TECHNISCHE UNIVERSITÄT
ILMENAU

Department of Computer Science and Automation
Neuroinformatics and Cognitive Robotics Lab

# Attribute-based Person Re-Identification by Deep Learning

Master's thesis to obtain the degree Master of Science
in Computer Science

**Thomas Golda**

Supervisor: Dipl.-Inf. Markus Eisenbach
Professor in charge: Univ.-Prof. Dr. Horst-Michael Groß
Neuroinformatics and Cognitive Robotics Lab

This master's thesis was submitted on the 9th of December 2016 at the Department of Computer Science and Automation of the Technische Universität Ilmenau.

***Acknowledgment***

*I take this opportunity to express gratitude to all my friends who supported me in any conceivable way. I would particularly like to thank Josephine R., who counter-checked my master's thesis and gave me helpful tips for my English writing; Christoph B., who gave me lots of feedback on the subject-specific topics; Patrick S., who supported me in spots with the mathematically correct expressions; Last but not least, Markus E., who showed much patience with me and provided helpful feedback.*

*Many thanks to all of you!*

Statutory Declaration: ”I declare on oath that I completed this work on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this, nor a similar work, has been published or presented to an examination committee.“

Ilmenau, 9th of December 2016

Thomas Golda

# Contents

# Introduction 1

Starting with the motivation in Section 1.1, this chapter will afterwards give a short description of the aim of this thesis (Section 1.2). The last part (Section 1.3) additionally serves as a short overview of the structure and different topics presented in this study.

## 1.1 Motivation

Person re-identification is an important task especially in the field of surveillance, but also in mobile robotics. It describes the problem of identifying people across images that have been taken using different cameras, or across time using a single or multiple cameras. In the past, a variety of approaches has been developed to identify persons by use of different kinds of features. An overview of features used for person re-identification is illustrated in Figure 1.2.

**Figure 1.1:** Common and challenging situation in clinical environment
*Narrow hallways with many people walking in different speeds and directions make it difficult for mobile robots to navigate towards a target or follow a person. (Taken from* [Eisenbach et al., 2015]*)*

When working with images of faces, biometric features show a good performance for re-identification. The main drawback of this kind of visual features emerges when

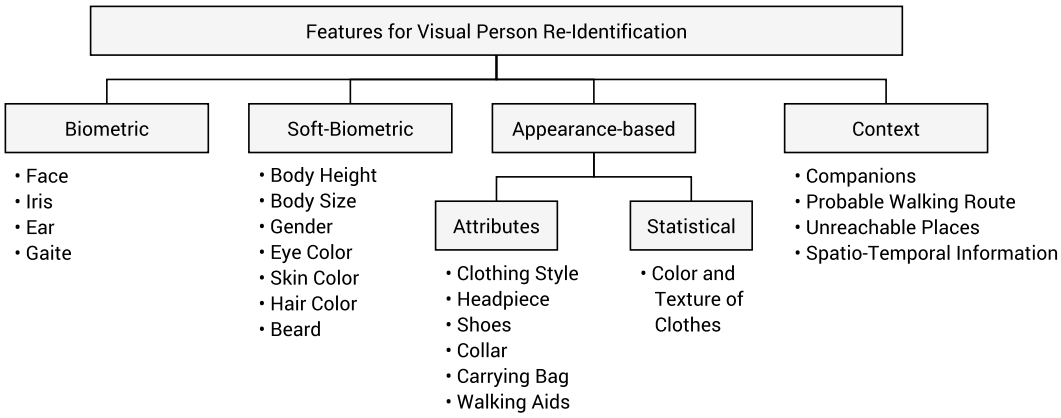



**Figure 1.2:** Different kinds of features for visual person re-identification *(Based on* [Eisenbach et al., 2015]*)*

choosing an application field like mobile robotics. Typically it cannot be assured that a robot is able to observe a person from the front, especially in the socially assistive robotics field, where a typical task can be to follow a certain person. In this case, robots usually see target persons from different perspectives and hence do not always have the possibility to take an image of their face. This kind of situation is illustrated in Figure 1.1, where the robot has to follow the person with the black jacket without being distracted by other people. For instance in the research project ROREAS at Ilmenau University of Technology, robots were developed to serve as coaches for walking and orientation training of stroke patients in clinical environments. [Gross et al., 2016] Therefore, the task was to select alternative features for re-identification, which also allow to identify persons from varying perspectives with a certain uncertainty.
A specific example for view invariant features are *attributes* like clothing style or accessories. Descriptions based on this kind of features apply in most cases independently of the viewing perspective. As stated by [Cheng et al., 2011], when observing people, humans pay attention to salient features, like special clothing, in order to identify them. This was examined by using eye-trackers. Thus, it seems natural to choose attributes as features for re-identification of persons when working with robots, even if such a description can be ambiguous.

## 1.2 Aim of the master's thesis

In recent years neural networks again began to enjoy great popularity, especially in the field of computer vision. Many different developments in the past led to better performance, particularly when working with neural networks with many hidden layers, making them attractive for different purposes. Therefore more and more researchers consider neural networks for their projects. This "reincarnation" of neural networks is often referred to as *Deep Learning*, on which further information will be given in Section 2.2.

Inspired by the great success of different Deep Learning techniques, this thesis aims to develop and evaluate a method to perform person re-identification based on attributes and Deep Learning, and to study whether it achieves the performance of state-of-the-art algorithms. Therefore, the thesis focuses on existing approaches for person re-identification based on Deep Learning and tries to combine promising concepts with an already existing Convolutional Neural Network architecture [Eisenbach et al., 2016] for person detection. The just mentioned model was developed by researchers of the Neuroinformatics and Cognitive Robotics Lab at the Ilmenau University of Technology. As previously stated, its purpose is to detect people in images, taken by a mobile robot in a clinical environment, hence it has already obtained some knowledge about the appearance of people. Since detecting and identifying persons are related problems to some degree, one can suppose that the person re-identification network will benefit of the already acquired knowledge of the person detector.

## 1.3 Structure of the Thesis

The thesis consists of six chapters, whereas the first chapter gave an introductory view on the topic. In the following paragraphs, the remaining five chapters are presented by giving short overviews of their contents.

**Chapter 2.** This chapter discusses the basic topics needed to understand this thesis. Section 2.1 describes the person re-identification task in a short manner and shows the

difficulties and problems occurring when trying to identify a single person seen in a certain image in another one. Finally the fundamentals of artificial neural networks and a detailed look on Convolutional Neural Networks, which play an important role within the scope of this study, are given in Section 2.2.

**Chapter 3.** In this part of the thesis an overview of different state-of-the-art methods for person re-identification will be presented. The first Section 3.1 gives basic information about general existing algorithms mostly used when trying to describe and identify persons. Section 3.2 takes a deeper look on Deep Learning algorithms for person re-identification. In concrete terms, Section 3.2.1 and Section 3.2.2 focus on attribute-less respectively attribute-based techniques by delineating substantial concepts and ideas of different related papers.

**Chapter 4.** The main part of this master's thesis is divided into to parts. The first part of the proposed algorithm, which focuses on the prediction of semantic attributes, will be presented in this chapter. Therefore, it gives an overview of the used methods in order to build a system, which is capable of predicting attributes for given images of people. Section 4.1 shows details about the network architecture, followed by information about the generation of the attribute dataset and further algorithmic details in 4.2. Finally, different performed experiments are discussed in 4.3.

**Chapter 5.** The second part of the algorithm focuses on performing the actual re-identification. It builds upon the results of the method presented in Chapter 4. For the purpose of the re-identification of people, two different network architectures are examined. In Section 5.1 the architectures of both networks are presented, followed by details about the performed training in Section 5.2. Finally, Section 5.3 both attempts are evaluated, discussed, and compared to another selected re-identification algorithm that uses semantic attributes in.

**Chapter 6.** The final chapter summarizes the results of the developed and presented method for person re-identification in Section 6.1. Last but not least, Section 6.2 shows possible ways for future work.

# Background 2

This chapter gives an overview of different aspects concerning this thesis. First, a short presentation and explanation of the person re-identification task and it's difficulties will be given in Section 2.1. Second, a brief look at Deep Learning will present some general information, to give the reader a basic understanding of this advanced and on-trend machine learning technique (Section 2.2). More details on the used Deep Learning method will be given in Chapter 4 and 5.

## 2.1 Person Re-Identification

As mentioned in the introduction of this master's thesis, person re-identification is an important task especially in field like surveillance, or mobile robotics. Furthermore, in environments where a robot faces many people, detecting and identifying the right persons is very important. Depending on the purpose of such a robot, it should be able to distinguish between different persons. Particularly when following a target person, the robot should be able to find his target person all the time. Therefore, it is necessary to extract features in order to build a system that can perform the re-identification. The typical steps performed in re-identification of persons are presented in the following shortly. First features of the target person are extracted at the beginning of the task. In the next step, for all detected persons feature representation have to be generated as well. Based on these features, the different persons have to be compared and afterwards ranked by their similarity. In order to achieve this, some kind of features have to be used, which are view-invariant, in order to find persons independent

from the view perspective. For this purpose, this master's thesis focuses on semantic attributes, which describe a person by their appearance. Examples are clothing style or accessories. However, large variations in viewpoint and lighting across different views can cause two images of the same person to look quite unalike whereas images of different people might look very similar. Figure 2.1 gives an example of two people looking fairly similar on the first look. [Ahmed et al., 2015] Moreover, different poses, blurring effects, image resolutions, camera settings and occlusions add to the difficulty of matching different images taken of the same person. [Li et al., 2014]

**Figure 2.1:** Different people looking similar
*Two different persons viewed from different perspectives in a similar environment. Both look very similar due to their chosen clothing style, which shows how difficult the person re-identification task can get in certain situations. (Pictures taken from PETA dataset.* [Deng et al., 2014]*)*

Even for humans it can get difficult to tell whether images of two people represent the same or different persons. An overview of certain re-identification systems is given in Chapter 3, where some approaches for this task are presented.

## 2.2 Deep Learning

In the past few years, Deep Learning has generated much fascination in Machine Learning. Particularly, many breakthrough results in different fields of research like

pattern recognition and computer vision led to this development. The main idea of Deep Learning is to take artificial neural networks as a basis for the model architecture. Even though neural networks look back at a decades-long history, they encountered a raising interest in recent years. This is mainly caused by the availability of inexpensive, parallel hardware, namely GPUs and computer clusters, and massive amounts of data as well as methodical improvements (e.g. Dropout, ReLUs and Convolutional Neural Networks).

In the following section a short introduction into the basics of artificial neural networks will be given, followed by some information about Convolutional Neural Networks, which play an important role in this thesis.

### 2.2.1 What are Artificial Neural Networks?

In the 1950s and 1960s the so called *perceptron* was introduced in [Rosenblatt, 1958], which can be seen as one of the first artificial neural networks. An example of a perceptron is given in Figure 2.2, which shows a single neuron taking three inputs and computing a single output.

The task of a perceptron is to take a binary decision by computing an output value depending on several binary inputs $x_i$ and some corresponding weights $w_i$ as shown in Equation 2.1.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_i w_i x_i \leq \text{threshold} \\ 1 & \text{if } \sum_i w_i x_i > \text{threshold} \end{cases} \tag{2.1}$$

Due to the simplicity of the *activation function* chosen for such a kind of neuron, the perceptron itself is not capable of representing complex behaviors. Therefore, alternatives have been developed which are more suitable for purposes of learning more complex functions. Especially so called *sigmoid neurons* can frequently be found in artificial neural networks.

A more sophisticated and powerful model are *multilayer perceptrons*. The main idea behind multilayer perceptrons is to combine multiple neurons into groups or *layers*

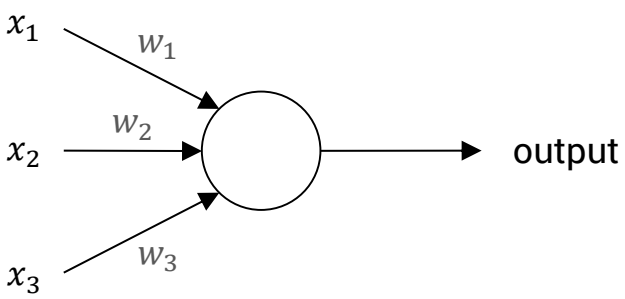



**Figure 2.2:** Simple perceptron
*The perceptron gets three inputs $x_1$, $x_2$ and $x_3$ and in combination with the weights $w_1$, $w_2$ and $w_3$ computes a decision function to generate a binary output value. (Based on* [Nielsen, 2015]*)*

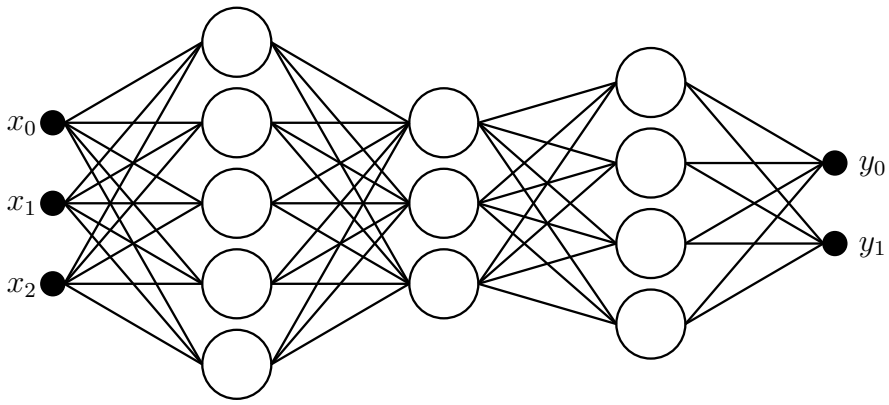



**Figure 2.3:** Example of a multilayer perceptron
*This example of a multilayer perceptron takes three inputs $x_0$, $x_1$ and $x_2$ which are fed forward. It is a combination of three hidden layers with five, three and four neurons and an output layer with two neurons $y_0$ and $y_1$. This structure results in 3 · 5 + 5 · 3 + 3 · 4 + 4 · 2 = 50 weights. This allows to compute more complex functions compared to a simple approach as shown in Figure 2.2.*

which are connected to the subsequent layer. Figure 2.3 outlines such a multilayer perceptron where each neuron is connected to all neurons in the subsequent layer. [Nielsen, 2015] Multilayer perceptrons had to deal with many critical problems, which made it difficult to train these networks. They were limited in size concerning width (number of neurons per layer) as well as depth (number of layers) and thus affected efficiency and performance of the model. In recent years, different discoveries and developments in the field of neural networks made it possible to use those models much more efficiently than before and led to the breakthrough of neural networks under the new term *Deep Learning*. [Lecun et al., 2015]
Apart from multilayer perceptrons there are even more architectural models for artificial neural networks which are used for different tasks, but a presentation of all these would go beyond the scope of this master's thesis. Therefore, only one specific type of neural network will be presented in the next section.

### 2.2.2 Convolutional Neural Network

Since the breakthrough of Deep Learning, so called *Convolutional Neural Networks*, which are biologically-inspired variants of multilayer perceptrons, enjoy great popularity. Concerning their structure, Convolutional Neural Networks differ significantly from multilayer perceptrons. The main difference hereby is, that neurons do not have to be connected to all neurons in the subsequent layer. It is more common to connect neighborhoods of neurons with single neurons in the following layer, which leads to a reduction of the number of weights which have to be adapted by the network. Hence, it is possible to say in regard of layers, that Convolutional Neural Networks try to exploit local patterns in the input, whereas multilayer perceptrons considers the whole input in some kind of

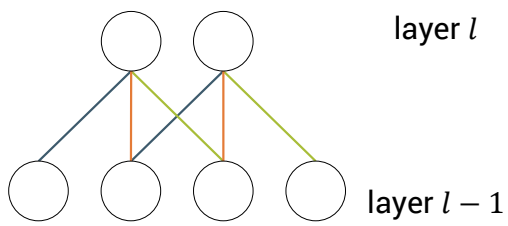
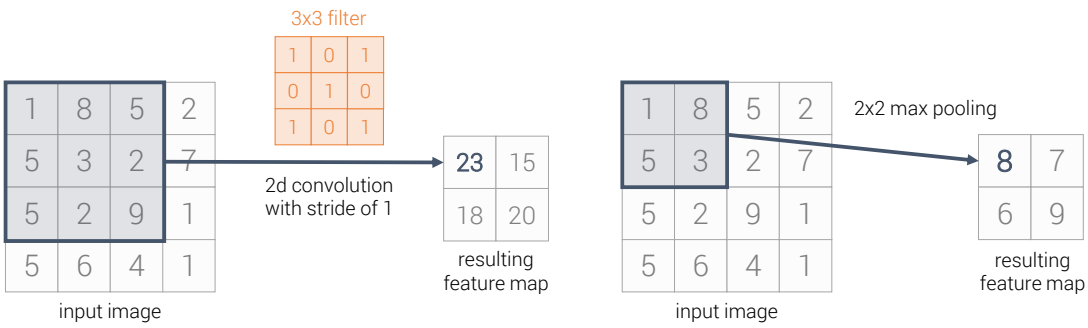



**Figure 2.4:** Concept of Convolutional Neural Network
*Neurons have connection to a subset of neurons in the subsequent layer. The number of connected neurons can differ between the layers.*




(a) Convolution (b) Max pooling

**Figure 2.5:** Concept of two-dimensional convolution and max pooling
*Figure 2.5(a) shows a two-dimensional convolution on a 4x4 image with filter size 3x3 and a stride of 1. The blue box moves over the underlaying matrix with a step-size of 1. In total, four different positions are possible. The resulting feature map is of size 4x4. At each position the filter (orange) is multiplied componentwise with the content of the blue box and summed up. The result is written to the corresponding position in the new feature map. Similar to the convolution, max pooling uses such a window to reduce the image size as shown in Figure 2.5(b). This time, the step-size is 2 and at each position the maximum value of the blue box is computed and written to the resulting feature map.*

global manner. [Krizhevsky et al., 2012] To reduce the amount of parameters even more, Convolutional Neural Networks use *shared weights*. For a better understanding of this difference, the basic concept presented above is illustrated in Figure 2.4. As an example, two layers from a one-dimensional Convolutional Neural Network were chosen to explain the concept. Even if there are six edges between layer $l-1$ and $l$ as displayed, the network only has to learn three of them, because edges with the same color share the same weights. Such groups of independent weights are called *filters*.

Despite the reduced amount of weights, Convolutional Neural Networks introduce new hyper parameters. When designing a network structure, one has to consider how many filters of which size and stride (i.e. step-size) are aspired to use.

Another kind of layers, so called *pooling layers*, use the same hyper parameters as convolutional layers but do not have weights which have to be learned. The aim of these

layers is to reduce input complexity, but also to preserve local information. Typical pooling layer types are *max pooling* and *average pooling*, which compute the maximal and average activation respectively of a neighborhood of neurons. Figure 2.5 illustrates how two-dimensional convolution and max pooling work.

These differences in the construction of multilayer perceptrons and Convolutional Neural Networks lead to different behaviors of both types of networks. In general, multilayer perceptrons can be seen as complex function approximators which try to compute a mapping between inputs and outputs. Often it is necessary to extract features before training a multilayer perceptron. Convolutional Neural Networks on the other hand do not need a manual feature extraction, since the filters learn useful features on their own. In many cases, after a sequence of convolutional and pooling layers, a multilayer perceptron follows. Its purpose is to perform a classification based on the output of the convolutional layers. [Lecun et al., 2015] Therefore, Convolutional Neural Networks can be seen as end-to-end systems, which extend multilayer perceptrons by the ability to work on raw data without the need of manually obtained features.
In Addition to the information about multilayer perceptrons and Convolutional Neural Networks, Table 2.1 gives an overview over some of the most famous Convolutional Neural Network variants, which in the past showed how powerful this kind of model can be. Especially for the task of visual person re-identification the deep architecture presented in [Krizhevsky et al., 2012] is often considered as a starting point for further developments.

| **Name** | **Comment** |
|---|---|
| LeNet | First successful application of Convolutional Neural Networks developed by Yan LeCun in the late 1990's. [Lecun et al., 1998] |
| AlexNet | Submitted to the ImageNet ILSVRC challenge in 2012. Significantly outperformed previous methods. Similar to LeNet in architecture, but deeper and with more convolutional layers. [Krizhevsky et al., 2012] |
| GoogLeNet | Winner of ILSVRC 2014 developed by researchers from Google. Introduced an *Inception Module* that reduced number of parameters in the network dramatically. [Szegedy et al., 2014] |
| VGGNet | Runner-up of ILSVRC 2014, which showed that the depth of a network is a critical component for good performance. [Simonyan and Zisserman, 2014] |
| ResNet | Winner of ILSVRC 2015. The architecture is missing fully-connected layers and uses special *skip connections* that feed the output of the two preceding layers into the actual layer. |

**Table 2.1:** Overview of famous Convolutional Neural Networks
*This table shows a list of common Convolutional Neural Network architectures which got famous for their achievements.* [Karpathy, 2015]

## 2.3 Summary

This chapter gave a short overview over the basics of person re-identification and neural networks. First, person re-identification is an important task in many different fields and gets much attention by researchers. Some existing techniques that try to solve this problem will be shown in Chapter 3. Second, Deep Learning is not a completely new topic but benefits of advances in research and hardware development in recent years and therefore enjoys great popularity in almost all of the computer vision community and beyond. Much more could be told, especially about the topic *Deep Learning*, but this would go beyond the scope of this study. For a good overview of Deep Learning see [Lecun et al., 2015].

# State of the Art

# 3

The main objective of this chapter is to give an overview of existing techniques for solving the person re-identification problem as presented in Section 2.1. Furthermore, in the scope of this thesis the chapter will show potentially useful existing concepts and explain why other methods cannot be applied. Starting with some general approaches,

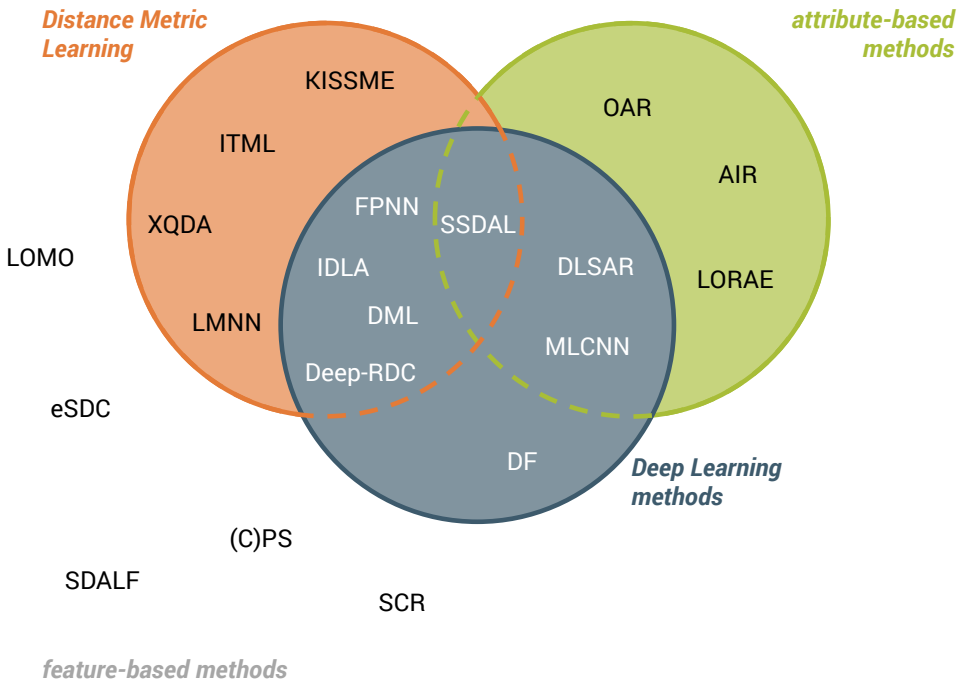



**Figure 3.1:** Overview of state-of-the-art methods for person re-identification
*The existing methods for person re-identification can be organized into different categories as outlined by this figure.*

like (Distance) Metric Learning, and some feature-based and non-deep-learning-related attribute-based methods in Section 3.1, the chapter continues with a deeper look at already existing Deep Learning algorithms for person re-identification. Therefore, the first part presents attribute-less (Section 3.2.1) while the second part deals with attribute-based Deep Learning methods (see Section 3.2.2).

## 3.1 General Approaches

Existing methods for person re-identification that try to solve the problem without using Deep Learning, are divided in this master's thesis into three major groups: Feature-based, Metric Learning, and attribute-based methods. Some examples for each group are presented in the following.

### 3.1.1 Feature-based methods

A first group are *feature-based methods*, whose aim it is to perform person re-identification on specific handcrafted features extracted from images showing pedestrians. Different kinds of feature-based methods developed for person re-identification have already been presented in literature. Some of the selected and often mentioned representatives are eSDC [Zhao et al., 2013], SDALF [Farenzena et al., 2010], (C)PS [Cheng et al., 2011], SCR [Bak et al., 2010], and LOMO [Liao et al., 2015]. In [Zhao et al., 2013] (**eSDC**) the authors try to extract salient image regions in an unsupervised manner. By their definition, a salient region is discriminative in making a person stand out from companions, and reliable in finding the same person across digurefferent views. To obtain a description of an image, they divide it into a grid and compute dense color histograms and SIFT features for every single patch. They only consider adjacent patches within a neighborhood of rows in the grid for comparison. Afterwards, they compute salience scores on a reference dataset, which has to be fairly large to obtain good results. Last but not least, they proposed two ways to compute a similarity score between two images for re-identification. The score is based on the salient scores and similarities of single patches in both images.

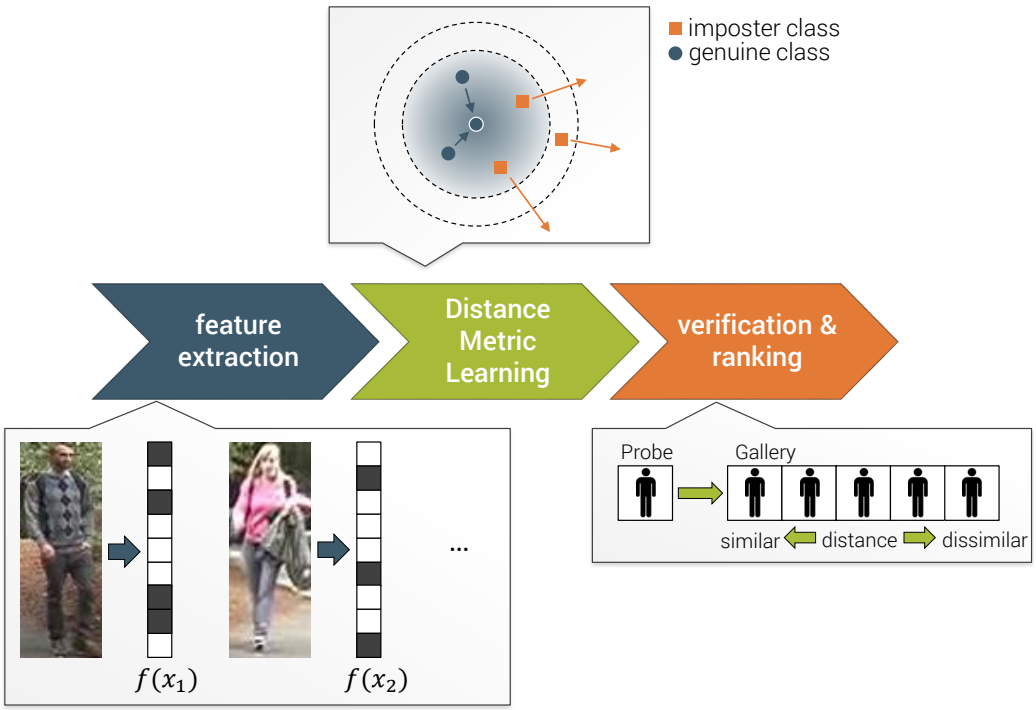



**Figure 3.2:** State-of-the-art procedure for person re-identification
*Many of the presented methods for person re-identification follow these steps. In the first stage features are extracted on which a distance metric learning algorithm performs subspace learning afterwards. Then, a query is performed to obtain a list of matches, depending on the given metric.*

The **SDALF** algorithm chooses a different approach. In a first step it subtracts the background to obtain the person. Afterwards, the person is first divided by an asymmetrical axis and the resulting two parts are again divided by a symmetrical axis. These steps allow to extract different body parts of a person. Furthermore, it computes three different kind of features to obtain the signature of an image: A weighted HSV histogram, Maximally Stable Color Regions and Recurrent Highly Structure Patches. To compare different images, a convex combination of differences between corresponding features is computed and evaluated [Farenzena et al., 2010]. For further information see [Sorge, 2013].

The **(C)PS** algorithm works similar to SDALF. In a first step the body model is decomposed into a set of six different parts like torso or arms using *Pictorial Structures*.

The features chosen are the same as in [Farenzena et al., 2010] except the Recurrent Highly Structure Patches. Furthermore, the distances between two images are realized and combined in a similar way. [Cheng et al., 2011]
Similarly to the just presented method, **SCR** extracts body parts in a first step. Therefore, it trains a HOG-based body part detector to obtain a set of different regions like head, torso, legs and arms. After normalizing colors in the particular regions, a feature vector for each pixel in such a region is constructed. The resulting feature vectors are composed out of pixel position, the particular values, and the gradients of each color channel at that position. Additionally, a covariance matrix between these features is computed for each region. In order to compare two images a pyramid matching was chosen which consists of multiple levels. Based on the computed covariance matrices on these levels, comparisons are conducted for each of the regions of interest. The size of those regions decreases with every subsequent level. To achieve robustness towards outliers, the proposed method considers only the $k$ biggest covariance matrix distances on certain levels for the final dissimilarity metric. [Bak et al., 2010]
Finally, the **LOMO** features remain. In order to obtain them, the algorithm first preprocesses images with the Retinex algorithm [Jobson et al., 1997] to achieve improved consistency of lighting and color in different views. Afterwards, the image is divided into small overlapping patches. Each of these patches is described by a HSV color histogram and a SILTP descriptor [Liao et al., 2010]. The feature vectors in each horizontal group are reduced to a single remaining feature vector by taking the maximum value of corresponding bins. This is done for three different image scales.
Methods like those presented above are well-suited to be used in combination with algorithms from the group to be presented next (Metric Learning), as outlined in Figure 3.2. Since this thesis addresses Deep Learning and uses neural networks for automatic feature extraction hand-crafted features are dispensable.

