in the final network architecture.

1.2 Methodological Approach

The following methodological approach is used to solve the minutiae extraction problem:

1. Anguli [Ans11] is used to generate 30.000 synthetic ridge patterns.

2. The commercial minutiae extractor Verifinger [Ver10] is used to generate the minutiae map ground truth for training. For the purpose of this thesis, it is assumed that the minutiae extractor works perfectly on the binary ridge pattern. Verifinger is currently one of the best minutiae extractors, according to fingerprint competitions [CFFM07].

3. Non-linear distortions are used to change the synthetic ridge pattern and to model the contact region of real fingers.

4. All the minutiae are removed, which are not in the contact region of the impression generator to avoid having minutiae points in empty regions.

5. Anguli creates multiple impressions of the fingerprints using the previously generated minutiae map. This is done by using different contact regions for the fingerprint and a randomized noise model.

6. A refinement network is trained using the SimGAN framework described in Chapter 7.

7. The impressions are used for training the U-shaped Neural Network (NN) to find a mapping from a fingerprint image to a corresponding minutiae map.

8. During training on-the-fly augmentations as described in Chapter 7 are used to make the trained model invariant to different operations.

9. An extraction algorithm is used as described in Chapter 3 to get a minutiae list from the output of the model on a particular fingerprint image. This includes extracting the angle and orientation of every minutia point.

10. For evaluation purposes the matching score of the trained model on specific fingerprint datasets is compared to state-of-the-art fingerprint minutiae extraction algorithms using the same minutiae matcher BOZORTH [WGT+04] in Chapter 9.

1.3 Organization of the remaining Thesis

The rest of this thesis is organized as follows, Chapter 2 lists related work. Chapter 3 is a summary on the discipline of biometric authentication using fingerprints. In Chapter 4 the theory of deep learning is described and the methodology used throughout this thesis is presented. Chapter 5 describes the experimental setup, the languages and frameworks used to program and test the algorithm developed in this thesis. The algorithm itself is described in Chapter 6. The data used to train the algorithm is explained in Chapter 7. The architecture of the neural networks is visualized in Chapter 8. Moreover, the evaluations are visualized in Chapter 9. Finally, Chapter 10 draws conclusions.

CHAPTER 2

Related Work

The main focus of this work is to improve the accuracy of fingerprint matcher by improving on the current state-of-the-art in minutiae extraction by using deep NNs. Another focus is the generation of a dataset, which allows a trained minutiae extractor to generalize to real datasets. In this chapter related works to these focus points are discussed, reviewed and compared to this work.

2.1 Literature study

In this section the related literature and a summary is shown. For better readability the works are separated into 5 categories, namely automatic fingerprint matching, deep learning, synthetic fingerprint generation, generative adversarial networks and binary segmentation.

Automatic fingerprint matching algorithms have been in the focus of biometric research since 1892 [Gal92]. Approaches in literature use minutiae matching as preferred way to match fingerprints [MMJP09]. Extracting minutiae from binary images is done by applying thinning and then a 3×3 filter over the image [HWJ98, Got12]. However the binarization of fingerprint images is susceptible to image quality [HWJ98]. To work reliably in the presence of creases and bruises, excessive dryness and sweat [CCG07] image enhancement algorithms were developed [HWJ98, CCG07, Got12, Ver10] and still form the current state-of-the-art in fingerprint matching competitions [CFFM07].

With [SPVS14] there is recent research in using neural networks for the minutiae extraction problem. Recently there has been some research in using Convolutional Neural Networks for fingerprint feature extraction in a forensic setting [TGF16].

Deep Learning has recently been the focus of many works in computer vision [ABGM14, HZRS16, MSM16, TGF16, RFB15, SIVA17]. With the rise of GPUs as a general computing platform [NBGS08] and the recent generation of immense datasets [Kit14], deep

learning has been applied to a variety of similar problems in computer vision like medical image segmentation [RFB15, MNA16], natural image segmentation [LLS15], fingerprint feature extraction [TGF16] and iris recognition [GJ16].

Synthetic fingerprint generation is also closely related to this work. Deep learning needs approximately 5.000 labeled samples per class to work well [GBC16]. In the context of minutiae extraction the concept of a class is not well defined though. Chapter 7 is devoted to the artificial generation of a realistic dataset. To my best knowledge the works in [Ans11] and [CMM04] are the only works related to the generation and enhancement of simulated fingerprints. The fingerprint generator described in [CMM04] is also used in fingerprint competitions such as the FVC 2000 [MMC+02a].

Generative Adversarial Networks (GAN) are used to find suitable, real looking refinements for syntehtic fingerprint images. GANs were first pioneered in [GPAM+14]. Since the proposal of adversarial training, there have been advances in stabilizing training like minibatch discrimination [SGZ+16] or energy-based training [ZML16]. The closest work based in the area of adversarial training to this thesis uses local adversarial loss to refine eye and hand images [SPT+16]. There is also a work, which generates real looking synthetic images from a ground truth [CGM+17]. Still, stable GAN training is an active research area [AB17].

Binary Segmentation on fingerprints is addressed by reformulating the minutiae extraction problem into a binary segmentation problem. U-shaped Convolutional Neural Networks (CNN) are used to solve such problems in medical application [RFB15, MNA16, ÇAL+16]. The architecture used in this work is most closely related to the architecture used for biomedical image segmentation in [DVC+16]. The main difference is, that Wide Residual Blocks as explained in Chapter **??** are used instead of Bottleneck Blocks for their superior performance.

2.2 Analysis of Literature

The problem with current solutions to the minutiae extraction problem is, that the results are unsatisfying [TGF16]. Current approaches work with time consuming image enhancement routines [Ver10, WGT+04] and even then the images need to fulfill specific quality measures like dpi for extracting stable minutiae points [Got12].

There has been a shift away from enhancement routines to new methods like stacked denoising sparse autoencoders [SPVS14] or pre-trained CNNs [TGF16]. Both those approaches suffer from a shortage of labeled examples and use fingerprints annotated from experts. In [SPVS14], 258 images form the training set, while in [TGF16] 4463 fingerprint images are used. To combat the shortage of training data [SPVS14] works on patches and [TGF16] works by using a fine-tuning approach with a CNN. Both approaches have room for improvement with a maximum accuracy on the amount of detected minutiae of less than 0.60.

Another large part of this thesis is the generation of real looking fingerprints. There are only two works to the knowledge of the author, that try to do this [CMM04, Ans11]. In [CMM04] a commercial fingerprint generator is described, while [Ans11] describes a freely available one. While [CMM04] is the basis for [Ans11], however, some functionality was not implemented and their main focus was on generating a large scale datasets of fingerprint images and not to provide the most realistic fingerprint images possible. With this approach it is possible to generate 1 million fingerprint images in a matter of days [Ans11]. In [CMM04], the fingerprints generated have higher variations and more of the operations observed in nature, like non-linear distortions or morphological operations, than the fingerprints generated by Anguli [Ans11]. Still the problem with SFinge is, that it is possible to reliably distinguish them from real fingerprints even without domain knowledge [CMM04]. The fingerprints created by SFinge miss operations observed in the real world as shown in Chapter 7. Additionally, the authors of SFinge themselves have open issues with their approach [CMM04].

2.3 Comparison to Existing Solutions

Most similar to our work is [TGF16]. In [TGF16] a fine-tuned network is used as pre-processing and outputs a map of regions of interest. They use logistic regression and region pooling to find minutia points in those regions. Compared to [TGF16] the problem is reformulated as a binary segmentation task. This task is then solved by the network directly. Because synthetic data is used to train our model, it is possible to train the model from scratch to be especially suited for the minutiae extraction task.

Other solutions using neural networks like the sparse autoencoder used in [SPVS14] have considerably worse precision compared to the CNN approach [TGF16]. Solutions based on fingerprint enhancement [HWJ98, Got12, Ver10, WGT+07] suffer from similar problems of having unsatisfactory accuracy.

Compared to other solutions developing synthetic fingerprints, deep learning is used to learn, how to generate fingerprints indistinguishable from real fingerprints from data. The algorithm described in [CMM04] generates fingerprints, which have similar matching error rates as fingerprint images used in real datasets as shown in [MMC+02a]. However, due to the large differences to real fingerprints as shown in Chapter 7, they are still unsuitable for training a minutiae extraction network on their own. In contrast the approach developed in this thesis uses real datasets to find suitable refinements and therefore the number of different fingerprints observed is dependent on the real datasets provided. Competitive results are shown even with only 150 real annotated fingerprint images.

CHAPTER 3

Biometric Authentication using Fingerprints

Fingerprints are the most widely used and accepted form of biometric authentication [MMJP09]. This chapter is about fingerprints and their representations. Fingerprints are generally represented using ridges and valleys [Ash99] as is shown in Figure 3.1. The ridges and valleys of a fingerprint projected on a two dimensional surface form a ridge pattern. One instance image of a ridge pattern is called an impression of a fingerprint. In Figure 3.1 the whole ridge pattern of a single fingerprint is shown. Injuries such as superficial burns or cuts do not change the ridge structure. The ridge pattern is also duplicated by new growing skin [MMJP09].

Fingerprints are generally described using one of the following three levels of detail [MMJP09]:

1. The overall global ridge flow pattern.
2. Minutiae points.
3. Pores and local shape of ridges.

The three levels of description of a fingerprint are further described in Section 3.1. In Section 3.2, the current process of fingerprint feature extraction is explained in detail. The last section describes the process of using those features for a reliable identification.

3.1 Fingerprint Representations

Fingerprints are represented by ridges and valleys which form a ridge pattern. The characteristic information in those ridge patterns is described in one of three levels of

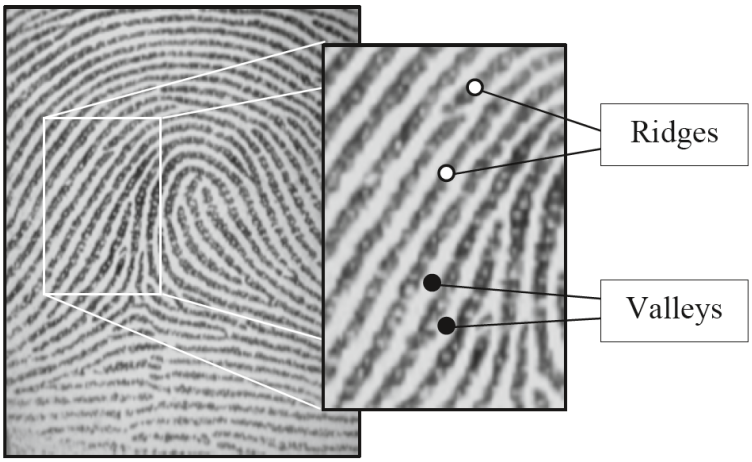


Figure 3.1: Example fingerprint with a detailed description of the ridges and valleys. Taken from [MMJP09]

detail. While no specific resolution is given for level one features, a 500 dpi resolution is required for level two features and 1000 dpi for level three features [MMJP09].

Level one features are regions, where the ridges assume distinctive shapes. These three regions are classified into three typologies, namely loop, delta and whorl [MMJP09]. The synthetic fingerprint generator Anguli [Ans11] was used to generate ridge patterns of all three patterns as shown in Figure 3.2. Various fingerprint matching algorithms pre-align fingerprints using level 1 features or dismiss fingerprints with different level 1 features. However, only using level 1 features is not distinctive for matching [MMJP09].

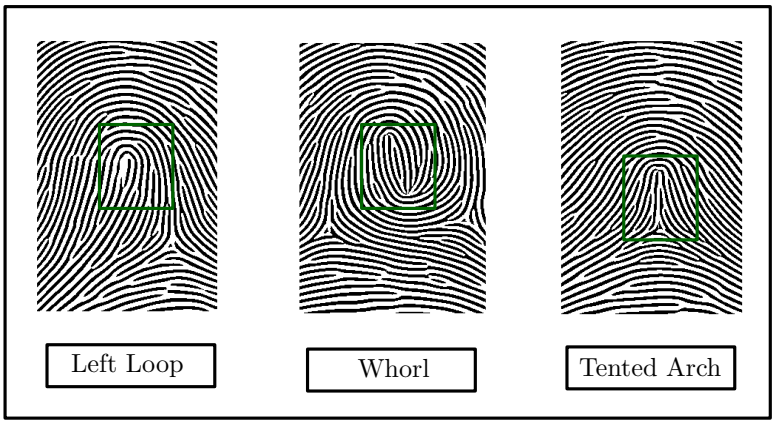


Figure 3.2: The three typologies of fingerprints are shown here.

The minutiae points are the level 2 features of a fingerprint. In the context of fingerprints, minutiae correspond to points which represent irregularities in ridges [MMJP09]. There

are over 150 such irregularity patterns [Moe71]. The most commonly found minutiae types are shown in Figure 3.3. All of those patterns can be reduced to combinations of the bifurcation and termination patterns. The FBI only distinguishes between ridge endings and bifurcations for its minutiae detection algorithm [McC04]. Moreover, an ending changes to a bifurcation and vice versa, if the values for valleys and ridges are reversed [MMJP09]. This observation makes finding endings and bifurcations a dual problem. The algorithm proposed in this thesis does not distinguish between different types of minutiae and just outputs the position, orientation and quality of a minutiae point.

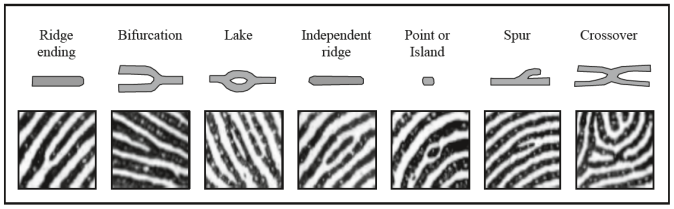


Figure 3.3: The seven most common minutiae types taken from [MMJP09]. Note that all the minutiae types can be reduced to a combination of endings and bifurcations. A Lake for example are just two nearby bifurcations in the opposite direction.

In practice level two features, also called minutiae, are most commonly used for fingerprint matching and are also accepted as proof of identity in the courts of law in almost all countries [MMJP09]. One of the main reasons for using minutiae landmarks is the observation, that minutiae points are stable over time [Gal92]. Even small wounds like cuts or burns heal in such a way that the minutiae information in a fingerprint is preserved. Additionally, even though there can be over 100 minutiae in a fingerprint, only 12-15 are needed to trust a fingerprint match with high confidence [MMJP09]. For the reliable detection of minutiae points a dpi of 500 is recommended [MMJP09].

Level 3 features consists of further finer details in corresponding fingerprint images [MMJP09]. All fine ridge details like shape, width and edge contour belong to level 3. By zooming into real fingerprints, the pores in the fingerprint ridges become visible as shown in Figure 3.4, given a resolution of at least 1000 dpi. It has been reported that 20-40 pores are sufficient to claim the identity of an individual [Ash99]. Additionally, the amount of pores along a centimeter of ridge varies between 9 to 18. Therefore, only 5 centimeters of overlapping ridges are needed for reliable identification using level 3 features. This is the reason why automatic matching techniques cannot use level three features since their reliable detection requires a resolution of at least 1.000 dpi [MMJP09]. The fingerprint databases described in Chapter 5 do not provide the resolution for level three feature extraction.

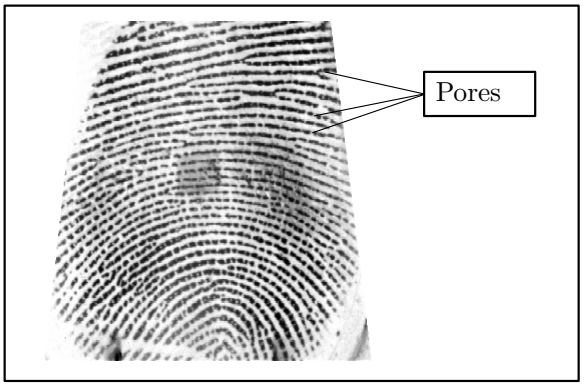


Figure 3.4: Shows the location of the pores in the ridges of the fingerprint. The fingerprint is taken from the UareU dataset. This is the only dataset, where pores are actually visible, which was collected as part of this thesis.

3.2 Fingerprint Minutiae Extraction

Most automatic fingerprint matchers use minutiae as their feature representation [MMJP09]. Therefore, minutiae detection is an extremely important task in biometric authentication and a substantial amount of research has been committed to finding more accurate algorithms for this problem [TGF16]. Still, these algorithms rely on fingerprint images with a dpi of at least 500, stable illumination and large overlapping contact regions. In Figure 3.5 the result of a minutiae extraction process is shown. Approaches exist, where the image is converted into a binary image [MMJP09]. Those methods benefit from image enhancement done a priori [HWJ98]. Therefore, image enhancement algorithms have been developed [HWJ98, CCG07, Got12] and are still the best performing methods according to fingerprint competitions [Ver10]. Given a binary image, the detection of minutiae points is done using thinning [ACL81]. This means shrinking the ridges until they are only a single pixel thick. Different algorithms have been proposed to find a mapping from a thinned image to minutiae points [MMJP09]. The current state-of-the-art uses a graph based approach using principal curves to follow ridges and find endings and bifurcations [ZMZ11].

Recently, authors have proposed methods, like the one described in this thesis, which work directly on the gray-scale images [TGF16]. The motivation for this approach is, that a significant amount of information is lost in the binarization process even with image enhancement steps [MMJP09]. Recent methods use neural networks to find a mapping between gray-scale images and minutiae positions, by learning a mapping from data [SPVS14, TGF16]. They did not evaluate on any of the fingerprint datasets that is used in this thesis and they did not provide a SDK to compare results. Additionally, access to the minutiae extraction dataset used in their work for evaluation has not been granted.

(a) (b)

Figure 3.5: The original synthetic fingerprint shown in (a). In (b) the minutiae detail is extracted with their angles using Verifinger [Ver10]

3.3 Fingerprint Matching

Fingerprint matching means comparing two fingerprints to each other and determining either a degree of similarity (a score between 0 and 1) or a binary decision of match or no match. The most common form of fingerprint matching is minutiae matching and is in essence a point cloud matching algorithm [MMJP09]. The main difficulties in fingerprint matching, as described in [MMJP09], are summarized below:

1. Linear transformations: The fingerprint may be rotated and displaced in respect to the sensor. This may result in only a partial overlap between two impressions of the same finger.

2. Non-linear distortion: By mapping the three dimensional finger over a two dimensional sensing area results in non-linear distortions. Therefore it is not possible to match fingerprints as rigid patterns.

3. Pressure, noise and skin conditions: Because of varying pressure of the fingerprint onto the sensor some parts of the finger might not be seen by the sensor or only modeled with lower pixel intensity values. Additionally the sensor might be dirty and introduce noise into the fingerprint impression. Various skin conditions like sweat, skin disease and humidity in air have profound impact on the extracted impression.

4. Feature extraction errors: In this work a new minutiae extractor is proposed. This is done, because current solutions are not perfect. Additionally, it is never claimed, that the approach in this work is perfect and therefore a matching algorithm has to account for errors in the feature extraction stage.

Given a template fingerprint T and a new input fingerprint I there are multiple ways to calculate a similarity score [MMJP09]. The minutiae matcher BOZORTH3 [WGT+07] is used for matching the minutiae of each impression of a fingerprint to each other in this thesis. BOZORTH3 calculates the matches using the similarity score given in (3.1), where k, n and m denote the matched minutiae and the minutiae points found in T and I, respectively. The algorithm finds matches using the following steps:

1. Create intra-fingerprint minutiae comparison tables, by computing the differences in rotation and translation for every minutia in T to every minutia in I.
2. The number of matches k is found by trying out every minutia point in T at every minutia point of I, as a base point and counting the number of minutiae that fit in I, using the minutiae tables obtained in step 1. Small deviations in rotation and translation are allowed to still count as a match [WGT+07].
3. The similarity score is calculated as in (3.1), given the maximum minutiae found in step 2.

$$\text{score} = \frac{k}{(n+m)/2} \tag{3.1}$$

The final output of BOZORTH3 is a similarity score. To use this score for matching a threshold is needed. This threshold determines the value at which to trust a match. To determine the threshold the Equal Error Rate (EER) on a sample set is used. The EER is defined as the error rate given by the threshold, where the rate of false matches is equal to the rate of true mismatches.

CHAPTER 4

Deep Learning

Deep Learning is the scientific field of using deep artificial neural networks for specific applications [GBC16]. Neural Networks (NN) have been part of scientific research since the introduction of the perceptron in [Ros58]. Even CNNs and Recurrent Neural Networks (RNN) have been part of research for decades [Fuk79, LBBH98]. But without the enormous amount of parameters used in recent networks [HZRS16, SVI+16, SIVA17] and the ability to train those networks [BSF94], CNNs performed below expectations in the early days of research [GB10]. However, with the rise of GPU - programming [NBGS08] and easily accessible datasets, e.g. ImageNet [KSH12], due to the recent data exposition [Kit14] and advances in training deep neural networks, CNNs are able to shatter most records set by other state-of-the-art methods in fields of computer vision, such as object classification [SLJ+15, HZRS16], medical image segmentation [RFB15] or natural language processing [ZJZF16].

Design choices in deep learning are an open topic due to the amount of time required to train large models and the amount of hyperparameters to consider for evaluation [MSM16]. One work evaluated design choices in NNs [MSM16], but this list is incomplete and only tested on a single dataset. Most of the recent research in improving accuracy for deep learning problems has gone into finding ways to reduce the parameter requirement for a given accuracy [SIVA17]. Then, by increasing the amount of parameters, additional accuracy is gained [HZRS16, SIVA17]. Following this intuition, many ideas have achieved accuracy improvements for a specific parameter quantity on a specific dataset, like residual connections [HZRS16], group convolutions [XGD+16] and seperable convolutions [Cho16].

Because of the cost of annotating images, instead of generating them [Ans11], research has gone into finding ways to need less images to perform well, or on the other hand generating real looking examples [GPAM+14]. One of those methods is to use fine-tuning, which has been applied to various problems in computer vision with little annotated training examples like plants [RCC15] or chimpanzees [FRS+16].