### 3.1.2 Distance Metric Learning

As already mentioned in the introduction of this chapter, a second group of methods, so called *Distance Metric Learning* (or: *Metric Learning*) algorithms, which follow a quite

different approach, exist. These methods try to learn the Matrix $M \in S_+^n := \{A \in \mathbb{R}^{n \times n} \mid x^T Ax \geq 0, A = A^T\}$ of a Mahalanobis distance in order to get small values for corresponding class members and bigger distances for imposter class members. Equation 3.1 shows the formula for the Mahalanobis distance between two points $x$ and $y$.

$$d_M(x, y) = \sqrt{(x - y)^T M(x - y)} \tag{3.1}$$

A selection of methods from Figure 3.1 is presented shortly in the following paragraph. First, **LMNN** [Weinberger et al., 2006] learns a Mahanalobis distance metric in the k-nearest neighbor classification setting using semidefinite programming. The learned metric attempts to keep k-nearest neighbors in the same class, while keeping examples from different classes separated by a large margin.

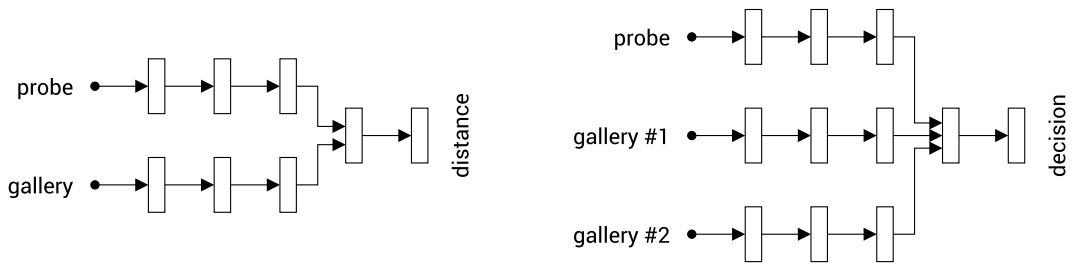
**ITML** [Davis et al., 2007] follows an information-theoretic approach to Metric Learning, where it tries to minimize the Kullback-Leibler divergence between two multivariate Gaussians. In other words, the algorithm tries to learn the Mahalanobis distance metric, which satisfies the constraints concerning distances between genuine and imposter class members, and to keep the corresponding matrix $M$ as close to the original one as possible. In addition, ITML is much faster than LMNN.

**KISSME** [Köstinger et al., 2012] considers a log likelihood ratio test of two Gaussian distributions, namely the probabilities for two points $x_i \in \mathbb{R}^n$ and $x_j \in \mathbb{R}^n$ being similar or dissimilar. The final Mahalanobis matrix for the learned metric results from projecting the difference of the inverted covariance matrices of both distributions onto the cone of positive semidefinite matrices. Since KISSME is a non-iterative algorithm, it can be computed extremely fast and hence outperforms LMNN and ITML concerning computation speed. For further information about KISSME see [Vorndran, 2015]. Finally, in addition to the already presented LOMO features [Liao et al., 2015] provides an own Metric Learning algorithm (**XQDA**). Similarly to KISSME, XQDA takes the covariance matrices of two Gaussian distributions in order to obtain the Mahalanobis matrix $M$. The combination of LOMO and XQDA outperforms all other presented feature-based and Metric Learning methods in respect to re-identification performance. [Kulis, 2013] Similar to XQDA, many more, especially non-linear, Metric Learning al-

gorithms exist, but presenting them all would go beyond the scope of this master's thesis. A detailed overview and comparison of these is given in [Xiong et al., 2014].

### 3.1.3 Attribute-based methods

Last but not least, a short look on existing attribute-based techniques will lead this part of the chapter to an end. Some state of the art methods for person re-identification based on attributes present different ontologies and describe how to construct suitable sets of attributes. In studies they also explore how different kinds of attributes perform for the purpose of re-identifying persons. As shown in Figure 3.1 three main methods exist and will be presented in the following paragraphs. Two algorithms, AIR and OAR, were introduced in [Layne et al., 2012] and [Layne et al., 2014] respectively in order to perform person re-identification based on attributes. However, as already mentioned, the main contribution of these papers are studies on the effect of using attributes for person re-identification and which kind of attributes preferably to be chosen to offer an optimal set for this purpose. AIR builds up upon the SDALF algorithm, which is extended by an additional objective, namely a weighted $L2$ distance between attribute vectors. It relays on a simple training of support vector machines in order to learn to predict attributes for a given image of a person. From the set of all attributes the algorithm step by step takes single attributes and computes corresponding weights. This is done as long as including further attributes improves the re-identification performance. The procedure of the second algorithm, OAR, differs slightly from AIR. Instead of an iterative expansion of the used attribute set, from beginning OAR takes all attributes into account, balances the dataset, and trains an SVM. This leads to a slight improvement in re-identification performance and an overall similar performance compared to AIR. However, the last algorithm, LORAE [Su et al., 2015], follows a different idea and tries to exploit dependencies between attributes. For example, assuming a specific gender, a certain clothing style may be more likely to appear than others. Therefore, LORAE converts binary attributes into continuous ones in order to learn attribute correlation. These new attributes allow to infer possibly hidden semantic information. For example, a skirt and a handbag




(a) Two input branches

(b) Three input branches

**Figure 3.3:** Examples of a Siamese Convolutional Neural Networks
*This figure shows two examples of siamese networks, which have a branch for each input. After three layers the branches are merged and continue as a single branch for two layers. Typically, networks with two branches are used to determine a distance between inputs. Those with three branches perform a decision which gallery image is closer to the probe image.*

can indicate a female gender. With this approach, LORAE outperforms all algorithms presented in this section and achieves even comparable performance to Deep Learning based methods that are examined in the following section.

## 3.2 Deep Learning Algorithms

As already mentioned in the introduction to this chapter, different kinds of Deep Learning algorithms for person re-identification exist, of which the majority does not use semantic attributes for finding matching persons. Therefore, the second part of the chapter is divided into two sections: The first one presents algorithms not using semantic attributes (Section 3.2.1), whereas the second one focuses on existing attribute-based methods (Section 3.2.2).

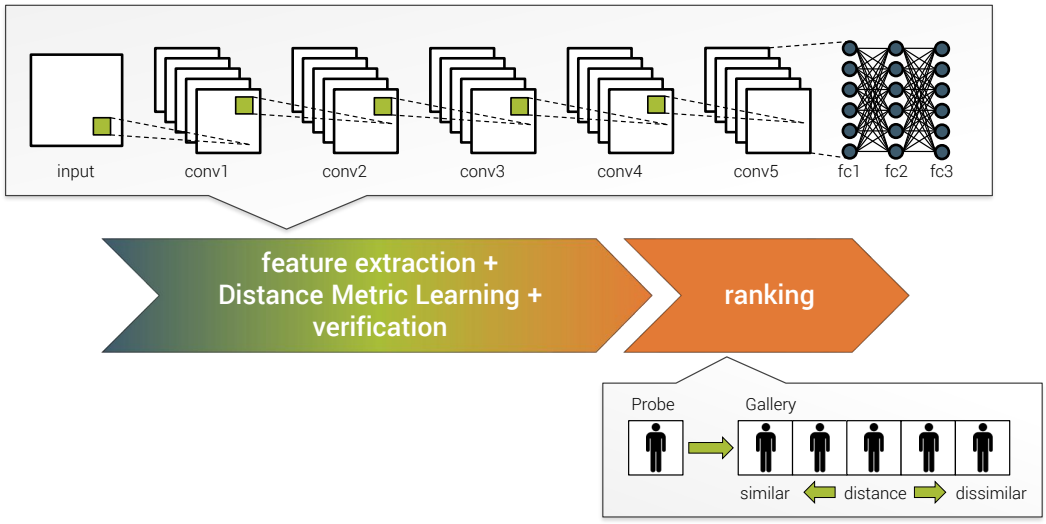



**Figure 3.4:** Steps for person re-identification with Deep Learning
*Deep Learning based methods make use of its capability to extract features automatically. Therefore unified systems are often used to combine single steps, as shown in Figure 3.2, in an end-to-end system.*

### 3.2.1 Attribute-less

In the literature, a variety of different algorithms can be found, which compare different inputs in order to decide if two images show the same or different persons in most cases. All presented techniques have in common that they train Convolutional Neural Networks for person re-identification being a part of an end-to-end system. This is shown schematically in Figure 3.4, which illustrates the procedure compared to the feature-based way shown in Figure 3.2. Except of [Wang et al., 2015] all methods use the so called Siamese Convolutional Neural Networks or Triplet Networks, where the first one is a modified version of conventional Convolutional Neural Networks, which often have multiple inputs with different inner branches. An exemplary visualization of such can be found in Figure 3.3, showing how typical Siamese Convolutional Neural Networks look like. The latter can be seen as a special form of siamese network with three branches. There are many ways how such a network could look like. They

differ in the number of inputs and outputs, and the depth of the merging layer. The following paragraphs introduce and describe some of the existing methods.

**Filter Pairing Neural Network (FPNN).** In [Li et al., 2014], the authors proposed a novel *Filter Pairing Neural Network (FPNN)* which is a first example for a Siamese Convolutional Neural Network. It takes two images from two different but fixed viewpoints as its input and tries to jointly handle the problems of misalignment, photometric, and geometric transforms by using so called *patch matching layers* to match the filter responses of local patches across views, and other convolutional and maxpooling layers to model body parts displacements. Instead of working on the original feature maps, after the first layer of the network (convolution and maxpooling) the resulting feature maps are sliced into horizontal stripes. Afterwards, these are divided into patches of the same size, which again result – due to the two input images – in filter pairs. Patches are compared only in corresponding stripes and result in *displacement matrices*, which encode the spatial patterns of patch matching under the different features. In the next step, a *maxout grouping* layer reduces groups of displacement matrices similar to maxpooling. The resulting filters capture local patterns of part displacements. At the end, the network decides whether two images show the same person or not. For performance evaluation the authors used the CUHK03 dataset [Li et al., 2014] where the algorithm outperformed different existing feature-based algorithms. This paper is one of the first attempts to apply Deep Learning to person re-identification.

**Deep Features (DF).** As indicated previously, [Wang et al., 2015] do not use Siamese Convolutional Neural Networks but adopt the architecture for a single Convolutional Neural Network as proposed by [Krizhevsky et al., 2012] which is pre-trained on a subset of the ImageNet dataset [Deng et al., 2009]. Afterwards, a fine-tuning is performed using the CUHK03 dataset. For this purpose they keep the weights of all convolution layers and reinitialize the fully-connected layers randomly. Additionally the authors replace the original 1000-way classification layer and feed the output of the second fully-connected layer to an $N$-softmax, which produces a

prediction distribution over $N$ person labels. For fine-tuning, they set the learning rate for the retained weights and layers to a much smaller value than for the reinitialized fully-connected layers in order not to change those pre-trained weights too fast. After training, the output of the last fully-connected layer is taken as the feature representation of the input image. On the basis of those 4096-dimensional vectors, they measure similarities between different observations of pedestrians and obtain similarity rankings to predict the correct match.
Similar to the FPNN, the authors compare their algorithm with different feature-based and Metric Learning methods, as well as with the FPNN itself. DF outperforms all of them and achieves about 50% better performance on rank-1 matching rates than FPNN. [Wang et al., 2015]

**Deep Features with Relative Distance Comparison (Deep-RDC).** In [Ding

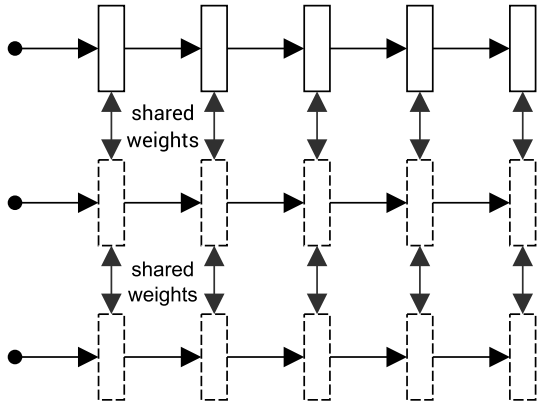



**Figure 3.5:** Simplified illustration of a special triplet network
*Three identical branches share weights for each corresponding layer and can alternatively be seen as three copies of one network.*

et al., 2015], the authors attempt to combine Distance Metric Learning with Deep Learning in the context of person re-identification. Instead of minimizing the classification error, the objective of their work is to maximize the relative distance between images. For that reason they use a so called *Triplet Network*, which can be seen as a special kind of siamese network with three inputs, where the main difference is that

this kind of network does not have dedicated branches for each input. As outlined in Figure 3.5, the network can be seen as three parallel branches with shared weights, where each branch produces its own output. However, sharing weights between branches actually leads to a single branch. For an easier understanding, this thesis will stay with the idea of three branches instead of a single one. By using a special loss function, the training process tries to achieve a more similar representation for the top and middle branch and a more dissimilar representation for the top and bottom branch. Typically, the input of the upper branch is referred to as anchor or query image, whereas the input of the middle and bottom branch is referred to as positive sample image and negative sample image respectively. Together, these three images are combined to a so called *Triplet*. The underlying Convolutional Neural Network is composed of two convolutional, two maxpooling and a single fully-connected output layer. In order to compare two images, the $L2$-distance between the outputs of the network for the corresponding images is computed on the basis of which it is possible to rank images.
The performance of this algorithm was tested on the i-LIDS and VIPeR dataset. It outperforms all other algorithms on i-LIDS on all ranks and on VIPeR on rank-1 and rank-5 matching rates, with comparable performance metrics on the remaining ranks. Comparing overall performance with FPNN and DF is difficult to achieve, because both were trained and evaluated only on the CUHK03 dataset.

**Improved Deep Learning Architecture (IDLA).** Similar to [Li et al., 2014], the authors of [Ahmed et al., 2015] take advantage of Siamese Convolutional Neural Networks and introduce supplementary new layers. Starting with a simple convolutional and maxpooling layer with shared weights between branches, so called cross-input neighborhood differences are computed. The concept is illustrated in Figure 3.6 for a $k \times k$ neighborhood with $k = 3$. Afterwards a $k \times k$ maxpooling with a stride of $k$ is performed to reduce the size of these maps and return to the original size. With this procedure, the authors try to add robustness to positional differences in corresponding features of two input images. In a subsequent step, another round of convolution and maxpooling follow without shared weights. The output for both

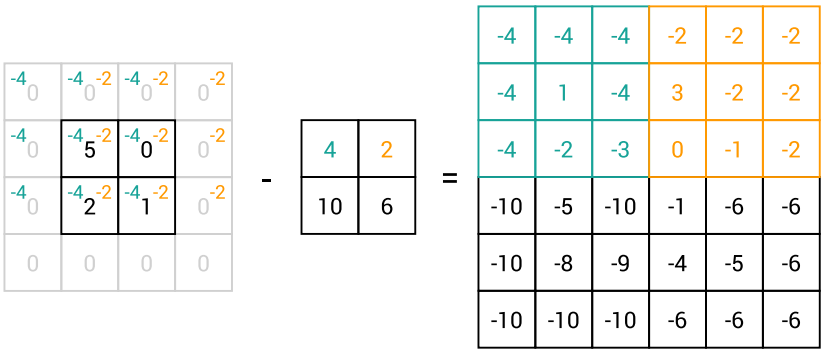



**Figure 3.6:** Cross-neighborhood difference example for two 2x2 feature maps
*In this example, the four from the right 2x2 feature map is subtracted from all cells in the neighborhood around the five in the left 2x2 feature map, which leads to the green block in the resulting matrix. This procedure is repeated for two (see orange box in target matrix) and also for all remaining elements in the right feature map.*

processed neighborhood difference maps is passed into an fully-connected layer of size 500. After this, a second fully-connected layer with a two-way softmax activation follows, which indicates that both images show the same or different person respectively. For evaluation, the authors trained and evaluated on the same dataset. Therefore, they used the already mentioned CUHK03, the CUHK01, and the VIPeR dataset. On CUHK03, the algorithm outperforms all other evaluated algorithms and achieves about 1.3-times the rank-1 matching rate of DF, whereas on VIPeR IDLA outperforms most tested feature-based algorithms, but stays inferior to Deep-RDC. [Ahmed et al., 2015]

**Deep Metric Learning (DML).** As the name of the algorithm already reveals, in [Yi et al., 2014] the authors chose the way of Metric Learning in combination with Deep Learning. In contrast to [Ding et al., 2015] they do not work with triplet networks, but attempt to learn to decide if two images show the same person by using only two instead of three input images. For this, they use two identical Convolutional Neural Networks which do not share weights. This is followed by the Cosine similarity as the connection function for evaluating the relationship between the outputs of

both Convolutional Neural Networks, because its output is bounded to $[-1, 1]$ and it is invariant to the magnitude of samples. Furthermore, the underlying Convolutional Neural Network is composed of three identical branches with two convolutional layers, each followed by a maxpooling layer, all together finally leading to one fully-connected layer. The first convolutional layers have shared weights over all three branches, whereas the second convolutional layers are independent of each other. The input to this three-branched network are images of pedestrians, vertically divided into three overlapping parts. In cross dataset experiments (training on i-LIDS and CUHK, evaluation on VIPeR) the DML achieved comparable results to feature-based methods, which were trained and evaluated both on VIPeR. DML achieves nearly twice the performance of its cross dataset experiments, when training is performed on VIPeR as well. This is, approximately, the performance of IDLA, which is why DML stays inferior to Deep-RDC [Yi et al., 2014].
Although the presented algorithms show promising performance for the task of re-identifying humans, they will not be considered further since this master's thesis focuses on using semantic attributes as feature representation.

Although all these algorithms show promising results for the re-identification of persons, they are not considered further, since the focus of this master's thesis lays on using semantic attributes as features for the re-identification.

### 3.2.2 Attribute-based

After taking a look on the first part of Deep Learning based algorithms that are not related to semantic attributes, the chapter continues with those that are. Some of the concepts presented in 3.2.1, for example Siamese Convolutional Neural Networks, can be found in this part of the chapter as well. Using Deep Learning methods for attribute prediction is a very challenging task as Deep Learning algorithms typically require large amounts of labeled data. However, labeling datasets with attributes is a very sophisticated task, which is why existing datasets are relatively small compared to typical Deep Learning datasets.

**Deep Learning for Semantic Attribute Retrieval (DLSAR).** In his dissertation, [Pala, 2016] tries to predict semantic attributes by using Deep Learning techniques. For this, three Convolutional Neural Networks are used, one for each body part: Head, torso and legs. Each of these tries to predict attributes of a given person by analyzing a certain body part. Beside the first convolutional layer of the network for the head part, the architecture of all Convolutional Neural Networks is identical. Furthermore, the main topic is the comparison of two different multi-label losses for training. Specifically, binary cross-entropy and pairwise ranking are examined, where the first one describes the average over many cross-entropy losses. Each attribute is considered a binary classification problem, for which cross-entropy can be used as the loss function. However, the pairwise loss tries to optimize the area under the ROC curve for each attribute, which leads to a slightly better overall performance than binary cross-entropy. The PETA dataset, which at the moment is the largest person dataset labeled with attributes, is used as training data. Additionally, in order to train a Convolutional Neural Network the data was augmented by horizontally flipping each image and altering the intensity of color pixels. Furthermore, attributes that were not present in at least 1% of the dataset and those regarding shoe parts, were discarded. [Pala, 2016] A direct comparison with already presented methods cannot be performed, due to the selected performance metrics and the focus on comparing the mentioned loss functions instead of evaluating re-identification performance.
Concerning this master's thesis, the dataset used by [Pala, 2016] as well as the binary cross-entropy will be used as well. Due to the fact, that the architecture for the own network is predetermined by the pretrained network, the concept of having a network for each part of the body cannot be adapted.

**Multi-label Convolutional Neural Network (MLCNN).** Similar to the method presented in [Pala, 2016], the authors of [Zhu et al., 2015] divide images of persons into different regions. Instead of pure semantic splitting, the input image is split into fifteen overlapping regions as illustrated in Figure 3.7. Each region is input to a Convolutional Neural Network which has as many input branches as available regions exist.

The outputs of these networks have predefined connections with a following output layer predicting 21 attributes. Additionally to the Convolutional Neural Network, they also extract features and perform metric learning to obtain a low-level feature representation. In the application phase, for two images, a probe image and one from a gallery set, low-level feature distance and attribute distance are computed and fused together, which is then used for ranking the gallery images. This procedure achieves slightly better performance on the VIPeR dataset compared to DML and IDLA but stays inferior to Deep-RDC.

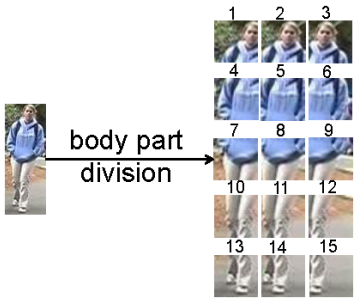



**Figure 3.7:** Division of an input image

*Images of the dataset are split into fifteen overlapping regions. [Zhu et al., 2015]*

This master's thesis will not adapt anything from this paper. Especially, the architecture cannot be used, due to the same reason as mentioned in the presentation of the DSLAR algorithm.

**Semi-supervised Deep Attribute Learning (SSDAL).** Finally, the SSDAL algorithm from [Su et al., 2016] is presented. It uses a three-staged process in order to train a Convolutional Neural Network to learn and predict semantic attributes. For the architecture, the authors adapt the AlexNet by [Krizhevsky et al., 2012] and change the output layer to a 105-dimensional sigmoid output layer. In the first stage, this network is trained on the PETA dataset $T$ in a supervised manor to obtain a system which is capable of labeling images of persons with attributes. Afterwards, a second, independent dataset $U$ is taken, for which labels are predicted with the trained network. The weights of the AlexNet are then fine-tuned by triplet training on $U$ in the second stage. Contrary to existing approaches, the aim is not to tell which images show the same person, but to produce similar labels for images showing the same, and different labels for unsimilar persons. In order to avoid learning meaningless attributes, changes leading to attributes differing too much from the original ones are penalized. As a last step, the first stage procedure is repeated with combined datasets

$T$ and $U$, where this time $U$ is labeled by the refined network from the second stage. Already the trained network from Stage 1 achieves good performance and beats almost all presented algorithms in this chapter. In combination with XQDA [Liao et al., 2015], SSDAL outperforms all existing Deep Learning based and most attribute-based methods on the percentage of correct matches on the VIPeR dataset. Only LORAE and some metric learning based methods achieve similar results.
Due to the similarity of the first stage of the SSDAL algorithm and the prerequisite of this work to use a pretrained network, this paper will be taken as the direct competitor.

## 3.3 Summary

Many promising attempts have been conducted in order to solve the problem of person re-identification. Feature-based methods benefit of the relatively fast way of comparing images of persons due to already handcrafted features, whereas Deep Learning proved to deliver even better features, leading to an overall improved performance on re-identification quality. It seems natural to extend the automatic construction of these descriptors by further steps, in order to enhance the performance even more. The aim of this thesis and its research is to develop an algorithm – based on an already existing architecture – using semantic attributes. Due to this fact not all concepts presented on the last pages, for example splitting up input images, are applicable. The broadly used siamese networks, however, show a good performance in almost all of the presented papers and hence seem to be suitable for the scope of this master's thesis. The original idea for this master's thesis was to adapt the concept of siamese networks for the multilayer perceptron part of the network. Unfortunately, due to the limited time, this could not be implemented. Additionally, [Su et al., 2016] showed that adapting an existing architecture can lead to a good performance in re-identifying persons. Similar to [Su et al., 2016], this master's thesis will examine whether using a pretrained network architecture has a positive effect on the prediction of attributes. In the following chapter, some of the concepts shown beforehand, will be adapted in order to build a system which allows to identify persons by using Deep Learning and semantic attributes.

# Classification of Semantic Attributes

# 4

As already mentioned in Chapter 1, the algorithm developed in this master's thesis consists of two stages: the prediction of semantic attributes on images of persons, and the re-identification of persons based on these attributes. This chapter focuses on the first stage, which has the purpose of predicting attributes on images of persons. In order to achieve this, an already existing Deep Convolutional Neural Network [Eisenbach et al., 2016], which takes images of persons as input is adapted. Section 4.1 presents the basic architecture of the model and discusses changes that were made for this master's thesis. Detailed algorithmic information about the training process is given in Section 4.2. Finally, different variations of the basic model architecture are evaluated in Section 4.3. In addition, some self-made implementations of other existing architectures are compared to the finally developed architecture as well.

## 4.1 Model Architecture

The first part of this chapter takes a look at the architecture of the developed model. Therefore, the original architecture proposed by [Eisenbach et al., 2016] is shortly presented in Section 4.1.1. In addition to that, the modifications made in order to use the network are introduced in Section 4.1.2 and Section 4.1.3. Similar to the AlexNet [Krizhevsky et al., 2012], the architecture of [Eisenbach et al., 2016] consists of eight layers. These mainly differ in the number of filters and neurons respectively, leading to an overall reduced model size for the latter.

| Layer | Type | Nb. | Activation | Filter | Pooling | Dropout |
|---|---|---|---|---|---|---|
| 1 | conv | 64 | ReLU | $5 \times 5$ | $2 \times 2$ | – |
| 2 | conv | 128 | ReLU | $4 \times 4$ | $2 \times 2$ | 0.25 |
| 3 | conv | 128 | ReLU | $3 \times 3$ | $2 \times 2$ | 0.25 |
| 4 | conv | 128 | ReLU | $2 \times 2$ | – | 0.3 |
| 5 | conv | 256 | ReLU | $2 \times 2$ | – | 0.4 |
| 6 | fc | 1000 | ReLU | – | – | 0.5 |
| 7 | fc | 1000 | ReLU | – | – | 0.5 |
| 8 | fc | 2 | Softmax | – | – | 0.5 |

**Table 4.1:** Layers of the original Convolutional Neural Network
*This table shows a list of all existing layers in the Convolutional Neural Network with additional information about number of neurons and filters respectively, filter size, and dropout rate at each layer. Layers 1 to 3 are followed by maxpooling. Additionally, the output layer is shown here as layer 8. (conv - convolutional, fc - fully-connected).*

### 4.1.1 Network Structure

The network proposed by [Eisenbach et al., 2016] was initially developed with the aim of detecting persons. This is done by taking an input image and deciding whether it shows a person or not. Thus, the network solves a binary classification problem. The architecture of the network is characterized by five convolution layers, followed by three fully-connected layers. Except the output layer, each of the remaining layers uses dropout regularization in order to avoid overfitting. The filter sizes begin with $5 \times 5$ in the first layer and are reduced by one at each layer. At layer 4 and 5 the final size is $2 \times 2$. In addition to that, the first three convolution layers are followed by maxpooling. In total, this architecture consists of over four million weights. For further information about the original architecture see Table 4.1 and [Eisenbach et al., 2016].

As shown in the following, some changes on the network are necessary in order to use the network for the purpose of attribute prediction.

### 4.1.2 Output Encoding

As mentioned above, the original paper [Eisenbach et al., 2016] focuses on detecting people. Hence, the authors try to solve a single(-label) binary classification problem: *person vs. non-person*. Classifying images of persons into semantic attributes, though, results in a multi-label classification problem. This means, the main question changes from "Does input $x$ belong to a certain class $C$?" to "Which $S \subseteq \mathcal{C} = \{C_1, \ldots, C_n\}$ does input $x$ represent?". This differs slightly from the multi-class classification, where each input belongs to exactly one class, since here, multiple attributes can be active at the same time. Analogous to the multi-label problem, this can be seen as the task of finding a subset $S \subseteq \mathcal{C} = \{C_1, \ldots, C_n\}$ with $|S| = 1$ for each input $x$. Table 4.2 illustrates the mentioned problems.

| **Problem** | **Available classes** | **Classes per sample** |
|---|---|---|
| single-label | 2 | 1 |
| multi-label | $n$ | $\leq n$ |
| multi-class | $n$ | 1 |

**Table 4.2:** Comparison of single-, multi-label and multi-class problems
*The single-label problem is a special case (n=2) of the multi-class problem. The multi-label and multi-class problems differ only in the number of predicted classes.*

As the table shows, changes on the output layer are necessary in order to apply the existing architecture on the problem of predicting semantic attributes. This is due to the larger number of classes the network has to predict. Taking a look on the architecture overview in Table 4.1, therefore, reveals that the size of the output layer has to be expanded from two to 44 neurons.