4.1 Introduction

In this section an overview on the history of deep neural networks is given. Additionally, the theory used to explain the success of deep neural architectures is explained. Finally, a description of the vanishing and exploding gradient problem is given.

4.1.1 Historical Evolution of Convolutional Neural Networks

One focus point in deep learning research is finding new architectures [LBBH98, SZ14, HZRS16, SLJ+15], that work for specific use cases. In this section the main advancements in deep learning architecture design are highlighted.

The first modern CNNs was introduced by Lecun in [LBBH98]. This network model already had convolutional layers and max pooling layers stacked together with two fully connected layers as a final classifier. This method yielded state-of-the-art performance on the MNIST dataset at the time. The MNIST dataset contains handwritten digit images. However, the algorithm did not scale well, because of the processing power required to train deeper models and the problems associated with actually training those deeper models [BSF94].

Only recently with the introduction of the AlexNet [KSH12] in 2012 and its victory at the Imagenet competition, deep learning gained widespread attention. This was the beginning of training large networks on GPUs to achieve better performance. Their model consisted of 60 million parameters, which takes months to train on a single CPU [KSH12]. Additionally, they used the Rectified Linear Unit (ReLU) to combat the vanishing gradient problem, plaguing previous deep models [BSF94]. Moreover, to combat overfitting, they already deployed an early form of dropout proposed in [HSK+12], where half the neurons in all layers were omitted. Eventually the idea proposed in [HSK+12] was further developed with the introduction of the dropout layer [SHK+14], which is heavily used in modern architectures [DVC+16, ZK16].

The next step in deep learning was the VGG net [SZ14], which is still widely used as a base network today [FRS+16, RCC15] and is built into deep learning frameworks like keras [Cho15] or caffe [JSD+14]. This network signals the rise of actual deep networks by using 11 to 19 layers with more than 100 Million parameters . However since the VGG net, the focus has shifted from increasing the number of parameters to changing the network architecture [SIVA17].

Building on top of the theoretic work of [ABGM14] and the rise of Network in Network architectures, the inception architecture was formed in [SLJ+15]. The network was not split up in layers anymore, but in layer blocks, containing convolutional layers with different paths through one layer block. The U-net-v1 architecture proposed in Chapter 8 uses these so called inception blocks. The inception architecture was developed further in [SVI+16]. Batch normalization layers [IS15] are used instead of dropout to combat overfitting. Since its introduction in [IS15], batch normalization is used in [ZK16, SVI+16, SIVA17, HZRS16, MSM16]. Additionally, batch normalization

helps not only against overfitting, but also against the vanishing gradient problem by normalizing activations in the network itself and improving gradient flow through the network. The introduction of batch normalization made training deeper models possible, like the ones proposed in [SLJ+15, HZRS16] with more than 40 and 100 layers, respectively. In [SVI+16], several design principles were introduced to improve the performance of the model for a given amount of parameters. These principles include using spatial aggregations or increasing width and depth at the same time for the best performance improvements given a specific number of parameters.

Another milestone in deep learning history is the usage of residual connections, also called skip connections, in [HZRS16]. Residual connections skip the subsequent layer and connect to one of the following layers directly. This improves the gradient flow through the network by providing direct connections were the gradient can flow faster. Better gradient flow not only accelerates training, but also helps finding a better minimum [HZRS16].

This concept was further developed with long skip connections for U-shaped networks in [RFB15] to improve the state-of-the-art in medical image segmentation. The importance and effect on the gradient was researched in [DVC+16]. In that paper, the gradual improvement of the weights with the usage of residual connections is demonstrated.

The current state-of-the-art in deep learning incorporates those skip connections. The current iteration of the inception architecture uses skip connections to reduce the number of parameters and training time, while maintaining state-of-the-art accuracy [SIVA17]. Additionally, by making heavy usage of residual connections DenseNet [HLW16] achieved a new state-of-the-art on the CIFAR-10 dataset. DenseNet uses dense blocks, where every layer is connected to every following layer leading to a linearly increasing number of feature maps in a given layer. That architecture is also used in U-shaped form for the semantic segmentation problem [JDV+16].

4.1.2 Theory behind Deep Learning

With the recent success of deep learning in domains of computer vision [SLJ+15, FRS+16, TGF16], it is interesting to look at the approach from a theoretical point of view. Researchers tried to prove the practical success of deep nets theoretically [ABGM14, KTR16]. Starting point for such works is [KS06], who proved that learning neural networks is NP-hard even for random input and shallow network architectures. This means, by adding a single hidden layer the problem on arbitrary input is already NP-hard. Still it is interesting to look at the resulting functions locally to find reasons on why deep learning works the way it does and hopefully improve on current methods.

Even though the problem is NP-hard researchers tried to find solutions by restricting the input space [MPCB14]. Therein, the complexity of the functions computable is studied in terms of the number of linear regions in the model. Another theoretical point of research is on information flow in a neural network [SVI+16]. The entropy changes between layers and a lower bound on the encoded information has been defined in [KTR16].

Another theoretic work [ABGM14] proved, that the probability distribution of any dataset can be represented by a large, very sparse deep neural network. This prompted the shift away from dense, shallow networks in [SZ14] to the sparse, deep networks with 101 layers in [HZRS16] with better accuracy for the same number of parameters on the ImageNet dataset than previous state-of-the-art methods [SZ14]. However, width is still important as shown in [ZK16]. In [SVI+16], it is explained that depth and width are a tradeoff and the best performance is achieved by increasing both equally.

4.1.3 Vanishing / Exploding Gradient Problem

The best known problem for training deep neural networks is the vanishing / exploding gradient problem [GB10]. Those phenomena occur, when the gradient gets smaller per layer until zero is reached in an intermediate layer of the network. In contrast, the gradient would explode, if it got larger in every layer until it reaches infinity and updates the weights to undefined values. This leads to layers being left untrained or wrongly trained, which defeats the purpose of using deep neural networks in the first place [DVC+16]. This problem was particularly amplified by using sigmoid as the activation function, which clamps the input space between zero and one [KSH12].

In [HZRS16] the problem is addressed by carefully initializing the model weights and to use a so called warm-up phase. This means using a low (0.001 in [HZRS16]) starting learning rate for the first few epochs to initialize the model. Afterwards, the learning rate is set to a higher learning rate which is used for the remaining training (0.1 in [HZRS16]).

Another part of the solution is to use activation functions, which do not clamp the input space to an arbitrary range of values [KSH12]. This is one of the reasons, why Regularized Linear Units (ReLU) are used as activation function in recent works as defined in (4.1). Also, ReLU is computationally inexpensive compared to other popular functions like sigmoid or tanh [KSH12]:

$$y = \max(0, x) \tag{4.1}$$

The usage of skip connections is also motivated by the idea to combat the vanishing gradients [HZRS16]. Skip connections are used to improve the gradient flow through the network as explored in detail in [DVC+16]. Therein it is shown that the weights are learned throughout the network instead of only at parts of the network, by analyzing the weight distribution of the network.

4.2 Neural Network Performance Tuning

In this section, various pitfalls and possibilities to squeeze out more performance of a given architecture are explained.

4.2.1 Weight Initialization

Weight initialization is part of deep learning research [GB10]. Wrong initialization leads to longer convergence times or failure to converge at all. Therefore a weight initialization algorithm is proposed in [GB10]. Therein the weights are initialized as samples from a normalized Gaussian distribution depending on the amount of feature maps in that particular layer. In [HZRS15], that approach was modified to better fit the non-linearities used in todays network models, like the ReLU function. The models introduced in this work also use ReLU as their activation function and therefore the initialization scheme proposed in [HZRS15] is used.

4.2.2 Optimizers

Different optimizers have been proposed in literature [DHS11, KB14, RHW85]. Adam [KB14], RMSProp [TH12] and Stochastic Gradient Descent (SGD) [RHW85] are compared in respect to their performance on the minutiae extraction task. Those three algorithms were selected based on their prevalence in related binary segmentation works [DVC$^+$16, RFB15].

SGD is a first order optimization technique, which in essence calculates the gradient and moves along the gradient using a learning rate. Since the inception of stochastic gradient descent a modification in using momentum [Nes83] has been suggested.

Adam [KB14] is another first order optimization technique, which uses order one and order two momentum to improve weight updates and to converge faster. Adam does more calculation than SGD and has therefore higher computational costs and memory requirements.

For most of my networks, Adam worked best because of its speed in optimization as shown in Chapter 9. Adversarial training is done using the SGD algorithm, because of the clearer weight updates, which is important in understanding the changes made to the model after each update.

A problem concerning both, Adam and SGD, is choosing a learning rate. This is a compromise between speed of the algorithm and actually converging to a good minimum. Additionally, the learning rate is dependent on the content of the dataset, the dataset size and the architecture of the model. Smaller models can make use of higher learning rates [HZRS16]. For models with more than 10 million parameters a startup phase can severely reduce training time. This means starting with a large learning rate and then after a few epochs train with a low learning rate [HZRS16].

One policy for choosing the learning rate is to chose a large initial learning rate and after the validation loss stops decreasing, reduce the learning rate to a tenth of what it previously was [MSM16]. Another policy is to use learning rate decay [MSM16]. Learning rate decay reduces the learning rate every epoch by a specified amount, which depends on the epoch as given by:

$$lr = L_0 \cdot \frac{1}{(1 + \gamma \cdot i)} \tag{4.2}$$

,where lr, L_0, γ and i are the resulting learning rate, initial learning rate, decay parameter and the number of the current iteration, respectively.

Cyclic learning rates [Smi17] is another policy for selecting the learning rate. Here the learning rate is increased and decreased according to a lower and upper bound in a triangular pattern. The idea is to not only decrease the learning rate as the training prolongs, but to periodically decrease and increase the learning rate. For this reason the following equations are used :

$$step = |\frac{epoch}{stepsize} - 2 \cdot \lfloor 1 + \frac{epoch}{2 \cdot stepsize} \rfloor + 1| \tag{4.3}$$

$$lr = base_{lr} + (max_{lr} - base_{lr}) \cdot max(0, (1 - step)) \tag{4.4}$$

where $epoch$, $stepsize$, max_{lr} and $base_{lr}$ denote the current epoch, half the length of the cycle, the maximum learning rate and the minimum or original learning rate, respectively. A graphical comparison of setting a cyclic learning rate compared to step learning or learning rate decay is illustrated in Figure 4.1.

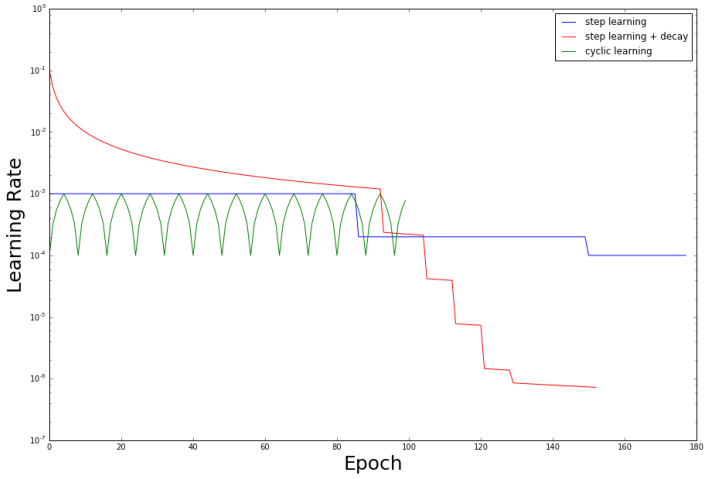


Figure 4.1: Comparison of the learning rate scheduling techniques.

With this approach three hyperparameters are required to be set, namely $stepsize$, max_{lr} and $base_{lr}$ instead of just a learning rate, as with the other approaches. However, according to [Smi17], there exist a semi-supervised ways to set max_{lr} and $base_{lr}$. This is done by running the model for several epochs with a linearly increasing value for the learning rate. The resulting plot of the accuracy is then used to determine working values

for the base learning rate and the maximum learning rate. Useful values for the *stepsize* are $2 - 10$ according to experiments conducted in [Smi17].

A comparison of the performance required on a fingerprint dataset using the WRN as described in Chapter 8 is shown in Figure 4.2. Therein is shown, that step learning produces the lowest loss and therefore step learning is used throughout this work.

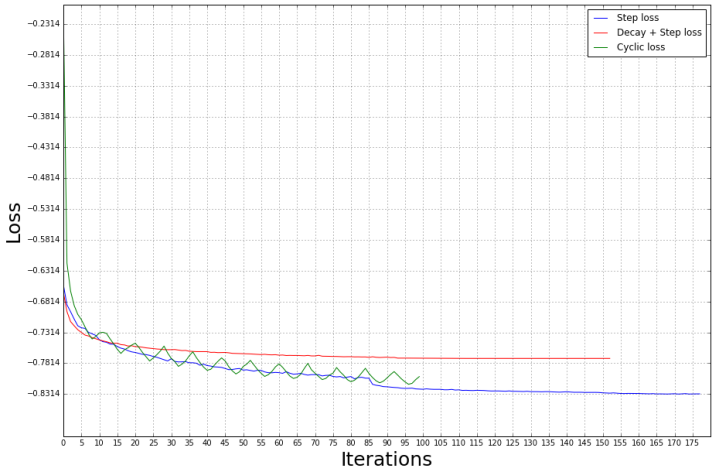


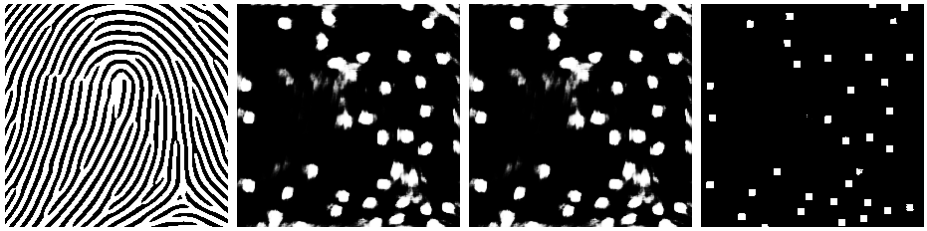
Figure 4.2: Comparison of the performance obtained with various learning rate scheduling techniques with WRN as described in Chapter 8 . The validation loss is used as comparison metric.

4.2.3 Loss Functions

Two loss functions are used in this thesis, namely binary cross entropy (or log loss) and dice loss (or negative F-Score), due to their successful usage in related binary segmentation tasks [RFB15, DVC+16]. An illustration of the different outputs produced by the models is shown in Figure 4.3.

Dice Loss

Dice loss is a smoothed version of the negative dice coefficient (F-Score). Dice Loss is used in related works in biomedical image segmentation [DVC+16]. Using dice loss as in (4.5), where y_{pred} is the prediction made by the model and y_{cor} is the ground truth. The advantage of using dice loss instead of binary cross entropy is that dice loss produces visually cleaner images as shown in Figure 4.3. Therein the images are almost binary. This means that no threshold is needed to post-process the images. This result has also been reported in [DVC+16].



(a) (b) (c) (d)

Figure 4.3: Shows the output of the minutiae extraction network on a sample input ridge pattern in (a). (b,c) show the output using the binary cross entropy loss, where in (c) a threshold of 0.5 is applied to generate a binary output. In (d) dice loss is used. Here, the output is nearly binary even without post processing

$$loss = -\frac{2 \cdot (y_{pred} \cdot y_{cor})}{y_{pred} + y_{cor}} \tag{4.5}$$

Binary Cross Entropy

Another choice used in related works is binary cross entropy [DVC+16]. Binary cross entropy is defined in (4.6), where y_i, p_i and N denote the i-th ground truth, the i-th prediction and the set of valid pixel indices in the image. The final loss output is the average log-loss for every sample in the image. The output of networks using this objective are noisier than the ones using dice loss as shown in Figure 4.3. Therefore a threshold is used to produce a useful output. Additionally, training visualizations using this objective also validate on a rounded dice coefficient by using 0.5 as a threshold.

$$loss = -\frac{1}{|N|} \sum_{(i,j) \in N} (y_{ij} \log(p_{ij}) + (1 - y_{ij}) \log(1 - p_{ij})) \tag{4.6}$$

4.3 Special Deep Learning Architectures

In this section, special deep learning architectures and techniques are explained, like the autoencoder, fine-tuning and GANs.

4.3.1 One-class Learning using Autoencoder

Autoencoders learn a representation of the training data [GBC16]. This can be used to tackle problems with only one class and the objective is to differentiate this class from any other class. An example autoencoder for 4 inputs is shown in Figure 4.4. The objective of an autoenccoder is to reconstruct a given class as well as possible given samples from this class. The reconstruction error is then used to discriminate between the class and any other class. Using this insight, only samples of a single class are needed for training.

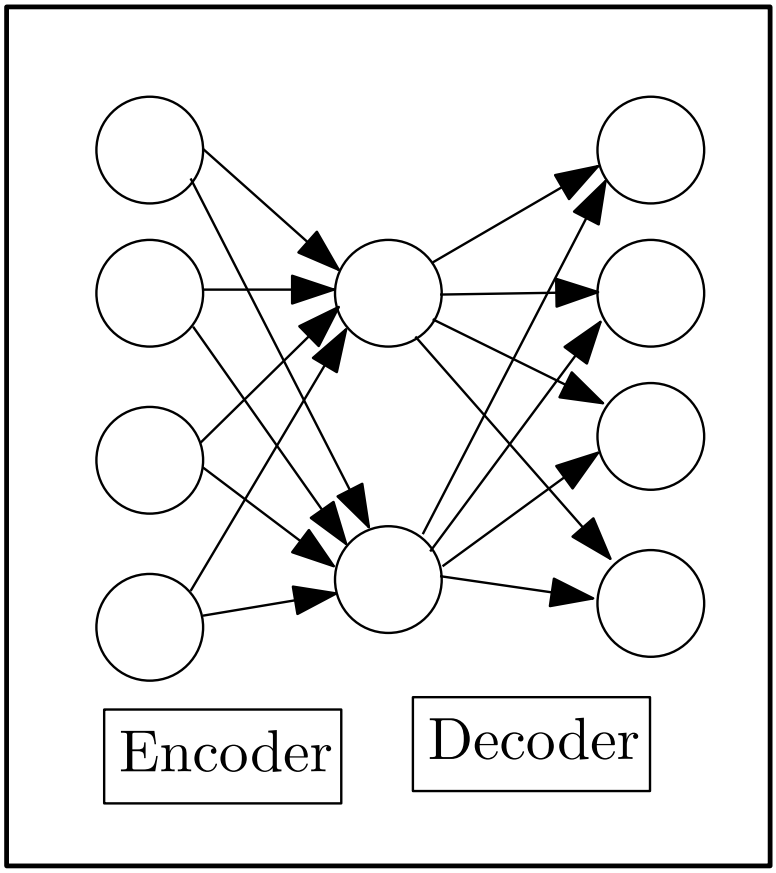


Figure 4.4: A sample autoencoder for 4 inputs

This approach has already been applied to the minutiae extraction problem in [SPVS14], by using two stacked denoising autoencoders [VLL+10] to learn one class each. One autoencoder learns the class of an image batch showing minutiae in it and the other one learns the class of an image batch showing no minutiae in it. Then in order to find batches showing minutiae in a new fingerprint, the reconstruction is compared between the two autoencoders and classified accordingly [SPVS14].

4.3.2 Fine-tuning

Fine tuning is used to reuse learned features on similar problems to the problem at hand [FRS+16, RCC15]. CNNs learn very general features at the initial layers, like corners [GBC16]. In fine tuning, weights learned on a general task are saved and used as a basis for new tasks [TGF16].

In [FRS+16] two base networks for the chimpanzee re-identification task are tested. One network was trained on ImageNet, while the other one was trained on the VGG Faces, which is a human faces dataset. Currently, in literature the chimpanzee re-identification

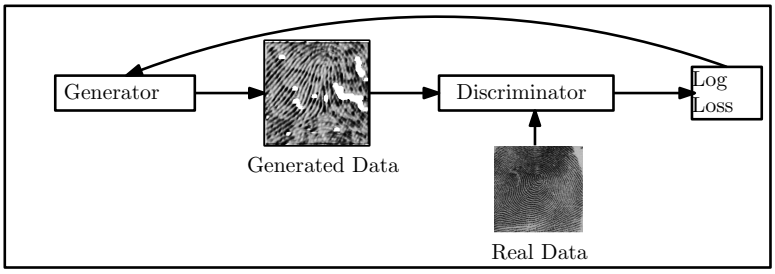


Figure 4.5: Illustration of a Generative Adversarial Network for fingerprint generation.

task is closely related to the human re-identification task, because of the similarity in features. However, a pre-trained network on the more general model outperforms the more specialized one [FRS+16]. This result illustrates the generalization capability of the base network. Networks trained on ImageNet [KSH12] are used for various computer vision problems in literature and in practice [FRS+16, TGF16].

4.3.3 Generative Adversarial Networks

GANs [GPAM+14] are derived from game theory to learn generative models [SGZ+16]. A GAN consists of a generator and a discriminator network as illustrated in Figure 4.5.

The goal of a GAN is to train a generator model that creates realistically looking images from random noise. To achieve this goal, a discriminator network is trained to distinguish between synthetic and real data. This discriminator network is continuously improved based on the output of the generator, while the generator looks for a distribution that is accepted by the current discriminator. This results in a min-max optimization problem, where the optimum is a Nash equilibrium [SGZ+16]. This is a very difficult problem, where optimization algorithms only exist for specialized cases without any feasible way to apply them to the GAN problem [SGZ+16]. The cost functions are non-convex, the parameters continuous and the parameter space is extremely high-dimensional [SGZ+16].

In literature [GPAM+14, SPT+16], SGD is used to optimize both the discriminator and the generator in turn. However, improving the generator can increase the cost of the discriminator and vice versa. This makes GAN training highly unstable and very tricky to get right [AB17]. One such instability is the Helvetica situation where all the input data points only lead to a single output [GPAM+14], which in turn is moved around by the generator until this point is not classified correctly by the discriminator. Then the discriminator learns to classify this point correctly. This process is then continued infinitely. Another problem is that the training is known to produce artifacts in the generated images [SGZ+16]. This problem has been addressed in other works, by adding minibatch discrimination [SGZ+16], historical averaging [SPT+16] or by energy-based training [ZML16].