In addition, the original softmax activation has to be replaced by a sigmoid activation. This decision was made due to the properties of the sigmoid activation, which are beneficial for the multi-label problem. A closer look on Equation 4.1 and 4.2 reveals that the activation of a sigmoid neuron is independent of the activation of the other neurons in the output layer. This is not the case for softmax activated neurons. For the softmax output function, each activation is normalized by the activations of all $n$

neurons in the output layer. In combination with the exponential influence, in most cases this leads to one dominant output.

$$f_{\text{sigmoid}}(x_i) = \frac{1}{1 + e^{-x_i}} \tag{4.1}$$

$$f_{\text{softmax}}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{4.2}$$

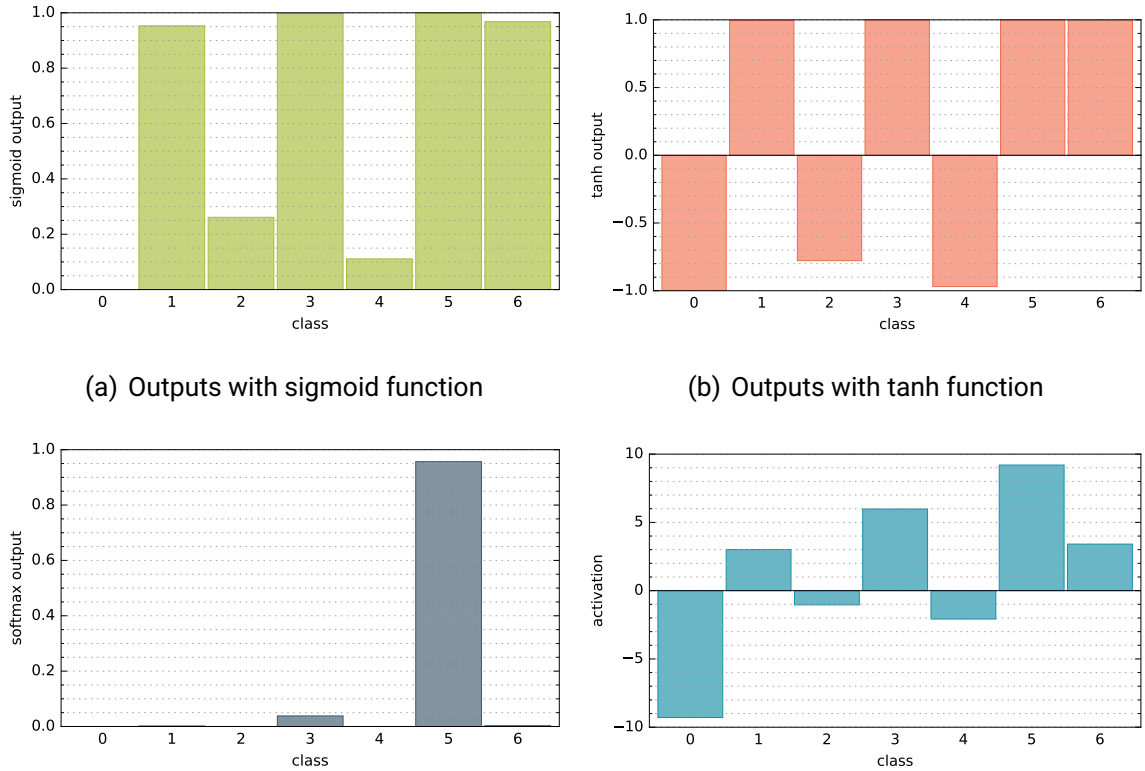
Figure 4.1 illustrates the consequences of these behaviors. It shows the actual activation of each neuron before applying an activation function (4.1(d)), as well as the output with three different activation functions. The two plots on the left-hand side display the resulting sigmoid (4.1(a)) and softmax (4.1(c)) output. Additionally, another popular activation, the hyperbolic tangent activation (4.1(b)), is shown in the upper right. The hyperbolic tangent activation can be written as a function of the sigmoid activation as Equation 4.3 shows.

$$\begin{aligned} f_{\text{tanh}}(x_i) = \tanh(x_i) = \frac{\sinh(x_i)}{\cosh(x_i)} = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}} = \frac{1 - e^{-2x_i}}{1 + e^{-2x_i}} = \\ = \frac{1 - e^{-2x_i}}{1 + e^{-2x_i}} + 1 - 1 = \frac{2}{1 + e^{-2x_i}} - 1 = 2 \cdot f_{\text{sigmoid}}(2x_i) - 1 \end{aligned} \tag{4.3}$$

Therefore, the tanh activation will not be examined further in this master's thesis. Instead the master's thesis uses the sigmoid function as used in the literature.
Another way to implement an output for multi-label classification would be to aggregate different attributes into groups. Some examples for suitable groups are *gender*, *hair color*, or *age*, since they form cliques of attributes, where typically only a single element applies to a person. In that case, each group would be activated by softmax outputs. This should be addressed in future work.

### 4.1.3 Inner changes on the Network

In order to improve the performance of the network, the activations for each layer were changed, compared to the original architecture by [Eisenbach et al., 2016]. One way to do this, is to use so called Exponential Linear Units (ELUs) [Clevert et al., 2016] instead of the broadly adapted ReLUs. These differ slightly from ReLUs and allow negative input values to have influence on the training of the network as illustrated




(a) Outputs with sigmoid function

(b) Outputs with tanh function

(c) Outputs with softmax function

(d) Activations of neurons from the last layer

**Figure 4.1:** Comparison of sigmoid, tanh and softmax output for seven classes
*Figures 4.1(a) and 4.1(c) show how the actual values for each neuron differ when applying sigmoid or softmax output function respectively on exemplary activations as seen in Figure 4.1(d). Additionally, the tanh output is shown in Figure 4.1(b) as well.*

in Figure 4.3. As stated by [Clevert et al., 2016], this can have a positive impact on the training speed as well as the quality of the weights. This decision was made, since ELUs outperformed ReLUs in almost all experiments, which will be shown in Section 4.3. In addition, training was performed by using stochastic gradient descent. Due to the limited time, the batch size was not evaluated further, but set to 256 for all experiments as proposed by [Eisenbach et al., 2016]. A summary of the most important information given above is presented in Table 4.1.

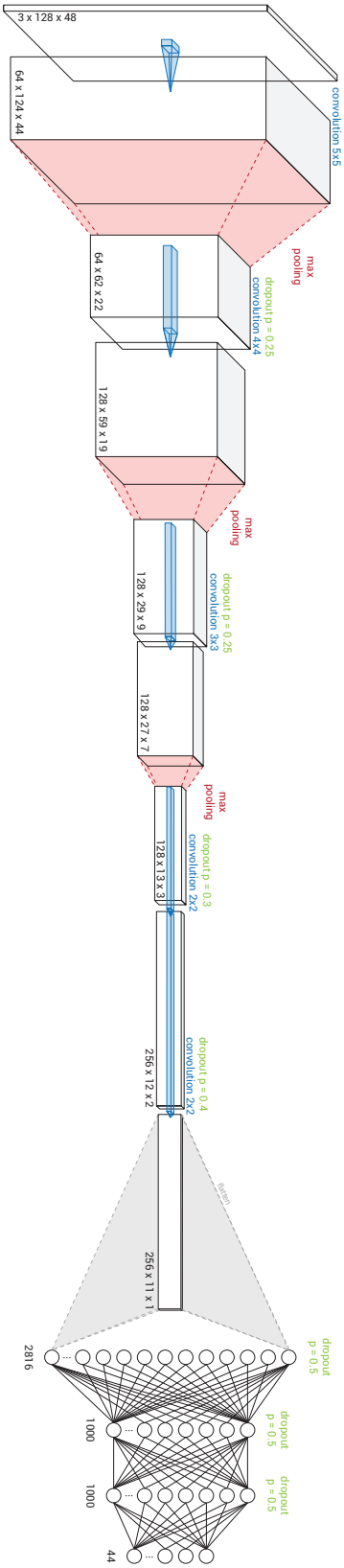
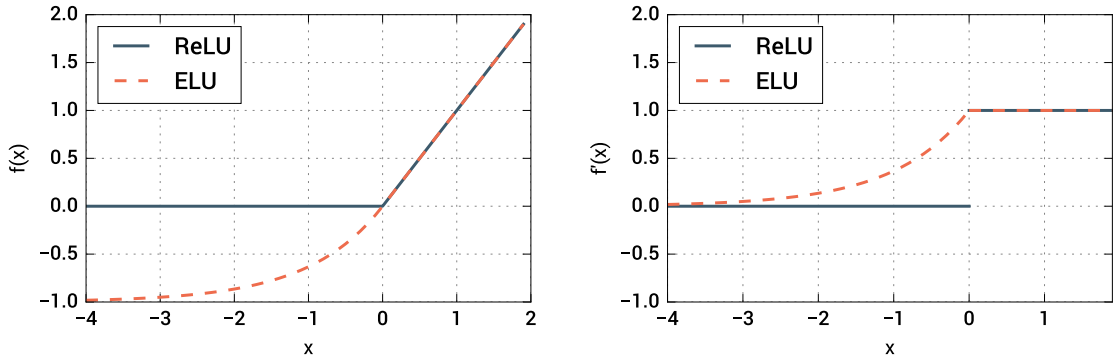



**Figure 4.2:** Architecture of the basic Convolutional Neural Network

*This model is almost identical to the model presented in [Eisenbach et al., 2016]. It consists of five convolution layers, followed by three fully-connected layers. The last fully-connected layer serves as a 44-dimensional output layer with sigmoid activation. For more detailed and compact information about each layer of the network architecture see Table 4.1.*




(a) Graph of the functions

(b) Graph of the derivatives

**Figure 4.3:** Visualization of the ReLU and ELU, as well as their derivatives
*The left hand side shows the graph of both functions, ReLU and ELU. In addition to that, the right hand side shows the corresponding derivatives. Since the derivative of the ELU does not drop automatically, when reaching 0, it allows allows negative input values to have at least a small influence on the training of the weights.*

## 4.2 Training for Attribute Recognition

After introducing the developed model in the first part of the chapter, the following paragraphs focuses on learning to recognize attributes. First, the dataset used for model training, namely PETA, is presented in Section 4.2.1. Second, the preprocessing of the dataset is described in Section 4.2.2, followed by general information about data augmentation and details about its implementation within the scope of this master's thesis in Section 4.2.3. Finally, Section 4.2.4 takes a short look at the used loss function.

### 4.2.1 Attribute Dataset PETA

As mentioned in the State of the Art (Chapter 3), the PETA (PEdesTrian Attribute) dataset [Deng et al., 2014] is currently the largest labeled dataset for learning attributes. It is composed of multiple datasets for person re-identification and detection. By combining multiple small and middle sized datasets, the resulting dataset achieves a size of 19,000 images of 8,705 different individuals. Each person is labeled with 65

different attributes, for instance *hair color*, *gender*, *age*, and *accessories*. These 65 attributes can be split in 61 binary and four multi-class labels. Each of the multi-class labels has eleven different peculiarities. In order to use them, the multi-class labels are "flattened", which results in 105 binary labels altogether. Even though PETA is the largest available dataset for attribute learning, it might be of insufficient size for training a deep neural network from scratch, but big enough for finetuning a pre-trained network. Therefore, it is inevitable to use techniques like data augmentation to mitigate its influence. The performed steps will be presented later in Section 4.2.3. When using PETA, one faces the problem that the frequencies of the attributes in the dataset differ substantially. Some attributes appear only once or twice in the whole dataset. *Blue hair* and *pink hair* do not appear at all, although they are listed as available attributes. This makes it impossible for the model to learn them. Therefore, of the 105 attributes only 44 attribute that can be found in at least 5% of the dataset are used for training the network. For further details regarding information about the dataset, e.g. the frequency of different attributes, see Appendix B.

### 4.2.2 Preprocessing the Dataset

Preprocessing the input is necessary, since the images of the dataset have different shapes. Furthermore, in order to use the adapted network structure including pre-trained weights, the input images have to be converted into a particular form. Since the PETA dataset is very heterogeneous and differs in image shape and size, all contained images are resized to the same size of $128 \times 48 \times 3$ pixel. The benefit of this procedure is, that the network has to learn to find the person within the image, which makes it robust over badly cropped bounding boxes. Nevertheless, it is doubtful if the network is able to adapt this behavior by training on such a small dataset, since most of the images are already cropped correctly showing only the body. Typically, pixel values at each color channel range between 0 (*dark*) and 255 (*bright*). For the training, the values of each pixel are normalized to the range between $-1$ and 1.

Each person in the dataset is associated with a set of attributes. These sets contain all attributes that are represented by the corresponding person. In order to train on

**Algorithm 1** Initial dataset generation

1: $S_{train} \leftarrow \emptyset$ ▷ Final training dataset
2: $S_{valid} \leftarrow \emptyset$ ▷ Final validation dataset
3: Construct list $L_{im} = \{x_0, ..., x_n\}$ of all available images from PETA dataset
4: Generate set $D$ of person identifiers from $L_{im}$ ($|D| = m$)
5: Construct list $L_{att} = \{\mathbf{y}_0, ..., \mathbf{y}_m\}$ of attribute vectors for persons $p \in D$
6: Split $D$ into two disjoint sets $D_{train}$ and $D_{valid}$ ▷ Random or by sub-dataset
7: **for** $i \in \{train, valid\}$ **do**
8:  **for all** $p \in D_i$ **do**
9:  Gather all images $X_p = \{x \in L_{im} \mid x \text{ shows person } p\} \subset L_{im}$
10:  **for all** $x_j \in X_p$ **do** ▷ $h$ height, $w$ width of image
11:  Normalize image: $x_j \leftarrow \frac{2x_j}{255} - 1 \in [-1, 1]^{h \times w \times 3}$
12:  Resize image: $x_j \leftarrow f(x_j)$ with $f : [-1, 1]^{h \times w \times 3} \rightarrow [-1, 1]^{128 \times 48 \times 3}$
13:  $S_i \leftarrow S_i \cup \{(x_j, \mathbf{y}_p)\}$
14:  **end for**
15:  **end for**
16: **end for**

these attribute sets, they have to be converted to binary vectors. The elements of these vectors are '1' when a particular attribute is available for the person and else '0'. Each resulting attribute vector $\mathbf{y}_p \in \mathbb{R}^{105}$ is then paired with all corresponding images $x_j \in X_p$ of person $p$. By writing the preprocessed data back to disk, this step has to be performed only once for all following trainings.

The described procedure is shown in Algorithm 1 in a more compact way. All performed steps are necessary due to the requirements of the adapted model structure.

### 4.2.3 Data Augmentation

For the purpose of Deep Learning, it is very important to have large amounts of data. As stated earlier, the used PETA dataset is very small compared to typical datasets for Deep Learning, which consist of millions of samples. Hence, the model is very likely to overfit, while training it on the dataset. Therefore, a necessary step is to augment the

data, which increases the size of the dataset. This is typically done by transforming the already available data. Depending on the type of data, different kinds of augmentation techniques can be used. In the case of images, there exist different transformations that can be performed randomly: *translation*, *rotation*, *flipping*, *stretching*, and *shearing* are examples for such transformations. Even adding a bit of noise to the input can be seen as a special kind of (random) data augmentation. However, it is very important to perform operations that preserve labels. Since many attribute labels relate to colors, adding noise is critical due to possible changes in color appearances. Therefore, this kind of augmentation was not considered within this master's thesis. In addition to the decision what kind of augmentation to use, the question at which point of the algorithm to perform augmentation must be answered. Concerning this, two kinds of data augmentations exist: *online* and *offline* respectively. In the scope of this master's thesis, data augmentation is performed online at each epoch of the training. This means, for each image of the original training dataset, a transformed copy is generated. The training is then performed on these copies. This is repeated in every epoch.

**Algorithm 2** Data augmentation at epoch *m*

1: $S_{train} \leftarrow \{(x_0, \mathbf{y}_0), ..., (x_{n-1}, \mathbf{y}_{n-1})\}$ ▷ Training dataset
2: $S^m_{aug} \leftarrow \emptyset$ ▷ Augmented dataset for epoch $m$
3: $p_{flip} \leftarrow 0.5$ ▷ Probability for flipping image
4: **for all** $(x_k, y_k) \in S_{train}$ **do**
5: $\theta \leftarrow$ draw uniformly from $[-5°, 5°]$
6: $\Delta x \leftarrow$ draw uniformly from $[-0.05 \cdot w, 0.05 \cdot w]$ ▷ width $w$ of image
7: $\Delta y \leftarrow$ draw uniformly from $[-0.05 \cdot h, 0.05 \cdot h]$ ▷ height $h$ of image
8: Flip image horizontally $x^{aug}_k \leftarrow flip(x_k)$ with probability $p_{flip}$
9: Rotate image $x^{aug}_k \leftarrow rot(x^{aug}_k, \theta)$ around center of image
10: Shift image $x^{aug}_k \leftarrow shift(x^{aug}_k, \Delta x, \Delta y)$
11: $S^m_{aug} \leftarrow S^m_{aug} \cup \{(x^{aug}_k, \mathbf{y}_k)\}$
12: **end for**

The downside of this decision is that each epoch of training takes more time, whereas the algorithm benefits from lower memory and disk space consumption at the same time. Additionally, performing augmentation online results in a potentially larger "virtual" dataset, compared to a pre-augmented dataset. Here, the augmentation includes randomly flipping, rotating, and translating images. Algorithm 2 shows the performed augmentation. It is important to know, that the different performed transformations like translation and rotation, lead to "empty" pixels. This is shown in Figure 4.4 by the example of rotation and translation. As previously reported in 4.2.2, all pixel values range between -1 and 1. This means, a medium gray color corresponds to pixels with value 0. Filling empty pixels with 0's is beneficial to the training, since they do not influence subsequent layers.

**Figure 4.4:** Empty pixels
*Rotation and translation lead to lost (gray) and empty (shaded) pixels.*

### 4.2.4 Loss Function

Last of all, this part of the chapter takes a look at the used loss function. This function is very important for the training, as it quantifies the training performance of the model and serves as the objective for the used optimizer. Therefore, in order to train the network, the broadly used cross-entropy ($CE$) is adapted. As [Patrice Y. Simard, 2003] showed, using cross-entropy for training classifiers improves performance and allows to train even faster compared to traditional mean-squared error. For a single sample $x_k \in X$, a binary classifier $f : X \mapsto [0, 1]$ with $f(x_k) = \tilde{y}_k$, and the teacher $y_k \in \{0, 1\}$, the cross-entropy can be written as displayed in Equation 4.4.

$$\mathrm{CE} = -y_k \ln \tilde{y}_k - (1 - y_k) \ln(1 - \tilde{y}_k) \tag{4.4}$$

This equation can be expanded in order to compute the loss of $n$ samples.

$$\mathcal{L}' = -\frac{1}{n} \sum_{i=1}^{n} y_i \ln \tilde{y}_i + (1 - y_i) \ln(1 - \tilde{y}_i) \tag{4.5}$$

Since the problem addressed in this master's thesis is a multi-label instead of a binary classification problem, the current loss function has to be adjusted. As previously

stated, the model predicts $c$ classes for a sample $x_k$. Each class $C_l$ can be seen as a binary classification problem. For each class $C_l$, the cross-entropy formula in Equation 4.4 can be adapted, leading to the formula 4.6.

$$\mathrm{CE}_l = -y_{kl} \ln \tilde{y}_{kl} - (1 - y_{kl}) \ln(1 - \tilde{y}_{kl}) \tag{4.6}$$

Extending this formula analogous to Equation 4.5 leads to the following final loss function, which combines multiple losses.

$$\mathcal{L} = -\frac{1}{n \cdot c} \sum_{i=1}^{n} \sum_{j=1}^{c} y_{ij} \ln \tilde{y}_{ij} + (1 - y_{ij}) \ln(1 - \tilde{y}_{ij}) \tag{4.7}$$

This loss function was also used by [Pala, 2016] and [Su et al., 2016] for the purpose of attribute classification.

## 4.3 Experiments

With the knowledge about the chosen architecture and training procedure in mind, this chapter takes a look on the practical part of this master's thesis. It begins with the information about the used validation dataset in Section 4.3.1. The different metrics used for evaluation are introduced and explained in Section 4.3.2, followed by information about the environment of the experiments in Section 4.3.3. Finally, Section 4.3.4 evaluates and discusses different examined approaches.

### 4.3.1 Validation Dataset

In the scope of this master's thesis, the VIPeR dataset is used for validation at all experiments. Since it is a sub-dataset of PETA, it was excluded from the training dataset. The VIPeR dataset is composed of 1,264 images of 632 persons. For each person, there are two images, each from a different perspective. Further and more detailed information about the composition of the complete PETA dataset can be found in Section 4.2.1 and Appendix B.

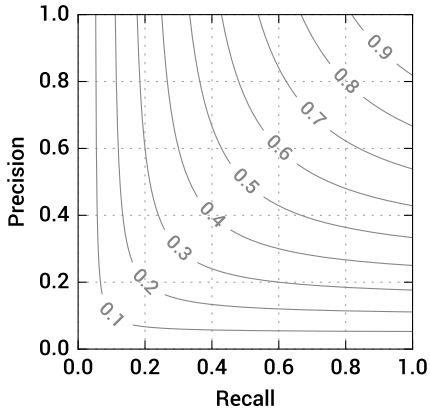



**Figure 4.5:** Contour lines of the $F_1$ score with reference to precision and recall
*The $F_1$ score takes values between zero and one, with small values indicating poor and high values good classification performance.*

### 4.3.2 Evaluation Metrics

In order to evaluate the classification performance of the developed system, different metrics were taken into account. The metrics were chosen for comparison with existing approaches and own variations of the used architecture. In the following, these metrics are shortly presented.

#### $F_1$ score

For general attribute prediction performance, the $F_1$ score was chosen. The $F_1$ score describes the harmonic mean between precision $= \frac{tp}{tp+fp}$ and recall $= \frac{tp}{tp+fn}$ of a classifier.

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.8}$$

The $F_1$ score is a special case ($\beta = 1$) of the $F_\beta$ score. The parameter $\beta \in \mathbb{R}^+$ controls the influence of the precision within the score. A greater value $\beta > 1$ weights recall higher than precision. On the other side, small values $0 < \beta < 1$ weight precision higher than recall. This general form is shown in Equation 4.9.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \tag{4.9}$$

Since precision and recall range between 0 and 1, the following property is always satisfied.

$$0 \leq F_\beta \leq 1 \qquad \forall \beta \in \mathbb{R}^+ \tag{4.10}$$

An $F_\beta$ score value near 1.0 indicates a good performance of the classifier. Figure 4.5 illustrates the behavior of the $F_1$ score with reference to precision and recall.

### Attribute Classification Accuracy

This Attribute Classification Accuracy was proposed by [Su et al., 2016]. Taking this metric into account allows to compare the performance of the developed model to the SSDAL algorithm presented in Section 3.2.2.

For a sample $x_k \in X$, let $n_k \in \mathbb{N}$ be the number of ones in the attribute label vector $\mathbf{y}_k = (y_1^k, \ldots, y_n^k) \in \{0,1\}^n$. For the predicted attribute label vector $\tilde{\mathbf{y}}_k = (\tilde{y}_1^k, \ldots, \tilde{y}_n^k) \in [0,1]^n$, the index set

$$I_k = \{\ i \mid \tilde{y}_i^k \text{ is one of } n_k \text{ largest elements of } \tilde{y}_k\ \} \subset \mathbb{N} \tag{4.11}$$

contains the indices of the top-$n_k$ components of the predicted attribute label vector. Additionally, the set $J_k$ contains the indices of all components of $y_k$ that equal '1'.

$$J_k = \{\ j \mid y_j^k = 1\ \} \subset \mathbb{N} \tag{4.12}$$

Therefore, Equation 4.13 shows the resulting formula for the Attribute Classification Accuracy.

$$\text{ACA} = \frac{|\ I_k \cap J_k\ |}{n_k} \tag{4.13}$$

From $|\ I_k\ | = |\ J_k\ | = n_k$ directly follows that $0 \leq |\ I_k \cap J_k\ | \leq n_k$. Hence, the final score ranges between 0 and 1. For a better understanding of the ACA score, Figure 4.6 illustrates how to determine the score with an example.

A major drawback of this metric is that only a fraction of all attribute labels is considered for the calculation of the accuracy. For instance, if all $\tilde{y}_i^k$ have a high value and $I_k = J_k$, the score would be 1.0 despite the fact that all remaining attribute labels are predicted wrong. However, in order to compare the performance to the SSDAL algorithm, the metric is considered as well.

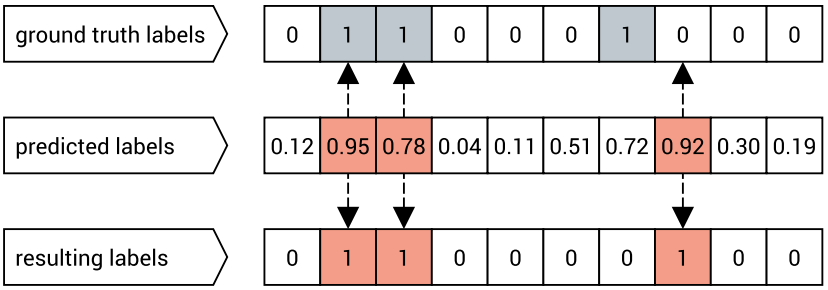



**Figure 4.6:** Attribute Classification Accuracy

*The number of 1's in the ground truth label vector is $n_k = 3$. Therefore, the three largest values of the predicted label vector are assumed as 1 and compared to the ground truth label vector. For this example, the ACA = 2/3.*

### 4.3.3 Experimental Setup

The following paragraphs gives a short overview of the used hardware and software for all experiments performed. The here presented setups are used for every experiment in Chapter 4 and 5.

#### Hardware

All experiments presented in this master's thesis were performed on the machines listed in Table 4.3. Each computer has a built-in nVidia graphic card, which allows to perform many computations on the GPU instead of on the CPU. This is much faster and cheaper, compared to a similar number of CPUs with the same performance. Furthermore, due to the number of available computers, it was possible to run up to five experiments at the same time. Hence, substantially more experiments could be performed in the time for this master's thesis. Machine 0 was mainly used for development and first experiments, since it was the slowest computer of all as displayed in Table 4.4. The remaining computers were primarily used for conducting different experiments. Due to the massive amount of memory, machines 2 and 3 allowed to keep the complete dataset in memory, which was beneficial for some experiments.

| Mach. | CPU | GPU | Mem. |
|---|---|---|---|
| 0 | AMD Phenom II X4 945 | nVidia GeForce GTX 960 (4 GB) | 12 GB |
| 1 | Intel Core i5 6600 | nVidia GeForce GTX 970 (4 GB) | 8 GB |
| 2 | 2 × Intel Xeon E5-2650 v3 | 2 × nVidia Tesla K20c (5 GB) | 505 GB |
| 3 | 2 × Intel Xeon E5-2650 v3 | nVidia Tesla K20c (5 GB) | 505 GB |

**Table 4.3:** Different machines used for training
*Machine 0 was mainly used for development and training of prototypes. Actual experiments were performed on machines 1, 2 and 3. However, the two graphic cards of machine 2 were used rather for training two networks at once, than for boosting the performance of a single training process. Due to the fact of having multiple machines, significantly more experiments could be performed.*

| Machine | Avg. time |
|---|---|
| 0 | 36.4776 sec. |
| 1 | 17.2367 sec. |
| 2 | 32.2975 sec. |
| 3 | 32.3943 sec. |

**Table 4.4:** Duration of training epochs
*The final architecture as presented in Section 4.1.1 was trained on each machine for about 3,000 epochs. The right column shows the average duration of a training epoch.*

### Deep Learning Framework

For the implementation of the developed Deep Learning model, *Keras* [Chollet, 2015] was used. Keras is a minimalist, highly modular neural networks library written in Python and is capable of running on top of either TensorFlow or – *in this case* – Theano [Theano Development Team, 2016]. It was developed with focus on enabling fast experimentation and developing prototypes with little effort [Chollet, 2015]. Since the Neuroinformatics and Cognitive Robotics Lab mainly uses Python and Keras for Deep Learning development, the decision to use the mentioned library was predetermined.

### 4.3.4 Performance Comparison

After the presentation of relevant information on the previous pages, this chapter continues with the evaluation of the performed experiments. These were conducted in order to determine the best architecture and settings for the purpose of attribute prediction. All evaluations were done on VIPeR dataset, as mentioned in Section 4.3.1. Unless otherwise stated, all displayed plots in this chapter show the corresponding mean score values over all attributes.

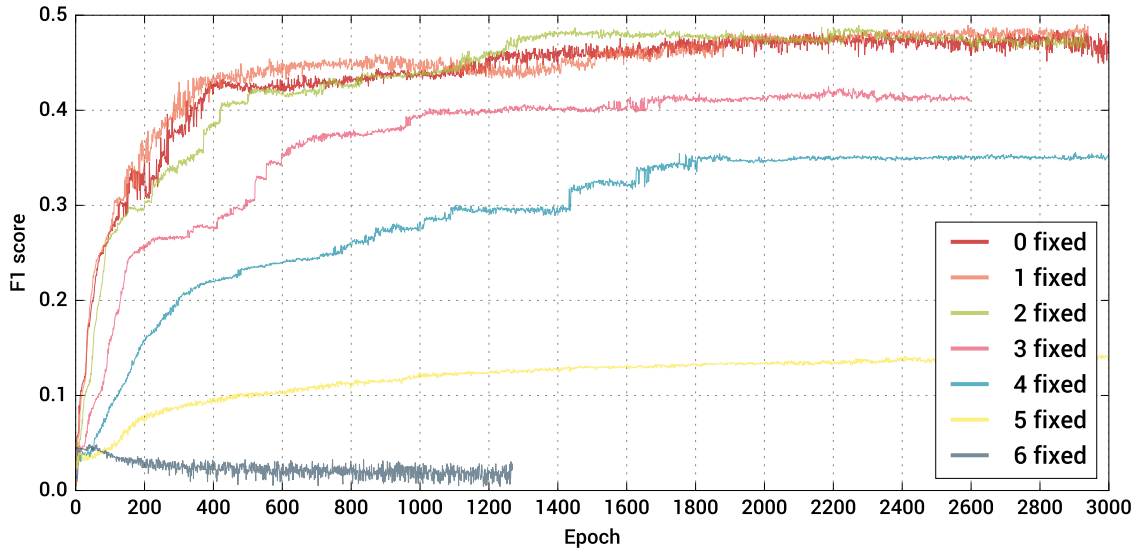



**Figure 4.7:** Different numbers of fixed layers for training

*This figures shows the $F_1$ scores for different trainings. Higher values are better. The experiment shows, that it is better to fix less layers. This means that filter in deeper layers are not as useful for the prediction of attributes, as the filters in the first layers. For details about the training, see Table A.1 in Appendix A.*

#### Experiments on pre-trained network

As already mentioned, the original idea for this master's thesis was to perform fine-tuning on a pre-trained architecture in order to predict attributes. However, different experiments showed that fine-tuning weights of the pre-trained network

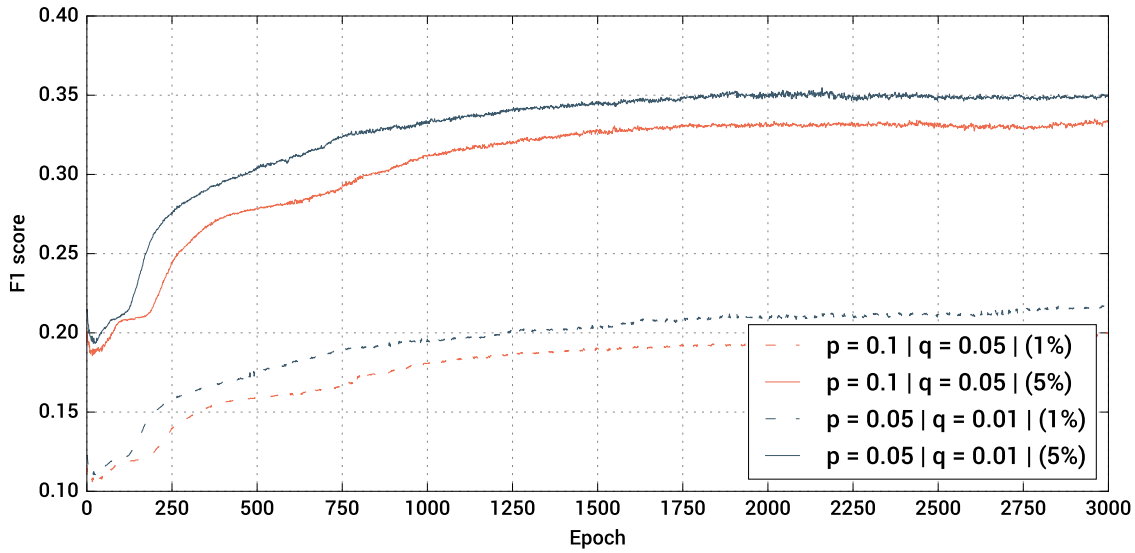



**Figure 4.8:** Two trainings compared by amount of noise added
*In both trainings (orange and blue) the weights for the first five layers were loaded. Additionally, the first three layers were fixed, i.e. excluded from training. The remaining network weights were initialized randomly. Of the two layers that were imported but not fixed, p percent of the filters were re-initialized randomly. Furthermore, zero-centered normal distributed noise (standard deviation of q) was added to the filters in both layers as well. The solid lines consider attributes which appear in at least five percent of the dataset. The dotted lines show those in at least one percent of the dataset. The training was performed on all attributes. The curves show, that by removing filters from the network, or disturbing them by adding noise, the performance worsens. Therefore, the pretrained network contains weights that are already useful and contribute to the prediction of attributes.*

was inferior to a training from scratch. As mentioned earlier, the model consists of eight layers. Due to the changes on the output layer, the existing weights cannot be imported for the last layer. This means that, at most, weights for the first seven layers of the model can be imported. First, Figure 4.7 shows seven trainings on a subset of the available attributes, namely the upper body colors. The upper body color is one of four multi-class attributes and can assume one out of eleven different colors. This first experiment investigated the influence of the number of fixed layers

on the training of the network. Therefore, each time, all weights were loaded from the original network, but different numbers of layers were excluded from training. For instance, the yellow curve shows the performance when fixing the first five layers. Apparently, excluding too many layers from training leads to a significant drop of the classification performance. However, the best performance was achieved by experiments fixing not more than the first two layers. These results imply that the imported weights need even more adaption in order to be useful for classifying upper body colors. Since about 42 percent of all attributes are colors, it is important to have filters which react on this kind of features. Hence, keeping all filters from the pre-trained network could prevent the training from adapting the weights in the right manner.