Another problem is that there is no good metric for the realness of data [SGZ+16]. Most work in literature use human evaluators to determine the realness of the data [DCF+15].

However, even humans are prone to changes in their state of mind and are not a reliable measurement [SGZ+16]. For the ImageNet dataset, there is a solution by using the expected output of the inception model trained on ImageNet to quantify the quality of the resulting samples. This score is called inception score [SGZ+16] and is further described in Chapter 5.

Energy Based Network

Energy Based Generative Adversarial Networks [ZML16] (EBGAN) use an energy function as discriminator. Low energy is associated with real data and high energy with fake data. The energy function is modeled using an autoencoder, where the energy corresponds to the reconstruction error. This relates to the problem of one-class learning, where an autoencoder finds a condensed representation of the data and fails at reconstructing other examples [GBC16]. Following the theory, an autoencoder learns to encode the distribution of the training set and therefore produces larger errors for samples from another distribution.

The main advantage of using such an approach is that the discriminator training does not depend on the generator in the training process. Therefore the discriminator is trained prior to the generator and is used as a cost function. This leads to a more streamlined training process and less dependence on careful initialization of the hyperparameters. After the discriminator has finished training, the generator only learns to construct samples with low energy, by using the discriminator as a loss function.

Adversarial Training

The problem with the GAN approach is, that it is hard to generate reliable annotations for the generated data. This is especially troublesome, where annotating is time consuming like in the minutiae extraction problem [CMM04]. For this reason, the adversarial training approach is adapted to fit those applications. In [SPT+16] a refinement network is proposed based on simulated data to generate real looking examples with ground truth. The simulated data is constructed with existing methods, where the corresponding ground truth is supplied freely. The main challenge hereby is the regularization. The network has to be regularized in such a way, that the refinements learned are meaningful, while still preserving the annotation data.

Another interesting approach using adversarial learning is to generate real looking synthetic data out of the corresponding ground truth as shown in [CGM+17]. Here retina images are generated with a binary segmentation as annotation data. A neural network used for binary segmentation of retina images is supplied as regularizer. The generator is forced to generate new data, while deviations to the ground truth are discouraged by regularizing using the binary segmentation network on the generated image and comparing it to the ground truth. The main challenge with this approach is, that the resulting images depend heavily on the used binary segmentation network.

CHAPTER 5

Setup

This chapter gives an overview over the experimental setup used to run the experiments for this thesis. The whole process of testing the minutiae extraction problem is complicated. Multiple programming languages with various frameworks are used to train and evaluate the neural network models. Additionally parts of the programs are run on a GPU and other parts are run simultaneously on the CPU.

Evaluating the algorithm is a non trivial task. There are multiple components to be evaluated, like the refinement network and the minutiae extraction network. Then the metric of choice makes a vast difference and depends on the use-case. For example, the accuracy of detected minutiae might be better for one neural network, but the matching score might be worse, because the detected minutiae might be more stable.

The programming languages and frameworks used in this work are listed in Section 1. To learn the models proposed in this thesis specialized CUDA-servers are essential to speed up computations. The specifications of those machines is given in Section 2. A description of the datasets created and collected for this work is given in Section 3. Finally the evaluation metrics for the various parts of the algorithms are described in Section 4.

5.1 Software

Python is used for everything related to machine learning, like pre-processing the data and training the models. C++ is then used, when other fingerprint related Software Development Kits (SDKs) are needed, like the Verifinger SDK [Ver10] for fingerprint matching and matching-based visualization.

5.1.1 Python

Python [Ros95] is used as a tool for rapid prototyping. Python is an interpreted language, which enables the possibility of changing the code, while the program runs. This enables rapid tryouts of different tasks with a loaded model, without specifying this in advance and loading the model all over again. Using a higher level language as opposed to a more lower level one helps me on focusing on the actual challenging part of the thesis. This means finding the right architecture of the model and fine tuning the model to gain better performance.

Additionally, Python has a very rich deep learning based toolbox, based on their machine learning platform scikit-learn [PVG+11]. Pre-defined functionality for splitting data into training and testing sets, shuffling or cross-validation are built into this framework. All visualizations related to deep learning, which includes weight, output and performance visualizations were made using the matplotlib [Hun07] framework of Python.

Theano

Theano [ARAA+16] is primarily a mathematical framework to compute symbolic math expressions. For the use-case of deep learning some additional features were added, like gradient calculations and convolutions. Theano gives the user full control over the calculations performed by the neural network with performance being their primary goal. Theano provides the gradient and is used to simplify the creation of functions and variables by using Tensors. Theano works seamlessly on GPU. This means no additional code has to be written to train or evaluate the neural network on a GPU as opposed to a cpu. Theano also works in Windows, which helped developing the algorithms on a local Windows computer, before using the CUDA-Servers for final evaluations, which runs under Linux. This is the reason for using Theano as opposed to Tensorflow [AAB+15].

Keras

For rapid prototyping, Keras [Cho15] was used. Keras is a wrapper for Theano [ARAA+16] or Tensorflow [AAB+15] and is used to hide repetitive math definitions from the user, while still allowing to do those definitions if needed. For example, the dice loss equation specified in (4.5) was programmed using Keras. Keras works with both, Theano and Tensorflow tensors, and is not restrictive on the functionality of those frameworks.

In Keras the focus is more on building a working architecture and testing specific combinations of parameters, than on typing math equations. Keras also provides models pre-trained on ImageNet for anyone to use and fine tune. Keras allows specifying new mathematic equations using the underlying framework directly.

A contribution to the source code of Keras has been made as part of the thesis [Cho15].

5.1.2 C++

For the evaluation of the minutiae extraction algorithm C++ was used. The minutiae matcher BOZORTH [WGT+04] and the minutiae extraction algorithms Verifinger [Ver10] and MINDTCT [WGT+04] are all written in C++. To natively use those algorithms, C++ was selected to write an evaluation software. The minutiae extraction algorithm, written in Python, outputs a binary file containing the minutiae list for each fingerprint. This is then read and used in the C++ programs for the fingerprint matching evaluation. Therefore the data transfer between C++ and Python is done using files.

5.2 Hardware

For the experiments described in this thesis access to CUDA resources was required. For this reason, access to the servers located at the AIT and at the TU Wien was granted to me. The configurations of those machines are described in this section.

5.2.1 TU Wien

The TU Wien supplied the Ralph machine for this thesis, with the following specification:

CPU	1x Intel Core i5-4690 (4 cores @ 3.5 GHz)
RAM	16 GB (1600MHz, DDR3)
GPU	1x Nvidia GTX 980 (2048 CUDA-Cores @ 1126 MHz, 4GB GDDR5)
OS	Ubuntu 14.04 LTS

Table 5.1: Specification of the Ralph machine.

5.2.2 Austrian Institute of Technology

The Austrian Institute of Technology (AIT) supplied two additional, identical servers, which were used for teh thesis, with the following specifications:

CPU	1x Intel Core i7-6850K 6 Cores @ 3.60 GHz
RAM	64 GB (2133 MHz, DDR3)
GPU	2x Nvidia Titan X (3072 CUDA-Cores @ 1000 MHz, 12 GB GDDR5)
OS	Ubuntu 16.04 LTS

Table 5.2: Specification of the CUDA Server @ AIT.

5.3 Datasets

In this section, the collected datasets and the training dataset is described. To allow the minutiae extraction model to perform well, it is important that the training set is closely

related to the dataset observed in nature. For this reason, it is a critical task to compare simulated fingerprints to real ones and bridge the gap between them.

Collecting datasets is also important to be able to verify the performance of the minutiae extraction on a broad range of training data. The different datasets collected as part of this thesis are described in detail in this section and example images are shown.

5.3.1 Synthetic Fingerprint Generation

Due to the required amount of annotated biometric fingerprints to train the neural networks was not available, synthetic fingerprints were used as a substitute. The advantage of synthetic fingerprints is that there is no limit on the generated sample size. Therefore the problem of having not enough training examples and, consequently learning the training set with poor generalization performance was avoided.

The freely available synthetic fingerprint generator Anguli [Ans11] was used to generate the training dataset. Initially, Anguli generated 30.000 ridge patterns, where 3.000 were non-linearly distorted as described in Chapter 7. Then for every ridge pattern, 5 impressions were made using Anguli. Example images are shown in Figure 5.1. In total, 150.000 fingerprint impressions were generated for this work. 10.000 images are devoted as a test set. Out of the remaining 140.000, used for training, 15.000 were non-linearly distorted. Additionally to the 140.000 simulated fingerprints, 400 annotated fingerprints were used in the training set as well. Out of the 400 fingerprints 320 were taken from the NIST SD 4 dataset and 80 from the FVC2000 DB4 dataset.

(a) (b) (c) (d)

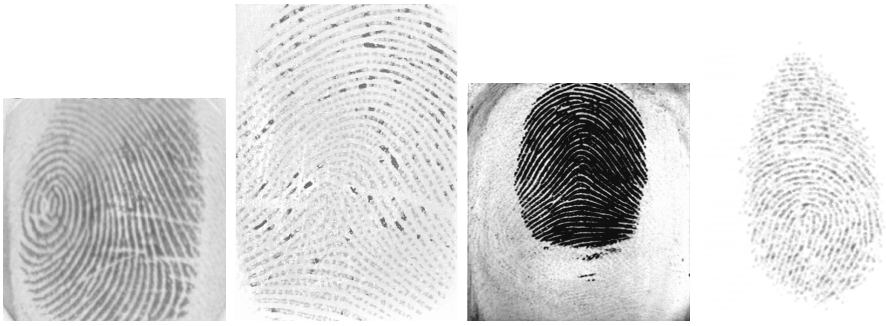
Figure 5.1: Example fingerprint impressions generated by Anguli. In (c,d) the underlying ridge pattern was non-linearly distorted, while (a,b) shows fingerprint impressions generated by Anguli, without additional applied distortions.

5.3.2 Fingerprint Verification Competition

The Fingerprint Verification Competition (FVC) uses four new databases for each of their competitions. The competition was hosted in the years 2000, 2002, 2004, 2006 and

one that is still ongoing. The competition has been getting harder every time it has been hosted and this is one of the reasons, why the classification rate has not improved over the years for those competitions [MMC+04]. For the ongoing competition, the dataset is not released yet and access to the database used in the competition for the year 2006 was not granted for this thesis.

Each competition contains four databases. Every database was acquired with different sensoric [MMC+02a]. This leads to changes in resolution, lighting and local contrast. Examples for each dataset are shown in Figure 5.2. Therein the large intra class differences are clearly visible, which are a result of using different sensors to obtain fingerprints.



(a) DB 1 (b) DB 2 (c) DB 3 (d) DB 4

Figure 5.2: A single example image for all the databases in the fingerprint competition FVC2000 [MMC+02a]. This visualization shows the intra class differences produced by using different sensor technology to obtain fingerprints.

Apart from changes in the pixel intensity values and local contrast in the databases, there are also differences in resolution of the fingerprint images. Moreover, the fingerprint in Figure 5.2(d) is actually synthetic and generated using SFinGe [CMM04]. Therefore different matching results are achieved by the various algorithms tried on such a dataset [MMC+02a]. Similar examples of the datasets from 2002 [MMC+02b] and 2004 [MMC+04] are shown in Figure 5.3 and Figure 5.4, respectively.

Every FVC competition consists of the same amount of fingerprints and all have four databases in them. Every database uses 110 different fingers with 8 impressions per finger. In total there are 880 images per database and 3520 images per competition. In total the FVC dataset has 14080 fingerprint images. The FVC dataset is used for evaluating the minutiae extraction performance based on the matching score between fingerprint images.

5.3.3 National Institute of Standards and Technology

The National Institute of Standards and Technology (NIST) is also involved in fingerprint research [WGT+04]. For this reason, multiple fingerprint databases were collected by

(a) DB 1 (b) DB 2 (c) DB 3 (d) DB 4

Figure 5.3: Example images for all the databases in the fingerprint competition FVC2002 [MMC+02b].

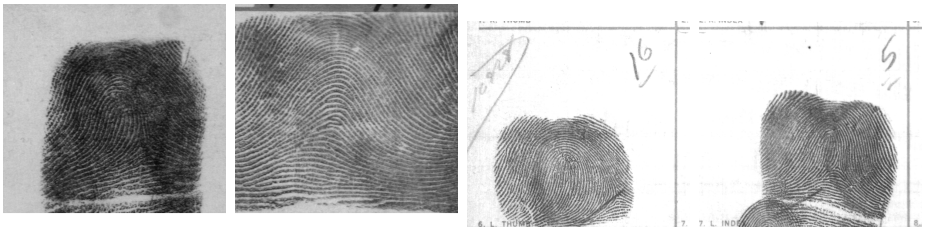
(a) DB 1 (b) DB 2 (c) DB 3 (d) DB 4

Figure 5.4: Samples of the fingerprint competition FVC2004 [MMC+04].

this institute. In this thesis the special database 4 and the special database 9 are used. Example images for both datasets are illustrated in Figure 5.5.

Special Database 4

The special dataset 4 consists of 2000 fingerprint pairs with a resolution of 512×512 pixels. This means, that a total of 4000 images are stored in this dataset. 300 images of this dataset have been annotated as part of this thesis to improve the discriminating ability of the trained model.



(a) (b) (c) (d)

Figure 5.5: In (a,b) samples from the special database 4 [WW92] and in (c,d) sample from special database 9 [Wat93] are shown. The resolutions of the fingerprint images themselves are the same, however the images of the special database 9 are larger. This leads to problems with training the refinement network.

Special Database 9

This dataset is not used for training the refinement network described in Chapter 7, because of the large areas without a fingerprint in them. Due to the random zoom operation, as described in Chapter 7, is used to prepare the training data, the preprocessed images may not include the fingerprint at all. Recall, that in the refinement network the real datasets are used to determine if the refined fingerprints are real. If background images are used to denote real fingerprints the refinement network learns unsuitable refinements. The reason for the large background is, that the acquisition process is done on all 10 fingers simultaneously. A large image with resolution 4096×1536 pixels is obtained through this process. Then the image is segmented into the 10 regions of size 832×767 pixels encapsulating a fingerprint each. Large regions are used to accommodate for the finger size of each individual and therefore large amounts of the images are comprised of background.

The dataset contains 5.400 such fingerprint cards of 2.700 individuals. This means 27.000 fingerprint pairs were collected as part of this dataset. This means this dataset consists of a total of 54.000 fingerprint images [Wat93].

5.3.4 Verifinger SDK Datasets

Along with the Verifinger [Ver10] SDK there are two freely available real fingerprint datasets appended for testing. This includes the Verifinger Sample DB and the UareU Sample Database [Ver10]. The Verifinger Sample DB contains 51 fingers and 8 impressions for each finger, thus 408 fingerprint images in total. The UareU Sample DB is used as a real dataset for the adversarial training and contains 65 fingers with 8 impressions each. Therefore a total of 520 fingerprint images are in this dataset. Examples for both datasets are illustrated in Figure 5.6.

(a) (b) (c) (d)

Figure 5.6: Sample of the UareU Dataset are shown in (a,b), while samples of the Verifinger Sample DB are shown in (c,d). Both datasets are part of the Verifinger SDK [Ver10].

5.4 Evaluation Metrics

Evaluation is important to determine the usefulness of an algorithm. In this section three different parts of the evaluation are highlighted. First, neural network comparison to find useful hyperparameter combinations. Second, the evaluation metrics used for the refinement network. Finally, the metric used for fingerprint matching, which is the main focus of this work.

5.4.1 Neural Network Comparison

The dice coefficient as defined in (4.5) is used as a rough estimate on how well specific models are performing. The dice coefficient denotes the amount of overlap between the predicted minutiae regions and the ground truth regions. This is mainly used to dismiss hyperparameter combinations that do not work at all and to decide between architectures, without having to evaluate the matching score.

5.4.2 Refinement Network Evaluation

The need for a quantitative score for generated data is not new and is addressed in research [SGZ+16]. Popular choices in GAN research are either purely visual [ZML16], using error rates of human discriminators [SGZ+16] or using the generated data for semi-supervised learning [Spr15]. The problem with human evaluation is that the results differ based on person and based on motivation and previous knowledge [SGZ+16]. To my knowledge the only quantitative evaluation metric for GANs is the inception score invented in [SGZ+16]. This score uses the inception model [SLJ+15] trained on ImageNet to ouput the conditional probability $p(y|x)$ of a generated sample. The evaluation metric is given in (5.1), where KL denotes the Kullback-Leibler divergence [KL51], and uses the insight that GANs should generate varied samples.

$$\text{score} = \exp(\mathbb{E}_x \text{KL}(p(y|x)||p(y))) \tag{5.1}$$

The problem with extending this approach to the minutiae extraction problem is that there is no standardized dataset to test on and the output space is very high dimensional, namely 224×224, making the evaluation time consuming. Additionally, this metric does not work well for training as reported in [SGZ+16].

Given this evaluation of the state-of-the-art, the evaluation used in this work is two-fold. First, the visual result is highlighted and compared to other versions of the algorithm and other synthetic fingerprint generators. Second, the matching performance is compared using the same model trained with a different refinement network and without a refinement network to observe the performance gain using this method as opposed to current synthetic fingerprint generators.

5.4.3 Fingerprint Matching

The main focus of this work is to propose a new robust minutiae extractor to allow a reliable identification of individuals using fingerprints. Therefore the matching score is the most important metric in this thesis. To achieve a good matching score, the extracted minutiae need to be consistent across multiple fingerprints. For this reason it is important to highlight the matching score of such a matcher opposed to an accuracy metric.

To determine the performance of the minutiae extraction algorithm proposed in this thesis, it is compared to other state-of-the-art works. For fairness reasons the same minutiae matcher, namely BOZORTH [WGT+04], is used. BOZORTH outputs a probability for a pair of fingerprints on how likely the corresponding minutiae show the same fingerprint. Therefore the classifier depends on a threshold at which point to trust the match. The Equal Error Rate (EER) is used to compare 3 algorithms with each other, namely Verifinger [Ver10], MinDTCT [WGT+04] and the one proposed in this thesis.

GAR is the Global Acceptance Rate and FAR ist the False Acceptance rate. GAR is defined as the percentage of the true matches, while the FAR is defined as the percentage of the false matches. GAR and FAR are controlled by using a threshold t, which controls if a specific matching score is counted as a match or not. The EER is defined as the rate obtained given by GAR at the value of t, where (5.2) holds.

$$1 - FAR = GAR \tag{5.2}$$

The EER is also used in [MMJP09].

CHAPTER 6

Implementation: Minutiae Extraction

In this chapter the new algorithm for minutiae extraction is explained in detail. In this thesis the minutiae extraction algorithm is reformulated as a binary segmentation task, which is solved using deep learning. This means learning a function that maps an input image to a probability mask for each pixel, indicating if a pixel is part of a minutiae point. The final minutiae map needs to be post-processed to match the specification of the matching algorithm. This chapter describes the algorithm used to extract a minutiae list from an arbitrary fingerprint image.

6.1 Training Algorithm

The training pipeline is visualized in Figure 6.1. The freely available synthetic fingerprint generator Anguli [Ans11] is used to generate the ridge pattern. An extension of Anguli, described in Chapter 7 is used to generate the training data for the neural network. The fingerprint images share the minutiae positions with their initial ridge pattern as shown in Figure 6.1. Then the commercial fingerprint minutiae extractor Verifinger [Ver10] is used to extract minutiae from the ridge pattern. In this work it is assumed that the minutiae extractor works perfectly on the binary ridge pattern. This is obviously not true in practice, but as shown in Chapter 9 Verifinger works good enough to learn from the data. Verifinger outputs a minutiae list, which is transformed into a minutiae regions map as shown in Figure 6.1. Regions instead of points are used following the insight that the minutiae points are not a single pixel, but whole regions of pixels. Also the neural network was unable to learn from single points as minutiae regions and therefore larger regions are used. The model then outputs a binary minutiae map to be used for evaluation. The network is then trained using dice loss on the ground truth and the output of the network. Dice loss is given in Eq. (4.5).

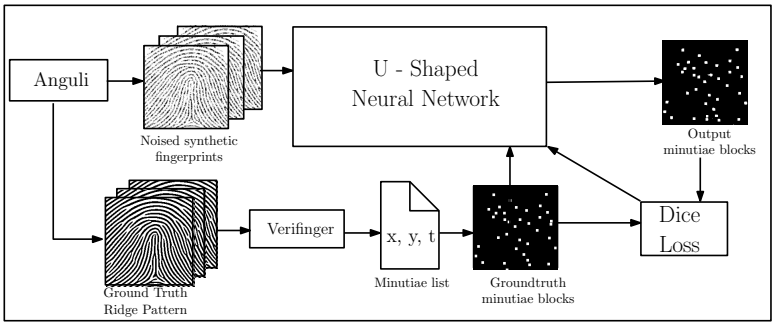


Figure 6.1: The training pipeline for the minutiae extraction network.

6.2 Fingerprint minutiae list generation

The neural network outputs a binary minutia region map. This region map is not directly usable by minutiae matching algorithms like BOZORTH3 [WGT+04]. BOZORTH3 expects a list of minutiae with the following properties:

1. Position in x, y coordinates of the image.
2. Orientation in degrees between 0° and 360°.
3. Quality measurement between 0 and 100, where 0 and 100 denote bad and good quality, respectively

BOZORTH3 expects natural numbers for each property. Recall, that the output of the neural network proposed in this work is a binary minutiae region map as explained in Chapter 8. Therefore this chapter describes the process of extracting the minutiae list, necessary for fingerprint matching, given this minutiae map.

6.2.1 Fingerprint Position and Quality Extraction

The neural network outputs a minutiae map for a 224×224 image crop of a fingerprint. As described in Chapter 5, the fingerprint images vary in size and are larger in size. For matching fingerprints it is favorable to use the whole fingerprint for matching. Therefore in this section the method used to predict on the whole fingerprint is explained. Additionally, the method to extract position and the quality extraction are also explained in this section.