Based on this result, additional experiments examined the effect of randomly re-initializing filters as well as adding noise to the filter weights. The aim of this experiment is to examine, if the imported but not fixed weights are useful at all. Figure 4.8 illustrates two selected trainings that were performed on all 105 attributes. In this experiment, two training runs are compared. The figure shows the performance evaluated on attributes that appear in at least 1% and 5% of the individuals of the dataset. Details about the training parameters can be found in Table A.2. Both experiments were identical except of the number of filters that were reset and the amount of noise added. At a first glance, both setups seem difficult to compare, due to simultaneous changes on the number of re-initialized filters and the amount of noise. Still, both changes have in common that they alter the existing weights. However, re-initializing is a more radical step, since each filter's weights get completely new random values. On the other side, adding a small amount of zero-centered normal distributed noise leads to weights that are similar to the original ones. Therefore, both can be seen as disturbance for the original layer weights. Summarizing the results displayed in Figure 4.8, in both cases – one percent and five percent of the attributes – the network with less disturbed weights performed best. This means, that the weights in the imported but not fixed layers are definitely useful for the attribute recognition. Therefore, the imported weights will not be discarded. Nevertheless, in the follow-

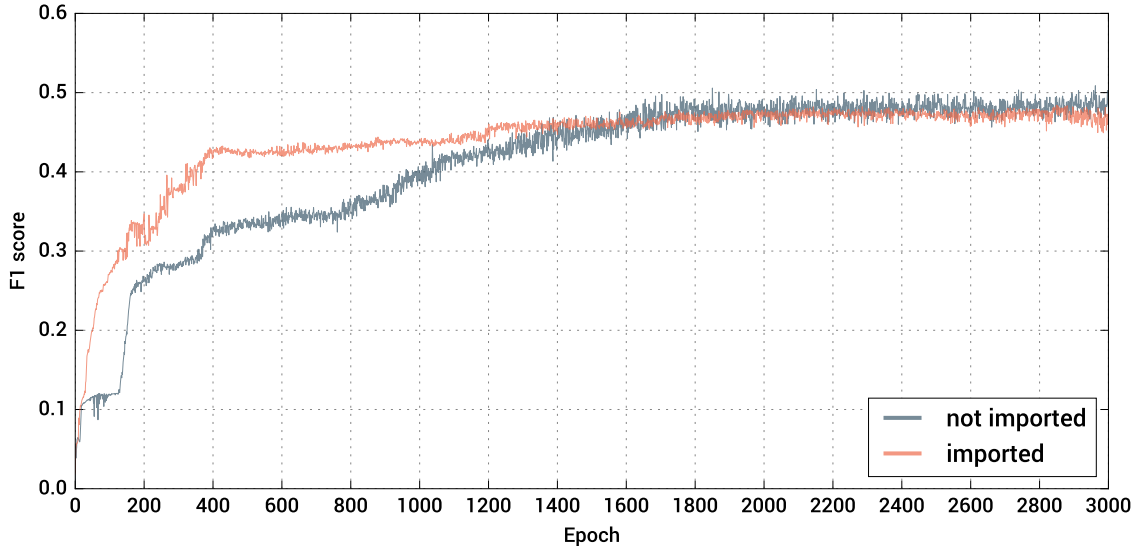



**Figure 4.9:** Training with and without imported weights

*Two networks with identical configurations were trained. One of them with imported weights, the other without. As the curves show, the pretrained network reaches faster a higher $F_1$ score and therefore performs better in early training stages. In later training stages, however, the network with randomly initialized weights performs slightly better. This result opens broad possibilities regarding architectural decisions. For detailed information about the training see Table A.1 in Appendix A.*

ing it will be examined how the network performs when no weights are imported at all.

In addition to the experiments from above, further were performed in order to compare the pre-trained version of the network, with a completely random initialized version. Figure 4.9 shows the results of this experiment. It is obvious that not importing any weights leads to a similar performance. However, it takes more epochs and, therefore, more time to achieve the same level of performance. In late training stages, the network with random initial weights performs slightly better, compared to the pretrained network. This means, that adapting the pretrained weights for the purpose of attribute prediction leads to fairly good results. Nevertheless, training from scratch seems equally well or potentially even better suited for the purpose of predicting attributes.

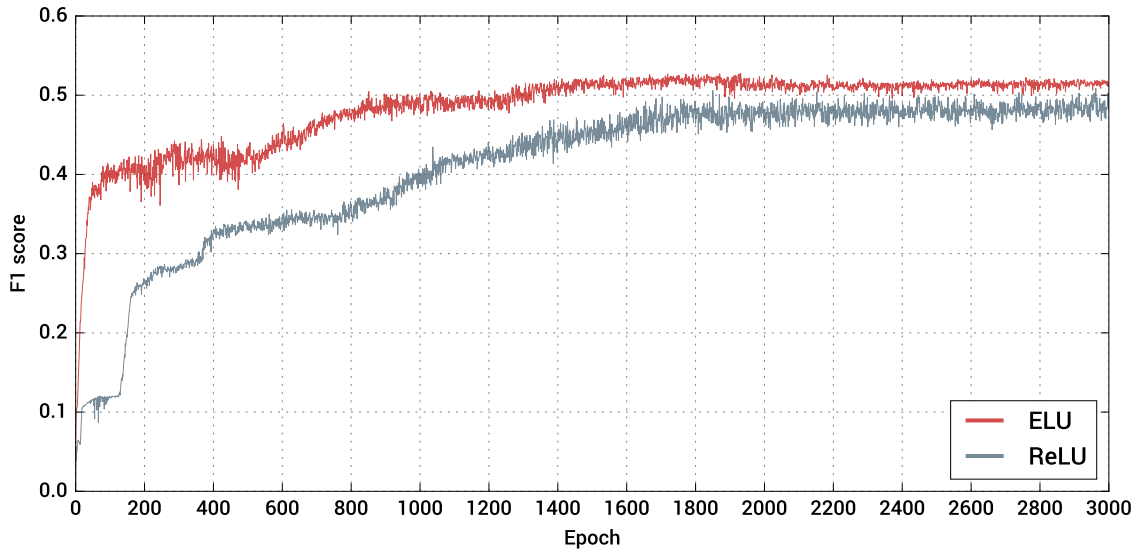



**Figure 4.10:** ELU versus ReLU

*This experiment examines the effect of using the ELU activation function. No weights are imported, so the architecture can be varied more freely. It is obvious, that with the same parameters, the ELU performs much better from the beginning. Even if the ReLU network comes closer at later stages, the ELU network stays superior. For detailed information about the training see Table A.3 in Appendix A*

This result opens broad possibilities regarding architectural decisions. Since keeping the weights of the pre-trained network is not necessary for predicting attributes, it offers the possibility to alter the original architecture. This possibly leads to even better results, which means the existing architecture is not limited to the original decisions made by [Eisenbach et al., 2016]. Furthermore, different techniques can now be used for training, for example several different optimizers or varying neuron activation functions. Especially, with imported weights, other optimizers were not applicable, since most of them like RMSprop, Adam, and Adagrad tend to behave very radically at the beginning of the training. Usually, this behavior destroys already trained weights, which makes them useless for the training.

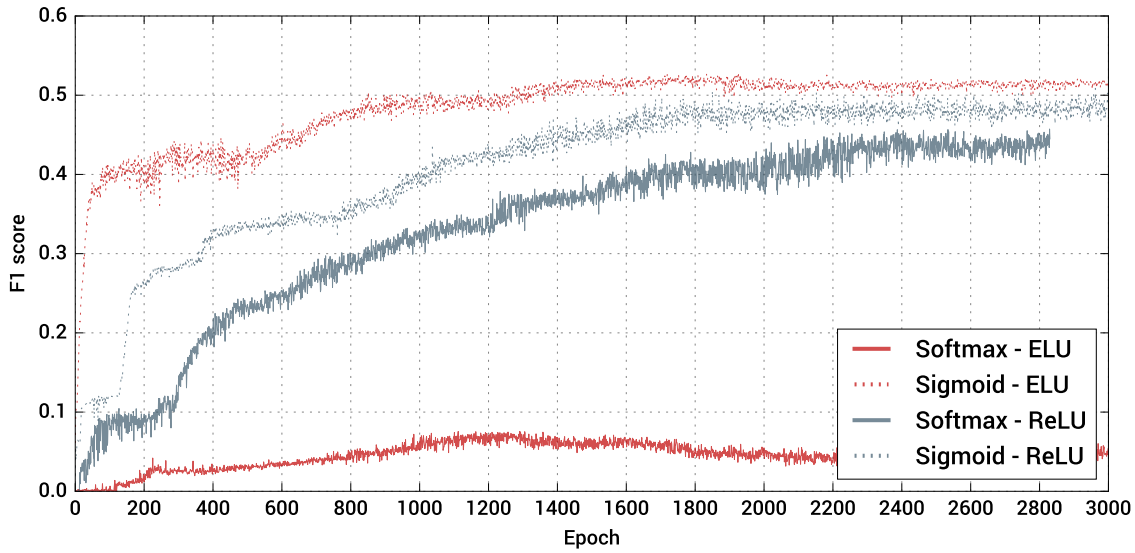
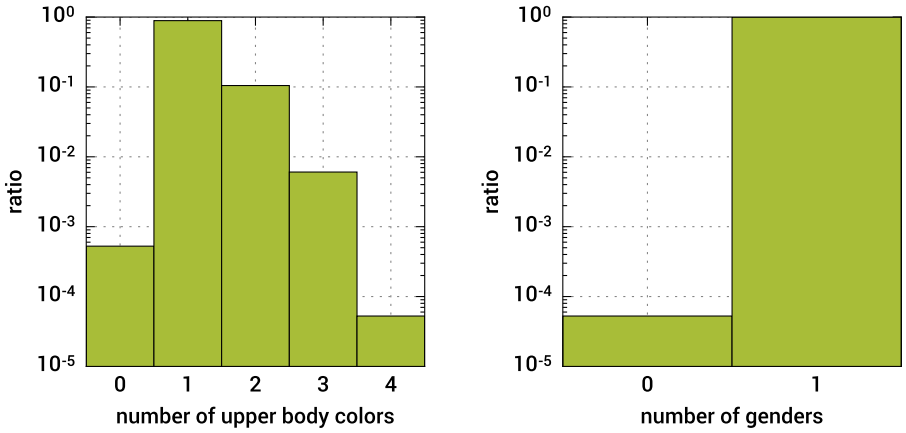



**Figure 4.11:** Softmax versus Sigmoid: Upper body color

*All networks were trained on upper body colors. According to [Deng et al., 2014], the upper body color is a multi-class attribute. Therefore, each image should be labeled with exactly one color. This experiment examines the effects on the classification performance, when using sigmoid or softmax activation function for the output layer. Both output functions are paired with ReLU and ELU activation for the hidden neurons. The experiment shows, that using sigmoid output activation performs better combined with ELU and ReLU activation function. The combination of ELU and Softmax, however, performs very bad. It seems that the combination of both activation functions is not suited for learning labels distributed like the upper body color (see Figure 4.12(a)). For further details about the training see Table A.4 in Appendix A.*

### Experiments without pre-trained weights

As mentioned above, not using pre-trained weights gives the opportunity to examine the effect of different changes on the architecture. Figure 4.10, therefore, shows that the use of ELU activation instead of ReLU boosts the performance even more. Not only does it reach a higher average $F_1$ score, but also a faster adaption to the problem as the first few hundred epochs already show. The reason as stated by [Clevert et al., 2016] is marginal influence of negative values. This may lead to a better adaption




(a) Upper body colors (b) Gender

**Figure 4.12:** Multi-class attributes
*Compared to the upper body color case, the gender labels (male and female respectively) show a "better" distribution of labels. Almost all persons have at most one label at the same time. Only one single person is labeled with neither female nor male gender.*

to the problem. In addition to that, the effect of different output functions was examined as well. Since the experiments shown in Figure 4.10 were performed on the upper body color – which is by definition of the PETA dataset a multi-class attribute – it seems natural to examine the effect of the softmax output function as well (see 4.1.2). Figure 4.11 displays the corresponding experiments. Therefore, four different combinations of ELUs (*red*) and ReLUs (*blue*) with sigmoid (*dotted line*) and softmax (*solid line*) activation were examined. Evidently, using sigmoid output achieves better results than using softmax output. Taking a look at the distribution of the number of upper body colors per person reveals, that the information given by [Deng et al., 2014] should be treated with caution. Actually, over ten percent of all persons are labeled with at least two different or no colors as illustrated in Figure 4.12(a). This is very likely to be the reason for the softmax output function to perform worse than the sigmoid output. Surprisingly, the performance is particularly poor for the combination of softmax output function and ELUs within the network. This result could be reproduced for multiple experiments. It seems that the combination of ELUs and softmax

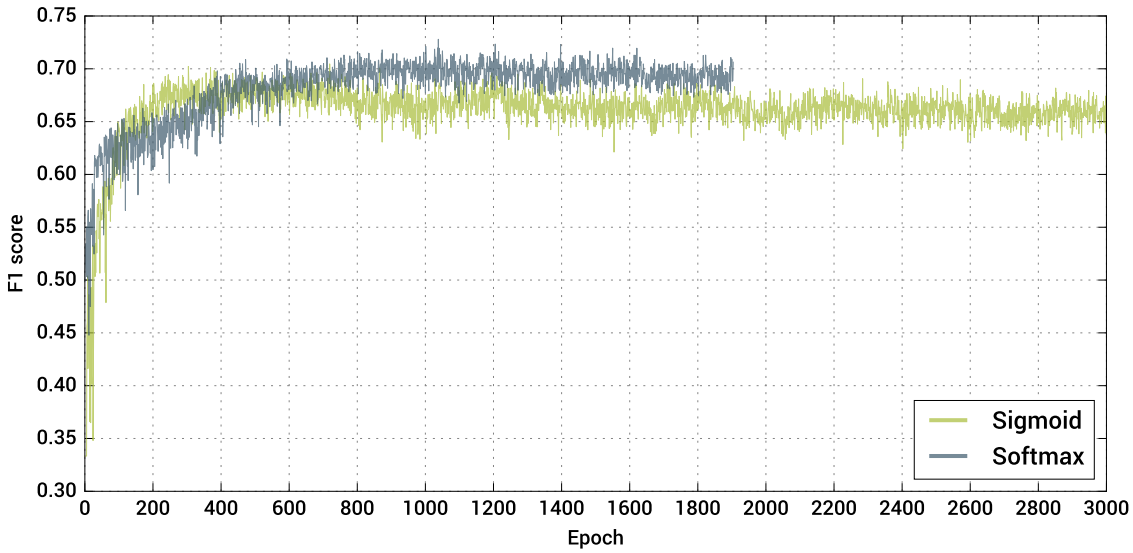



**Figure 4.13:** Softmax versus Sigmoid: Gender

*Both experiments were performed on gender attributes. The network with the softmax output performs better than the network with sigmoid output, as the higher $F_1$ score indicates. In addition to the experiment in Figure 4.11, this shows that the softmax activation function achieves better performance, if the distribution of the used labels is not too noisy. For details about the used learning parameters see Table A.4 in Appendix A.*

output is not suited for learning labels that are distributed like the upper body color. In addition, another experiment was performed on a different group, namely the gender of people. As Figure 4.13 shows, both ELUs and softmax work together and achieve very good results. In this case, softmax performs even better than sigmoid. This underlines the initial thoughts that softmax output should be avoided for multi-label problems. Compared to the experiments in Figure 4.11, each person has at most one label for gender as illustrated by Figure 4.12(b). Further research showed that only a single image in the whole dataset exists that has no gender label. This particular image is shown in Figure 4.14. Due to the difficult viewing angle and the bad cropping of the image, a final statement about the gender is not possible. Since this is a singular case, it has no considerable influence on the gender classification problem. Based on these results, the decision to use ELUs for the final version of the architecture was

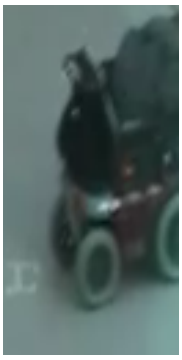


**Figure 4.14:** Genderless person
*This image is taken from the PRID dataset, another sub-dataset of PETA. It is the only occurrence of a person not labeled with any gender.*

made, since the networks with ELUs performed better in almost all cases. Motivated by the overall performance and the encountered difficulties of the softmax function, the decision has been made in favor of the sigmoid function for the output layer. Therefore, the sigmoid output function will be used for all following experiments. So far, stochastic gradient descent was used for optimizing the loss function, because it is broadly used by many researchers and shows good performance in many different cases. Another reason was, that the original weights of the pre-trained model were trained by using this optimizer. Due to the fact, that other optimizers often behave very radical, it was necessary to use the identical optimizer for the fine-tuning as well. For the initial experiments, the stochastic gradient descent parameters were chosen as proposed by [Eisenbach et al., 2016]. Thus, the learning rate was set to 0.01 and the momentum to 0.7. These are the parameters for all experiments performed so far.
Unless specified differently, for the following ones, only attributes that are available in at least five percent of all persons were considered. This decision is motivated by the assumption that labels that are available only in few samples are unlikely to contribute to the overall classification performance. Therefore, Figure 4.15 shows three different training runs. The red one used all of the 105 available attributes from the dataset for training. Afterwards, it was evaluated on 44 of the 105 attributes in order to compare it with the other trainings. The performance, though, stays below its direct competitor, depicted by the blue curve. This means, the training does not benefit from taking

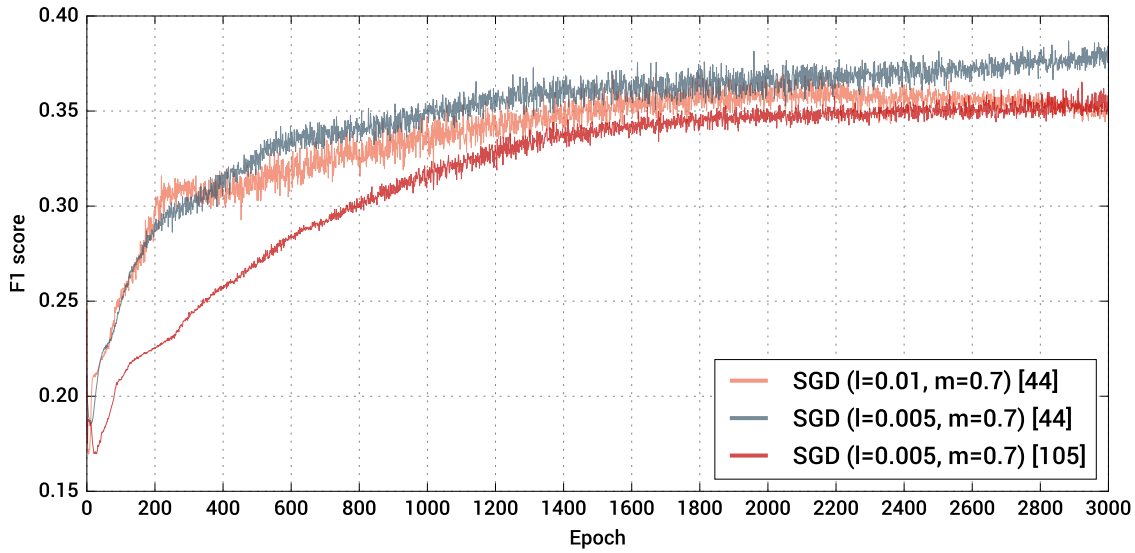
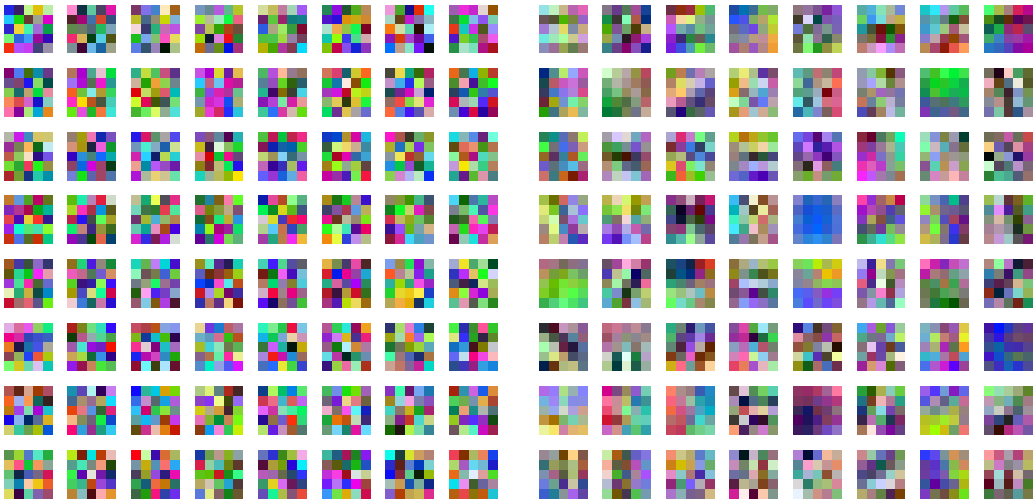



**Figure 4.15:** Parameter evaluation for stochastic gradient descent optimizer
*The stochastic gradient descent has multiple parameters to set up. Here, only learning rate and momentum were taken into account. The number in square brackets indicates the number of attributes on which the training was performed.*

the additional attributes into account. Both trainings differ only in the amount of presented labels, which is indicated by the number in square brackets. The blue and the orange curve show the average $F_1$ score when training directly on the attributes that are available in at least five percent of all persons. This time, the orange one is outperformed by the blue one as well. Of all tested learning rates, hence, a value of 0.005 has shown the best quality. Additional tests, so as to examine similar effects of the momentum were disregarded, since this would go beyond the scope of this master's thesis. As the experiment showed that considering only the attributes that are present in at least 5% of all individuals leads to a better classification performance, the decision was made to use only these for the rest of this master's thesis.
In a second experiment, the Adam [Kingma and Ba, 2014] optimizer has been examined in order to figure out whether the classification performance could be improved even more. The best results were achieved by Adam with a learning rate of $10^{-5}$. Independent of the chosen learning rates, all curves reach a hill after some hundred epochs, which is clearly visible in Figure 4.17(a). Despite the good performance on



(a) Adam (b) stochastic gradient descent

**Figure 4.16:** Filters from first convolution layer
*Resulting filters of the best network trained with Adam and stochastic gradient descent optimizer. The weights of the network trained with Adam look like they were in a very early stage of training. Since they correspond to the network with the best $F_1$ score, it seems that the network reached some local minimum. The weights of the stochastic gradient descent trained model, hence, seem much more cleaner, although many weights look still very noisy.*

the validation dataset, which is even better than the best stochastic gradient descent conducted, the weights still show a significant amount of noise. This indicates, that the network has not learned anything useful so far. Therefore, this leads to the assumption that the optimizer has reached a local minimum, suggested by the weights of the first convolution layer (see Figure 4.16). Furthermore, Figure 4.17(a) shows that the default learning rate is too high, as indicated by the orange curve. Decreasing the learning rate, hence, is beneficial to the performance on the validation data. But not all tested optimizers (SGD, Adam [Kingma and Ba, 2014], and Adagrad [Duchi et al., 2011]) were applicable. Especially, Adagrad [Duchi et al., 2011] performed surprisingly bad as shown in Figure 4.17(b). The optimizer seems to have problems with the ELUs, since the average $F_1$ score does not change at all for most of the time. This theory could not be verified yet due to the limited time for this master's thesis, and should

be addressed in future work.

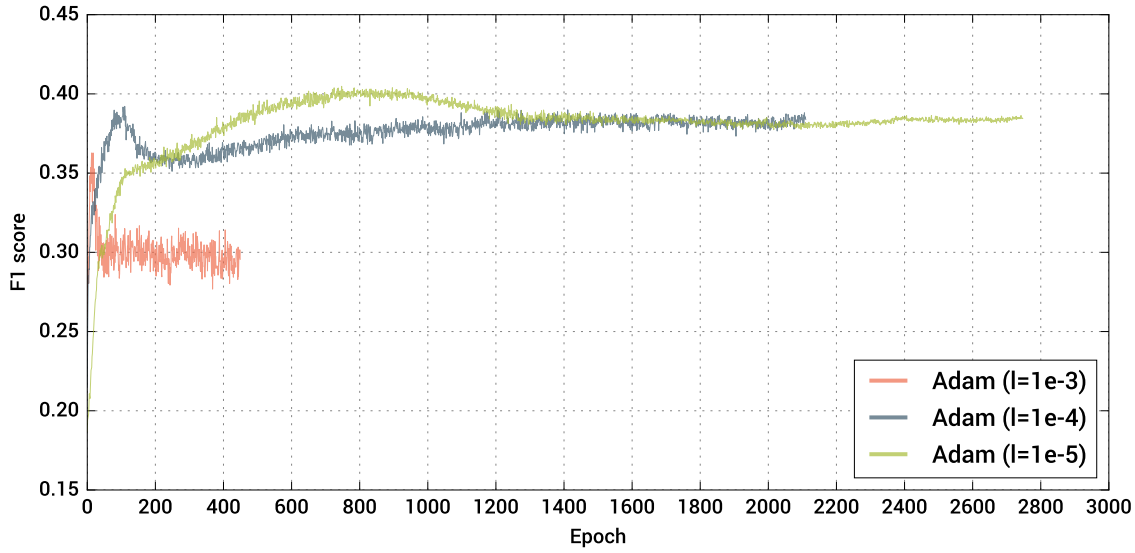
Finally, it can be summarized that almost all examined optimizers showed a fairly good performance. However, the most stable results were achieved by stochastic gradient descent, which needs more epochs and time respectively to achieve a comparable performance. Therefore, for the final architecture – in the following referred to as **attCNN** (attribute Convolutional Neural Network) – stochastic gradient descent was chosen as the optimizer, with a learning rate of 0.005 and a momentum of 0.7. These showed the best performance in all performed experiments. Even more optimizers exist that could be used for training, like Adamax [Kingma and Ba, 2014] or Adadelta [Zeiler, 2012]. But these were not taken into account, since analyzing them would go beyond the scope of this master's thesis.

### Comparison to other architectures and attempts

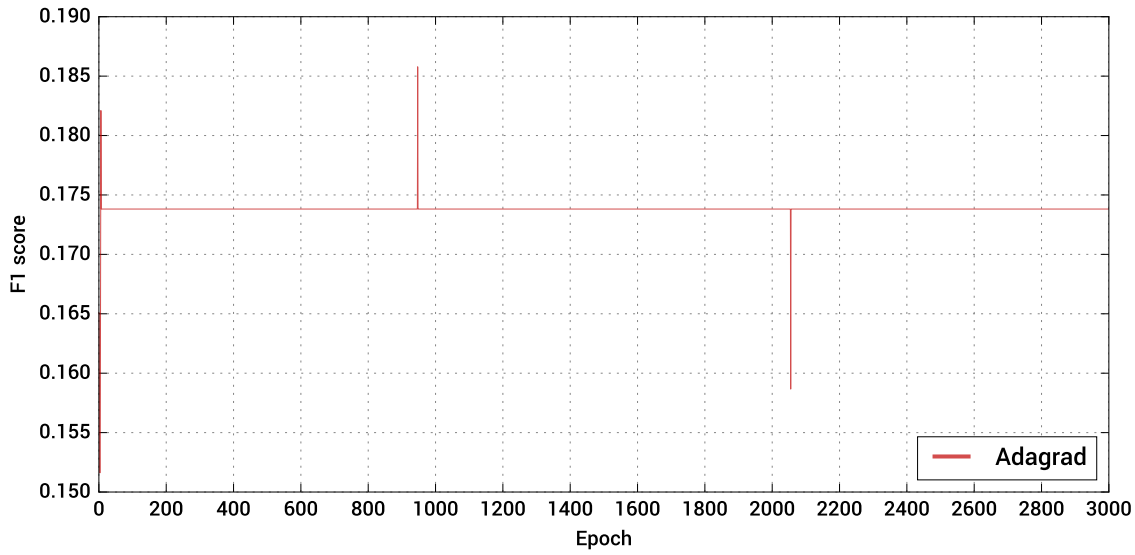
| Layer | Filt. | F. size | Activ. | MaxPooling | Dropout | Outp. Volume |
|---|---|---|---|---|---|---|
| conv1 | 32 | $9 \times 9$ | ELU | 2x2 | — | $60 \times 20 \times 32$ |
| conv2 | 64 | $7 \times 7$ | ELU | 2x2 | 0.25 | $27 \times 7 \times 64$ |
| conv3 | 128 | $6 \times 6$ | ELU | 2x2 | 0.25 | $11 \times 1 \times 128$ |
| fc1 | 500 | — | ELU | — | 0.3 | 500 |
| fc2 | 500 | — | ELU | — | 0.5 | 500 |
| output | 44 | — | sigmoid | — | 0.5 | 44 |

**Table 4.5:** Architecture of the reduced smallCNN model
*This architecture has less convolution layers and less fully-connected neurons. The parameters were chosen similar to the architecture shown in Table 4.1. By altering the network structure, the size of the network could be reduced from about four million to less than one and a half million weights.*

With the results from the previous experiments, the attCNN will be compared with different state-of-the-art methods, namely the approaches of [Pala, 2016] and [Su et al., 2016], in order to examine if it can outperform those. Motivated by the improvements achieved by modifying different parts of the original architecture, which led to the




(a) Adam




(b) Adagrad

**Figure 4.17:** Training with different optimizers

*Adam [Kingma and Ba, 2014] and Adagrad [Duchi et al., 2011] are further well known optimizer. It is recommended to use the default for most parameters. For Adam, however, the learning rate is variable and should be adapted to the problem. Furthermore, Adam seems to get stuck in a local minimum very fast. The experiment shows, that Adagrad seems to have problems, whereas Adam seems to be better suited than Adagrad for training the network.*

attCNN, a smaller model was derived from the attCNN, the so called smallCNN in order to examine, if the performance can be enhanced even more, with a less complex model. This mentioned model is evaluated here as well. The architecture for the latter, is shown in Table 4.5. The smallCNN has a lower number of layers, filters, and neurons in the fully-connected part of the network. It consists of less than one and a half million weights, which is a drastic reduction of the original attCNN size. Figure 4.18 shows

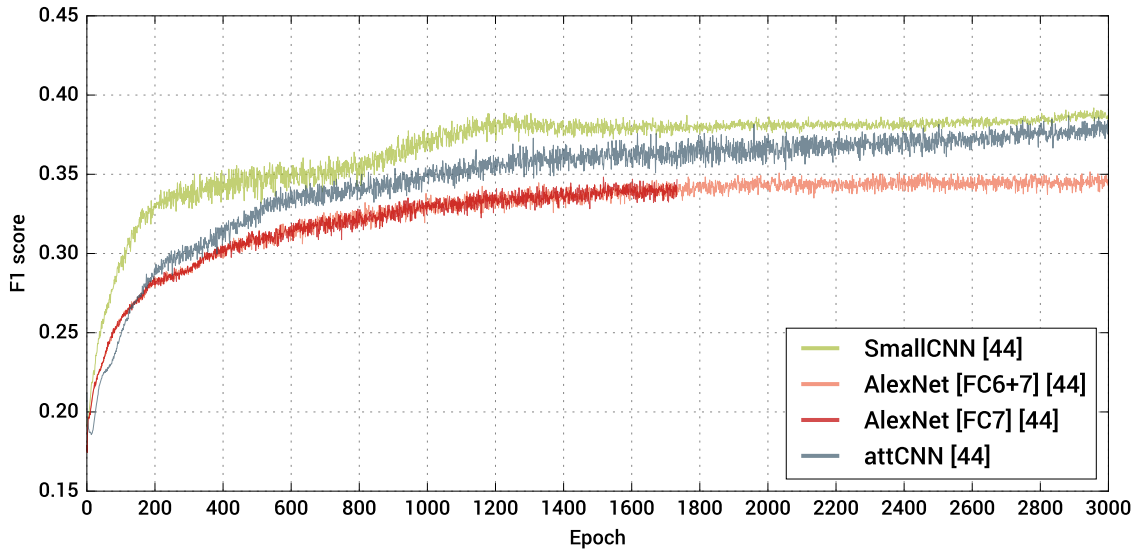



**Figure 4.18:** AttCNN vs. smallCNN vs. AlexNet (SSDAL [Su et al., 2016]) on 44 attributes

*The smallCNN achieves even better performance than the attCNN. Moreover, both beat the pre-trained AlexNet, for which the last (FC7) and last two fully-connected layers (FC6+7) respectively were trained.*

the consequences of these changes. Of all examined models, the smallCNN achieves the best performance, followed by the attCNN and the two AlexNet approaches. The most likely reason is the reduced complexity of the model, which is preferable for a small dataset. This reduction improves the generalization capability of the network and, thus, leads to better performance. As the green curve indicates, the performance of smallCNN is constantly above all other presented algorithms. However, over time, attCNN gets very close to the performance of smallCNN. As the smallCNN and the attCNN achieve almost the same performance, the decision was made to use the latter

for the following experiments, since it is more similar to the original adapted network by [Eisenbach et al., 2016].
In the following, the attCNN will be compared to approaches of [Pala, 2016] and [Su et al., 2016]. Since [Su et al., 2016] only used the attribute classification accuracy for evaluating the attribute prediction performance, this metric was adapted here as well. In addition, the reproduction of the network from stage 1 of the SSDAL algorithm was attempted. Since stage 1 simply consists of fine-tuning a pre-trained AlexNet, this could be achieved relatively fast. Due to the lack of information about the fine tuning, it was attempted to follow the procedure presented in [Krizhevsky et al., 2012], which seemed to be the most likely way [Su et al., 2016] implemented it. These are shown in Figure 4.18 as well. Both experiments used pre-trained weights for all layers except for the last or the last two layers respectively. Obviously, both trained AlexNets stayed below the competing attCNN and smallCNN. This outcome is also confirmed by the resulting Attribute Classification Accuracy (ACA) scores in Table 4.6.

| **Method** | **ACA (105)** | **ACA (44)** |
|---|---|---|
| SSDAL [Su et al., 2016] | 0.586 | — |
| AlexNet | 0.637 | 0.639 |
| attCNN | 0.651 | 0.654 |
| smallCNN | — | 0.655 |

**Table 4.6:** Comparison of attribute classification accuracy
*In order to compare attCNN, smallCNN, and the re-implemented fine-tuned AlexNet of the SSDAL approach [Su et al., 2016], the attribute classification accuracy was evaluated. Both, attCNN and smallCNN, outperform the SSDAL algorithm. The number in parentheses indicates the number of attributes on which the metric was evaluated.*

The score for SSDAL is taken from the paper as they evaluated the algorithm only on all 105 attributes. Since the AlexNets and attCNN were trained in the context of this master's thesis, both scores could be determined. Due to the limited time, smallCNN was only evaluated on the 5-percent-subset of the attributes and, therefore, does not provide any value for the training on all attributes.

The second algorithm, to which this master's thesis is compared to, is the already mentioned one by [Pala, 2016], which was presented in Section 3.2.2. [Pala, 2016] use three almost identical Convolutional Neural Networks for different parts of the body, namely head, torso, and legs. These use less but larger filters in the first layer, as well as less layers (only two convolution and two fully-connected layers). This results in a total of up to half a million weights per network.

Table 4.7 shows all attributes that were considered for training, meaning all attributes that are available for at least five percent of individuals. The table demonstrates that attCNN achieves better performance for almost all attributes compared to the tripartite attempt of [Pala, 2016]. Since [Pala, 2016] ignored some attributes, the corresponding entries in the table are marked with "—". Despite both the large architecture and the relatively small size of the dataset, the developed model achieves better results. Especially, splitting it into different body parts is not required in order to predict all considered attributes. It could be possible that the performance can be enhanced further, since the weights shown in Figure 4.16(b) seem not have reached their final form, which is indicated by their noisy appearance. In order to achieve this improvement, one step could be to add further regularization like L1- and L2-regularization to the layers of the network. Whether this improves the performance should be evaluated in future work.

| **Attribute** | **attCNN** | **Pala** | **Attribute** | **attCNN** | **Pala** |
|---|---|---|---|---|---|
| footwearBlack | **0.5255** | — | personalFemale | **0.7229** | — |
| footwearBrown | **0.0588** | — | lowerBodyFormal | **0.1389** | 0.026 |
| footwearGrey | **0.0945** | — | upperBodyFormal | 0.1565 | **0.163** |
| footwearWhite | **0.4903** | — | upperBodyJacket | **0.0766** | 0.049 |
| hairBlack | **0.8051** | 0.779 | lowerBodyJeans | **0.7717** | 0.701 |
| hairBrown | **0.3911** | 0.365 | footwearLeatherShoes | **0.3000** | — |
| lowerBodyBlack | **0.6954** | 0.587 | upperBodyLogo | **0.1070** | 0.104 |
| lowerBodyBlue | **0.6250** | 0.573 | hairLong | **0.7252** | 0.577 |
| lowerBodyGrey | 0.4336 | **0.442** | upperBodyLongSleeve | **0.7740** | 0.773 |
| upperBodyBlack | **0.7736** | 0.659 | personalMale | **0.7511** | — |
| upperBodyBlue | **0.6191** | 0.477 | carryingMessengerBag | **0.3018** | — |
| upperBodyBrown | **0.1595** | 0.128 | accessoryNothing | **0.5922** | — |
| upperBodyGrey | **0.3529** | 0.265 | carryingNothing | **0.4372** | — |
| upperBodyRed | **0.7569** | 0.724 | footwearShoes | **0.3867** | — |
| upperBodyWhite | **0.7006** | 0.618 | hairShort | **0.8228** | 0.649 |
| personalLess30 | **0.8480** | — | lowerBodyShorts | 0.3020 | **0.320** |
| personalLess45 | **0.2325** | — | upperBodyShortSleeve | 0.4248 | **0.467** |
| personalLess60 | **0.1159** | — | lowerBodyShortSkirt | 0.2155 | **0.275** |
| carryingBackpack | **0.5192** | — | footwearSneakers | **0.4379** | — |
| carryingOther | **0.1826** | — | lowerBodyTrousers | **0.6099** | 0.529 |
| lowerBodyCasual | **0.9905** | 0.972 | upperBodyTshirt | 0.3958 | **0.453** |
| upperBodyCasual | **0.9814** | 0.978 | upperBodyOther | **0.6179** | - |

**Table 4.7**: Comparison of area under the precision-recall-curve

*This table shows the attributes used for the training of the network. They are represented by at least five percent of the individuals. [Pala, 2016] focused on training separately on the head, torso, and leg part. The foot part and some soft biometric features were not considered for training and, therefore, marked with "—". For each attribute, the better value is highlighted in bold.*

## 4.4 Summary

As this chapter showed, predicting attributes is a difficult problem. Many different aspects have influence on the training of a Convolutional Neural Network for attribute prediction. Especially, the relatively small amount of labeled data is a crucial point. In order to mitigate this problem, different techniques and architectural details have to be considered. The chapter showed, that the initial idea to adapt a pre-trained network in order to get around the problem of small amounts of data, achieves fairly good results. With particular changes on the original architecture and training methods, the model, as trained from scratch, accomplishes even better results. This implies switching the activation from ReLU to ELU, decreasing the learning rate, and reducing the amount of labels considered for training, improves the classification performance.

The final attCNN, hence, outperforms the architecture of [Pala, 2016] in almost all cases and does not need splitting into different body parts. Furthermore, considering the performance metric introduced by [Su et al., 2016], the attCNN presented in this chapter achieves better results than the SSDAL algorithm [Su et al., 2016]. Further reduction of the network size leads to a slightly better performance. Since the attCNN seems not to have reached its final performance yet, it might be able to outperform the smallCNN with further improved training.

Despite the evaluation of many parameters, the experiments suggest, that the performance might be improved further. Therefore, in future work e.g. further regularization techniques and experiments using modified architectures should also be addressed. But this is beyond the scope of this master's thesis.

# Attribute-based Person Re-Identification

# 5

The first part of this master's thesis focused on the prediction of semantic attributes. For this reason, a Deep Convolutional Neural Network was introduced in order to recognize attributes on images of persons. The aim of the second part of the developed algorithm is to perform person re-identification based on these predicted attributes. Given an attribute vector of a certain person, it is necessary to rank all other persons from the probe set by their attribute representation. Therefore, two different approaches were developed. First, a distance network was implemented, which uses triplet-based training in order to learn a distance function, which allows to rank attribute vectors by these distances. Furthermore, a second network, which classifies two persons by their similarity to a probe person, is presented afterwards. The architectures of these different models is presented in Section 5.1, followed by details about the underlying dataset and training in Section 5.2. Finally, the performed experiments and results are shown and discussed in Section 5.3.

## 5.1 Model Architecture

As mentioned above, this part of the chapter addresses the architectural details of the two networks. Both have in common that they are multilayer perceptrons. As input, each network takes multiple attribute label vectors $\mathbf{x} \in \mathbb{R}^d$, which were predicted by the attCNN presented in Chapter 4. Regarding the internal structure, both networks consist of two hidden layers, both having an identical amount of neurons. They mainly differ in the number of inputs and the desired output. First, this part of the chapter

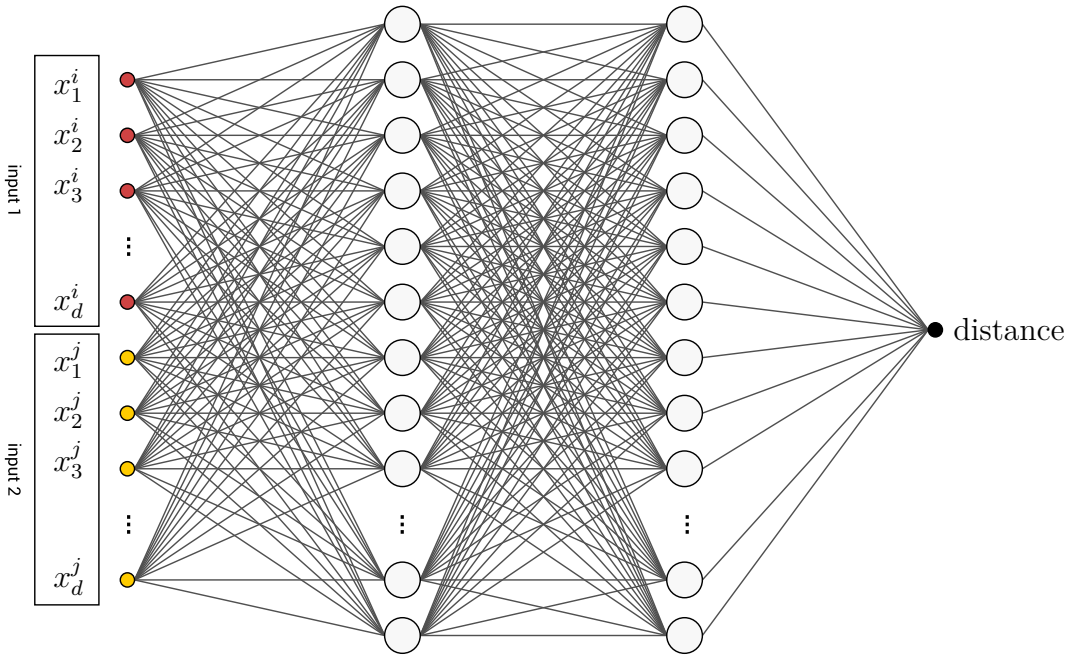



**Figure 5.1:** Distance network architecture

*The network consists of two d-dimensional inputs. The output of the network is a single neuron that emits the final distance value. In addition to the shown neurons, each layer has bias neurons as well.*

takes a look on the distance network. Afterwards, the classification network will be presented in detail.

### 5.1.1 Distance Network

As already mentioned, the distance network approach was developed to solve the re-identification problem by learning a distance function and rank the gallery images base on the distance to the probe image. Similar persons should have a small distance, compared to dissimilar ones with a larger distance. Therefore, the task for this network is to solve a regression problem. In particular, its aim is to learn a function $D : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$. This function assigns a distance score $D(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}$ to two $d$-dimensional inputs $\mathbf{x}_i$ and $\mathbf{x}_j$.

In order to achieve the desired behavior, the network is equipped with two input layers. Each input layer takes the attribute representations $\mathbf{x}_k \in \mathbb{R}^d$ of an image $i$ for person

| | Distance Network | | | Classification Network | | |
|---|---|---|---|---|---|---|
| **layer** | **neurons** | **activation** | **dropout** | **neurons** | **activation** | **dropout** |
| input 0 | $d$ | — | — | $d$ | — | — |
| input 1 | $d$ | — | — | $d$ | — | — |
| input 2 | — | — | — | $d$ | — | — |
| hidden 0 | 875 | ReLU | — | 1000 | ReLU | — |
| hidden 1 | 875 | ReLU | 0.5 | 1000 | ReLU | 0.5 |
| output | 1 | linear | 0.5 | 2 | softmax | 0.5 |

**Table 5.1:** Overview of the network structures
*Both network architectures are summarized in this table. It shows the architectural differences between both networks. Analogous to the network from the first stage, the input layers do not use any dropout regularization. Examining the effect of using dropout for the input layers as well should be covered in future work.*

$k$. Both input layers are fed into the first hidden layer, which is followed directly by the second hidden layer. For simplicity reasons, both hidden layers are identical. This includes the choice of ReLU activation function for each neuron, as well as the number of neurons per layer. In addition to the just mentioned properties, both hidden layers are equipped with a dropout rate of 0.5. This serves, analogous to Chapter 4 as regularization, which aims to avoid overfitting of the network. Analogous to the network from the first stage, the input layers do not use any dropout regularization. Examining the effect of using dropout for the input layers as well should be covered in future work. Since the desired output of the network is a distance score, it is sufficient to use a single neuron with linear activation as output. The information given above is summarized in Table 5.1. In addition to that, Figure 5.1 visualizes the network.

### 5.1.2 Classification Network

The second developed architecture follows a slightly different approach. This time, three instead of two inputs are fed into the network. In opposite to the distance network, the classification network takes three inputs $\mathbf{x}_0$, $\mathbf{x}_1$ and $\mathbf{x}_2 \in \mathbb{R}^d$ and decides

which pair $(\mathbf{x}_0, \mathbf{x}_1)$ or $(\mathbf{x}_0, \mathbf{x}_2)$ is more similar. In order to achieve this, all three inputs are passed to the first hidden layer of the network. Despite the number of neurons in both hidden layers (1,000 and 875 neurons, see experiments in Section 5.3.3 for further details), the layers themselves are identical between both networks regarding activation of single neurons and regularization techniques. Since the network has to decide which pair is more similar, the output layer differs from the output layer of the distance network by the number of neurons. Specifically, this means that it uses two neurons with softmax activation. The first neuron is active, if the pair $(\mathbf{x}_0, \mathbf{x}_1)$ is more similar than the second pair $(\mathbf{x}_0, \mathbf{x}_2)$. Otherwise the second neuron is active. As mentioned earlier in this master's thesis, the softmax output leads to one dominant neuron activation in almost all cases. This suits the desired behavior. Thus, the function learned by the network can be seen as $F(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$ with $F : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}^2$. The information presented above is summarized in Table 5.1. Analogous to the distance network, Figure 5.2 visualizes the architecture of the classification network.

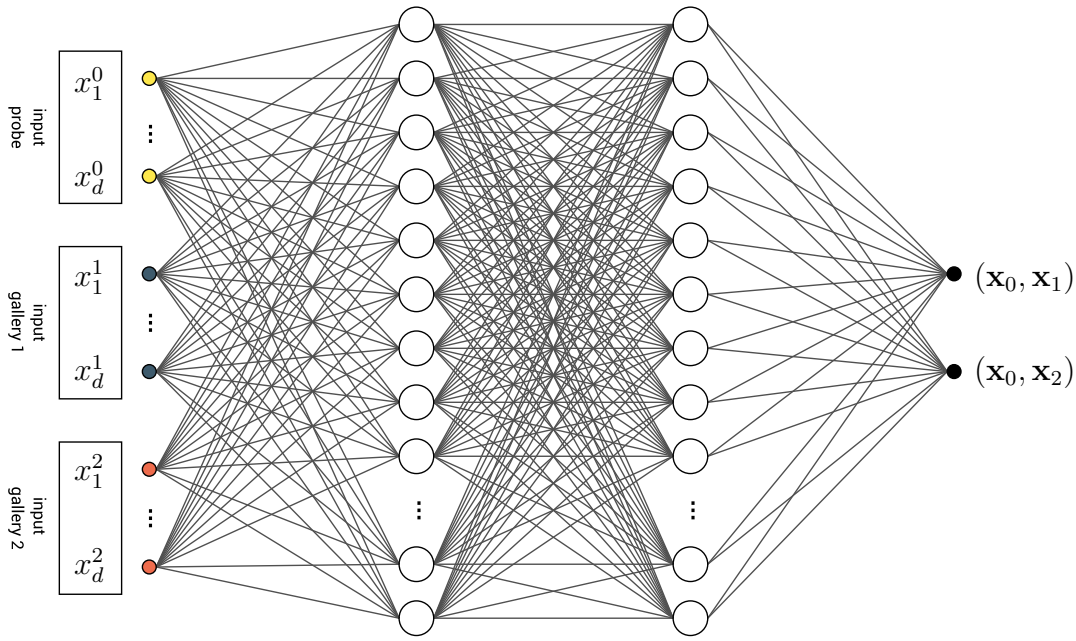



**Figure 5.2:** Classification network architecture

*The network consists of three d-dimensional inputs and two output neurons. Furthermore, two output neurons indicate which pair is more similar. In addition to the shown neurons, each layer has bias neurons as well.*

## 5.2 Training of the Networks

This part of the chapter takes a look on the training of both approaches. This includes the dataset (Section 5.2.1), which is the same for both networks, as well as different data augmentation techniques (Section 5.2.2) used in the scope of this chapter. Furthermore, the slightly different training procedures and the used loss function (Section 5.2.3) are presented here, too. Finally, the chapter takes a look on the used ranking techniques for both networks (Section 5.2.4).

### 5.2.1 Triplet Dataset

As mentioned above, this section introduces the dataset used for the re-identification part of the algorithm. Different to the first part of the algorithm introduced in Chapter 4, the approaches developed here use attribute vectors $\mathbf{x} \in \mathbb{R}^d$ as inputs. These attribute vectors are generated by employing a dataset independent from the PETA dataset used in Chapter 4, which consists of multiple images of different persons. This is necessary, since the Deep Convolutional Neural Network that provides the attribute labels for the training of the distance and classification network has been trained on the PETA dataset. As a consequence, the resulting attribute label vectors would be probably to close to the ground truth labels, thus limiting the the achievable re-identification performance of the two networks. For each of these images $I \in \mathbb{R}^{128\times48\times3}$, the attCNN $M$ was used to predict attribute vectors $\mathbf{x} := M(I) \in \mathbb{R}^d$ . Hence, the dimension of the attribute vectors is $d = 44$. Based on these vectors, so called *triplets* were constructed. Within the scope of this master's thesis, a triplet $t = (\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n)$ is composed of three representations $\mathbf{x}_i^1$, $\mathbf{x}_i^2$, and $\mathbf{x}_j \in \mathbb{R}^d$ of two different individuals $i$, $j$: an anchor vector $\mathbf{x}_a = \mathbf{x}_i^1$ describing a certain person $i$, a second representation of the same person $\mathbf{x}_p = \mathbf{x}_i^2$ and a third one $\mathbf{x}_n = \mathbf{x}_j$ that describes a different person $j$. These three representations, are referred to in the literature as *anchor*, *positive*, and *negative* sample. Figure 5.3 shows the conceptual idea behind the introduced triplets.

As already mentioned, these triplets are used for training both the classification and the distance network. Therefore, it is possible to generate a single dataset instead of

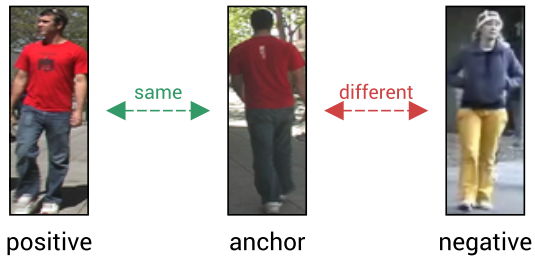



**Figure 5.3:** Conceptual idea of a triplet
*Anchor and positive image show the same person from varying perspectives. The negative image shows a different person. (Images taken from [Deng et al., 2014].)*

a different dataset for each approach. In Chapter 4 the problem was, that the used dataset, namely PETA, was relatively small and not suited very well for training deep neural networks. By constructing triplets as defined above, this problem can be eliminated. Even with relatively small datasets, the number of possible triplets explodes very fast. This will be shown in the following.

### Derivation of the formula for the size of the dataset

Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ be the set of the attribute representations of all available images for $n$ individuals of a certain image dataset. Therefore, $X$ can be split by individuals into $n$ different subsets $X_1, \ldots, X_n \subseteq X$. Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ be the set containing all person-subsets of $X$. $\mathcal{X}$ is a partition of $X$ and therefore the following conditions hold:

1. $\emptyset \notin \mathcal{X}$
2. $X = \bigcup_{X_i \in \mathcal{X}} X_i$
3. if $X_i, X_j \in \mathcal{X}$ and $X_i \neq X_j$ then $X_i \cap X_j = \emptyset$

It is simple to prove that $\mathcal{X}$ is a partition of $X$, since every person has at least one image ($\rightarrow$ Condition 1) and each image is assigned to exactly one individual ($\rightarrow$ Condition 3). Condition 2 is satisfied by definition of $\mathcal{X}$. The fact, that $\mathcal{X}$ is a partition

of $X$ makes it simpler to determine the maximum size of the dataset.
In order to compute the total number of producible triplets, two sets $\hat{X}_i$ and $\bar{X}_i$ are defined for each person $i$ in the following way. The first set $\hat{X}_i$ consists of pairs of attribute vectors that describe the same person. Pairs $(\mathbf{x}_k, \mathbf{x}_k)$ of the same representation are excluded from the dataset.

$$\hat{X}_i = \{ \ (\mathbf{x}_k, \mathbf{x}_l) \in X_i \times X_i \mid \mathbf{x}_k \neq \mathbf{x}_l \ \} \tag{5.1}$$

Obviously, $\hat{X}_i = \emptyset$ if and only if one attribute vector for person $i$ exists. $\bar{X}_i$ on the other hand holds all attribute vectors not describing person $i$.

$$\bar{X}_i = X \setminus X_i \tag{5.2}$$

In order to generate triplets, each possible pair of anchor and positive match from $\hat{X}_i$ can therefore be combined with each image in $\bar{X}_i$ showing a different person. The total number $\tilde{N}$ of possible triplets can be computed as follows.

$$\begin{aligned} \tilde{N} &= \Big| \bigcup_{i \in \{1,\ldots,n\}} \left( \hat{X}_i \times \bar{X}_i \right) \Big| = \sum_{i=1}^{n} |\hat{X}_i| \cdot |\bar{X}_i| \\ &= \sum_{i=1}^{n} \left( |X_i \times X_i| - |X_i| \right) \cdot |X \setminus X_i| \\ &= \sum_{i=1}^{n} |X_i| \cdot (|X_i| - 1) \cdot (|X| - |X_i|) \end{aligned} \tag{5.3}$$

### Market-1501 for triplet dataset creation

Now, with the formula derived in Equation 5.3, it is possible to determine the total number of producible triplets. In order to create triplets, the Market-1501 dataset [Zheng et al., 2015] was employed, which is independent to the PETA dataset used for the first part of this master's thesis. Market-1501 consists of over 30,000 images for 1,501 different individuals. For the construction of the triplet dataset, images that do not show persons labeled with an identifier are disregarded, which reduces the size of the original Market-1501 dataset to 26,051. In addition to that, one randomly collected part of the VIPeR dataset, consisting of one half of the individuals including all their attribute vectors, was included as well. The VIPeR dataset has not been

used for training of the attCNN, therefore it is uncritical to use it. Hence, for the final set $X$ results that $|X| = 26{,}051 + 632 = 26{,}683$. By using the formula obtained in Equation 5.3 follows that the total number of producible triplets is $\tilde{N} = 15{,}224{,}645{,}260$.

Due to the lack of time and space, the size of the training dataset was set to $N = 2^{20} = 10^{20 \cdot \log_{10} 2} \approx 1$ million samples. This training dataset is constructed independently from the training step. Therefore, in a first step, $N$ pairs of different persons $i$ and $j$ are generated randomly. For each pair of individuals, corresponding attribute vectors are generated randomly, two for person $i$ and one for person $j$. These three vectors are then combined to a triplet and added to the dataset. For the training of both networks, $\frac{1}{16}$ of the generated training set is randomly chosen and used for validation and, therefore, excluded from training.
In addition to the training dataset, another small set of triplets for testing the performance of the classifier is constructed as well. The dataset consists only of attribute vectors generated from the second part of the VIPeR dataset. Furthermore, this is done ten times with different random partitions of VIPeR. This is necessary, since [Su et al., 2016] uses this evaluation procedure. The generation of the triplets for the second dataset is done completely deterministic by pairing each individual $i$ with all other individuals $j$. The just described procedure is summarized in Algorithm 3.