The neural network works on 224×224 images. However, fingerprints often have different resolutions ranging from 326×357 to 832×768 as shown in Chapter 5. The fingerprints still use a predetermined dpi of 500. Therefore to use the same dpi in the network model it is favorably to use crops of the images. However, a single crop does not entail the whole image and results in missed minutiae points. To circumvent this multiple crops are used. Multiple crops up to 32 do not cause a significant slow down, because the output

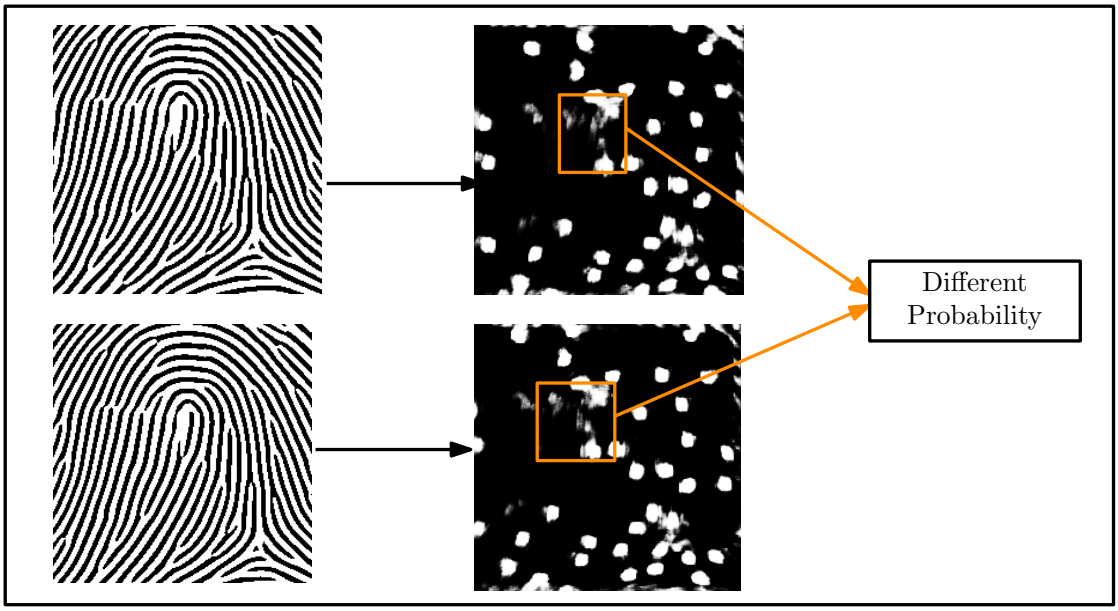


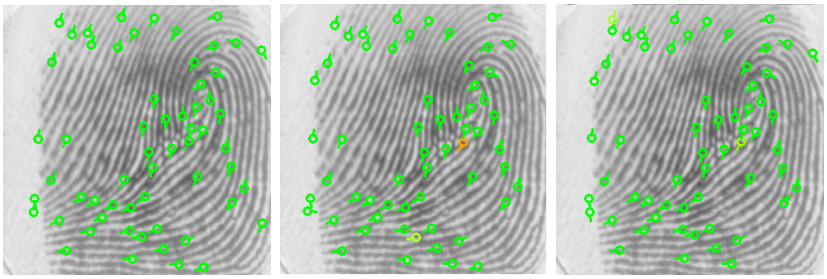
Figure 6.2: Showcase of the differences between a predictions on the ridge pattern and its slightly translated ridge pattern.

can be computed in parallel on the machines using a Titan X. The problem thereby is on how to combine overlapping regions and how to deal with contrasting information.

This problem is addressed using another insight into deep learning. Namely, that the neural network is very sensitive to change and produces different outputs for minimally translated images as shown in Figure 6.2. However, using the insight, that the model is less likely to do the same mistake twice, leads to a majority vote method on overlaying regions.

Therefore randomly placed 224×224 crops are used to detect minutiae in the fingerprint. The number of crops used for evaluation is a hyper parameter for the feature extraction stage. A comparison of the minutiae found using different numbers of crops is shown in Figure 6.3. Therein minutiae that are detected using a specific amount of crops vanish with a different amount.

The centroid of every minutiae region is used to extract the final minutia position. To do this the binary connected components are used. The connected components are found using a depth-first search on the binary input image and outputs all connected regions using the 4-neighborhood connection. The centroid for every component C is calculated using (6.1) for the x coordinate and (6.2) for the y coordinate. The problem with this approach is, that overlapping minutiae regions are combined into one minutia point. Still, as shown in Chapter 9 the algorithm works well enough.



(a) (b) (c)

Figure 6.3: Illustration of the detected minutiae using 8 (a), 16 (b) or 32 (c) crops.

$$centroid_x = \frac{1}{|C|} \sum_{p \in C} p_x \tag{6.1}$$

$$centroid_y = \frac{1}{|C|} \sum_{p \in C} p_y \tag{6.2}$$

Following the idea, that the quality of a minutia correlates with the confidence of the prediction by the neural network, the connected component is also used for the quality property. The output of the network, as shown in Chapter 4, is almost binary using dice loss. Therefore the probability is not suited for this task. However, it is assumed that the prediction of the model correlates with the size of the extracted minutia region. Following this idea, the resulting quality measurement is calculated using (6.3).

$$quality = \min(2 \cdot |C|, 100) \tag{6.3}$$

6.2.2 Fingerprint Orientation Estimation

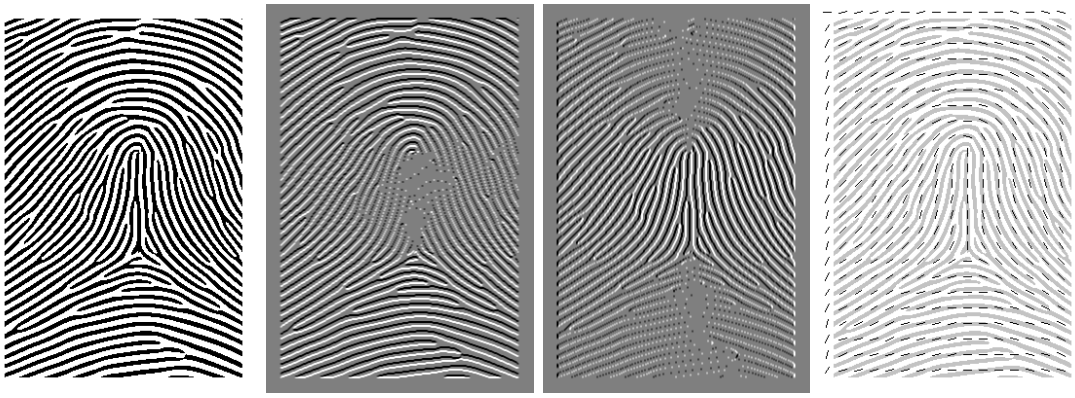
To match the extracted minutiae positions to each other, the direction of the corresponding ridge is used [WGT+04]. For the minutiae extraction algorithm, the focus was on learning the minutia position instead of the orientation. The algorithm proposed in [HWJ98] was used for a rudimentary orientation estimation. This means splitting the image in 16×16 blocks, where the orientation of the image is estimated. The angle θ of a block B is calculated using the following set of formulas:

$$V_x(B) = \sum_{i,j \in B} 2 * S_x(i,j) * S_y(i,j) \tag{6.4}$$

$$V_y(B) = \sum_{i,j \in B} S_x^2(i,j) * S_y^2(i,j) \tag{6.5}$$

$$\theta(B) = \arctan(\frac{V_x(B)}{V_y(B)}), \tag{6.6}$$

where S_x, S_y symbolize the gradient of the image in the x and y directions, respectively. For the calculation of the gradient, a Sobel [Sch00] filter is used. The output of such a filter in both directions can be seen in Figure 6.4. Therein the response of the Sobel filter in a particular direction is shown.



(a) Original ridge pattern of Fingerprint. (b) Sobel response in x direction. (c) Sobel response in y direction. (d) Orientation estimation of the Fingerprint.

Figure 6.4: Orientation Field calculation of a sample fingerprint.

The output of (6.6) can be seen in Figure 6.4d. The direction of the image follows the ridge pattern and therefore the direction in a block corresponds to the direction of the ridge in a minutiae block. Because the arctan used in (6.6) only corresponds to degrees between $-90°$ and $90°$, local statistics are used to determine if the ridge is between $-90°$ and $90°$ or between $90°$ and $270°$. This means looking at the 3×3 neighborhood and using the direction with higher values. This corresponds to the neighborhood containing the ridge [HWJ98]. The result of this approach is shown in Figure 6.5.

Comparatively, in this thesis a second point based approach is also used in this thesis. This approach works by looking at every minutia point in isolation and calculating the orientation in a 10×10 pixel grid around the minutia point. The orientation is calculated in the same way as in the orientation field, using (6.6) for the 10×10 pixel patch. One such block is shown in Figure 6.6.

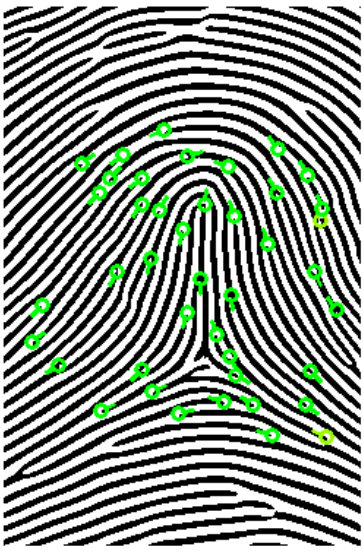


Figure 6.5: Example fingerprint with overlaying minutiae points. The angles are extracted using the orientation field. The minutiae are extracted using the center as a crop region.

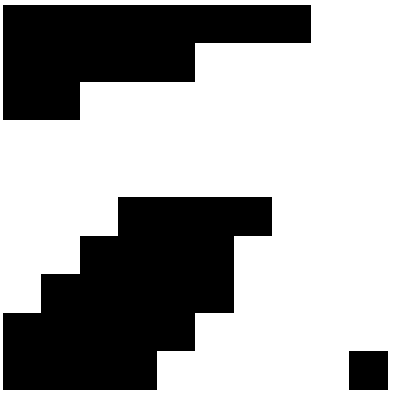


Figure 6.6: Example of a minutia block

CHAPTER 7

Implementation: Data Generation

This chapter is split into two parts. First in Section 7.1 the offline data acquisition, generation and modifications are explained. Then in Section 7.2 on the fly augmentation to the previously acquired fingerprints is described.

7.1 Offline Data Generation

Fingerprint images are tedious to acquire, due to privacy protecting legislation, and a boring process for both the people involved and the volunteers [CMM02]. For these reasons synthetic fingerprint generators are used in competitions [MMC+02a, CFFM07].

7.1.1 Synthetic Fingerprint Generators

To my best knowledge only two such generators exist, namely Anguli [Ans11] and SFinGe [CMM04]. Those approaches work as follows:

1. A fingerprint shape model, a directional map model and a density map model are combined to create a ridge pattern. For Anguli a sample ridge pattern is shown in Figure 7.1(a).
2. So called impressions, which are variations of the same fingerprint as described in Chapter 1 are extracted from the ridge pattern generated in step 1 by extracting the fingerprint along contact region. Then a noise model is applied on top of that. SFinGe also uses morphological operations and non-linear distortions to create more realistic fingerprints [CMM04].
3. Fingerprint impressions are randomly translated and rotated. This operation was turned off for the purpose of this thesis to preserve the minutiae information, which is calculated from the binary ridge pattern.

4. SFinGe [CMM04] additionally uses a background generator to make the fingerprints more challenging.

For this thesis, only Anguli was available to be used to generate training data, so additional operations were implemented as part of this thesis to make the fingerprints more realistic. The operations described in the following sections are randomly applied to fingerprints and their minutiae maps.

7.1.2 Non-linear distortion

To model the contact region of a fingerprint random non-linear distortions are used. By using non-linear distortions, changes to the local ridge frequency are created. Non-linear distortions are implemented as follows. Figure 7.1(b) shows the two radii visualized by circles used in the non-linear distortions equation given in (7.1). The two circles represent the regions where different functions are used to calculate the rotation applied to each pixel. Equation (7.1) is used to model the rotation, where r_1 and r_2 denote the circle radii and ϕ and d the resulting rotation angle and the distance to the center pixel. The constant k is used to control the resulting rotation and was set to 0.03 to generate the result shown in Figure 7.1c.

$$\phi(d) = \begin{cases} 0, & \text{for } d \leq r_1 \\ k \cdot (d - r_1) & \text{for } d \leq r_2 \\ k \cdot (r_2 - r_1 + (d - r_2)^2) & \text{for } d > r_2 \end{cases} \tag{7.1}$$

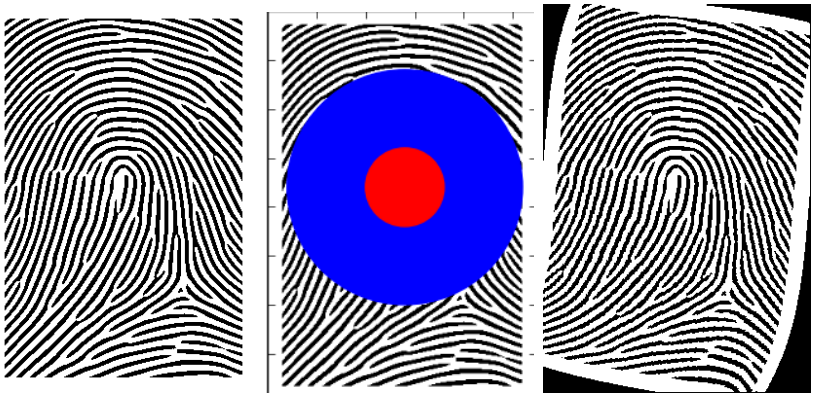
The circle radii are randomly varied in $[35 - 45]$ and $[95 - 125]$ and are given in pixels. An illustration of the two radii is shown in Figure 7.1b.

Due to discretization and rounding errors, not all the pixels are rotated accordingly and so the image is pre-initialized to -1. After the distortion operation, pixels with value -1 are assigned to a value decided by their neighborhood through a majority vote. In case of a tie, the value of a ridge is used. The final result is illustrated in Figure 7.1c.

Finally, to preserve the annotation data, the transformation is also applied to the minutiae map. Using these new ridge patterns, Anguli is able to generate new impressions. This is visualized in Figure 7.2.

7.1.3 Fingerprint Minutiae Annotations

To additionally increase the discriminative power of the training dataset, some fingerprints were annotated as part of this thesis. For this purpose annotation software was written to help annotating fingerprints. A screenshot of this computer program is illustrated in Figure 7.3. A fingerprint of the NIST sd04 [WW92] dataset is shown with the minutiae information initialized with the Verifinger [Ver10] minutiae extractor.



(a) (b) (c)

Figure 7.1: A single ridge pattern by Anguli is shown in (a). The circles, where the different distortions are applied to are shown in (b). By applying (7.1) with the radii of those circles the ridge pattern in (c) is generated.

(a) (b) (c)

Figure 7.2: The distorted ridge pattern is shown in (a). In (b) and (c) corresponding impressions generated by Anguli are shown.

As illustrated in Figure 7.3 the minutiae extracted by the Verifinger SDK need to be checked manually in order to correct annotation errors. Most errors occur in regions with little contrast and along artifacts like the black line at the top in Figure 7.3. Also some minutiae points are just not found by Verifinger, which has to be corrected using the annotation software.

In this approach the orientation and the quality of the minutiae are not considered. Verifinger supplies the initial orientation and quality and, if corresponding minutiae are not changed, this information does persist. Changed minutiae have their quality and their orientation initialized to 100 and 0 degrees respectively. This information is not used by the training and evaluation algorithms.

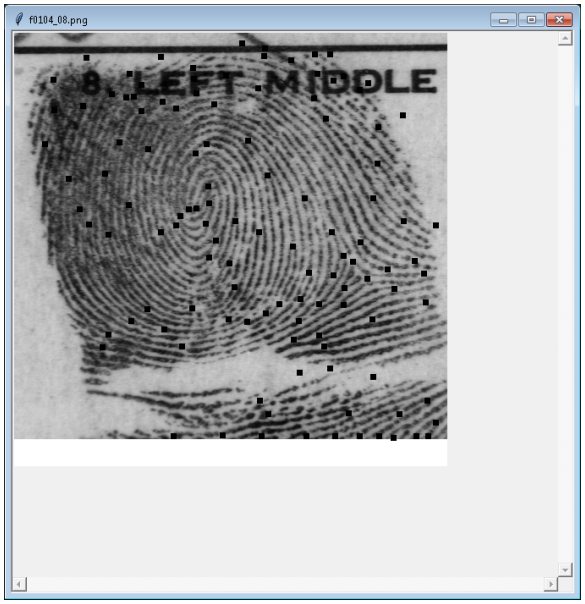


Figure 7.3: Screenshot of the annotation program with a sample initial fingerprint taken from the NIST Special Dataset 4 [WW92].

7.2 Data augmentations

The minutiae extraction model needs to be invariant to various operations, like linear transformations, lightning changes, pixel intensity changes and noise [MMC+02a]. Therefore on the fly augmentation are used in the training process.

7.2.1 Linear transformation

To make the minutiae extraction CNN invariant to linear transformations, the following operations are applied to the training data on the fly:

1. **Translation Invariance**: Fingerprint images are randomly translated by a value between $[-25, 25]$ pixels in x and y directions illustrated in Figure 7.4(b).
2. **Rotation Invariance**: The input images are rotated, by a random degree between 0 and 180 degrees around the center of the image. Additionally random mirroring is used to add additional invariance to larger rotation.
3. **Scale Invariance**: The input image size of the CNN is 224×224 pixels, while the resolution of the fingerprints generated by Anguli is 400×275 pixels. Real fingerprints use different resolutions [MMC+02a]. To gain limited amount of scale invariance, the fingerprint is first resized by a random factor between $[1, \frac{\max(h_1,w_1)}{\min(h_2,w_2)}]$, where h_1 and w_1 denote the input resolution of the CNN and h_2, w_2 denote the resolution of the fingerprint. This value is 0.815 for the fingerprints generated by

Anguli. After the resize operation, a random crop is used as the final image. This operation is called random zoom in this thesis.

Example affine transformations are visualized in Figure 7.4.

(a) (b) (c) (d)

Figure 7.4: In (a) the original simulated fingerprint is shown, while (b-d) show linear transformations of this fingerprint used for training. (b) shows a random translation and (c) shows a random rotation, while (d) shows a random zoom, resulting in an image size, which is usable by the CNN.

7.2.2 Other augmentations

Fingerprints also change appearance under various conditions. For example, wet fingers look vastly different than normal fingerprint images [CMM04]. For this reason, the following additional augmentation operations are used:

1. **Random Blurs**: The images are randomly blurred with a gaussian kernel, where the variance varies to simulate noisy fingerprints.

2. **Morphological operations**: Grayscale dilation and erosion are used to model wet and dry fingerprint images [CMM04]. These operations do not change the minutiae positions in the image and therefore are not applied to the minutiae map.

3. **Random channel shift**: A specific percentage of rows or columns of the image are omitted. This way the noise in the image is modeled. Also the CNN avoids to rely on specific rows or columns to classify minutiae.

The operations described above are illustrated in Figure 7.5.

(a) (b) (c) (d)

Figure 7.5: In (a) the original simulated fingerprint is shown, while (b-d) shows augmented fingerprints used for training. (b) shows a gaussian blurred fingerprint and (c) shows an erosion operation on this fingerprint. In (d) the result of a chnanel shift is shown.

7.2.3 Refinement Network

As described in Chapter 4, a GAN is used to generate data indistinguishable from real data. The problem with this approach is, that there is no reliable way to add annotation data to the generated data. In contrast, current synthetic data generators enable an automated way of annotation [Ans11].

Therefore, to still apply the GAN paradigm to the minutiae extraction problem simulated fingerprints with annotation data are used as input. A so called *refinement network* or *refiner* is learned, instead of a generator network in the original approach. This neural network is used to refine fingerprints, while preserving the annotation data, instead of generating entirely new data. Such a refinement network is then used to augment data on the fly.

The preservation of the annotation data is done by regularizing the refinement network in order to preserve the underlying ridge structure of the fingerprints. In [SPT+16], this is accomplished by using the Mean Squared Error (MSE) between the input image and the generated output image. The refinement network and a discriminator network are combined to form the SimGAN, named after the network used in [SPT+16].

The refinement network is used to refine simulated fingerprints until it is not possible to discriminate them from real ones. To ensure this, the refinement network is trained using a second network, namely the discriminator, which is responsible for discriminating between real and simulated fingerprint images. The SimGAN converges, when the discriminator is unable to learn a discrimination between the generated data and the real data. The architecture details of the refinement and the discriminator networks is given in Chapter 8

Training the SimGAN

The process of generating real looking simulated fingerprints is visualized in Figure 7.6. Anguli generates various synthetic fingerprints to be refined. Those fingerprints are combined with the real fingerprints and are fed into the SimGAN. Then the algorithm works as follows:

1. The refinement network is applied to the simulated data and the output is combined with the real data. The refinement output has the label zero, while the real data has the label one. In deep learning, some operations are only applied at the training stage, like dropout and batch normalization, and produce different outputs with different value spreads then in the testing stage. Using the testing stage of the refinement network to produce the samples used by the discriminator in the training stage leads to divergent SimGAN training.

2. The discriminator is trained on the dataset, which was prepared in Step 1, for one epoch.

3. The discriminator and all its layer weights are frozen. This is done to be able to use the discriminator solely as a loss function for the refinement network. In this stage, there is no need to know in which way a single item has to be refined.

4. The refinement and the discriminator networks are combined to form the SimGAN. The output of the refinement network is the input to the discriminator, while the output of the discriminator is the final output of the model. Because the discriminator is frozen in Step 3 the gradient is backward propagated through the SimGAN and only updates the weights of the refinement network.

5. The label for each simulated fingerprint is set to one, which is the class of the real fingerprints. This is done to allow the SimGAN to learn the refinement of fingerprints so they are perceived as real by the discriminator.

6. Using the dataset constructed in Step 5, the SimGAN is trained for two epochs.

7. The discriminator and all its layers are unfrozen again and the whole process starts with Step 1 until the maximum number of iterations is reached.

There is no actual convergence condition to the refinement network. Therefore, the refiners used in this thesis are trained for 20 iterations. In the experiments conducted the difference afterwards are negligible and the risk of collapsing to a single output increased with the number of epochs.

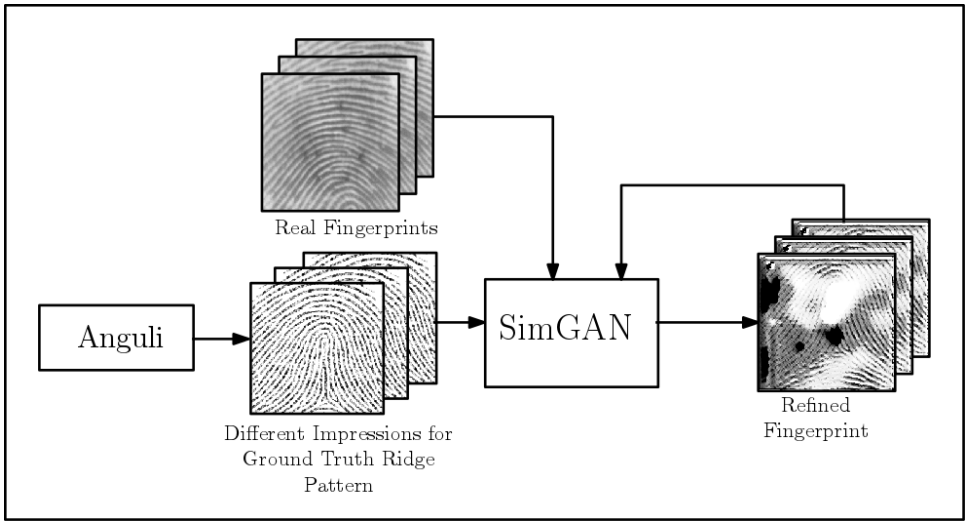
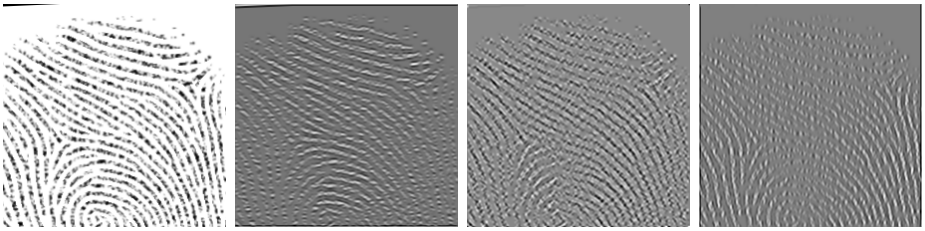


Figure 7.6: Training Pipeline of the SimGAN.

Regularization

In the literature MSE is used on the original image as regularizer to discourage the refinement network from applying large refinements [SPT+16]. This work proposes the usage of the Hessian image instead of the original image for regularization, following the idea to allow the network more freedom in the refinement process. The Hessian represents the ridges independent of the pixel intensity values as illustrated in Figure 7.7. The Hessian is the second spatial image derivative and is calculated by using the Sobel filters twice. The Hessian is calculated as part of the NN. Depending on the directions of the Sobel [Sch00] filter either the xx, xy or the yy Hessian is produced. The resulting ridge pattern is not supposed to deviate strongly from any of these directions and therefore all of those directions are used for regularization. To penalize deviations from the Hessian MSE is used, due to its successful usage in related literature [SPT+16].

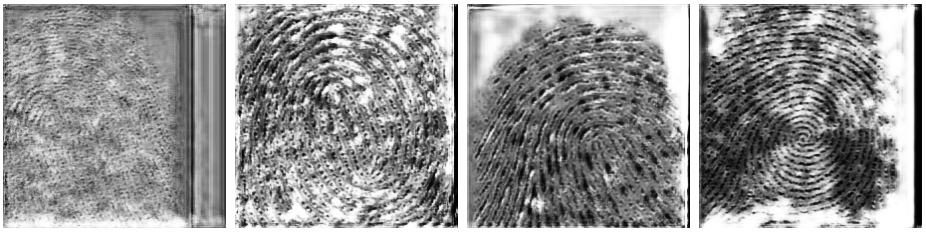


(a) (b) (c) (d)

Figure 7.7: In (a) the original simulated fingerprint is shown, while (b-d) show the Hessian of the fingerprint image in xx, xy and yy directions.

By controlling the regularization parameter of the MSE in the loss function, different refiner networks are learned. Those refiner networks then produce different output

fingerprints as visualized in Figure 7.8. The structure of the ridge pattern is preserved even with little weight on the regularization. However, the structure is less visible and as shown in Figure 7.8(a). The illustration in Figure 7.8 also shows, that increasing the value of the regularization parameter improves the visibility of the ridge pattern.

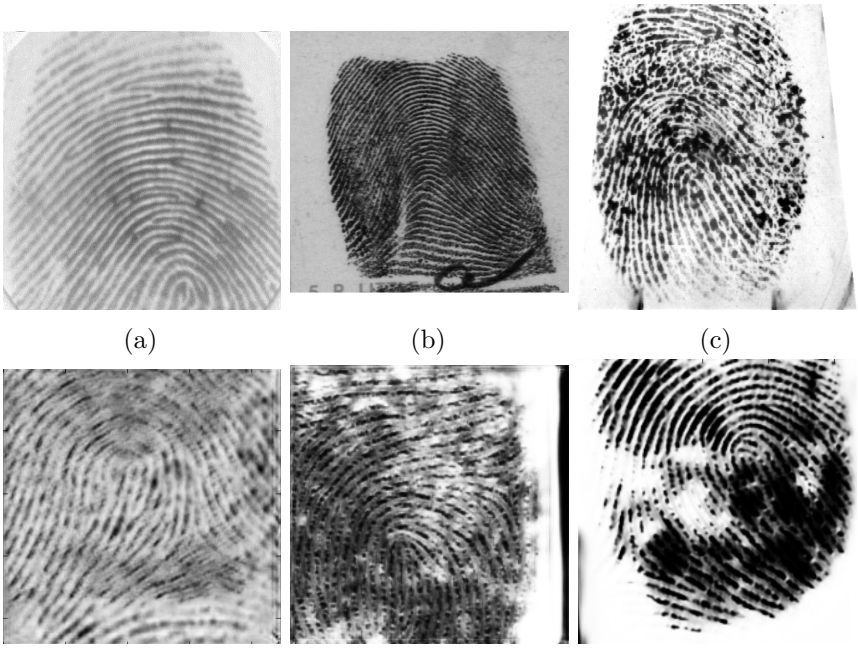


(a) (b) (c) (d)

Figure 7.8: Refined outputs of a SimGAN trained with local adversarial loss. The only difference is the value of the regluarization parameter. (3 in (a), 5 in (b), 8 in (c) and 16 in (d)) Note, that the ridges and valleys are more visible when higher weighting is applied.

Different Real Datasets

The refinements calculated by the refinement network are dependent on the real dataset used for discrimination. Example images are shown in Figure 7.9. Therein large differences in real datasets and the impact of using different real datasets for training a refinement network are shown. One characteristic in the sample from the FVC200 [MMC+02a] is the inhomogeneous brightness distribution as shown in Figure 7.9(a). This is then adopted by the refinement network in Figure 7.9(d). Similar behavior is shown by the other two examples illustrated in Figure 7.9. Therefore, the real dataset used for generating new fingerprint images needs to be diverse in its appearance to allow the refinement network to generate a diverse dataset.



(a) (b) (c)

(d) (e) (f)

Figure 7.9: Output of the refinement network, when trained on a particular dataset. (d) was trained on the FVC2000 DB 1 dataset with a sample shown in (a). (e) was trained on the sd04 dataset, with a sample shown in (b). Finally (f) was trained on the UareU dataset with a sample shown in (c).

CHAPTER 8

Implementation: Neural Network Design

Various architectures are proposed in literature for NNs [SZ14, HZRS16, SVI+16, SIVA17, XGD+16, Cho16] with performance improvements for tasks in computer vision. The ideas described in those papers are used to design the novel architecture used in this thesis for the minutiae extraction task. As explained in Chapter 6 the minutiae extraction task is reformulated as a binary segmentation task. This chapter is split into three parts. In Section 8.1, the similarities between all architectures are explored and evaluated. In the next section, the minutiae extraction models are explored and compared. Finally, the refinement and discriminator network architectures are described and evaluated.

8.1 Network Architecture

This section is about the common elements in the different neural networks. This ranges from hyperparameters like batch sizes and dropout to the general layout of the model. The evaluation of those experiments prompted the development of the following abstract architecture.

8.1.1 Abstract Architecture

To speed up the development of new architectures, an abstract architecture was developed, where the similarities between the different architectures are collected. The basis of all the minutiae extraction and the refinement models forms a so called U-Shaped Fully Convolutional Neural Network. The U-Shaped architecture is first proposed in [RFB15], while the first fully convolutional neural network is introduced in [SDBR14]. "Fully convolutional" means that down-sampling is accomplished using convolutional layers instead of pooling layers. U-shaped architectures have successfully been used to address

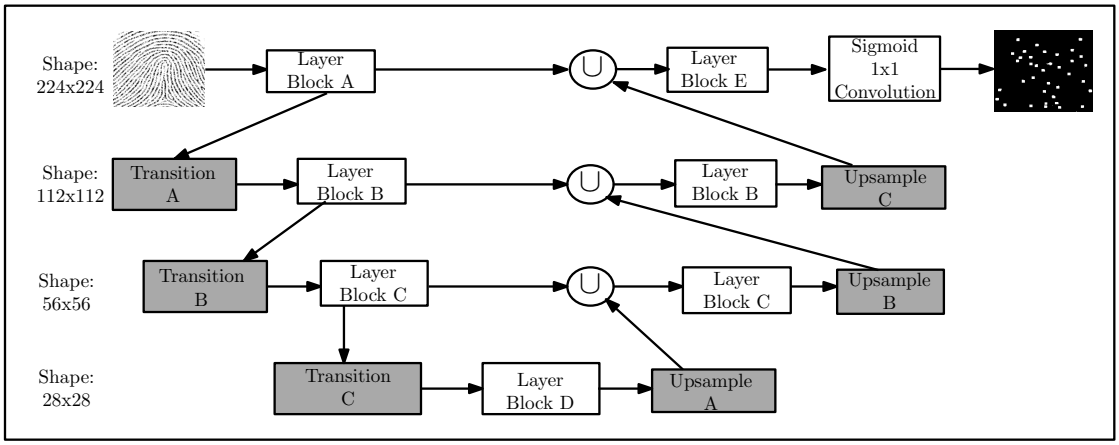


Figure 8.1: Shows the abstract network architecture, inspired by the U-shaped Architecture proposed in [RFB15].

segmentation problems like semantic segmentation [JDV+16] and biomedical image segmentation [RFB15, MNA16, DVC+16].

The general architecture is visualized in Figure 8.1. This part of the network is implemented in this thesis as an abstract model to rapidly prototype small changes to the network architecture, like changing layer blocks and different amount of layers. In this chapter, before every convolution, batch normalization and a ReLU activation is applied unless stated otherwise.

The U-Shape of the network is illustrated in Figure 8.1. The U-shape is accomplished with Transition and Upsample blocks. Convolutional layers with a 2×2 stride are used for transitioning downwards. This has the advantage of being able to learn downsampling layers and not to lose information in contrast to 2×2 pooling, where $\frac{3}{4}$ of the information is lost. For the Upsample blocks, a so called Upsample Layer is used, which in essence just repeats each pixel in a 2×2 grid and therefore doubles the width and height of the resulting feature maps. After the Upsample block a 3×3 convolution without prior batch normalization and ReLU activation is applied, before the feature maps are passed on to the next layer block.

The final activation function of those U-shaped networks is sigmoid as defined in (8.1). The final layer is not preceded by a batch normalization and does not use a ReLU. However a dropout layer is applied before the final layer. This is done according to the observation in [DVC+16], that dice loss, as defined in (4.5), benefits from dropout before the final layer. The sigmoid function has an output range of $[0, 1]$. By using sigmoid as the final activation function, the network tries to learn a mapping from a pixel to the probability, that this pixel belongs to a minutiae region, given the whole image.

$$f(t) = \frac{1}{1 + e^{-t}} \tag{8.1}$$

8.1.2 Effect of the image size on network architecture

The amount of parameters in a convolutional layer is not dependent on the image size [LBBH98]. Therefore it is possible to use a large image as input and still keep the number of parameter to optimize low. However, because of the large amount of intra class differences as explored in Chapter 7, a reasonably large batch size of at least 16, preferably 32 or 64 is needed for the training to work well as shown in Figure 8.2.

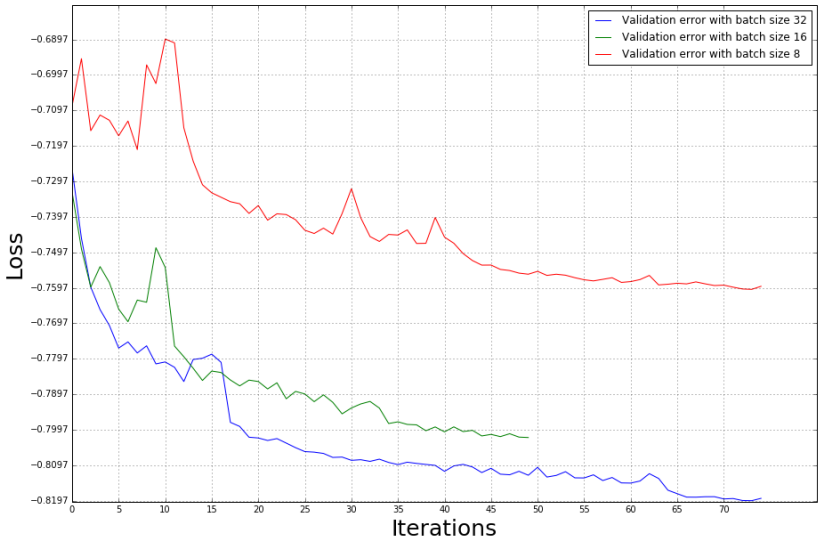


Figure 8.2: Validation Loss on the U-net-v1 network with different batch sizes.

However the model needs to fit into the VRAM to be optimized. For this reason the limiting factor for the image size is the VRAM in the GPU. Because the images calculated in one batch have to be stored in VRAM in order to calculate the gradient. The WRN-V2 network as described in Chapter 8 uses 11.314 MB of the 12.189 MB available VRAM in a Nvidia Titan X GPU. This illustrates the point, that segmentation is starving for VRAM, because the output size is the same as the input size. To preserve information flow the amount of information contained in subsequent layers should not change too drastically [SVI+16].

8.1.3 Hyperparameters

The optimizer used to train the model is one hyperparameter [Nes83]. In literature, either Adam [KB14], SGD with nesterov momentum [Nes83] or RMSProp [TH12] are used [RFB15, DVC+16, GBC16]. Fig 8.3 shows a comparison between the different optimizers using the WRN-V1 model. The jumps when using the adam optimization occurs where the learning rate is cut. SGD and RMSprop use learning rate decay and

therefore the jumps are not as noticeable. Adam worked best while optimizing the U-net shaped networks. It did find a better optimum faster than SGD and RMSprop.

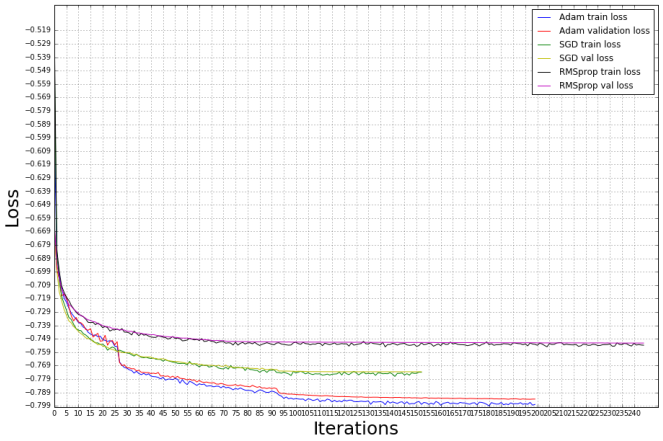


Figure 8.3: Equal error rate plot on the synthetically generated data.

Another hyperparameter is the number of samples per epoch. It is possible to increase the samples per epoch arbitrarily, without feeding the same images into the network over and over again, because of the on-the-fly augmentation. In this work, this is non-trivial, because epochs have an influence on the learning rate, depending on the learning rate scheduling. In step learning, when the validation error stops decreasing, the learning rate is cut. If the model is trained in a single epoch, the learning rate is never cut. This leads to a difference in using more or less samples per epoch. Therefore the training algorithm needs longer to react to a non-decreasing validation loss. The impact of this phenomenon is shown in Figure 8.4.

8.1.4 Dropout

Dropout [SHK+14] is heavily used in literature [KSH12, ZK16, HLW16]. In [DVC+16] it is reported, that dice loss benefits especially from dropout. Therefore it is promising to incorporate dropout into our network architecture. Because dropout is not used for the validation score, the validation score can be higher than the training score as shown in Figure 8.5. In Keras [Cho15], the validation loss is calculated after the training set has been traversed completely. Therefore the difference between the training and the validation loss is larger in the first 2 epochs as shown in Figure 8.5.

In Figure 8.6, different architectures based on the DenseNet, described in the following section, are compared. The difference in the architectures is the amount of dropout layers used. One network uses no dropout. The second one uses dropout in the final convolution layer. The last model uses dropout as part of its layer blocks. As shown in Figure 8.6, using dropout is detrimental for the performance of the network.

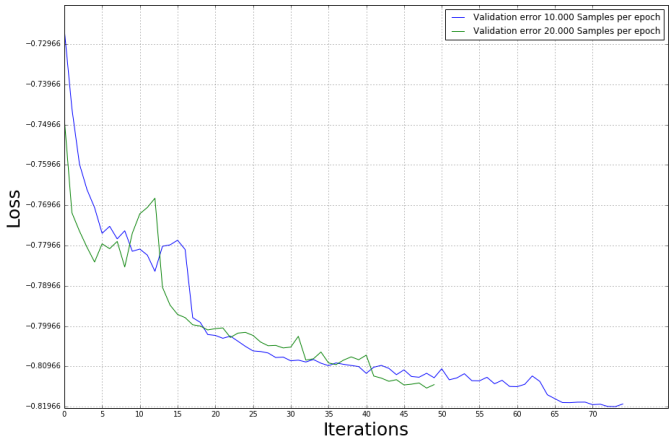


Figure 8.4: Validation error on the U-net-v1 network with different amount of samples per epoch.

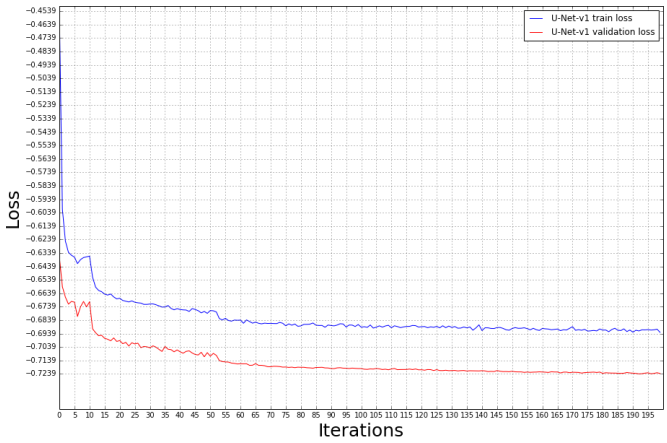


Figure 8.5: Training and validation loss for the U-net-v1 network.

8.2 Minutiae Extraction Networks

Recall, that the minutiae extraction problem is reformulated as a binary segmentation task. For this reason the networks proposed here are similar to the ones proposed for other binary segmentation tasks in medicine [RFB15, DVC+16]. In this section, the different networks programmed for the minutiae extraction problem are explained.

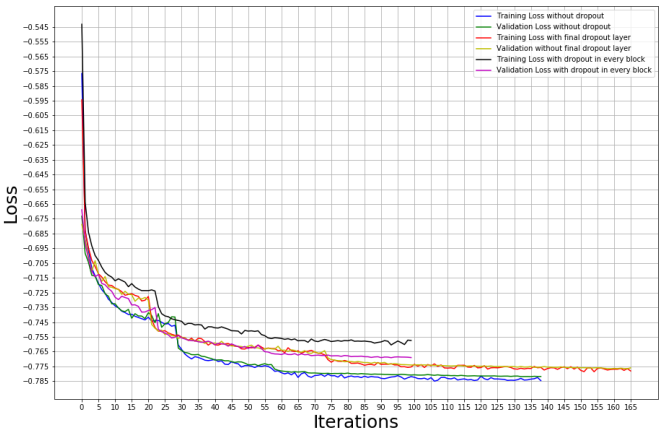


Figure 8.6: Performance of the dense network with dropout enabled/disabled.

8.2.1 Inception Style Architecture

The first network was developed with the thought of fusing the state-of-the-art in segmentation [RFB15] with the state-of-the-art in classification [SIVA17] and was named U-Net-v1. In Figure 8.7, this network, namely the u-net-v1, is illustrated. This figure is primarily used to showcase the complexity of the models used in this thesis. This model uses inception blocks [SIVA17] on its downward path and residual inception blocks on its upward path as shown in Figure 8.7, giving a weight distribution of its filter kernels as visualized in Figure 8.8. Therein the value of the norm of every filter kernel is visualized per layer. This model is also different in the way long residual connections are used. Because of the results in [SZ16], weighted residuals are used here. This is accomplished by learning the weight of the residual by applying a 1×1 convolution on the residual connection.

The complete specification of the network, split into its layers can be found in Tab. 8.1. The parameter count is low in relation to the number of layers compared to the other architectures proposed in this thesis. This network was used to test out the U-Shaped approach. All the following architectures were built after the success of this one as indicated in Chapter 9. The other models are larger in the size of parameters. To still compare the performance, a larger model with a similar architecture was constructed. However, this particular architecture did not scale well and the performance decreased as the parameter count increased, which is why this model is not further used for evaluations.