### 5.2.2 Data Augmentation

Data Augmentation is an important topic for data driven methods, since its aim is to increase the number of data samples for training. So far, the obtained dataset can be used for training the distance and classification network. However, depending on the network used, different techniques for data augmentation are possible. First, some general techniques that are applicable for both approaches are presented in short: adding random noise and transforming images. In particular, adding random zero-centered normal distributed noise to the attribute vectors already serves as data augmentation. It is very difficult to tell how much this method increases the theoretical size of the

**Algorithm 3** Triplet dataset generation

1: $T \leftarrow \emptyset$
2: $V \leftarrow \emptyset$
3: $\mathcal{X} = \{X_1, \ldots, X_n\} \leftarrow$ load sets of images for all persons $i \in I$ from Market-1501
4: Split $I$ into training set $I_t$ and testing set $I_v$ ▷ $I_t, I_v$ include each one half of VIPeR
5: Split $X$ into training set $X_t$ and testing set $X_v$ ▷ $X_t \cap X_v = \emptyset, X_t \cup X_v = X$
6: Load attCNN model $M : \mathbb{R}^{128\times48\times3} \mapsto \mathbb{R}^d$
7: Randomly sample $N$ identifiers $i_k \in I_t$ with replacement into multiset $A$
8: **for all** $i_k \in A$ **do**
9: Randomly sample identifier $i_l$ from $I_t \setminus \{i_l\}$
10: Randomly sample two images $x^1_{i_k}$ and $x^2_{i_k}$ for $i_k$ and one $x_{i_l}$ for $i_l$ from $X_{i_k}$ and $X_{i_l}$
11: $T \leftarrow T \cup \{(M(x^1_{i_k}), M(x^2_{i_k}), M(x_{i_l}))\}$ ▷ Predict vectors and add triplet to dataset
12: **end for**
13: **for all** $i_k \in I_v$ **do**
14: $V_k \leftarrow i_k \times (I_v \setminus \{i_k\})$ ▷ pair $i_k$ with all other $i_j$ from second half of VIPeR except $i_k$
15: **for all** $(i_u, i_w) \in V_k$ **do**
16: Randomly sample images $x^1_{i_u}$ and $x^2_{i_u} \in X_{i_u}$ and one $x_{i_w} \in X_{i_w}$
17: $V \leftarrow V \cup \{(M(x^1_{i_u}), M(x^2_{i_u}), M(x_{i_w}))\}$
18: **end for**
19: **end for**

underlying dataset. In a continuous world, this leads to an infinite number of possible samples. Since computers work in a discrete and bounded world, this is obviously not the case. In addition to that, it is hard to determine the proper standard deviation $\sigma$ of the normal distribution. Too much noise (high value for $\sigma$), however, will have a negative effect on the data, because information is lost. Therefore, it is necessary to find a suitable amount of noise, which has to be large enough in order to have any effect, but cannot be too large in order to retain the information within the data. Due to the limited time for this master's thesis, adding random noise was not examined further.

Instead of augmenting the dataset by direct manipulation of the attribute vectors, it is

also possible to perform the augmentation on the image dataset. For this, all methods presented in Algorithm 2 can be used. The simplest way would be to horizontally flip each image, which doubles the size of $X$. Let $X^*$ be the resulting theoretical dataset with $|X^*| = 2 \cdot |X|$. $X_i^*$, $\hat{X}_i^*$ and $\bar{X}_i^*$ are defined based on $X^*$ analogous to 5.1 and 5.2. Therefore, the new total number $\tilde{N}^*$ of possible triplets is at least eight times larger as shown in Equation 5.4.

$$\begin{aligned}
\tilde{N}^* &= |\bigcup_{i \in \{1,\dots,n\}} \left(\hat{X}_i^* \times \bar{X}_i^*\right)| = \sum_{i=1}^{n} |\hat{X}_i^*| \cdot |\bar{X}_i^*| \\
&= \sum_{i=1}^{n} |X_i^*| \cdot (|X_i^*| - 1) \cdot (|X^*| - |X_i^*|) \\
&\geq \sum_{i=1}^{n} 2 \cdot |X_i| \cdot (2 \cdot |X_i| - 2) \cdot (2 \cdot |X| - 2 \cdot |X_i|) \\
&= \sum_{i=1}^{n} 2 \cdot |X_i| \cdot 2 \cdot (|X_i| - 1) \cdot 2 \cdot (|X| - |X_i|) \\
&= 8 \cdot \tilde{N}
\end{aligned} \tag{5.4}$$

This way of augmenting the data was not employed, either. The reason is, that it is not possible to perform the augmentation with the here used models at training time (i.e. live), which is the preferred way. Performing the augmentation live, theoretically allows to explore a much larger virtual dataset $\tilde{N}$. If the augmentation is performed beforehand, the training is limited to the generated data for the whole training. In order to benefit from the augmentation, it is necessary to generate more samples before training, which take even more disk space. This, however, could be done as well by creating a larger training dataset without data augmentation, since the theoretical dataset provides enough possible triplets. Therefore, in this case the data augmentation by transforming images is not beneficial to the training.

Beside these methods, that could be used for both networks, for the classification network, however, another additional way of augmenting the data exists. Specifically, the size of the dataset can be increased by simply swapping the second and third element of each triplet, which is shown in Figure 5.4. This swapping of the elements of the triplets is referred to as *inner triplet shuffling*. By doing this, the size of the virtual dataset can be doubled, since for each sample $(\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n)$ the corresponding sample $(\mathbf{x}_a, \mathbf{x}_n, \mathbf{x}_p)$ can be used as well. At each epoch, for each of the available triplets

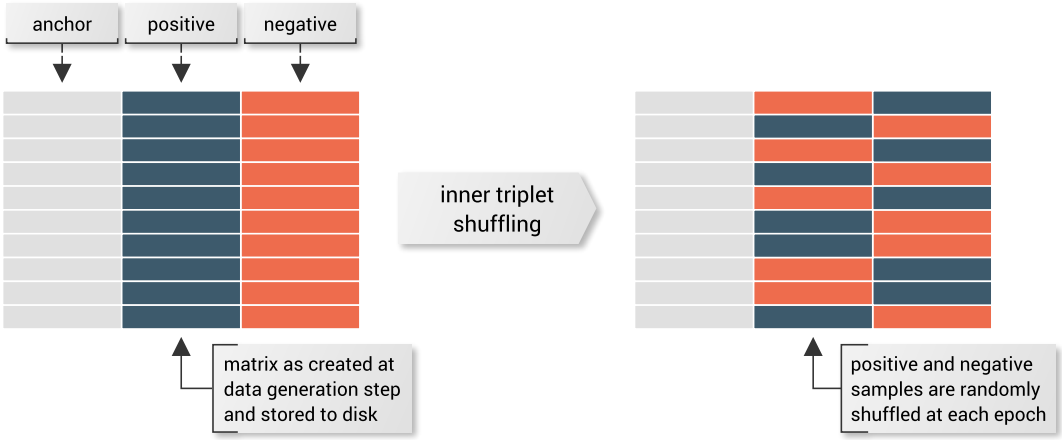



**Figure 5.4:** Data augmentation by inner triplet shuffling
*Each row represents a triplet sample. Initially, samples are sorted as shown in the left part of the figure. The anchor members stay untouched at the first position. For each sample, positive and negative members are switched randomly at each epoch. In addition to that, triplets are shuffled for training as well.*

the algorithm determines with a probability of 0.5, whether it will be flipped or not. Obviously, the ground truth labels have to be swapped then, too. The benefits of this kind of augmentation are that it can be performed live, and it is cheap concerning computation time. Yet, this is not applicable for the distance network due to the numerical nature of the problem and the way the network is trained. This is mainly due to the used loss function, which will be shown in the next section. Performing the shuffling is important, in order to avoid the network to develop a behavior that simply always predicts the first sample as more similar. If the data is not shuffled, this is very likely to happen.

Therefore, it can be summarized, that data augmentation was not used for the training of the distance network. The classification network, however, uses the necessary inner triplet shuffling.

### 5.2.3 Loss Function and Training

Beside the architectural differences, the two approaches differ in their training, too. In this part of the chapter, all relevant information for the understanding of both methods are presented.

#### Distance Network

As already mentioned earlier, the distance network presented in this master's thesis tries to predict a distance value between two attribute vectors. Specifically, the network tries to decrease the distance between attribute vectors that describe the same person, whereas the distance between attribute vectors showing different people will be increased. For this purpose, the so called *Triplet Loss*, as used for example by [Schroff et al., 2015] and [Su et al., 2016], was adapted. The main concept as explained above can be expressed as shown in Equation 5.5, where $f_{\text{dist}} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is an arbitrary distance function.

$$f_{\text{dist}}(\mathbf{x}_a, \mathbf{x}_p) + \alpha \leq f_{\text{dist}}(\mathbf{x}_a, \mathbf{x}_n) \tag{5.5}$$

Instead of simply trying to achieve that the distance between anchor vector $\mathbf{x}_a$ and positive vector $\mathbf{x}_p$ is smaller than the corresponding distance between anchor vector $\mathbf{x}_a$ and negative vector $\mathbf{x}_n$, a margin parameter $\alpha \in \mathbb{R}^+$ is included. This margin controls how big the distance between these pairs should be at least. By subtracting the distance between anchor and negative sample from both sides of the equation, the formula can be brought into the form shown in Equation 5.6.

$$f_{\text{dist}}(\mathbf{x}_a, \mathbf{x}_p) + \alpha - f_{\text{dist}}(\mathbf{x}_a, \mathbf{x}_n) \leq 0 \tag{5.6}$$

This leads directly to the final loss function as shown in Equation 5.7, where the arbitrary distance function $f_{\text{dist}}$ is replaced by the function computed by the distance network $D(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}$.

$$\mathcal{L}_{\text{dist}} = \sum_{i=1}^{n} \max\{\ 0,\ \alpha - D(\mathbf{x}_a, \mathbf{x}_n) + D(\mathbf{x}_a, \mathbf{x}_p)\ \} \tag{5.7}$$

This loss is non-negative and reaches its minimum, if the condition from Equation 5.6 is fulfilled by all triplets in the training dataset. In order to train the distance network with the triplet dataset, a method based on two networks, which share their weights, was developed. Figure 5.5(a) illustrates how the training of the distance network is performed. As explained in Section 5.2.1, the attribute vectors are provided by the attCNN. Each of the distance network instances, gets a pair of attribute vectors and outputs a corresponding scalar. Since both instances are identical due to the shared weights, both outputs can be used for computing the triplet loss.

### Classification Network

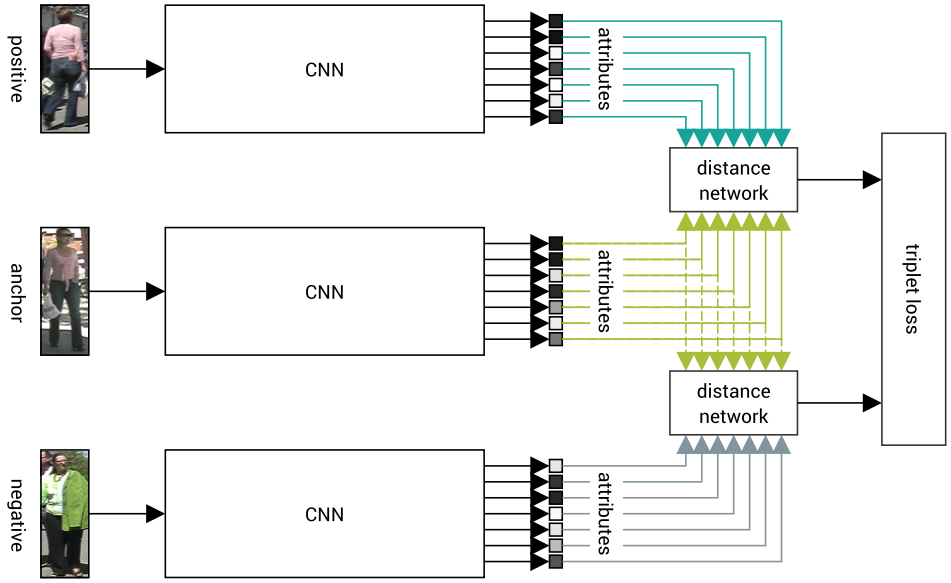
Similar to the attCNN, the classification network uses the cross entropy for training. This time, due to the fact that the network only has to classify which of the two representations $\mathbf{x}_p$ and $\mathbf{x}_n$ introduced earlier is more similar to the anchor $\mathbf{x}_a$, the loss function can be simplified. For the classification network, $F : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}^2$ with $F(\mathbf{x}_a, \mathbf{x}_0, \mathbf{x}_1) = \tilde{\mathbf{y}} = (\tilde{y_0}, \tilde{y}_1)^T \in \mathbb{R}^2$ is the function of the network. In other words, the classification network takes three inputs, where the first one is the anchor element, and decides which of the remaining two elements is more similar to the anchor. This is done by returning a two-dimensional vector, where the elements $y_i$ provide the probability that the corresponding elements $x_i$ is the more similar one. The corresponding ground truth label vector is $\mathbf{y} = (y_0, y_1)^T$, where $y_i = 1$ if $x_i$ is more similar to the anchor and 0 else. Therefore, the loss function can be written as shown in Equation 5.8.

$$\mathcal{L}_{\text{class}} = -\sum_{i=1}^{n} y_0 \ln \tilde{y}_0 + y_1 \ln \tilde{y}_1 \tag{5.8}$$

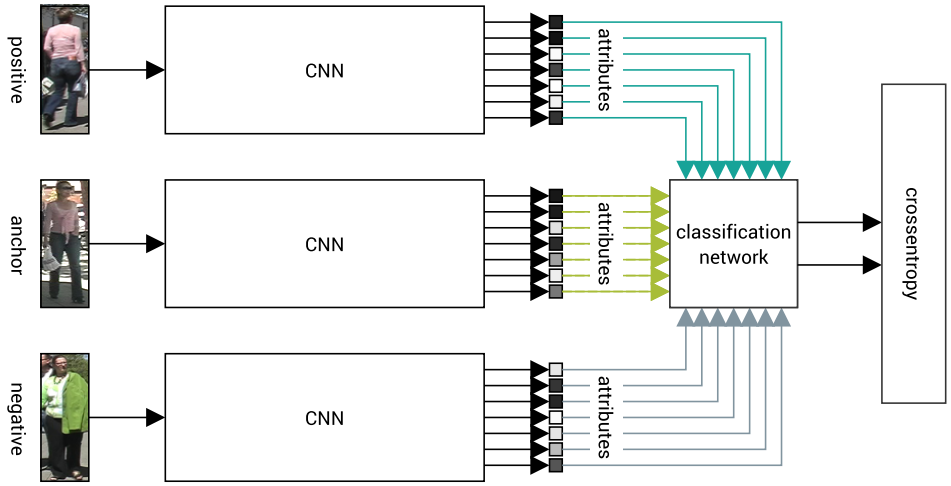
Due to the softmax activation, which ensures that $\tilde{y}_0 + \tilde{y}_1 = 1$, the formula can be rewritten depending on $\tilde{y}_0$ only without loss of generality. This is displayed in Equation 5.9.

$$\mathcal{L}_{\text{class}} = -\sum_{i=1}^{n} y_0 \ln \tilde{y}_0 + (1 - y_0) \ln(1 - \tilde{y}_0) \tag{5.9}$$

Analogous to Figure 5.5(a), Figure 5.5(b) illustrates the training of the classifier. Again, the attribute vectors are provided by the attCNN. This time, there is no need




(a) Distance network approach




(b) Classification network approach

**Figure 5.5:** Triplet training alternatives

*Two different triplet-based approaches were developed for re-identification. Both have in common that they take predicted attribute vectors as input. 5.5(a) computes distance values between two attribute vectors, whereas 5.5(b) decides which of two presented descriptions is more similar to the anchor.*

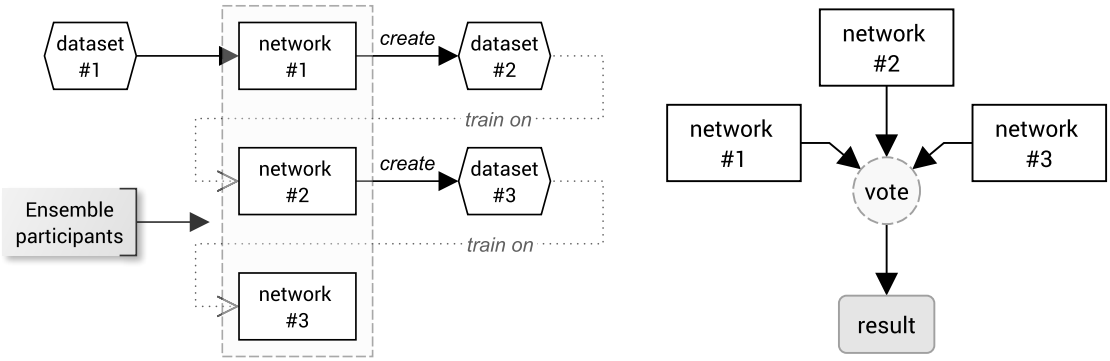
**Algorithm 4** Ensemble Training

1: $S^1_{train} \leftarrow \{ (\mathbf{x}^1_a, \mathbf{x}^1_p, \mathbf{x}^1_n), \ldots, (\mathbf{x}^m_a, \mathbf{x}^m_p, \mathbf{x}^m_n) \}$ ▷ Generate triplets (Algorithm 3)
2: $F_1, F_2, F_3$ initialize classification networks
3: Train network $F_1$ on $S^1_{train}$
4: **for all** $i \in \{2, 3\}$ **do**
5: Create $\tilde{S}^i_{train}$ with $|\tilde{S}^i_{train}| = 4 \cdot |S^{i-1}_{train}|$
6: $S^i_{train} \leftarrow \{ (\mathbf{x}^k_a, \mathbf{x}^k_p, \mathbf{x}^k_n) \in \tilde{S}^i_{train} \mid (\mathbf{x}^k_a, \mathbf{x}^k_p, \mathbf{x}^k_n)$ misclassified by $F_{i-1} \}$
7: Train network $F_i$ on $S^i_{train}$
8: **end for**

to use two instances of the network, since all three inputs are used directly by the classification network.

### Classification Ensemble

A simple way to build a classifier, which improves the performance, is to use an ensemble of networks. One possible way to build an ensemble is to use multiple instances of the same network. Here, the decision was made to use three networks $F_1$, $F_2$, and $F_3$. These network have to be trained independently. For this, $F_1$ is first trained on the dataset $S^1_{train}$ as introduced in Section 5.2.1. Afterwards, a second and four times bigger dataset $\tilde{S}^2_{train}$ is generated. This dataset is not used for training. Instead, it is employed to build another dataset $S^2_{train}$. This one consists of triplet samples from $\tilde{S}^2_{train}$ that were misclassified by the trained network $F_1$. On the obtained dataset, another randomly initialized copy of the classification network, $F_2$ is trained on. The just described procedure is then repeated again in order to obtain a third network $F_3$. It was also considered to use both networks, $F_1$ and the $F_2$, to generate the dataset for $F_3$. Therefore, this dataset would consist of samples that are difficult for both networks. As it turned out, after filtering, far too few samples remained for training another instance of the classification network. Hence, an even larger dataset would be needed to increase the total number of triplets. Generating such a dataset,




(a) Training phase (b) Application phase

**Figure 5.6:** Ensemble training and application phase
*In the training phase, the classification network is trained on the initial dataset. Afterwards, the network is used to create another dataset, which then will be used for training a second network. This is done a third time. The three obtained networks are combined to an ensemble afterwards.*

was not possible, since the time for this master's thesis was limited. Therefore, this modification of the training procedure was disregarded.
Figure 5.6(a) illustrates the procedure explained above, which is also summarized in Algorithm 4. These three networks $F_1, F_2$, and $F_3$ are then composed to an ensemble, which decides what element of the triplet is more similar to the anchor part by a majority vote. The concept is illustrated in Figure 5.6(b).

### 5.2.4 Ranking of Gallery Samples

So far, the distance and the classification network were presented. The last missing detail, which is necessary to perform re-identification, is the ranking of the gallery samples. Typically, for a given probe $\mathbf{x}_i$ and a gallery of multiple samples $\mathbf{x}_j$, the task is to order the gallery by similarity of the samples in descending order. As mentioned earlier, due to the different aims of both networks, the ranking has to be performed differently for each approach. The ranking techniques for the distance and classification network are presented separately for each network in the following.

### Distance Network

Ranking a gallery by employing the distance network is fairly simple. Due to the construction and training of the network, the distance network is able to provide scores for pairs of attribute vectors. Therefore, it is sufficient to determine the scores for all pairs $P = \{ (\mathbf{x}_i, \mathbf{x}_j) \mid \mathbf{x}_i \text{ probe vector}, \mathbf{x}_j \text{gallery vector} \}$. Afterwards, the elements $\mathbf{x}_j$ can be arranged in ascending order by the determined distance scores predicted by the distance network.

### Classification Network

The classification network decides which attribute vector of two individuals $j$ and $k$ from the gallery set is more similar to the attribute vector of the third individual $i$ from the probe set. Therefore, the gallery set has to be ranked by these pairwise comparisons, which makes the sorting of the gallery samples for each individual $\mathbf{x}_j$ more complex. For the classification network, two different ranking approaches were evaluated: an adapted sorting algorithm and a winning-based ranking. Both need the comparison matrix $\mathbf{M}_i$, which is defined for each probe $i$ as shown in Equation 5.10.

$$\mathbf{M}_i = \begin{pmatrix} 0 & m^i_{1,2} & \cdots & m^i_{1,n-1} & m^i_{1,n} \\ m^i_{2,1} & 0 & \cdots & m^i_{2,n-1} & m^i_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ m^i_{n-1,1} & m^i_{n-1,2} & \cdots & 0 & m^i_{n-1,n} \\ m^i_{n,1} & m^i_{n,2} & \cdots & m^i_{n,n-1} & 0 \end{pmatrix} \tag{5.10}$$

with $m^i_{j,k}$ defined as follows:

$$m^i_{j,k} = \begin{cases} 1 & \text{if } j \text{ is more similar to } i \text{ than } k \\ -1 & \text{if } k \text{ is more similar to } i \text{ than } j \\ 0 & \text{if both } j \text{ and } k \text{ are equally similar to } i \end{cases} \tag{5.11}$$

In the case of a perfect classification of all triplets, $\mathbf{M}_i$ is a skew-symmetric matrix, meaning $m^i_{k,j} = -m^i_{j,k}\ \forall i, j \in \{1, \ldots, n\}$ or simply $\mathbf{M}_i = -\mathbf{M}_i^T$.
For the first ranking method, the BubbleSort algorithm was adapted, with the slight

**Algorithm 5** BubbleSortRanking

```
Generate M as described in Equation 5.11
n ← Number of gallery samples                    ▷ number of gallery elements
Generate initial order of elements (e.g. by ascending identifier or better, random order)
for all i from n-1 to 1 do
    for all j from 1 to i do
        if m_{j,j+1} == -1 then
            swap(j,j+1)
        end if
    end for
end for
```

change that the comparison of two elements $j$ and $k$ is performed by looking at the matrix $\mathbf{M}_i$. First, the elements from the gallery set are put into a list on which the algorithm is performed afterwards. If the corresponding entry in $\mathbf{M}_i$ equals -1, then the elements are swapped. Algorithm 5 shows this procedure. With a certain probability, the classification network makes wrong decisions. These can disturb the sorting performed by Algorithm 5. Thus, this method has the major drawback, that it is not independent from the initial order of the list, which can lead to a highly ranked individual which has only "won" the comparison to some weak competitors.
Therefore, the second approach was designed to be more robust. This winning-based method counts the number of wins for all individuals. For the individual $j$, therefore, the number of 1's in the $j^{th}$ row, as well as the number of -1's in the $j^{th}$ column are counted. This is done for all individuals in the gallery set. Based on these numbers, the elements are arranged in descending order.

## 5.3 Experiments

With all the knowledge about the architecture and training of the networks, the chapter continues with the presentation of the performed experiments. Section 5.3.1 gives a short overview of the datasets used for validation and testing. Analogous to Chapter 4,

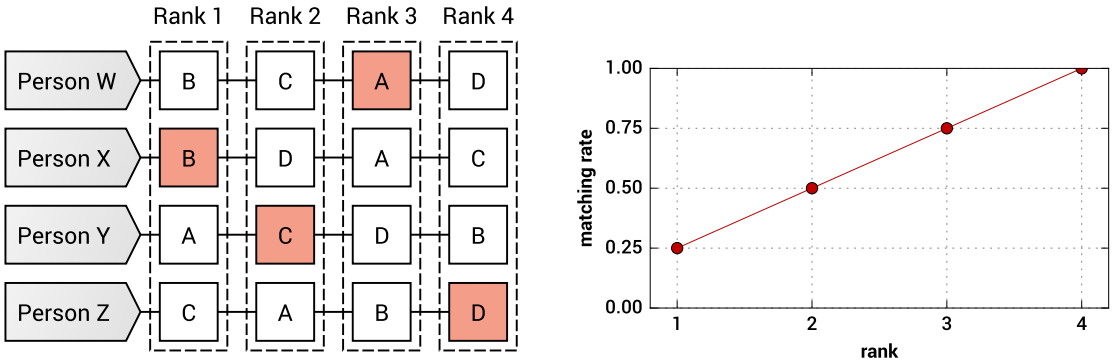
the employed metrics for the evaluation of the performance of the single networks are presented in Section 5.3.2. These experiments show the performance of both approaches developed and compare them with each other. Furthermore, the networks are compared to the SSDAL algorithm proposed by [Su et al., 2016].

### 5.3.1 Datasets for Validation

Similar to the first part of this master's thesis, the VIPeR dataset is used for testing in all experiments. Since VIPeR was excluded from training of the attCNN, it is uncritical to generate attribute vectors for the images of this dataset. The resulting attribute vectors can be used for training and testing of the distance and classification network. For evaluating the algorithms, the protocol for the VIPeR dataset is to split the dataset into half: one half can be used for training, the other half is reserved for testing. This is repeated ten times with different, randomly determined splits. Afterwards, the resulting performance metrics are averaged over these ten results.

### 5.3.2 Evaluation Metrics

In order to compare different experiments, evaluation metrics are necessary to quantify the performance of the tested networks. Therefore, depending on the performed experiments, different evaluation metrics were employed. These are: multiple rank-$k$ scores from the Cumulative Matching Characteristic (CMC) curve for the distance network, and $F_1$ score introduced in Section 4.3.2 for the classification network. In order to compare the networks regarding their re-identification performance, the cumulative match score is employed. The CMC curve describes the matching rate at specific ranks over all performed rankings. At rank $k$, the value describes the number of matches within the first $k$ ranks. For a better understanding, Figure 5.7 gives a simplified example for the computation of the CMC matching rates at different ranks. In addition to the CMC curve the Synthetic Recognition Rate (SRR) curve is employed for performance comparisons between the different networks. This metric shows, depending on the number of considered individuals $k$, the (synthetic) rank-1 score and can be interpreted as the probability of finding the corresponding gallery element at




(a) Rankings

(b) CMC curve

**Figure 5.7:** Example for computation of matching rates for different ranks
*Four different persons W, X, Y, and Z are compared to a gallery consisting of four individuals A, B, C, and D, resulting in a ordered list for each anchor. For each anchor, the correct match can be found at a specific position in the ranking. These positions of the correct matches are highlighted. From these matches, the corresponding matching rates can be determined. One out of four persons matched at rank 1 (25%), two out of four rankings matched within the first two ranks (50%) and so on. The corresponding CMC curve can be found in Figure 5.7(b).*

the first rank, when investigating $k$ targets. The score can be derived directly from the CMC metric as displayed in Equation 5.12, where $n$ is the number of persons in the probe gallery.

$$\mathrm{SRR}(k) = \mathrm{CMC}(\tfrac{n}{k}) \tag{5.12}$$

### 5.3.3 Performance Comparison

In order to achieve a good re-identification performance, it is necessary to investigate the influence of different architectures and other parameters for both developed networks. Therefore, different experiments need to be performed. They investigate the effects of certain changes and therefore help to choose the best network configuration and training parameters. First, the experiments for the distance network are presented. Afterwards, the same is done for the classification network and the presented ranking techniques. The obtained best configurations for the distance and classifica-

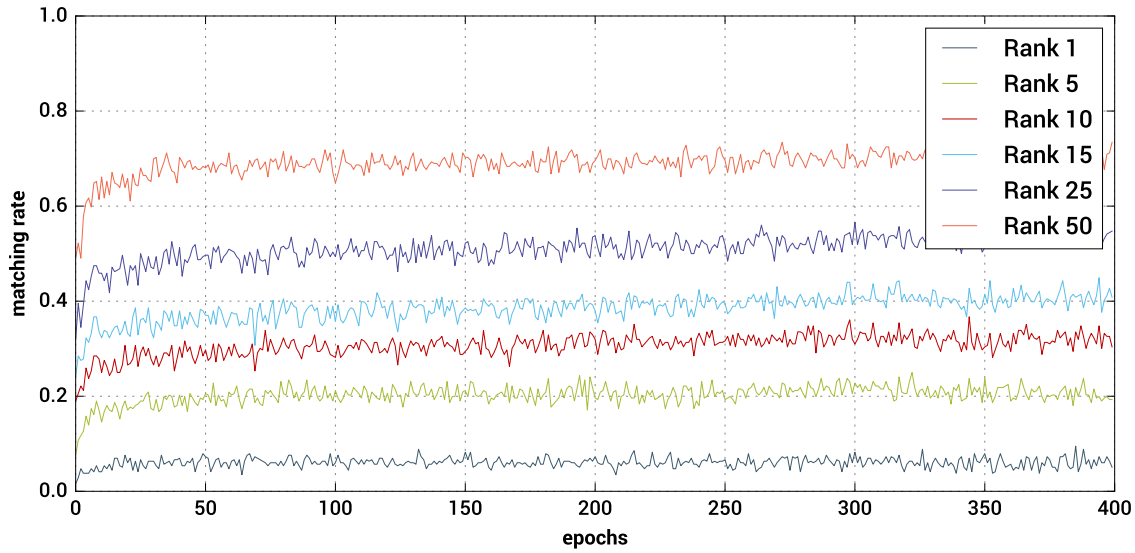



**Figure 5.8:** Trends of the rank scores of a certain distance network architecture
*The figure shows the progress of the selected rank matching rates over training epochs. As one can see, the scores stop to improve after several epochs. For details about the used network configuration see Table A.6 in Appendix A.*

tion network are then compared to each other in the third part of this section. In addition, the networks are also compared to the SSDAL [Su et al., 2016] algorithm, which currently achieves the best attribute-based person re-identification performance of a Deep Learning approach. Due to limited time, only some aspects of the developed networks could be investigated. As the distance network showed a bad performance from the beginning, more time was invested on the experiments for the classification network. Nevertheless, all obtained results for the distance network are presented here and compared to the classification network.