8.2.2 Other Architectures

Because of the scaling problem with inception blocks, as previously mentioned other layer blocks are considered as part of this work. One particular important work hereby

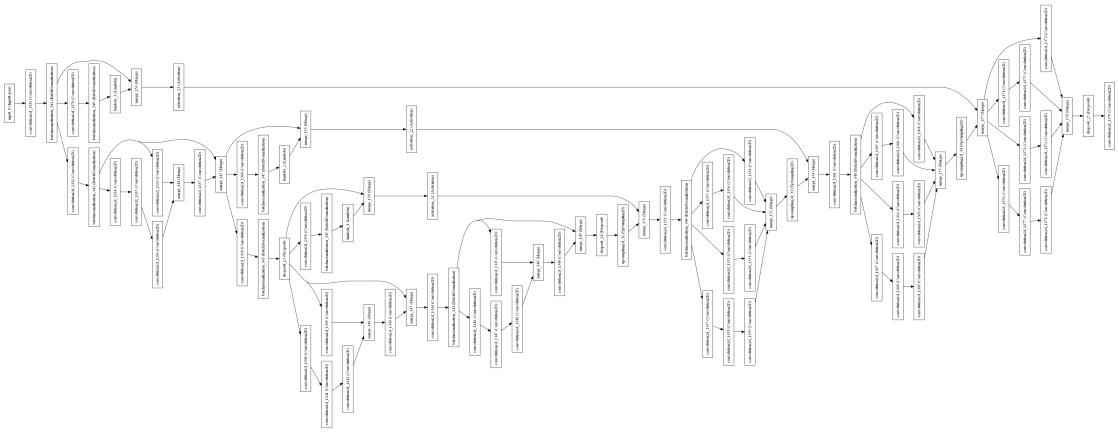


Figure 8.7: Complete U-Net-v1 model. This is the only model, where the complete graph is shown.

is [DVC+16], which investigated the correspondence of weight development over epochs with the usage of skip connections. Also the correlation between the weight development and the performance of the network is analyzed. Therefore three different architectures

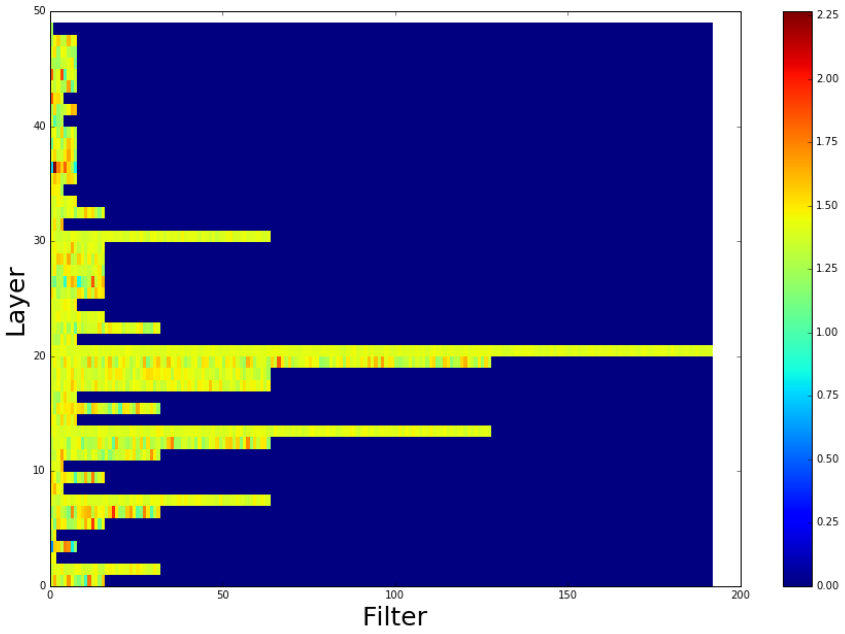


Figure 8.8: Norm of the filter weights for every convolutional layer

Layer	Amount in Model	Total Parameter Count
Convolution2D	49	527149
BatchNormalization	9	1216
InputLayer	1	0
Merge	15	0
Dropout	3	0
Lambda	3	0
UpSampling2D	3	0
Activation	3	0
In Total	86	528365

Table 8.1: U-net model specification.

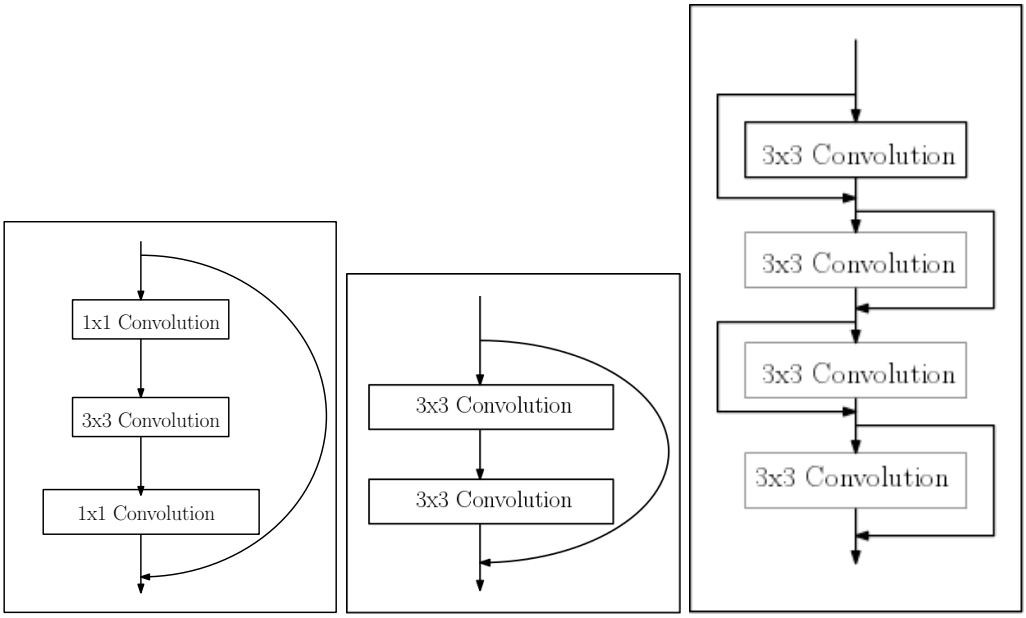
were developed using the following three different layer blocks:

1. ResUnet with Bottleneck Blocks [DVC+16]: As shown in Figure 8.9(a), a small 1×1 convolution is followed by a small 3×3 convolution. Afterwards a large 1×1 convolution follows. The insight, that large layers are often linear combinations of small layers, is applied here by using the final 1×1 convolutions. Additionally, 1×1 convolutions are used to shield computations and reduce the number of parameters.

2. WRN with Wide Residual Blocks [ZK16]: As shown in Figure 8.9(b), two large 3×3 convolutions are used. The key insight here is, that width is equally important as depth of the model and increasing both yields a higher accuracy per parameter than only increasing in width or in depth. In [ZK16], a streamlined residual layer block is introduced.

3. DenseNet with Dense Blocks [HLW16]: As shown in Figure 8.9(c) every layer is connected to every following layer leading to an ever increasing architecture. Using more residual connections helps with the gradient flow and allows the whole network to learn equally in each layer [HLW16]. Dense Blocks have already been applied to semantic segmentation in [JDV+16] with state-of-the-art results.

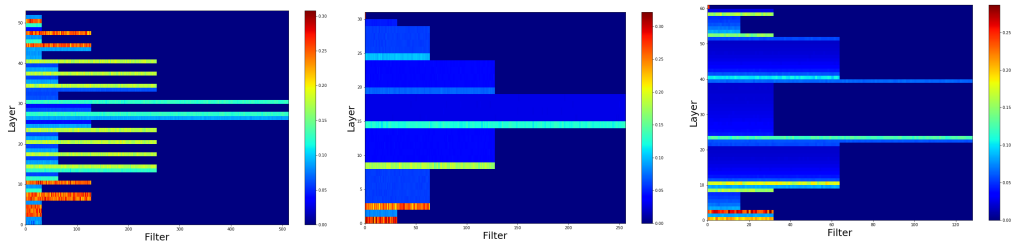
To give a sense about the final architectures, the norm of the filter weights for each layer is plotted in Figure 8.10. Therein every line represents one convolutional layer and the length of the line shows the number of filters in that layer. A stable weight distribution is needed for a network to learn well [DVC+16]. The networks are also nearly symmetric in their layers, due to the reuse of layer blocks on the downwards and upwards paths. One key characteristic of the WRN is the lower amount of convolutional layers and the larger amount of filters per layer compared to the other architectures. In contrast to the ResUnet, all of the filters are 3×3 convolutions. The DenseNet uses a merge layer per convolutional layer and therefore the visualization in Figure 8.10 is not indicative for the computational complexity required to run this model compared to the other architectures.



(a) (b) (c)

Figure 8.9: Shows the three layer blocks used in this thesis. The bottleneck block is shown in (a), the wide residual block in (b) and the dense block in (c).

The residual feature maps in a DenseNet are appended to the current feature maps, before the next convolutional layer is applied. In the ResUnet and the WRN the feature maps are added and therefore need to have the same input size as output size. Another interesting property of dense blocks is that the incoming feature map is passed through the entire block as well as a subset of the outgoing feature maps.



(a) (b) (c)

Figure 8.10: Shows the architectures developed using bottleneck blocks (a), wide residual blocks (b) and dense blocks (c).

A comparison of the three architectures based on their dice losses using the same amount of parameters of approximately 4.000.000 is given in Figure 8.11. The models are trained using step learning, which is the reason for the jumps in the loss function. As illustrated

in Figure 8.11, the wide residual connections proved to yield the best results.

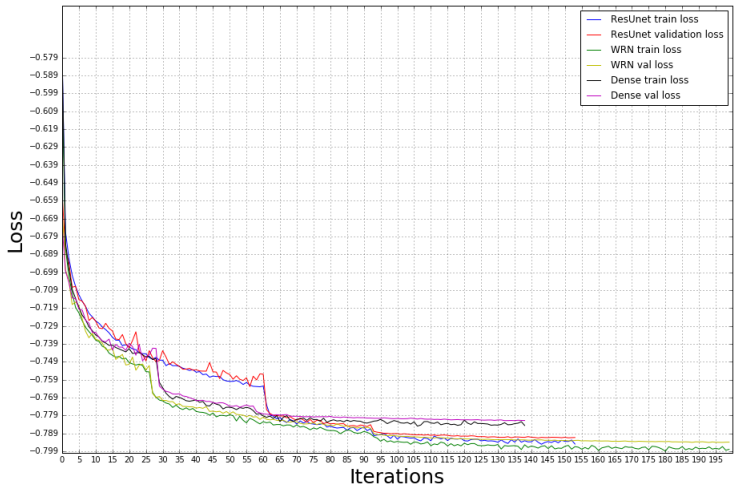


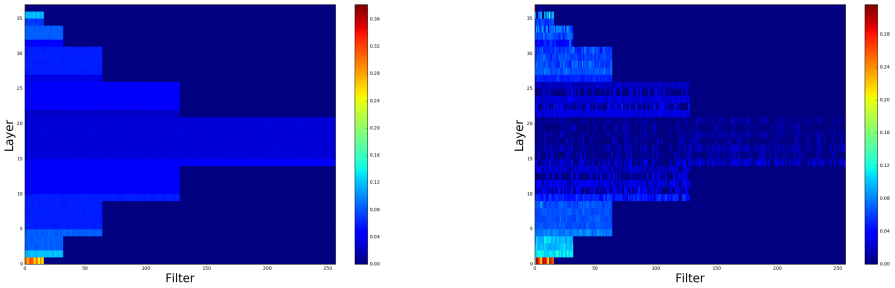
Figure 8.11: Model comparison between the three different models. The best performance is obtained by using wide residual connections.

8.2.3 Final Architecture

Based on the comparison, a final architecture was developed, named WRN-V2. This model is used to determine the final matching score on the fingerprint databases in Chapter 9.

Wide residual blocks are used, because they provided the best performance as shown previously. To be able to use 2.000.000 more parameters in the model and keep the batch size stable, four down-sampling and up-sampling levels are used instead of the three described in the abstract model. A visualization of the model is given in Figure 8.12. Therein the symmetric model is visualized. To preserve the amount of information in any cut separating the input to the output the amount of filters in a layer block is increased as the image size is decreased on the downwards path as illustrated in Figure 8.12. The first layer in such a block is either the down-sampling or the up-sampling convolution. Then every layer block uses two convolutions. Therefore the number of layer blocks in a level of the U-shape is countable by means of the number of layers with the same amount of filters. The first layer and the last layer are the only layers that use different kernel sizes for their filters and therefore their norms are higher than the other filters. The first layer uses a 5×5 kernel, while the last layer uses 1×1. 3×3 filters are used everywhere else. This gives a total of 37 Convolutional Layers and 6.048.997 parameters.

This model was then trained using cyclic learning with four steps and the learning rate is bounded in $[1e{-}6, 1e{-}4]$. Additionally, weight decay of $1e{-}6$ was used to combat



(a) (b)

Figure 8.12: Shows the WRN-V2 model pre training (a) and post training (b).

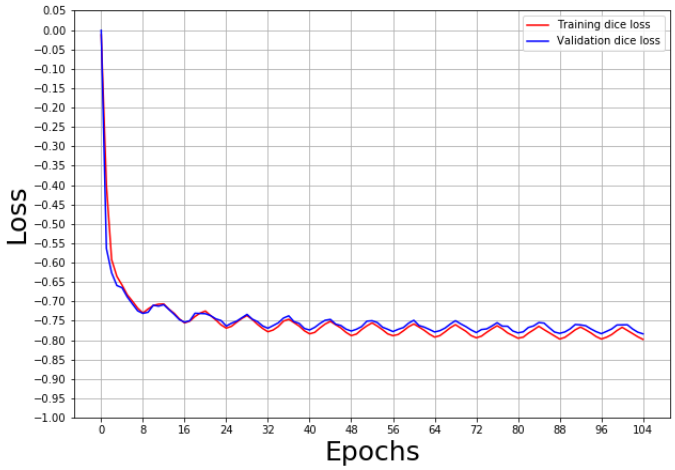


Figure 8.13: Training curve of the WRN-V2 network using cyclic loss.

overfitting by the model. The final training result is visualized in Figure 8.13. The resulting performance is similar to previous iterations, because this network is trained on data generated by the refinement network, instead of the augmented data used previously. The model shown in Figure 8.13 is used for the matching results in Chapter 9.

8.2.4 Failed Architectures

This is a short description of additional architectures that were programmed and tested, but did not produce the anticipated results. One such network would be the larger model based on the inception architecture. Another uses the ideas gathered in [XGD+16], where group convolutions are used. In this work it is proven that group convolutions are the same as aggregated residuals, which is what was used for this architecture. The insight

is, that very deep networks are possible using this kind of architecture. The weight visualization of that network is shown in Figure 8.14. The 1054 layer deep network is shown. Even with such a large amount of layers the model only has 2.213.057 parameters. Still, training this network took a day for every epoch instead of only 15 minutes for other architectures. Therefore this model was never trained to convergence.

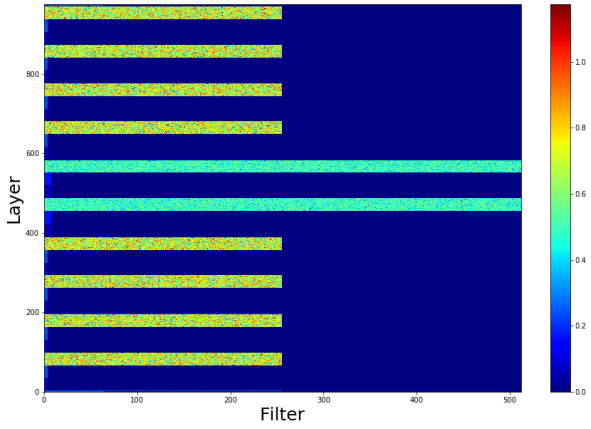


Figure 8.14: Weight visualization of the network using group convolutions as proposed in [XGD+16].

Additionally, an autoencoder was trained on patches without minutiae to learn how to encode and decode ridges. Then the idea is, that the autoencoder fails at reconstruction patches with minutiae in them with the same accuracy as on patches with minutiae in them. Then the patches with minutiae in them are detected by using a threshold on the reconstruction error. A 20% higher value than the average reconstruction error on patches without minutiae is chosen as a threshold. With this approach, an accuracy of 60% was achieved on detecting whether a patch has a minutiae in there or not. This is not even close to the performance acquired by the minutiae extraction framework as shown in Chapter 9.

Another approach was using the Unet-v1 architecture on a regression problem, where the minutia point itself had the largest value and the surrounding area had a decreasing value. That model produced a completely noisy image and failed to learn a meaningful mapping.

8.3 SimGAN Networks

In this section, the architecture of the SimGAN network is explored. The framework used in this thesis is largely based on [SPT+16]. Additionally, the insights gained by training minutiae networks are applied to that refinement network. The SimGAN is

a combination of two networks, namely the refinement network and the discriminator. Both can have vastly different architectures. The combination of the networks depends on the actual architectures of the networks used. The interaction between those two networks in the training phase is illustrated in Figure 8.15. Anguli generates the initial fingerprint images denoted as input. For training the refinement network the minutiae ground truth for the fingerprint images is unnecessary. The Hessian on the input images is calculated in advance by applying two Sobel filters on the image and are also used as input to the refinement network. Given the generated input images by Anguli, the Hessian of those images and a real fingerprint dataset the algorithm works as follows:

1. Freeze the weights of the discriminator and use the discriminator as a loss function for the refinement network. The refinement network uses the same number of generated samples as input as the discriminator.
2. Generate refined samples for the discriminator from anguli generated input images on which the refinement network was not trained on.
3. Update the discriminator with equal amount of generated data by the refinement network and real fingerprint images.
4. Repeat Step 1, 2 and 3 for 20 epochs.

One issue that appeared while using this algorithm is the difference between the training and the predict algorithm. Because batch normalization is only applied, while training the network the output of the predict algorithm is different to the one used to train the network. This has implications in having the discriminator network discriminate between a different value spread in training and as part of the adversarial training. This lead to the discriminator being 100% confident in discriminating the fake and the real samples and the SimGAN was 100% confident in fooling the discriminator.

TODO: read again

8.3.1 Refinement Network

The refinement network proposed in this thesis uses the same underlying abstract architecture as the minutiae extraction models. Therefore a U-net FCNN was used to refine the fingerprints. In [SPT+16], a refinement network was used for equally sized images and in there they did not make use of an U-shape and just used a succession of convolutional layers. The problem with not using a U-shape is, that for the same amount of parameters the run time is longer and the memory requirements for the GPU are larger. On-the-fly refinements are used. Therefore the runtime of the refinement network has an impact on training the minutiae extraction network. Therefore lower runtime is preferable. Because of the superior performance of wide residual blocks on the minutiae extraction task, they are also used for the refinement parts of the model. In essence, it is a smaller version of the WRN network proposed earlier.

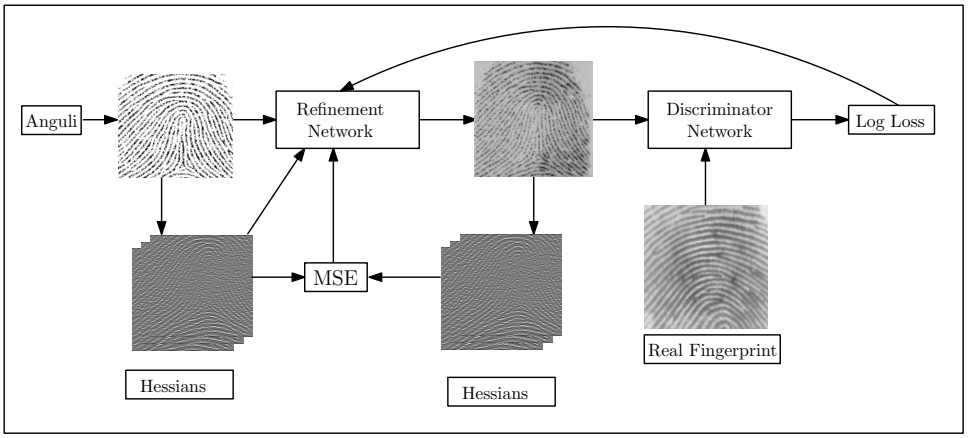


Figure 8.15: A visualization of the training phase of the SimGAN approach. Anguli generates the input fingerprint images, while the refinement network produces refined versions of the same fingerprint. The refinement network is trained on the log loss of the discriminator on the refined fingerprints and the mean squared error between the Hessians of the input and refined fingerprint. After every iteration of the refinement network the resulting refined fingerprints and a batch of real fingerprints are combined and used to train the discriminator network.

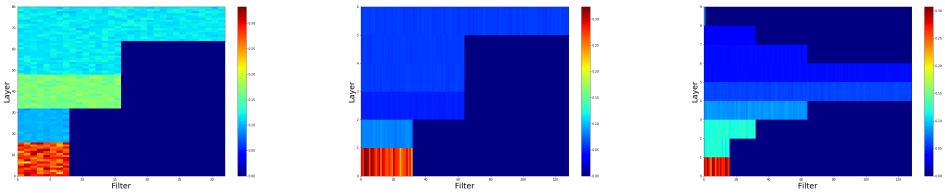
8.3.2 Discriminator

To determine the realness of the refined images another neural network is used. This is trained against the output of the images generated by the refinement network and the real images to reliably distinguish between them. The discriminator network proposed in [SPT+16] uses local loss. Here, this architecture is compared to a simple discriminator and an energy based approach using an autoencoder as network architecture [ZML16]. All three models are visualized in Figure 8.16.

For the autoencoder architecture, a convolutional autoencoder is used, similar to the U-shaped network, without long skip connections. The problem with a traditional autoencoder is the large amount of parameters needed to encode and decode an image of size 224×224. Using the traditional autoencoder to get to 100 neurons encoding, 5.017.600 parameters are needed in the single dense layer. The decoder part has the same numbers of parameter, making this network larger than the minutiae extraction networks. Therefore a fully convolutional neural network is used as autoencoder as illustrated in Figure 8.16(c).