### Distance Network

As mentioned in Section 5.3.2, the distance network was evaluated by comparing the matching rates at different ranks in order to compare different variations of the network configuration. Therefore, a set of ranks was selected in order to investigate the progress of the matching rates. The chosen ranks are: rank 1, 5, 10, 15, 25 and 50. These are

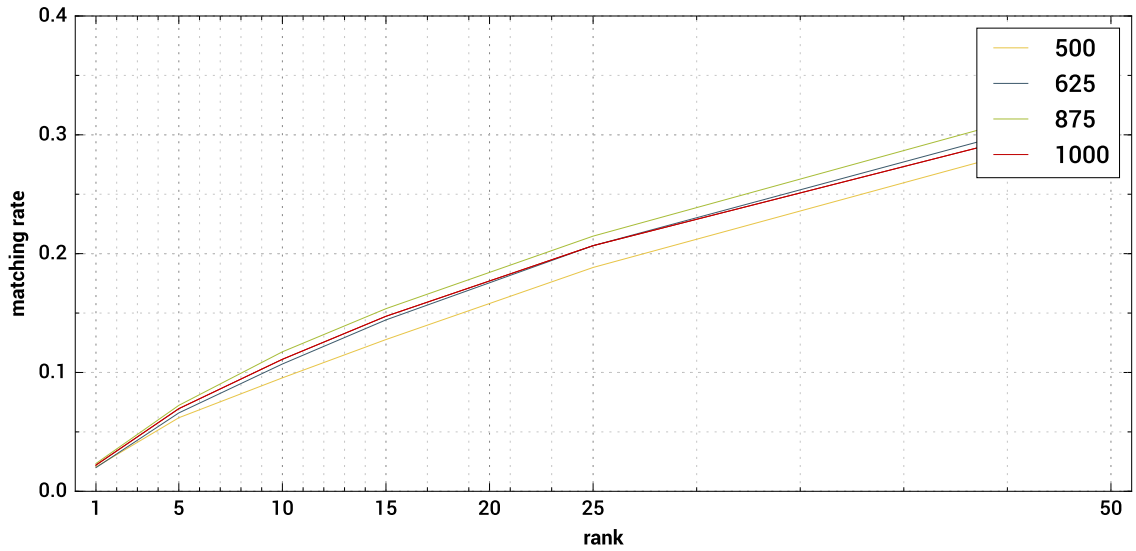



**Figure 5.9:** Rank scores of the validation dataset for certain distance network architectures

*The experiment examines the effect of different amounts of neurons per hidden layer on the matching rates. Since the original curves are very noise but stationary in some way, it was decided to evaluate the rank scores at each rank averaged over epochs. It is obvious that the changes do not have much impact on the matching rates. Nevertheless, the smallest network seems to lack complexity, since it stays behind all other architectures. The best performance, however, is achieved by a middle-sized architecture. Therefore the network with 875 hidden neurons was chosen as the final architecture. For details see Table A.6 in Appendix A.*

used in order to obtain a simplified CMC curve. As Figure 5.8 illustrates, the single rank-$k$ curves are very noisy, without a significant improvement over training epochs. Most of the performance improvement occurs in the first few epochs, before almost all scores seem to reach a plateau. Higher ranks compared to lower ones, however, show improvements over a longer time but stop to improve as well after few hundred epochs. This is the case for all conducted trainings. Independently from the chosen architectures or learning parameters, the network reaches the plateau within a few epochs. This makes it difficult to find an appropriate architecture for the distance network based on the rank performances, since almost all training results seem to be nearly

identical. Figure 5.9 shows the investigated architectures. The best performance was achieved by the network with 875 neurons per hidden layer. Reducing or increasing the size of the network, leads to an declining performance on all ranks. Especially, the smallest network shows the worst performance, but these are very marginal differences. Nevertheless, the architectures can be compared and the best performing one could be identified. Thus, for the final architecture and for later experiments the number of hidden neurons was set to 875. The corresponding CMC curve is shown in Figure 5.10. For further information about the training parameters, see Table A.6 in Appendix A. In addition to the experiment that investigated the network size, it could be examined how the learning rate or different neuron activations influence the performance of the network. In particular, different activations for the output neuron could be examined,

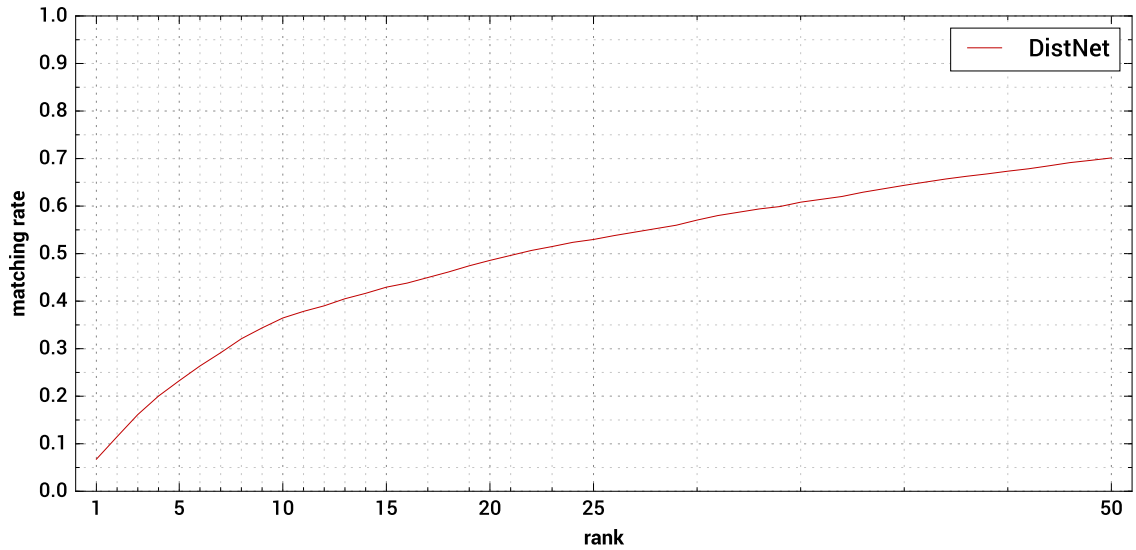


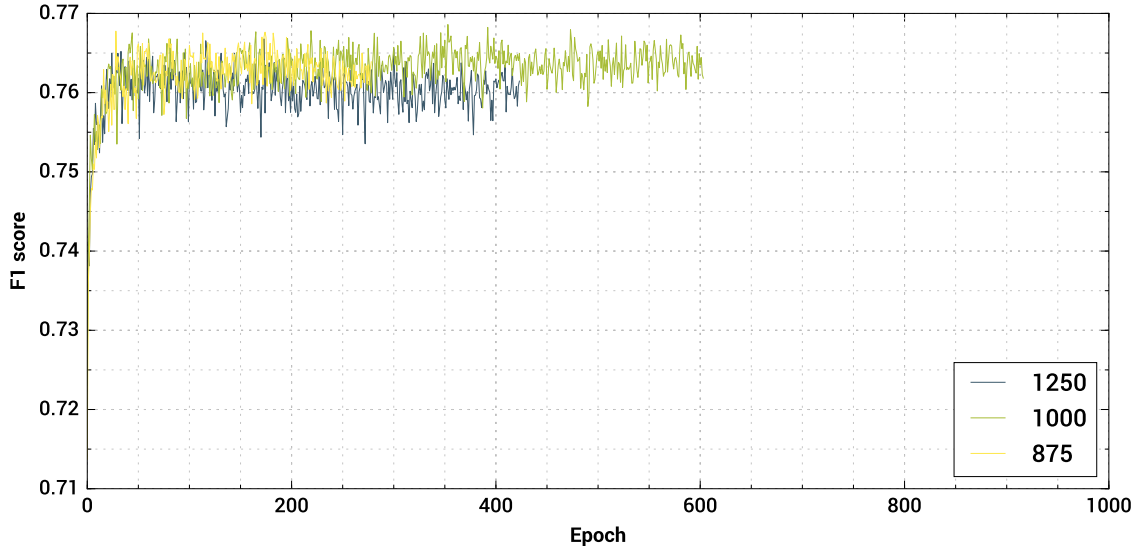

**Figure 5.10:** CMC curve for the distance network

*The re-identification performance for the distance network with 875 neurons per hidden layer. For further information about the training parameters, see Table A.7 in Appendix A.*

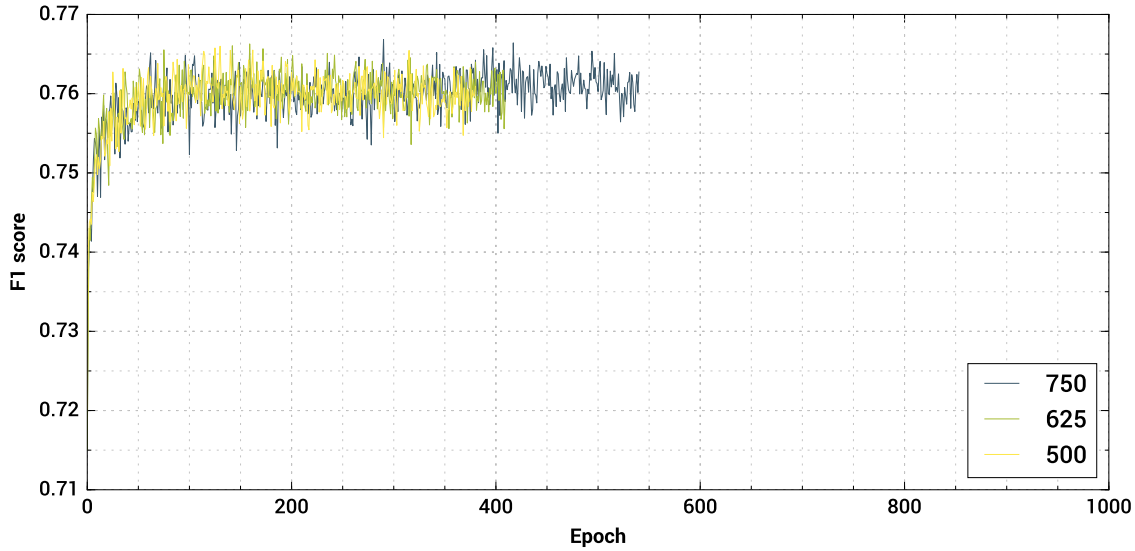
like e.g. sigmoid or ReLU activation. Additionally, the learning rate was not evaluated, but set to 0.05 with a momentum of 0.7. The influence of this parameters and further configurations of the network could be examined in future work. The following experiments focus on the classification network and the comparisons to the SSDAL algorithm.

### Classification Network

In order to determine the best architecture and parameters for the classification network, different aspects that influence the performance were examined. The first experiment, therefore, examines the influence of the network size, which is mainly affected by the number of neurons in the hidden layers, has on the classification performance. All trainings performed have in common, that they use the same number of neurons for both hidden layers, since most of the deep learning based methods that include multilayer perceptrons follow this procedure. The experiment investigated networks from 500 to 1,250 neurons per hidden layer. Starting with 500 neurons per layer, the size was increased by 125 neurons multiple times in order to examine a sufficient number of architectures. As Figure 5.11(b) shows, the smaller networks achieve an almost identical performance compared to all others. At some points, the curve of the network with 750 neurons per layer, achieves a slightly higher $F_1$ score and hence a better performance. Increasing the size of the network even more improves the performance, as shown in Figure 5.11(a). Choosing a network with 875 or 1,000 neurons respectively achieves the best performance compared to the other investigated architectures. The performance measured by the $F_1$ score starts to drop with even larger networks. The reason for the dropping performance with larger networks could be that the network are to complex for the problem, and therefore have less generalization capability compared to the networks with 875 and 1,000 neurons respectively. Altogether, these are slight differences in the performance and no significant changes in the $F_1$ score were encountered. The best performing architecture, which consists of 1,000 neurons per hidden layer, was chosen for the final experiments. The influence of training parameters as learning rate and momentum, or other optimizers was not examined, since this would go beyond the scope of this master's thesis. The parameters used in this experiment are presented in Table A.8 in Appendix A. It provides further information about the network, the performed training and the used dataset.
The ranking methods introduced in Section 5.2.4, are examined in the next two experiments shown in Figure 5.12 and 5.13. In the first experiment, for both methods, the same configuration of the classification network was applied. The only difference




(a) Larger sized models




(b) Smaller sized models

**Figure 5.11:** Impact of different numbers of hidden neurons per layer

*The curves show the performance measured by the $F_1$ scores over the training epochs for different architectures. Higher values are better. The legend shows the amount of neurons per hidden layer. The experiment shows, that the chosen number of neurons per hidden layer has only a small effect on the performance. Since the network with 1,000 neurons per hidden layer achieves the best performance, this one was chosen for all following experiments. For details, see Table A.8 in Appendix A.*

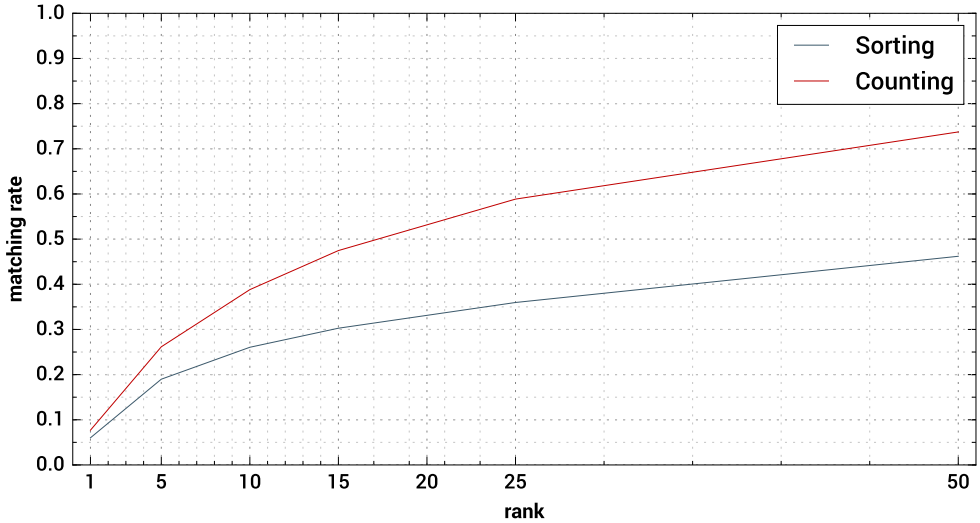



**Figure 5.12:** Comparison of both presented ranking techniques using the same network configuration

*The figure shows the CMC curves (evaluated at ranks 1, 5, 10, 15, 25 and 50) for the classification network using the two ranking techniques. Both presented methods are evaluated with the same configuration of the classification network. A higher matching rate indicates a better re-identification performance. It is obvious that the counting method performs much better at all ranks. This result highlights that the ranking by simply counting the number of wins is the method of choice. For further details on training parameters see Table A.9 in Appendix A.*

was the way of ranking the gallery for each probe. Analogous to the distance network, the matching rate was evaluated for the ranks 1, 5, 10, 15, 25 and 50. The results are shown in Figure 5.12. Apparently, the counting wins method achieves far better results than the sorting-based method. The reason for this is, that counting the number of times a certain individual wins against its "competitors", is more robust than sorting the elements, since the obtained score values are of a more general nature. Sorting algorithms, however, rank by few comparisons with certain other elements. Therefore, for a given probe $\mathbf{x}_a$, the element at rank 1 does not necessary has to be the most similar one. For example due to a decent initial ordering, the probe element might have been pushed forward, although far more similar elements can be found on later

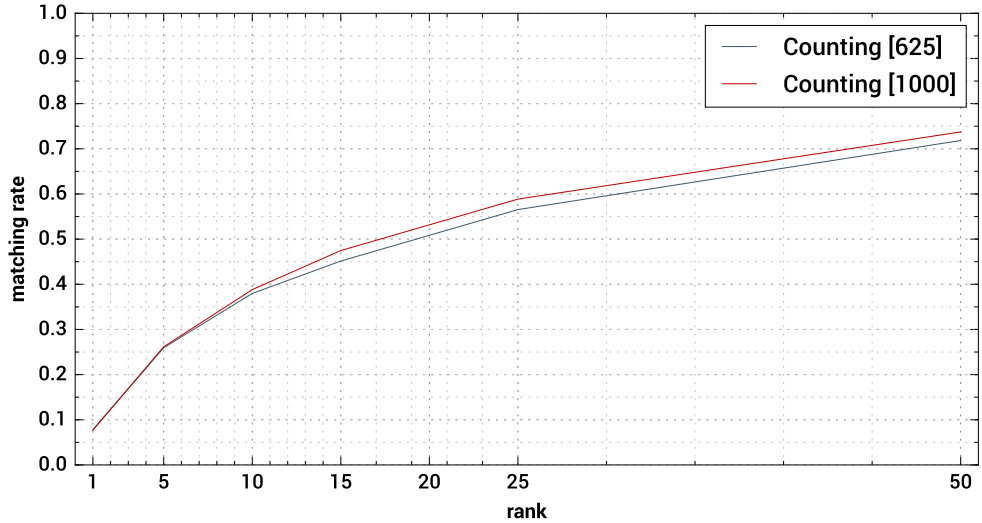



**Figure 5.13:** Ranking by counting wins on different architectures
*This experiment examines how the counting of wins performs with two different networks. It confirms the choice of architecture. Similar to the classification performance, the larger network shows a slightly better performance, than the smaller one. This is understandable, since an improvement in the classification performance implies more accurate comparisons results between different individuals and therefore better rankings.*

ranks. This problem might be mitigated, by ranking multiple times. First, the whole list is ranked. In a next step, the elements within the first and second half are ranked another time. This is repeated multiple times. Future work could examine, if this procedure improves the performance. In addition to the experiment for finding the best architecture for the classification network, Figure 5.13 highlights the findings of the experiment presented in Figure 5.11.

### Comparison with state-of-the-art methods

As mentioned in the beginning of the section, this part focuses on the re-identification performance of the distance network, the classification network and the SSDAL algorithm. Therefore, different experiments were conducted to investigate the performance of each approach. All experiments, presented in this part are evaluated on the VIPeR

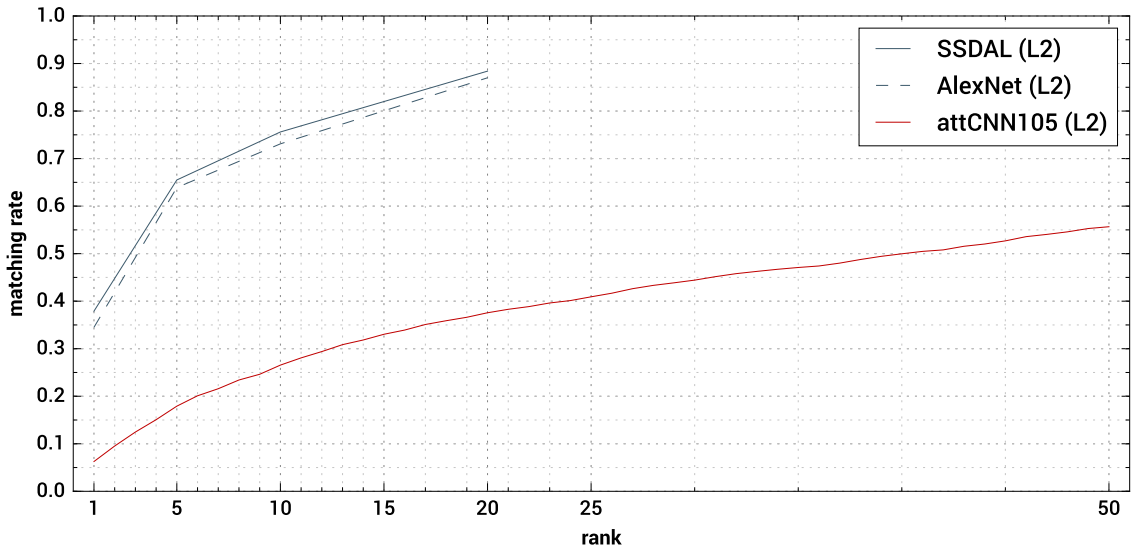



**Figure 5.14:** Comparison of SSDAL and attCNN
*This figure shows the CMC curves for the SSDAL algorithm (AlexNet with finetuning), the AlexNet itself (values for both taken from [Su et al., 2016]) and the attCNN (trained on 105 attributes), which was proposed in the first part of this master's thesis. Apparently, the SSDAL algorithm performs much better than the attCNN.*

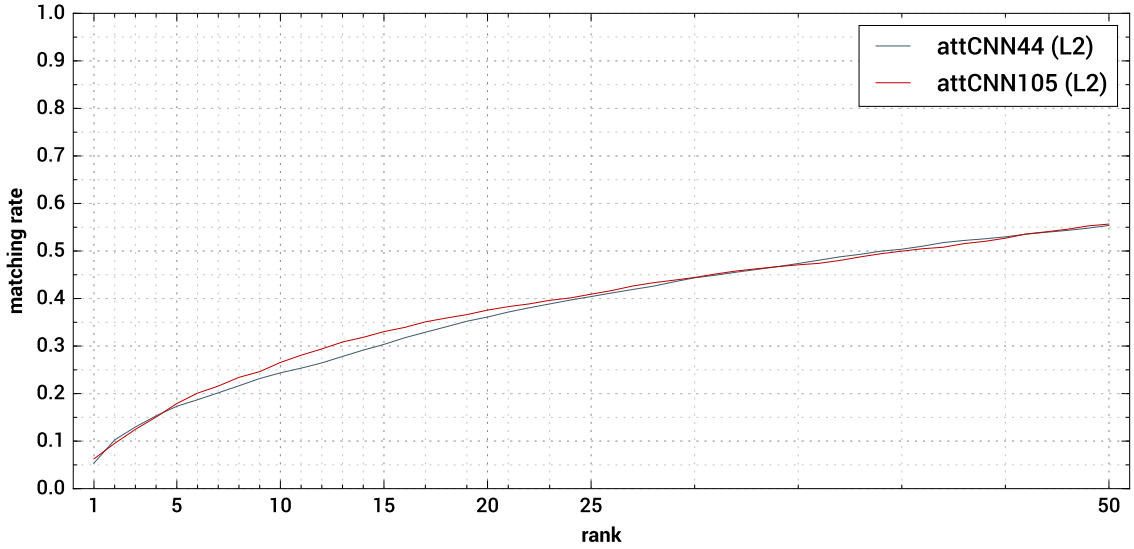
dataset.

The first experiment, shown in Figure 5.14 examines how SSDAL and attCNN perform, when ranking the gallery set by using the L2 distance on the predicted attribute vectors. This comparison is conducted, since the SSDAL algorithm uses exactly this metric as a distance measure for ranking. Obviously, SSDAL outperforms the attCNN developed in this master's thesis by a large margin. Even the first stage of the SSDAL algorithm, which is simply an adapted AlexNet, performs better than the attCNN.

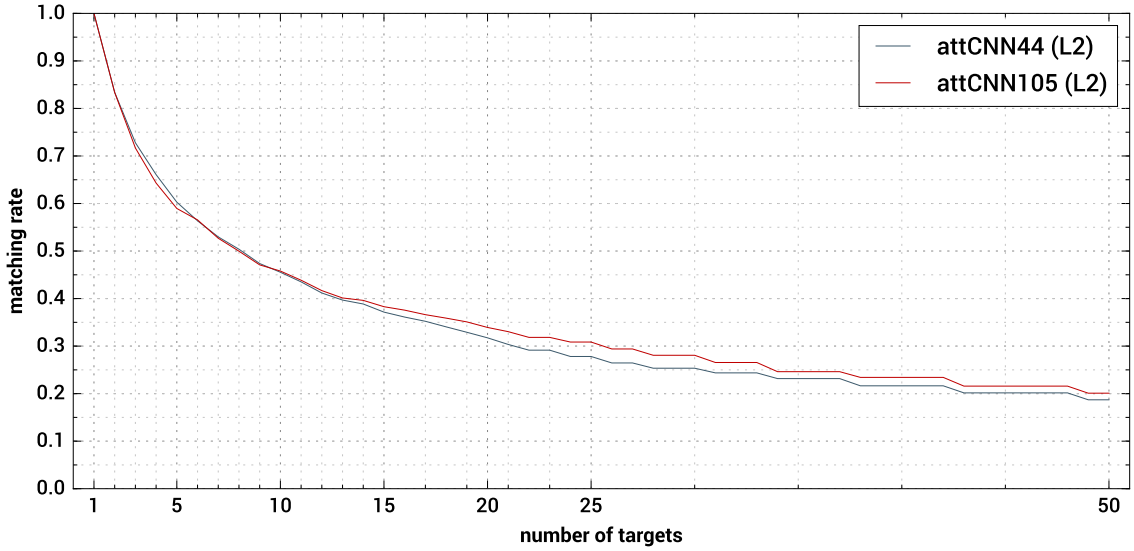
It is difficult to tell, what the exact reason for this significant better performance of the SSDAL algorithm is. One reason could be, that the used pretrained AlexNet is much larger and was initially trained on a huge and diverse dataset. This might lead to more diverse filters, which include filters that are better suited for classifying different attributes. The attCNN, however, was trained from scratch on a relatively small dataset. Therefore, the attCNN might have learned different filters that are useful for predicting semantic attributes, whereas some filters might not have been able to learn,

since to few examples were provided by the dataset. Although, the attCNN achieves a decent performance on attribute prediction, this might not be enough for competing with the AlexNet approach. Furthermore, the SSDAL algorithm uses different fine tunings in order to improve the attribute classification performance of the AlexNet. This could all be reasons for the better performance of the SSDAL algorithm. For a better comparison, the scores for some ranks provided by [Su et al., 2016] are listed in Table A.11 in Appendix A. These correspond to the curve for the SSDAL algorithm in Figure 5.14. All this should be addressed in future work.