The simple discriminator and the local loss networks are very similar. Both were modeled using a combination of convolutional layers followed by max pooling layers. The model in Figure 8.16(b) uses a single larger one to determine if this fingerprint is real. The local loss architecture uses the same principle with less filters per layer as illustrated in Figure 8.16(a). However the local loss discriminator uses 16 of those simple models. Each of these models is supposed to judge if this batch is part of a real fingerprint. The first

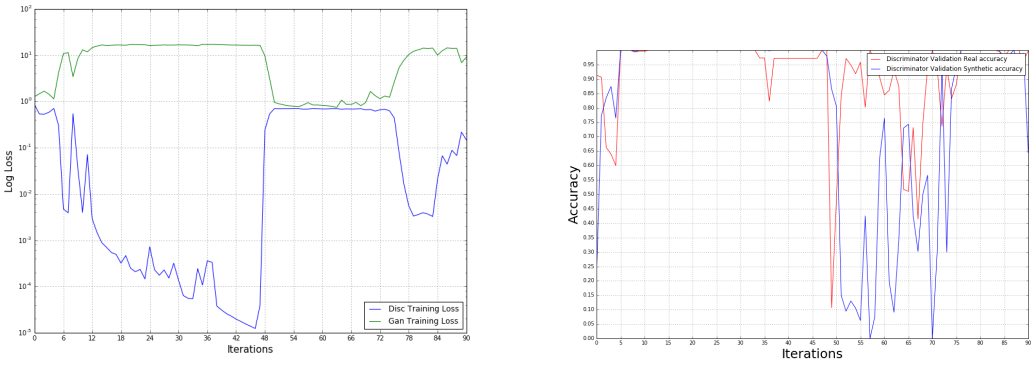
layer of such a model is a split layer, which splits the tensor into their local parts and feeds each part into one of the 16 models. This has the advantage of learning for every image patch separately, if this path belongs to a real fingerprint image.



(a) (b) (c)

Figure 8.16: Shows the norm of the weights in every convolutional layer of the models. In (a) a simple discriminator is shown, while in (b) a discriminator working on patches is shown. The illustration in (c) shows the autoencoder approach.

In Figure 8.17, the training loss and the resulting accuracy of the discriminator and the SimGAN are demonstrated. The problem of training such a network is visualized by showing that the loss jumps arbitrarily. This is the intended behavior of the SimGAN, because the training of the discriminator increases the loss of the generator and vice-versa. Even after the loss converges for both networks, the loss diverges again shortly afterwards. It is also hard to tell, when the model is actually converged, because both loss functions decrease and increase.

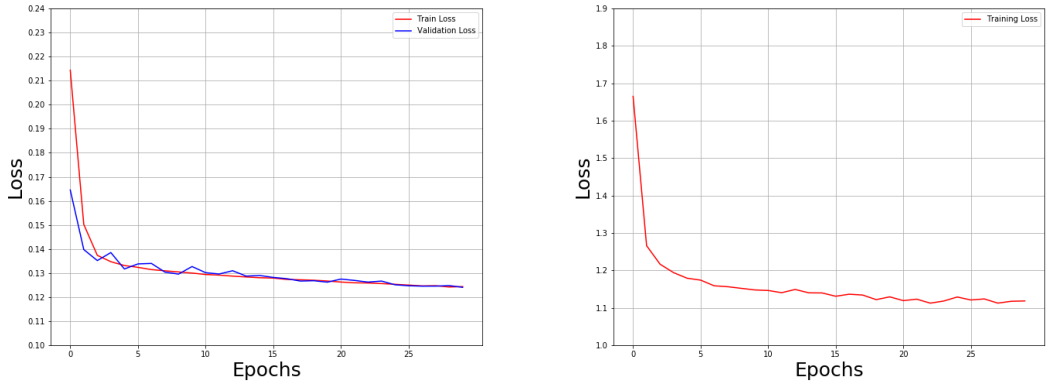


(a) Log Training loss of GAN. (b) Accuracy of the discriminator.

Figure 8.17: Visualization of the Training process of training the SimGAN.

In contrast, by using the energy based approach those arbitrary jumps are eliminated. First the discriminator is trained on its own on the real dataset and penalized by the reconstruction error as shown in Figure 8.18(a). Then the generator is trained using the discriminator as a loss function as shown in Figure 8.18(b). Therefore this approach

is very similar to training any other network in literature, because the training of one network has no impact on the other network. However, the results were visually not as good with this method as with using the local loss.



(a) (b)

Figure 8.18: Visualization of the Training process using the energy based approach. In (a) the discriminator is trained, while in (b) the generator is trained.

CHAPTER 9

Implementation: Evaluation and Visualizations

This chapter shows the fingerprint matching performance and further visualizations of the methods described previously. The evaluation uses metrics, which are described in Chapter 5. First the F-score on the generated dataset is shown. Then the minutiae extraction algorithm is evaluated based on the matching score on several fingerprint databases. For the purposes of this chapter the algorithm proposed in this work is called U-net FCNN. Next is a section about the visualization of the network intermediate outputs to provide insight into the inner structure of the neural networks. Also special images and interesting outputs are explored.

9.1 Evaluation on Synthetic Fingerprints

Annotated fingerprints were evaluated using two metrics. First we evaluated the amount of minutiae found using the F-Score and second the Equal Error Rate (EER) was monitored while matching the fingerprints as described in Chapter 5.

The F1 Score is defined:

$$\text{F-Score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}, \tag{9.1}$$

where Precision and Recall are defined in Equations 9.2 and 9.3, respectively.

$$Precision = \frac{TP}{TP + FP} \tag{9.2}$$

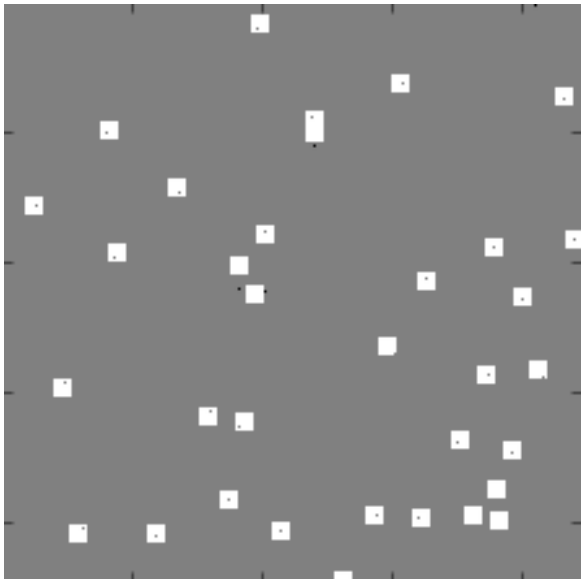


Figure 9.1: The points are the detected minutiae points and the white regions represent the ground truth.

$$Recall = \frac{TP}{TP + FN} \tag{9.3}$$

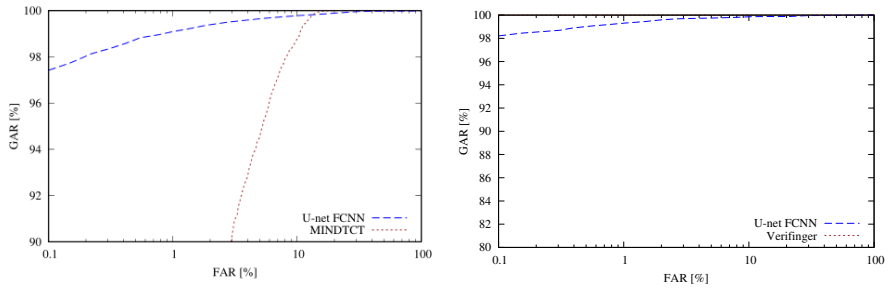
For our purposes, a True Positive (TP) is defined as a minutia point that is in a ground truth minutia region. A minutiae region is defined as a 6×6 region around a minutiae point. Visually this is argued, that a minutia point in the image does not correspond to a single pixel in the image, but a collection of pixels. 6×6 regions are used, because it gave us the highest accuracy while training the network. Experiments with 1×1 and 3×3 regions have also been conducted, but the accuracy dropped by 60% and 35% respectively. A minutia point that is not in a region is called a False Negative (FN). Finally, a False Positive is a minutia region without a minutia point in it. This is illustrated in Figure 9.1. The F-Score on the Anguli generated dataset, calculated using this method is **0.86**.

9.2 Fingerprint Matching

The main goal of this work is robust biometric authentication using fingerprints. The minutiae extraction is only one step in this process. Most important about the extracted minutiae is how well they are suited for fingerprint matching. For this reason, it is important to robustly detect the same minutiae points in different impressions of the same finger. Thus the EER was calculated on a test set using the BOZORTH3 [WGT+07] matcher.

9.2.1 Synthetic Fingerprints

Anguli generated test set was constructed for testing the algorithm. 10.000 true matches and 10.000 false matches are used to determine the EER. In Figure 9.2(a) the original proof of concept is shown. Using the U-net-v1 model, as described in Chapter 8, the error rate is already lower then for MINDTCT, which for this experiment was used to generate the ground truth minutiae for training, instead of Verifinger due to the fact, that it took some time until access to the Verifinger SDK was granted. This experiment shows, that it is possible to learn a better minutiae extractor, than the one supplying the ground truth and that the network does not only learn the operations used by the ground truth generator. The second experiment shown in Figure 9.2(b), which compares the approach proposed in this thesis to Verifinger prompted the switch to Verifinger as the ground truth generator, because of its superior results. Verifinger is able to match the Anguli fingerprints to each other without making any mistakes and is therefore hidden in Figure 9.2(b) by the black figure border.



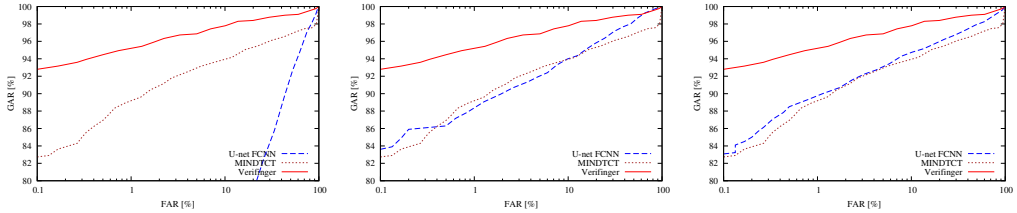
(a) (b)

Figure 9.2: Equal error rate plot using the U-net-V1 network on data generated by Anguli compared to MINDTCT (a) and Verifinger (b). Note, that Verifinger makes no mistake for this dataset and the line is at 100%.

9.2.2 FVC 2000 Databases

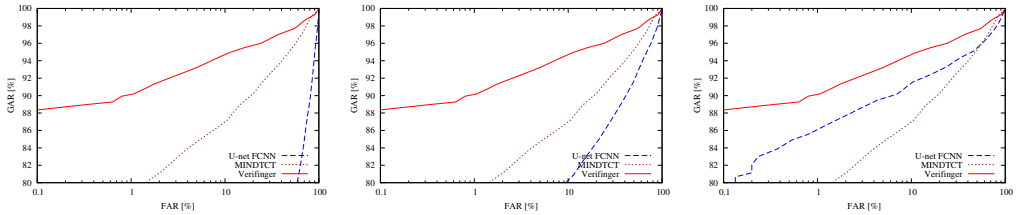
A comparison to the state-of-the-art is made in Figure 9.3 and 9.4. The figures show EER plots. Given a False Acceptance Rate (FAR) the Global Acceptance rate is plotted. The benchmark number using such a plot is the EER. This is the error rate where the $1 - FAR = GAR$ holds. Using these plots it is shown that the proposed algorithm does not set a new state-of-the-art and is beaten on by the commercial fingerprint minutiae extractor Verifinger [Ver10], but it is consistently better than MINDTCT. Also training with refinements leads to consistent improvements in the matching scores.

As shown in Figure 9.4 the improvements obtained by using refinements on the DB 3 dataset are higher than on the DB 1 dataset. Here the U-net FCNN approach is vastly better then the MINDTCT algorithm, but worse than Verifinger.



(a) (b) (c)

Figure 9.3: Matching scores compared to the state-of-the-art with the WRN-V2 model trained with (a) Anguli data, (b) augmented data, (c) refined data on the FVC2000 DB 1 dataset.



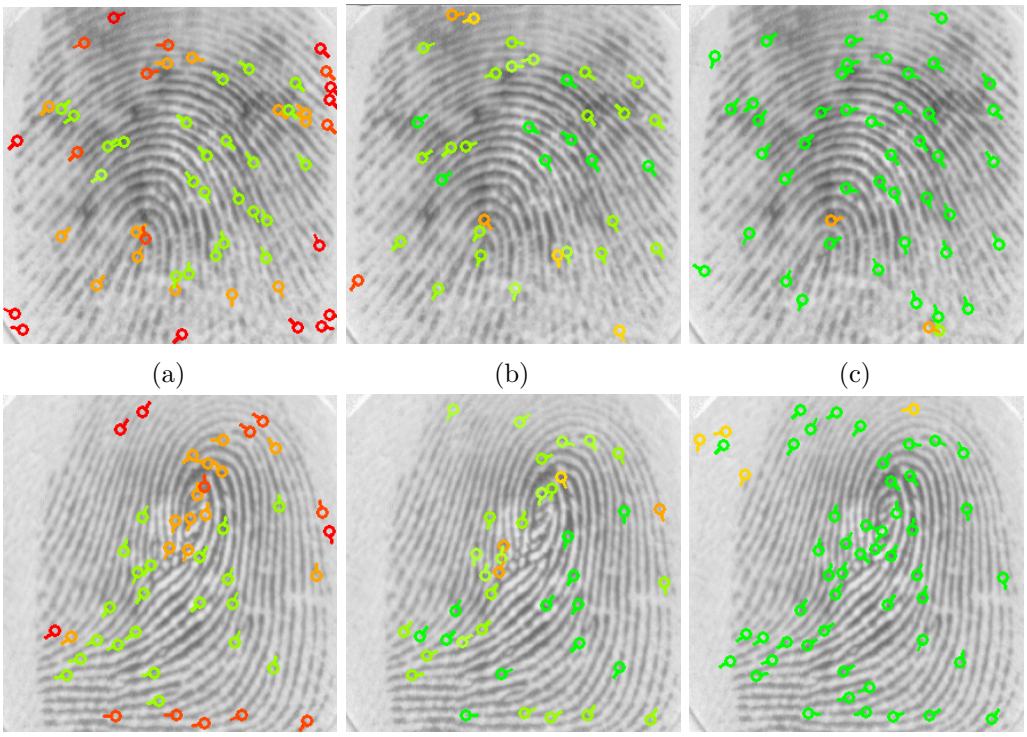
(a) (b) (c)

Figure 9.4: Matching scores compared to the state-of-the-art with the WRN-V2 model trained with (a) Anguli data, (b) augmented data, (c) refined data on the FVC2000 DB 3 dataset.

Sample outputs for the DB 1 dataset are shown in Figure 9.5. The actual minutiae positions are remarkably similar using all three approaches. Still, there are some slight differences in the center of the fingerprint and on the margins. One thing that is different is the quality estimate in the images. The algorithm proposed in this thesis nearly always outputs good quality, compared to the other approaches. Considering the quality estimation there are also some differences between the MINDTCT and the Verifinger algorithm. MINDTCT finds wrong minutiae points at the borders of the image of low quality. Verifinger and the FCNN do not find those minutaie points, which is desirable.

The orientation estimations are even more different. MINDTCT and Verifinger differ in their orientations by 180 degrees as illustrated in Figure 9.5(a,b). The U-net FCNN approach outputs a combination of both and is not actually consistent, which leads to a decrease in EER. This shows the weakness of the orientation extraction algorithm.

Sample results produced by the minutiae network for the DB 3 dataset are shown in Figure 9.6. The fingerprint images are different in values for ridges and valleys, lightning condition and rotation to the ones shown in Figure 9.5. In those images, MINDTCT finds low quality minutiae in the background. This time even Verifinger finds wrong minutiae points in regions with low contrast as in Figure 9.6(e) at the bottom of the fingerprint. The U-net FCNN approach is able to ignore those regions. However regions



(a) (b) (c)

(d) (e) (f)

Figure 9.5: Comparison of the outputs for a fingerprint of the FVC2000 DB 1 dataset using MINDTCT (a,d), Verifinger (b,e) and the U-net FCNN (c,f).

with changing rigde frequency are misclassified as minutiae as shown in Figure 9.6(f).

9.2.3 UareU Database

For this dataset, the refinement network was not trained using the UareU dataset. This is done to get a stable estimate of the test performance of the algorithm, without having seen the data in the training phase. The FVC 2000 DB 1 dataset was used instead as a real dataset for the refinement training. Still there is a significant matching gain by using the refinements. It shows the general applicability of the refinement network compared to manual augmentations. While the traditional augmentations also worked comparably to the refinement network for the FVC 2000 databases, there is only a 1.78% improvement by using the manual augmentations to using only the anguli generated data. In comparison by using the refinement network the EER is improved by 9.81%.

9.2.4 Different Orientation Configurations

Figure 9.8 is an illustration on the importance of the angle extraction algorithm. As shown therein, the performance of the orientation field algorithm severely outperforms

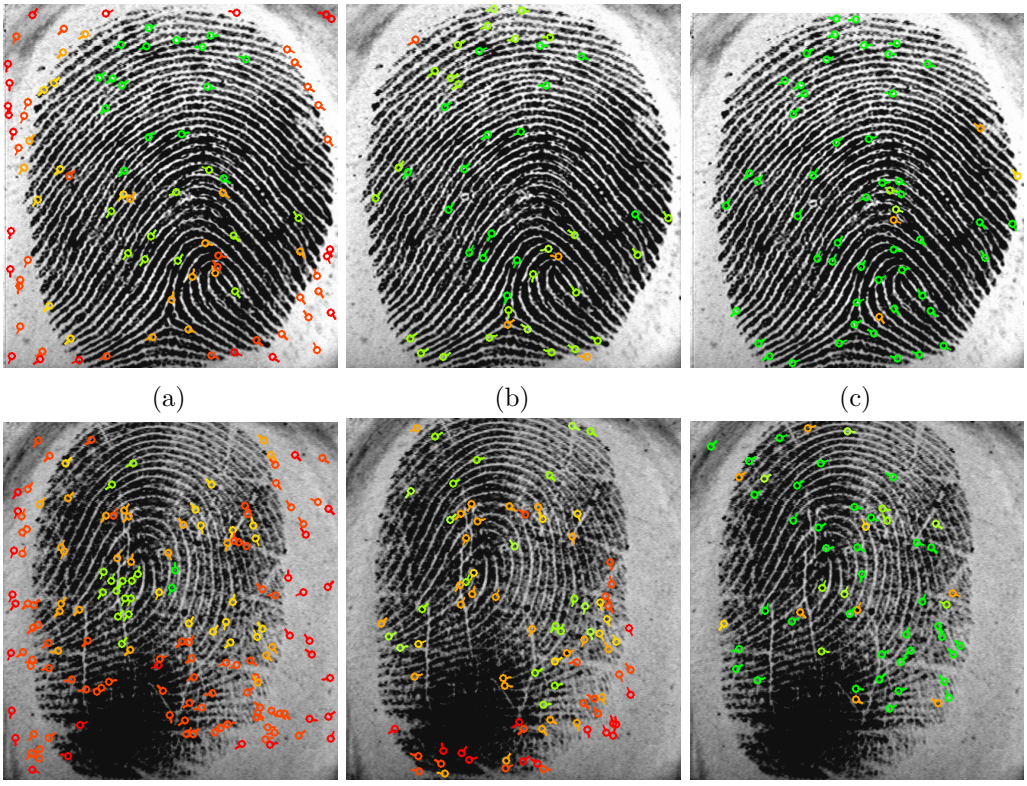


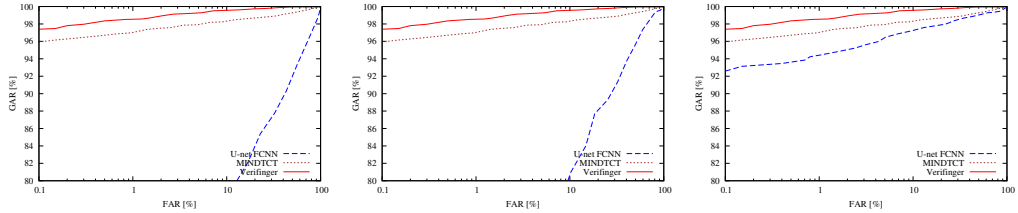
Figure 9.6: Comparison of the outputs for a fingerprint of the FVC2000 DB 1 dataset using MINDTCT (a,d), Verifinger (b,e) and the U-net FCNN (c,f).

the algorithm, which assigns the same orientation to all of its minutiae points, based on the matching score. The FVC2000 dataset used for this evaluation has only little rotation in its fingerprints and the matching performance decreased drastically by not using the angle information.

In this thesis two approaches to the angle extraction problem are proposed, namely using points based versus field based orientation extraction. The comparison is shown in Figure 9.9. Therein the actual EER of the point based approach is worse than the one using the orientation field. However, the point based extraction algorithm is more stable in its performance and outputs a lower error rate, if wrong matches are not permitted.

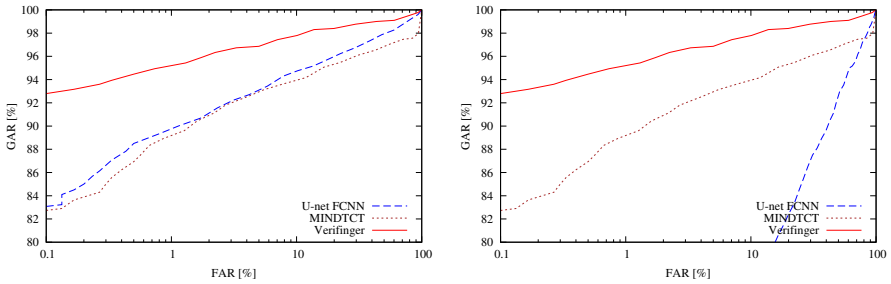
9.2.5 Results for Specific Number of Output Crops

The output algorithm uses multiple crops of the image as described in Chapter 8. A comparison of the matching scores is given in Figure 9.10. Even though the difference is not that significant, there is an improvement concerning the EER as shown in Table 9.1.



(a) (b) (c)

Figure 9.7: Matching score compared to the state-of-the-art with the WRN-V2 model trained with (a) Anguli data, (b) augmented data, (c) refined data on the UareU Sample DB [Ver10] dataset.



(a) EER plot with the orientation field extraction algorithm.

(b) EER plot with every angle set to 0 degree.

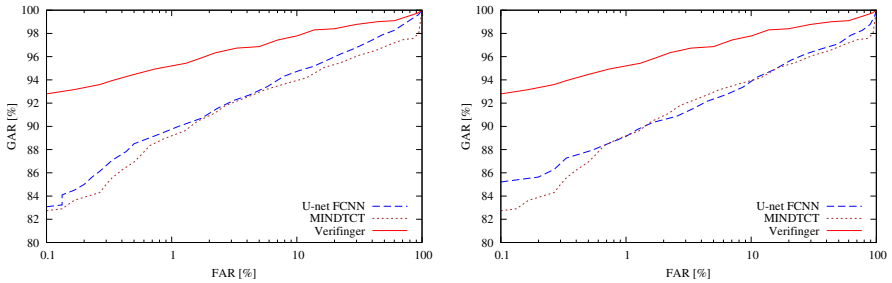
Figure 9.8: Comparison of the Equal error rate with orientation and without on the FVC2000 DB1 dataset.

Method	FVC2000 DB1 EER	Average Acquisition Time in sec
CNN 8 Crops	9.41%	0.022
CNN 16 Crops	8.99%	0.035
CNN 32 Crops	8.86%	0.061

Table 9.1: Table showing the difference in EER by using different amounts of crops

9.3 Results for Specific Input Data

The output of the neural networks for specific input data is presented. The minutiae extractor developed in this thesis, also manages to find minutiae in images not showing fingerprints and might therefore be fooled by synthetic images. Input data ranges from generated data to natural images. This section is also used to highlight the faults made by the proposed approach and the limitations of the neural network approach.



(a) EER plot with the orientation field extraction algorithm.

(b) EER plot with the point based angle extraction algorithm.

Figure 9.9: Comparison of the EER with point versus orientation field estimation on the FVC2000 DB1 dataset.

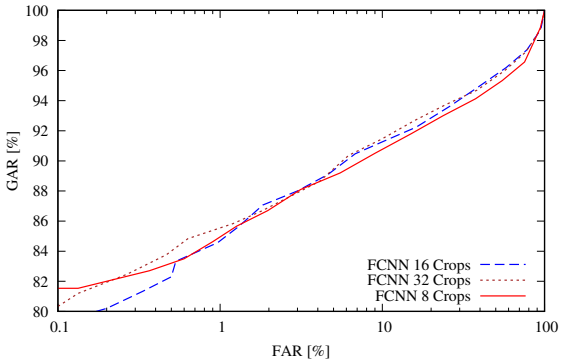


Figure 9.10: EER plot comparing different crop strategies.

9.3.1 FVC2000

In Figure 9.11 the output of the U-net FCNN is visualized. Because the model only works on resolutions of 224×224, crops of the images are used to output results. In Figure 9.11(a) the original fingerprint is illustrated, while (b) shows the output of the cropping process and the input to the network. The network output without any post-processing is visualized in Figure 9.11(c). This already corresponds to the minutiae positions as is shown by overlaying this output over the fingerprint in Figure 9.11(d).

9.3.2 Natural Images

To illustrate the behavior of the network it was also evaluated on natural images like a tree in Figure 9.12. A tree was chosen because the ridges of a tree are similar to minutiae. As shown in Figure 9.12 minutia can also be found in natural images as well and a fingerprint matcher can be fooled by using the same tree twice.

(a) (b) (c) (d)

Figure 9.11: Visualization of the output of the network on a sample fingerprint taken from the FVC2000 databases. (a) shows the whole fingerprint image, while (b) shows the network input. In (c) the network output is visualized and the output is then overlayed on (b) to produce (d).

(a) (b)

Figure 9.12: Visualization of the output of the network for a tree image. The original image is shown in (a). The output of the network displayed on top of the rotated grayscale tree image is illustrated in (b).

9.3.3 Empty Image

Interestingly, using an empty image resulted in found minutiae as shown in Figure 9.13.

By analyzing the inner outputs of the neural network the zero padding in combination with the bias values in the convolutions was responsible for adding structures to the image. Then batch normalization distributes the values through the image and structure

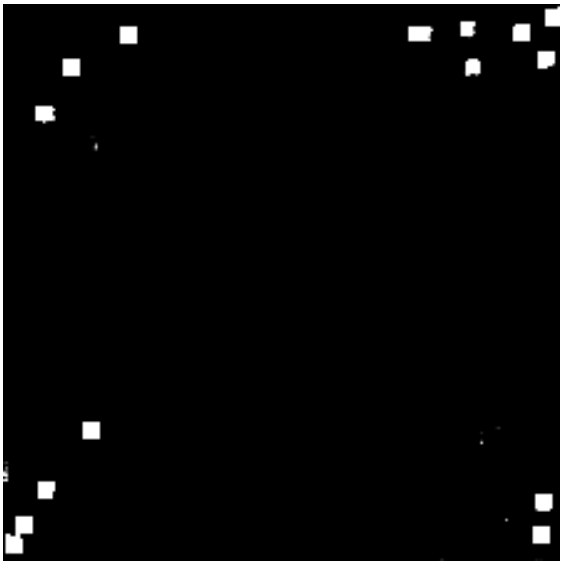


Figure 9.13: Output of U-net FCNN on an empty image.

is formed as shown in Figure 9.14. In the final layer, the network predicts that minutiae are in the image. This example is used to showcase the volatile nature of the neural network approach.

Figure 9.14: Shows the sample inner outputs of the neural network on an empty image. The first image shows, how the bias and zero padding add ridges into the inner images of the network. Then through batch normalization structure in the images is formed. This structure is then later classified as minutiae.

9.3.4 Intermediate Output Visualization

In Figure 9.15, the intermediate layer ouptuts of the network are illustrated. At the beginning, the network learns the orientation of the image. As previosly mentioned the orientation constitutes to the ridge pattern. This is similar to the algorithm used in [HWJ98]. Responses produced by the ridges are amplified. Later, the ouptut is transformed into a height map, where circles denote minutiae regions. Those circles are then slowly transformed into minutiae regions and form the final output. The network used for this viusalization is the U-net-v1 trained with dice loss.

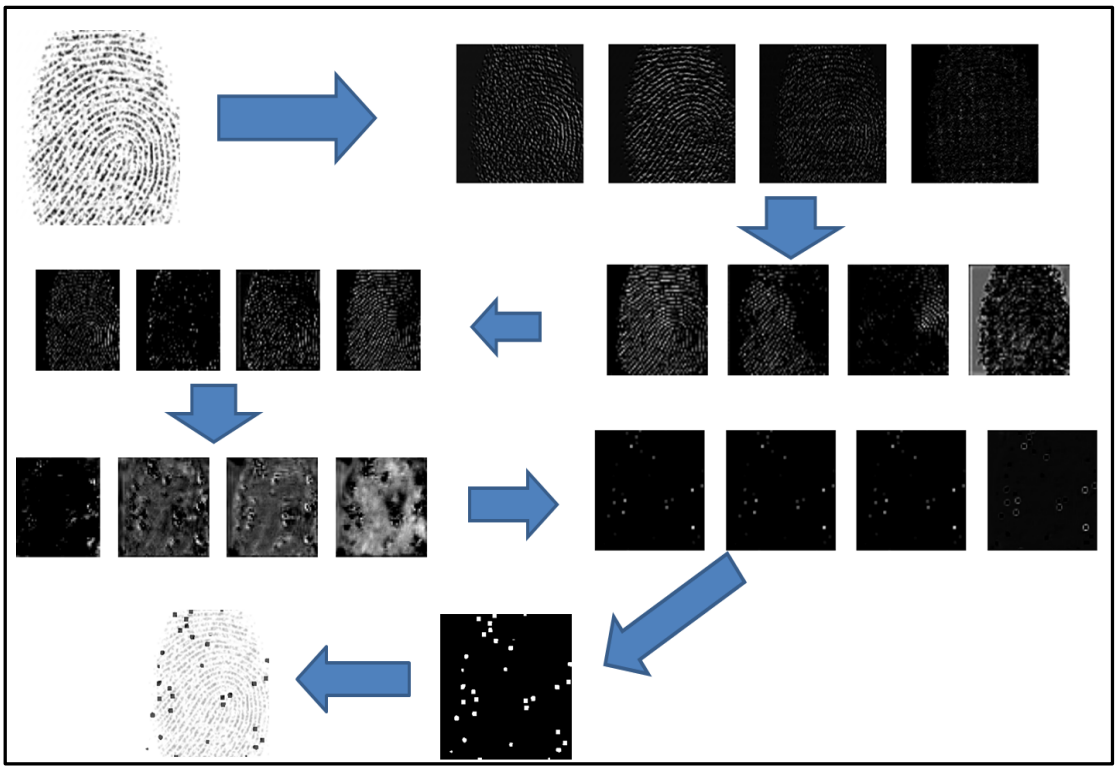


Figure 9.15: Intermediate layer visualization of layers 5 -> 15 -> 25 -> 35 -> 45 -> 50.

CHAPTER 10

Conclusions

In this thesis, a novel approach using a FCNN for the minutiae extraction problem is shown, following the idea to solve an equivalent binary segmentation problem. Current synthetic fingerprint generators are used to provide the training data and the corresponding clean ridge patterns. The annotations are generated by current matching technology, following the assumption, that current minutiae extractors are perfect on good quality fingerprints. In this thesis the assumption is proven, that a better minutiae extractor can be learned by using this data, than the minutiae extractor responsible for generating the data on that data. Further, it is shown in this thesis, that the performance translates to other real datasets with competitive results to existing solutions.

Also, the reliance on good training data and why existing synthetic fingerprint generators are inadequate to learn a good minutiae extractor are demonstrated. For this reason, the novel idea of learning suitable augmentations based on adversarial training to refine synthetic fingerprint images while preserving annotation data is introduced. This enables the generation of synthetic fingerprints, which are indistinguishable to real data and are thus used for training. A clear performance improvement is reported, by using the refined fingerprints instead of augmented fingerprints for training. Vastly different refiners are learned using different combinations of hyper parameters and real datasets, solving the problem of current fingerprint generators of only generating similar fingerprint images.

Additionally, the state-of-the-art in binary segmentation was discussed and used to propose a new architecture suited for the minutiae extraction problem. Improvements to the architecture are done by using insights found by trying out different combinations of hyperparameters. Concepts like cyclic learning or dropout were applied to the fingerprint minutiae extraction problem to check the applicability to this new use-case.

The initial research question was to improve on the state-of-the-art in fingerprint matching by improving the performance on the minutiae extraction task. While this result was not achieved in this thesis, comparable performance to other state-of-the-art minutiae

extractor was shown with our method. Open issues remain, as discussed in Section 10.1, which are likely to improve the accuracy of the minutiae extractor. However, a clear improvement to synthetic fingerprint generations is reported by using adversarial training with deep NNs. Using this method, fingerprints were generated, which were better suited for training a minutiae extraction network. This method is entirely data driven and can therefore be applied to other domains as well. This approach is used to generate data indistinguishable from a real dataset by a NN. A ground truth is provided for the generated data by the algorithm. Therefore, the need to annotate data is severely diminished as shown in Chapter 9. This method is likely to improve the accuracy of deep NN on domains where labeled data is scarce, like biomedical image segmentation [DVC+16].

10.1 Remaining Issues

The output of the neural network is post-processed using the position, orientation and quality extraction algorithms. Those algorithms, as shown in Chapter 9 do have issues and limitations. The position estimation can not deal with two overlapping minutiae regions and therefore combines them to one wrong minutiae. The orientation estimation algorithm uses local information about minutiae points to decide if the angle is between $0° - 180°$ or $180° - 360°$. This decision is not as consistent as existing solutions as shown in this thesis. Finally for the quality extraction algorithm outputs the same quality for 95% of the minutiae, which implies that the factor is too high.

Another issue is that the initial minutiae map provided by the Verifinger SDK, which is used for training the minutiae extraction network, is not perfect. This could be improved upon by using an initial fingerprint generator, which provides native access to the generated minutiae.

10.2 Future Work

Currently, the proposed architecture is similar to current state-of-the-art models in binary segmentation. Therefore the model will be compared to other state-of-the-art models on a semantic labeling benchmark in a future work to look at the general scalability of said network.

The minutiae extraction algorithm still has problems with the orientation extraction algorithm. A future topic could be to use the minutiae extraction network to also output the orientation field. Additionally, the final decider to use the orientation between 0° and 180° or between 180° and 360° could also be learned.

Another interesting future topic would be to try using the learned model for matching directly. This would be done by using a siamese network architecture, where two feature extraction networks are combined with a classifier on top. It would be interesting if we get better performance than by using an existing matcher.

Finally, collecting more real data is useful for both the minutiae matcher and the refinement training to generate better results.

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[AB17] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[ABGM14] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *ICML*, pages 584–592, 2014.

[ACL81] Carlo Arcelli, Luigi P Cordella, and Stefano Levialdi. From local maxima to connected skeletons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2):134–143, 1981.

[Ans11] Afzalul Haque Ansari. *Generation and storage of large synthetic fingerprint database*. PhD thesis, Indian Institute of Science Bangalore, 2011.

[ARAA+16] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat,

Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[Ash99] David R Ashbaugh. *Quantitative-qualitative friction ridge analysis: an introduction to basic and advanced ridgeology*. CRC press, 1999.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[ÇAL+16] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 424–432. Springer, 2016.

[CCG07] Sharat Chikkerur, Alexander N Cartwright, and Venu Govindaraju. Fingerprint enhancement using STFT analysis. *Pattern Recognition*, 40(1):198–211, 2007.

[CFFM07] Raffaele Cappelli, Matteo Ferrara, Annalisa Franco, and Davide Maltoni. Fingerprint verification competition 2006. *Biometric Technology Today*, 15(7):7–9, 2007.

[CGM+17] Pedro Costa, Adrian Galdran, Maria Inês Meyer, Michael David Abràmoff, Meindert Niemeijer, Ana Maria Mendonça, and Aurélio Campilho. Towards adversarial retinal image synthesis. *arXiv preprint arXiv:1701.08974*, 2017.

[Cho15] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[Cho16] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[CLJ14] Kai Cao, Eryun Liu, and Anil K Jain. Segmentation and enhancement of latent fingerprints: A coarse to fine ridgestructure dictionary. *IEEE transactions on pattern analysis and machine intelligence*, 36(9):1847–1859, 2014.

[CMM02] Raffaele Cappelli, Dario Maio, and Davide Maltoni. Synthetic fingerprint-database generation. In *16th International Conference on Pattern Recognition, 2002. Proceedings.*, volume 3, pages 744–747. IEEE, 2002.

[CMM04] Raffaele Cappelli, D Maio, and D Maltoni. Sfinge: an approach to synthetic fingerprint generation. In *International Workshop on Biometric Technologies (BT2004)*, pages 147–154, 2004.

[DCF+15] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[DVC+16] Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The importance of skip connections in biomedical image segmentation. In *International Workshop on Large-Scale Annotation of Biomedical Data and Expert Label Synthesis*, pages 179–187. Springer, 2016.

[FRS+16] Alexander Freytag, Erik Rodner, Marcel Simon, Alexander Loos, Hjalmar S Kühl, and Joachim Denzler. Chimpanzee faces in the wild: Log-euclidean cnns for predicting identities and attributes of primates. In *German Conference on Pattern Recognition*, pages 51–63. Springer, 2016.

[Fuk79] Kunihiko Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position- neocognitron. *ELECTRON. & COMMUN. JAPAN*, 62(10):11–18, 1979.

[Gal92] Francis Galton. *Finger prints*. Macmillan and Company, 1892.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GJ16] Abhishek Gangwar and Akanksha Joshi. Deepirisnet: Deep iris representation with applications in iris recognition and cross-sensor iris recognition. In *IEEE International Conference on Image Processing (ICIP), 2016*, pages 2301–2305. IEEE, 2016.

[Got12] Carsten Gottschlich. Curved-region-based ridge frequency estimation and curved gabor filters for fingerprint image enhancement. *IEEE Transactions on Image Processing*, 21(4):2220–2227, 2012.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[HLW16] Gao Huang, Zhuang Liu, and Kilian Q Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[HSK+12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[HWJ98] Lin Hong, Yifei Wan, and Anil Jain. Fingerprint image enhancement: algorithm and performance evaluation. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):777–789, 1998.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[JDV+16] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. *arXiv preprint arXiv:1611.09326*, 2016.

[JSD+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

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

[Kit14] Rob Kitchin. The real-time city? big data and smart urbanism. *GeoJournal*, 79(1):1–14, 2014.

[KL51] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[KS06] Adam R Klivans and Alexander A Sherstov. Cryptographic hardness for learning intersections of halfspaces. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 553–562. IEEE, 2006.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KTR16] Pejman Khadivi, Ravi Tandon, and Naren Ramakrishnan. Flow of information in feed-forward deep neural networks. *arXiv preprint arXiv:1603.06220*, 2016.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LLS15] Fayao Liu, Guosheng Lin, and Chunhua Shen. Crf learning with cnn features for image segmentation. *Pattern Recognition*, 48(10):2983–2992, 2015.

[McC04] R Michael McCabe. Fingerprint interoperability standards. In *Automatic Fingerprint Recognition Systems*, pages 433–451. Springer, 2004.

[MMC+02a] Dario Maio, Davide Maltoni, Raffaele Cappelli, James L. Wayman, and Anil K. Jain. Fvc2000: Fingerprint verification competition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):402–412, 2002.

[MMC+02b] Dario Maio, Davide Maltoni, Raffaele Cappelli, James L Wayman, and Anil K Jain. Fvc2002: Second fingerprint verification competition. In *Pattern recognition, 2002. Proceedings. 16th international conference on*, volume 3, pages 811–814. IEEE, 2002.

[MMC+04] Dario Maio, Davide Maltoni, Raffaele Cappelli, Jim L Wayman, and Anil K Jain. Fvc2004: Third fingerprint verification competition. In *Biometric Authentication*, pages 1–7. Springer, 2004.

[MMJP09] Davide Maltoni, Dario Maio, Anil Jain, and Salil Prabhakar. *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009.

[MNA16] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 565–571. IEEE, 2016.

[Moe71] Andre A Moenssens. *Fingerprint techniques*. Chilton Book Company London, 1971.

[MPCB14] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[MSM16] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of CNN advances on the imagenet. *arXiv preprint arXiv:1606.02228*, 2016.

[NBGS08] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, March 2008.

[NdALM16] Rodrigo Frassetto Nogueira, Roberto de Alencar Lotufo, and Rubens Campos Machado. Fingerprint liveness detection using convolutional neural networks. *IEEE Transactions on Information Forensics and Security*, 11(6):1206–1213, 2016.

[Nes83] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RCC15] Angie K Reyes, Juan C Caicedo, and Jorge E Camargo. Fine-tuning deep convolutional networks for plant recognition. In *Working notes of CLEF 2015 conference*, 2015.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[Ros95] Guido Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[Sch00] Hanno Scharr. *Optimal operators in digital image processing*. PhD thesis, 2000.

[SDBR14] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

[SHK+14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SIVA17] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284, 2017.

[SLJ+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*, pages 1–9, 2015.

[Smi17] Leslie N Smith. Cyclical learning rates for training neural networks. In *IEEE Winter Conference on Applications of Computer Vision (WACV), 2017*, pages 464–472. IEEE, 2017.

[Spr15] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

[SPT+16] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from simulated and unsupervised images through adversarial training. *arXiv preprint arXiv:1612.07828*, 2016.

[SPVS14] Anush Sankaran, Prateekshit Pandey, Mayank Vatsa, and Richa Singh. On latent fingerprint minutiae extraction using stacked denoising sparse autoencoders. In *IEEE International Joint Conference on Biometrics (IJCB), 2014*, pages 1–7. IEEE, 2014.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

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

[SZ16] Falong Shen and Gang Zeng. Weighted residuals for very deep networks. *arXiv preprint arXiv:1605.08831*, 2016.

[TGF16] Yao Tang, Fei Gao, and Jufu Feng. Latent fingerprint minutia extraction using fully convolutional network. *arXiv preprint arXiv:1609.09850*, 2016.

[TH12] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[Ver10] SDK VeriFinger. Neuro technology (2010), 2010.

[VLL+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[Wat93] Craig I Watson. NIST special database 9, mated fingerprint card pairs. *National Institute of Standard and Technology (February 1993)*, 1993.

[WGT+04] Craig I Watson, Michael D Garris, Elham Tabassi, Charles L Wilson, R Michael McCabe, and Stanley Janet. User's guide to NIST fingerprint image software 2 (NFIS2). *National Institute of Standards and Technology*, 2004.

[WGT+07] Craig I Watson, Michael D Garris, Elham Tabassi, Charles L Wilson, R Michael Mccabe, Stanley Janet, and Kenneth Ko. User's guide to NIST biometric image software (NBIS). 2007.

[WW92] Craig I Watson and CL Wilson. NIST special database 4. *Fingerprint Database, National Institute of Standards and Technology*, 17:77, 1992.

[XGD+16] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[ZJZF16] Zhuoyao Zhong, Lianwen Jin, Shuye Zhang, and Ziyong Feng. Deeptext: A unified framework for text proposal generation and text detection in natural images. *CoRR*, abs/1605.07314, 2016.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

[ZML16] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

[ZMZ11] Hongyun Zhang, Duoqian Miao, and Caiming Zhong. Modified principal curves based fingerprint minutiae extraction and pseudo minutiae detection. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(08):1243–1260, 2011.