Since the decision to restrict the training of the network on attributes from a subset of all attributes was made in Chapter 4, the corresponding network is examined in the next experiment. The aim of this experiment is to examine, whether the better performance in classifying attributes is also beneficial for the re-identification performance. Figure 5.15 shows the results of the experiment. Surprisingly, the network that uses all 105 attributes performs slightly better. Of course, as already mentioned in Chapter 4, rare attributes are well suited for re-identifying persons, however, they are difficult to learn. Although, the overall attribute classification performance of the attCNN44 is superior to the one of the attCNN105, the additional attributes seem to be beneficial for the re-identification. However, it seems that the better decision would have been to retain all attributes in order to achieve a better performance at re-identifying persons, even if it does not have a significant effect on the performance. At this point, it would be interesting to see, if using the pretrained weights, that were discarded in Chapter 4 would increase the re-identification performance even more. Unfortunately, for this experiment was not enough time left. In addition to the experiment in Figure 5.15, it was also examined if the re-identification performance can be improved by the developed triplet-based networks. For a first experiment the classification network was chosen in order to compare itself to the L2-based re-identification. Figure 5.16 shows that using the classification network leads to a significant improvement. As mentioned in Section 5.1.2, an attempt to improve the performance even more was to use an ensemble for classifying triplets. However, Figure 5.18 shows that this is not the case. Surprisingly, the achieved performance is even worse than working with a single network. The explanation for this can be found in Figure 5.19. Only two of




(a) Cumulative Match Characteristic curve




(b) Synthetic Recognition Rate curve

**Figure 5.15:** Comparison of the attCNN trained on 44 and 105 attributes
*Both curves show, that the two networks perform almost equal. Nevertheless, the network with all 105 attributes shows a slightly better performance at some points. Especially, in places where many people are detected, the attCNN105 is likely to perform better. As a consequence, it seems that the decision to reduce the number of labels was not beneficial for the re-identification performance.*

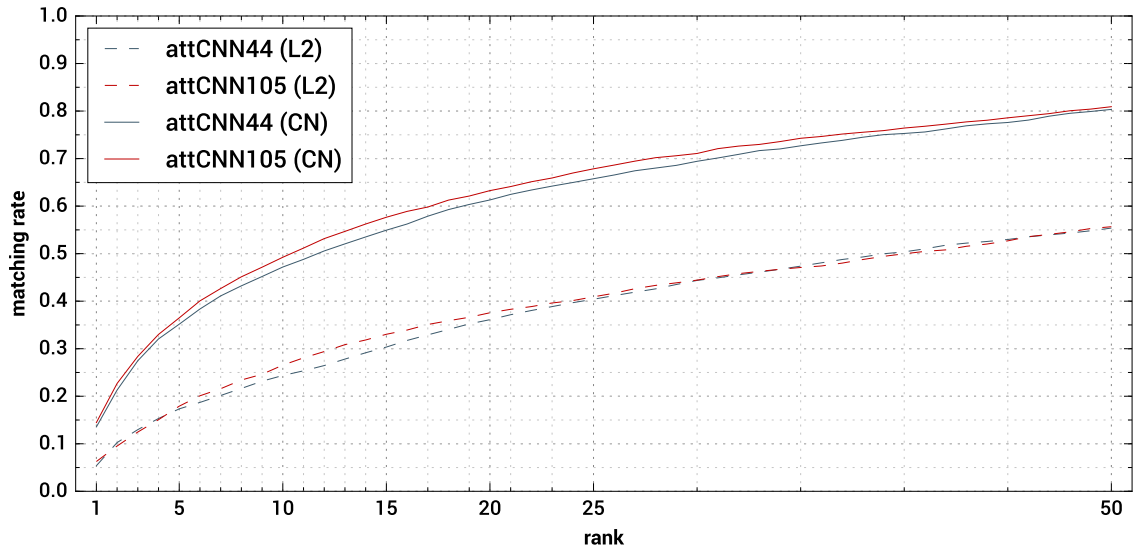
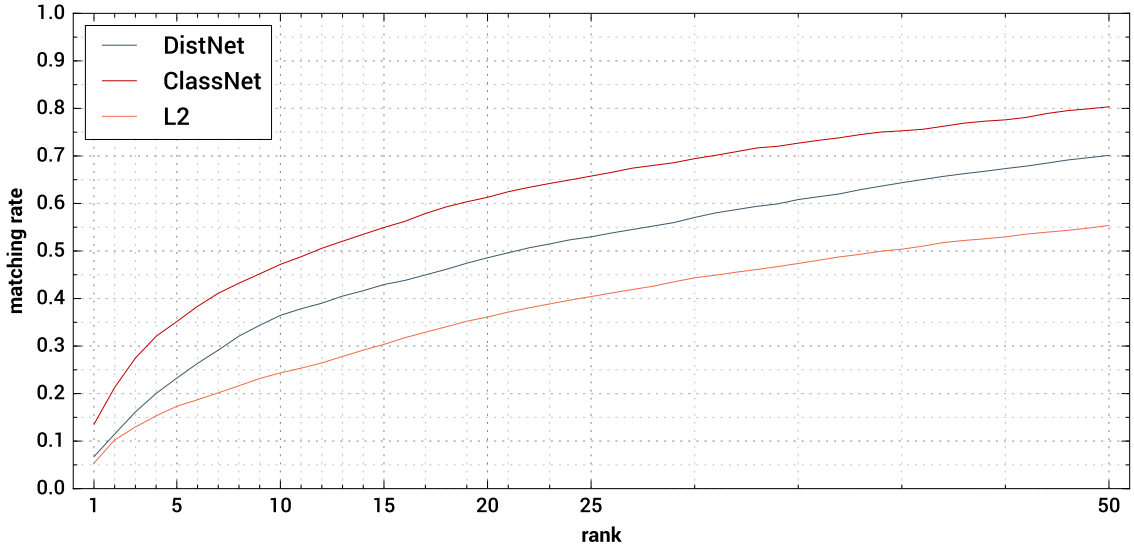


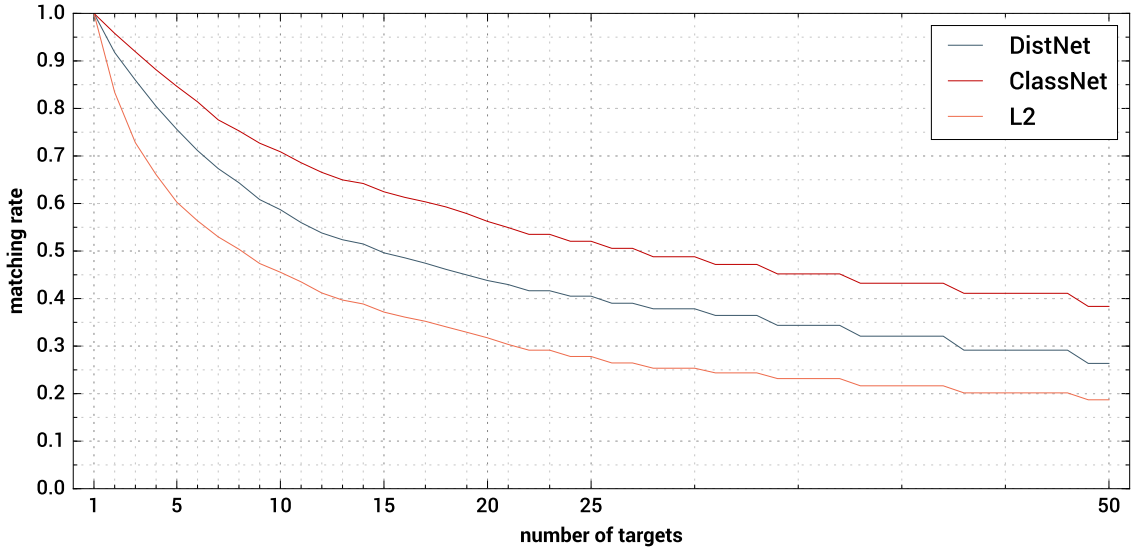

**Figure 5.16:** Comparison of the attCNN with L2-based ranking and classification network (CN) ranking

*The curves show, that the L2-based re-identification is inferior to the classification network with win-based ranking. Furthermore, the classification network has an equivalent effect on both, the attCNN44 and attCNN105, and therefore does not benefit from the different numbers of labels.*

three networks show a good performance, whereas the performance of the third one drops very fast. Consequently, this has a negative influence on the overall performance of the ensemble. The reason for the behavior of the second network can be found in its training. As explained earlier, the first classification network is used to create the training dataset for the second network. This is done by taking an initial, randomly generated dataset with over four million triplets. The first network is then used to filter this dataset in order to obtain all triplets that were misclassified, which were on average about 130,000 triplets. These are far to few triplets for training the second classification network. Therefore, one possible way to solve this problem could be to use a ten times larger initial dataset consisting of 40 million triplets. After filtering this dataset, the expected size of the remaining dataset would be about one million triplets. Another possibility would be, to train a smaller architecture on the dataset that can be trained by 130,000 triplets. Eventually, this would perform better and




(a) Cumulative Match Characteristic curve




(b) Synthetic Recognition Rate curve

**Figure 5.17:** Comparison of distance and classification network

*This experiment compares the distance and the classification network, using the earlier determined best network configurations. Apparently, the classification network achieves a significantly better performance at all shown ranks. However, both outperform the L2 distance based re-identification. For further details see Table A.10 in Appendix A*

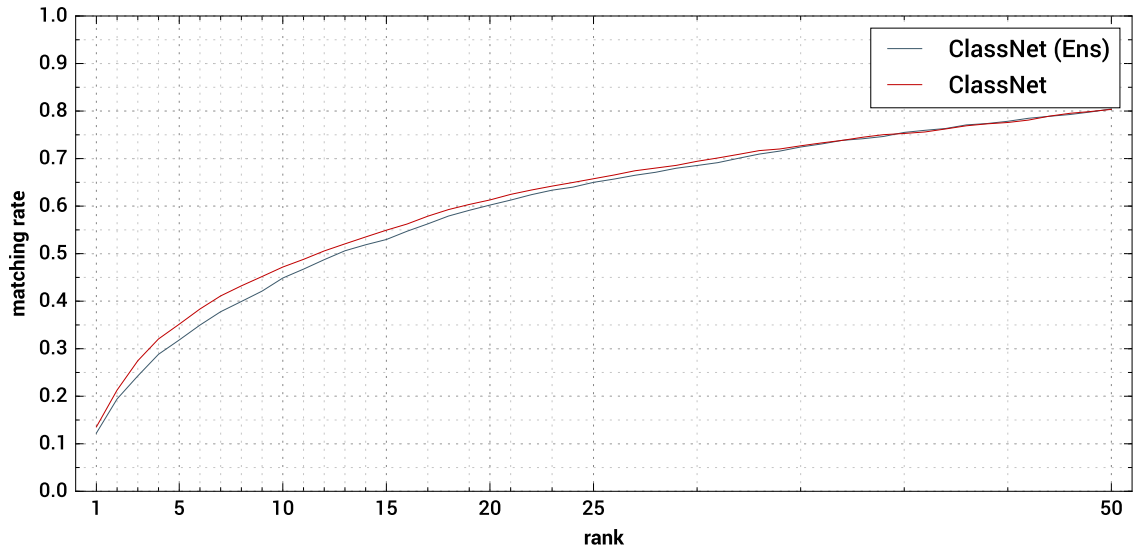



**Figure 5.18:** Classificaton network vs. Ensemble
*This figure shows the CMC curves for classification network and the ensemble of classification networks. One would expect the performance to improve, when using an ensemble approach. This is not the case. The ensemble reaches approximately the same performance. At earlier ranks, the performance is even slightly worser than the single network performance. As Figure 5.19 shows, this is due to the second network of the ensemble.*

therefore contribute to the performance of the ensemble. These ideas should be addressed in future work.

## 5.4 Summary

In this chapter, two conceptually different networks were presented, which try to solve the person re-identification problem based on semantic attributes. The first attempt, tries to learn a distance function, which is used to assign a distance score to pairs of attribute vectors. These scores are then used to perform the ranking of individuals from the probe gallery. The network itself uses pairs, however, the training of the network is triplet-based.
The second approach, tries to solve the mentioned problem by deciding which of the two

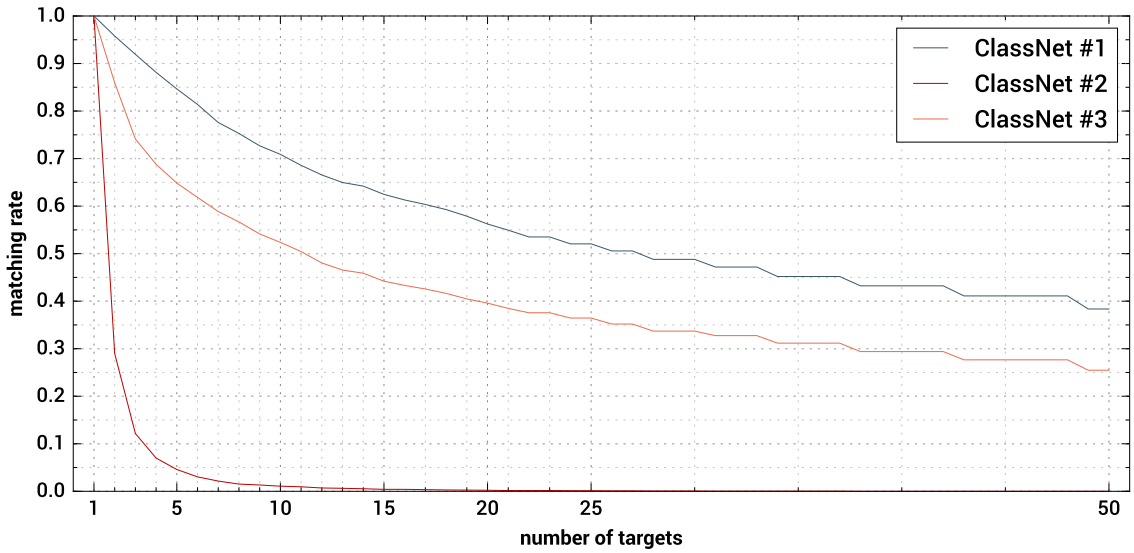



**Figure 5.19:** Evaluation of the performance of the single parts of the ensemble
*This figure shows the SRR curves for three parts of the ensemble as presented in Section 5.2.3. Network #1 performs well. Network #2, however, achieves a very bad performance. Most likely, due to the size of the used dataset. On average, the first network misclassifies about 3% of the original triplets (130,000 of 4 million triplets). This dataset is even with the used data augmentation techniques to small for the training of the network. Since the second network performs rather bad, many triplets are misclassified, so that the third network gets a much larger dataset and therefore is able to perform better again. This problem should be addressed in future work.*

attribute vectors is more similar to the anchor vector. For this network, certain ranking techniques are necessary. In this chapter, two methods for ranking were introduced and evaluated. In addition, for increasing the performance of the classification network, an ensemble was presented and evaluated as well. Finally, the two networks were compared with each other, as well as the SSDAL algorithm by [Su et al., 2016]. As already mentioned in the summary of Chapter 4, much more time has been invested into learning to predict attributes. Therefore, some aspects presented in this chapter should be addressed in future work.

# Conclusion

# 6

Finally, this chapter concludes the master's thesis. Section 6.1 focuses on giving an overview of the results obtained for both parts of this master's thesis: the recognition of semantic attributes, and the re-identification of persons by using the predicted attributes. Afterwards, Section 6.2 takes a look at further ways to improve the performance of the proposed algorithm

## 6.1 Summary

This master's thesis addresses the problem of person re-identification based on semantic attributes and Deep Learning. Therefore, it is divided into two different parts: the prediction of semantic attributes, and the re-identification based on these predicted attributes. In the first part a pretrained model is adapted in order to predict semantic attributes. This pretrained model is a Deep Convolutional Neural Network proposed by [Eisenbach et al., 2016], which was initially trained on images of persons in order to detect people. In order to use this model, it was adapted and extended. Different experiments were conducted to examine the suitability of the network for the task of recognizing attributes. Since the aim of the original architecture was to detect people, changes were necessary for the purpose of predicting semantic attributes. Therefore, the output layer of the Convolutional Neural Network was modified by increasing the number of output neurons to the amount of available attributes. Furthermore, the activation of the output neurons was changed to the sigmoid activation function, in order to use the network for the new task. The experiments

showed, that compared to other state-of-the-art methods based on Deep Learning and semantic attributes the adapted model achieves state-of-the-art classification results already. In addition to that, further experiments showed that different modifications on the architecture of the Convolutional Neural Network, as well as constraints on the used attributes enhanced the attribute classification performance of the original network. Especially, this master's thesis showed that the attribute classification performance of the modified Convolutional Neural Network could be improved by using so called Exponential Linear Units [Clevert et al., 2016]. Altogether, this part of the master's thesis showed, that the modified network achieves state-of-the-art performance when predicting semantic attributes.

As mentioned above, the second part of this master's thesis focused on the re-identification based on semantic attributes that were predicted by the Deep Convolutional Neural Network from the first part. For this purpose, two different approaches were examined for the purpose of re-identification. The first one tries to learn a distance function, which assigns a distance score to pairs of images. For this, a novel training procedure was proposed, which uses triplets in order to train the network. The second one tries to classify triplets into two classes, depending on which element of the triplet is more similar to the anchor element. These networks were both trained on a dataset of triplets, which was generated by using the Deep Convolutional Neural Network from the first part of the master's thesis. The images used for the construction of the triplet dataset were taken from a dataset, which is independent to the one used in the first part of this master's thesis. In a first experiment, it was examined how the Deep Convolutional Neural Network developed in the first part of this master's thesis performs compared to the SSDAL algorithm proposed by [Su et al., 2016], when applying the L2 distance for re-identification. The results showed, that the algorithm by [Su et al., 2016] performs better than the own developed approach. Further experiments showed that, by using the developed distance or classification network, a significant improvement of the re-identification performance could be achieved. As it turned out, the classification network achieved the best performance of both approaches that were mentioned above and brought the performance of the Convolutional Neural Network closer to the SSDAL algorithm.

## 6.2 Future Work

In order to improve the re-identification performance, different aspects could be examined in future work. As, the recognition of semantic attributes is a very important task for the re-identification of persons, it could be examined whether a more sophisticated output layer, using multiple softmax activated groups of neurons instead of one flat output layer improves the overall classification performance. These groups could contain for example a single or multiple corresponding semantic attributes. This idea was initially planned to be examined in this master's thesis. Due to the limited time this could not be considered and should be addressed by future work. Further improvement could be achieved by splitting images into different body parts. For each body part, a network could be trained to recognize semantic attributes as proposed by [Pala, 2016]. The output of these networks could be fused with the network developed in this master's thesisafterwards. The most promising enhancement, however, could be a fine-tuning by using an independent dataset as proposed by [Su et al., 2016]. The slight adjustment of the predicted attribute vectors could probably improve the performance even more.

# Tables and Implementation Details for the Algorithms

A

## A.1 Classification of Semantic Attributes

This part of the appendix contains additional information about the experiments performed in Chapter 4. The tables contain parameters for the performed trainings, grouped by experiments. In addition, they contain information about the used activation and output functions, as well as the numbers of fixed and imported layers from the original network by [Eisenbach et al., 2016].

| **Parameter** | **Value** |
|---|---|
| Optimizer | Stoch. Grad. Descent |
| Learning rate | 0.01 |
| Momentum | 0.7 |
| Activation | ReLU |
| Output | sigmoid |
| Nb. of imported layers | varying |
| Nb. of fixed layers | varying |

**Table A.1:** Parameters and Details for Experiment on page 47, 48, and 50
*This table contains the parameters used for the training of the networks in the first experiment, which has the purpose to examine the effects of the number of fixed layers on the performance of the network. The training was performed on a subset of the available attributes, namely the upper body colors. In Experiment 3, no layers were fixed at all.*

| **Parameter** | **Value** |
|---|---|
| Optimizer | Stoch. Grad. Descent |
| Learning rate | 0.01 |
| Momentum | 0.7 |
| Activation | ReLU |
| Output | sigmoid |
| Nb. of imported layers | 5 |
| Nb. of fixed layers | 3 |

**Table A.2:** Parameters and Details for the experiment on page 48
*This table contains parameters used for the training of the networks in the second experiment. Its purpose is to examine the effects removing filters or adding noise to imported filters.*

| **Parameter** | **Value** |
|---|---|
| Optimizer | Stoch. Grad. Descent |
| Learning rate | 0.01 |
| Momentum | 0.7 |
| Activation | ReLU |
| Output | sigmoid |
| Nb. of imported layers | varying |
| Nb. of fixed layers | varying |

**Table A.3:** Parameters and Details for the experiment on page 51
*This table contains the parameters used for the training of the networks in the fourth experiment. Its purpose is to examine the effects of using ELU activation on the performance of the network. The training was performed on the attributes that are available in at least 5% of the individuals. No layers were loaded here at all.*

| **Parameter** | **Value** |
|---|---|
| Optimizer | Stoch. Grad. Descent |
| Learning rate | 0.01 |
| Momentum | 0.7 |
| Activation | varying |
| Output | varying |
| Nb. of imported layers | 0 |
| Nb. of fixed layers | 0 |

**Table A.4:** Parameters and Details for Experiments on page 52 and 54
*This table contains the parameters used for the training of the networks in the fourth experiment. Its purpose is to examine the effects of using different activation functions in the output layer on the performance of the network. No layers were loaded here at all.*

## A.2 Attribute-based Person Re-Identification

Analogous to the previous section, detailed information, like learning rates, or specific characteristics of different performed experiments in Chapter 5 are given here. The tables contain parameters for the performed trainings, grouped by experiments. All used networks were trained on 44 semantic attributes.

### Distance Network

| **Parameter** | **Value** |
|---|---|
| $\alpha$ | 0.2 |
| Hidden neurons | 875 |
| Optimizer | Stoch. Grad. Descent |
| Learning rate | 0.005 |
| Momentum | 0.7 |
| Activation | ReLU |
| Output | linear |

**Table A.5:** Parameters and Details for Experiment on page 85 for the distance network

*This table contains the parameters used for the training of the network. $\alpha$ describes the chosen parameter of the loss function. The evaluation was performed on a subset of the training dataset which was excluded from training. The training includes half the VIPeR dataset as well.*

| **Parameter** | **Value** |
|---|---|
| $\alpha$ | 0.2 |
| Hidden neurons | varying |
| Optimizer | Stoch. Grad. Descent |
| Learning rate | 0.005 |
| Momentum | 0.7 |
| Activation | ReLU |
| Output | linear |

**Table A.6:** Parameters and details for investigation of the network size of the distance network

*This table contains the parameters used for training the network in the experiment on page 86. $\alpha$ describes the chosen parameter of the loss function. The evaluation was performed on a subset of the training dataset which was excluded from training.*

| **Parameter** | **Value** |
|---|---|
| $\alpha$ | 0.2 |
| Hidden neurons | 875 |
| Optimizer | Stoch. Grad. Descent |
| Learning rate | 0.005 |
| Momentum | 0.7 |
| Activation | ReLU |
| Output | linear |

**Table A.7:** Parameters and details for investigation of the network size of the distance network

*This table contains the parameters used for training of the final network presented on page 87. $\alpha$ describes the chosen parameter of the loss function. The evaluation was performed on a VIPeR, following the procedure presented in 5.2.1.*

### Classification Network

| **Parameter** | **Value** |
|---|---|
| Optimizer | Stoch. Grad. Descent |
| Learning rate | 0.01 |
| Momentum | 0.7 |
| Activation | ReLU |
| Output | softmax |

**Table A.8:** Parameters and Details for Classification Network Experiment on page 90

*This table contains the parameters used for the training of the classification network in the first experiment. The purpose of the experiment is to determine the best number of neurons in the hidden layers. The learning parameters were not evaluated further, but set to the values shown in the table. The training was performed on attribute labels predicted on images from PETA. VIPeR, which was excluded from training, was used for the evaluation.*

| **Parameter** | **Value** |
|---|---|
| Optimizer | Stoch. Grad. Descent |
| Hidden neurons | 1000 |
| Learning rate | 0.01 |
| Momentum | 0.7 |
| Activation | ReLU |
| Output | softmax |

**Table A.9:** Parameters and Details for Classification Network used for the ranking experiments on page 90

*This table contains the parameters used for the experiment that examines the performance of the presented ranking techniques. The training was performed on attribute labels predicted on images from PETA. VIPeR, which was excluded from training, was used for the evaluation.*

| **Parameter** | **Value** |
|---|---|
| Optimizer | Stoch. Grad. Descent |
| Hidden neurons | 1000 |
| Learning rate | 0.01 |
| Momentum | 0.7 |
| Activation | ReLU |
| Nb. of attr. | 44 |
| Output | softmax |

**Table A.10:** Parameters and Details for Classification Network used for the ranking experiments on page 96

*This table contains the parameters used for the experiment that examines the performance of the presented ranking techniques. The training was performed on attribute labels predicted on images from PETA. VIPeR, which was excluded from training, was used for the evaluation.*

| | **SSDAL** | **AlexNet (SSDAL)** | **attCNN105** | **attCNN44** |
|---|---|---|---|---|
| **Rank 1** | 37.9 | 34.5 | 6.96 | 5.4 |
| **Rank 5** | 65.5 | 63.9 | 17.9 | 17.3 |
| **Rank 10** | 75.6 | 73.1 | 26.6 | 24.4 |
| **Rank 15** | — | — | 33.0 | 30.3 |
| **Rank 20** | 88.4 | 87.0 | 37.6 | 36.1 |

**Table A.11:** Comparison of matching rates from the CMC curves on different ranks by using L2 distance for re-identification on page 92 and 94

*This table shows how the SSDAL algorithm as well as the underlying AlexNet perform, compared to the attCNN developed in this master's thesis for attribute prediction. The number at the end of the name indicates the number of used attributes. Hence, attCNN44 predicts only 44 attributes. Obviously, both attCNNs are of inferior performance compared to the AlexNet and the SSDAL algorithm.*

| | **ClassNet** | **CN Ensemble** | **DistNet** |
|---|---|---|---|
| **Rank 1** | 13.5 | 12.2 | 6.7 |
| **Rank 5** | 35.2 | 31.8 | 23.3 |
| **Rank 10** | 47.2 | 44.9 | 36.5 |
| **Rank 15** | 54.9 | 53.0 | 42.9 |
| **Rank 20** | 61.3 | 60.2 | 48.6 |

**Table A.12:** Values for the CMC curves of the classification network, the ensemble of classification networks and distance network architecture 96 and 97
*These values correspond to the experiments that involve one of these three architectures. The single classification network performs best of all developed approaches. These scores were obtained by training the corresponding network on predicted attribute vectors with 44 labels.*

# Pedestrian Attribute Dataset (PETA)

# B

## B.1 List of available attributes

This section gives some detailed information about the attributes existing in the used dataset, which also implies information about the frequency of each attribute in the dataset.

| Id | Name | Count | % |
|---|---|---|---|
| 0 | footwearBlack | 4070 | 46.7440 |
| 1 | footwearBlue | 20 | 0.2297 |
| 2 | footwearBrown | 729 | 8.3726 |
| 3 | footwearGreen | 42 | 0.4824 |
| 4 | footwearGrey | 1256 | 14.4252 |
| 5 | footwearOrange | 34 | 0.3905 |
| 6 | footwearPink | 16 | 0.1838 |
| 7 | footwearPurple | 13 | 0.1493 |
| 8 | footwearRed | 175 | 2.0099 |
| 9 | footwearWhite | 1753 | 20.1332 |
| 10 | footwearYellow | 32 | 0.3675 |

**Table B.1:** List of attributes - part #1
*This table shows the attributes 0 to 10 with their names, the absolute frequency of their occurrence in images from the dataset as well as the relative occurrence.*

| Id | Name | Count | % |
|---|---|---|---|
| 11 | hairBlack | 6859 | 78.7757 |
| 12 | hairBlue | 0 | 0.0000 |
| 13 | hairBrown | 1102 | 12.6565 |
| 14 | hairGreen | 1 | 0.0115 |
| 15 | hairGrey | 382 | 4.3873 |
| 16 | hairOrange | 2 | 0.0230 |
| 17 | hairPink | 0 | 0.0000 |
| 18 | hairPurple | 2 | 0.0230 |
| 19 | hairRed | 1 | 0.0115 |
| 20 | hairWhite | 99 | 1.1370 |
| 21 | hairYellow | 205 | 2.3544 |
| 22 | lowerBodyBlack | 4293 | 49.3052 |
| 23 | lowerBodyBlue | 1399 | 16.0675 |
| 24 | lowerBodyBrown | 310 | 3.5604 |
| 25 | lowerBodyGreen | 29 | 0.3331 |
| 26 | lowerBodyGrey | 2258 | 25.9332 |
| 27 | lowerBodyOrange | 8 | 0.0919 |
| 28 | lowerBodyPink | 21 | 0.2412 |
| 29 | lowerBodyPurple | 27 | 0.3101 |
| 30 | lowerBodyRed | 68 | 0.7810 |
| 31 | lowerBodyWhite | 398 | 4.5710 |
| 32 | lowerBodyYellow | 18 | 0.2067 |
| 33 | upperBodyBlack | 3661 | 42.0466 |
| 34 | upperBodyBlue | 644 | 7.3963 |
| 35 | upperBodyBrown | 495 | 5.6851 |

**Table B.2:** List of attributes - part #2

*This table shows the attributes 11 to 35 with their names, the absolute frequency of their occurrence in images from the dataset as well as the relative occurrence.*

| Id | Name | Count | % |
|---|---|---|---|
| 36 | upperBodyGreen | 264 | 3.0320 |
| 37 | upperBodyGrey | 1709 | 19.6279 |
| 38 | upperBodyOrange | 106 | 1.2174 |
| 39 | upperBodyPink | 161 | 1.8491 |
| 40 | upperBodyPurple | 276 | 3.1699 |
| 41 | upperBodyRed | 615 | 7.0633 |
| 42 | upperBodyWhite | 1882 | 21.6148 |
| 43 | upperBodyYellow | 128 | 1.4701 |
| 44 | accessoryHeadphone | 58 | 0.6661 |
| 45 | personalLess15 | 57 | 0.6546 |
| 46 | personalLess30 | 5857 | 67.2677 |
| 47 | personalLess45 | 2164 | 24.8536 |
| 48 | personalLess60 | 495 | 5.6851 |
| 49 | personalLarger60 | 131 | 1.5045 |
| 50 | carryingBabyBuggy | 36 | 0.4135 |
| 51 | carryingBackpack | 2518 | 28.9193 |
| 52 | hairBald | 140 | 1.6079 |
| 53 | footwearBoots | 249 | 2.8598 |
| 54 | lowerBodyCapri | 205 | 2.3544 |
| 55 | carryingOther | 1711 | 19.6509 |
| 56 | carryingShoppingTro | 13 | 0.1493 |
| 57 | carryingUmbrella | 19 | 0.2182 |
| 58 | lowerBodyCasual | 7786 | 89.4223 |
| 59 | upperBodyCasual | 7738 | 88.8710 |
| 60 | personalFemale | 3229 | 37.0851 |

**Table B.3:** List of attributes - part #3

*This table shows the attributes 36 to 60 with their names, the absolute frequency of their occurrence in images from the dataset as well as the relative occurrence.*

| Id | Name | Count | % |
|---|---|---|---|
| 61 | carryingFolder | 157 | 1.8031 |
| 62 | lowerBodyFormal | 892 | 10.2446 |
| 63 | upperBodyFormal | 892 | 10.2446 |
| 64 | accessoryHairBand | 394 | 4.5251 |
| 65 | accessoryHat | 306 | 3.5144 |
| 66 | lowerBodyHotPants | 36 | 0.4135 |
| 67 | upperBodyJacket | 518 | 5.9492 |
| 68 | lowerBodyJeans | 2621 | 30.1022 |
| 69 | accessoryKerchief | 14 | 0.1608 |
| 70 | footwearLeatherShoes | 1717 | 19.7198 |
| 71 | upperBodyLogo | 497 | 5.7081 |
| 72 | hairLong | 2024 | 23.2457 |
| 73 | lowerBodyLongSkirt | 133 | 1.5275 |
| 74 | upperBodyLongSleeve | 6785 | 77.9258 |
| 75 | lowerBodyPlaid | 13 | 0.1493 |
| 76 | lowerBodyThinStripes | 22 | 0.2527 |
| 77 | carryingLuggageCase | 105 | 1.2059 |
| 78 | personalMale | 5477 | 62.9034 |
| 79 | carryingMessengerBag | 2099 | 24.1070 |
| 80 | accessoryMuffler | 148 | 1.6998 |
| 81 | accessoryNothing | 7559 | 86.8152 |
| 82 | carryingNothing | 2356 | 27.0587 |
| 83 | upperBodyNoSleeve | 240 | 2.7564 |
| 84 | upperBodyPlaid | 308 | 3.5374 |
| 85 | carryingPlasticBags | 407 | 4.6744 |

**Table B.4:** List of attributes - part #4

*This table shows the attributes 61 to 85 with their names, the absolute frequency of their occurrence in images from the dataset as well as the relative occurrence.*

| Id | Name | Count | % |
|---|---|---|---|
| 86 | footwearSandals | 257 | 2.9516 |
| 87 | footwearShoes | 2780 | 31.9283 |
| 88 | hairShort | 6598 | 75.7781 |
| 89 | lowerBodyShorts | 487 | 5.5932 |
| 90 | upperBodyShortSleeve | 1652 | 18.9732 |
| 91 | lowerBodyShortSkirt | 466 | 5.3520 |
| 92 | footwearSneakers | 2719 | 31.2277 |
| 93 | footwearStocking | 390 | 4.4792 |
| 94 | upperBodyThinStripes | 222 | 2.5497 |
| 95 | upperBodySuit | 305 | 3.5029 |
| 96 | carryingSuitcase | 177 | 2.0328 |
| 97 | lowerBodySuits | 425 | 4.8811 |
| 98 | accessorySunglasses | 278 | 3.1928 |
| 99 | upperBodySweater | 164 | 1.8835 |
| 100 | upperBodyThickStripes | 89 | 1.0222 |
| 101 | lowerBodyTrousers | 4375 | 50.2469 |
| 102 | upperBodyTshirt | 840 | 9.6474 |
| 103 | upperBodyOther | 2232 | 25.6345 |
| 104 | upperBodyVNeck | 82 | 0.9418 |

**Table B.5:** List of attributes - part #5

*This table shows the attributes 86 to 104 with their names, the absolute frequency of their occurrence in images from the dataset as well as the relative occurrence.*

As Table B.1 to B.5 show, some attributes appear very rarely. For instance *hairBlue (12)* and *hairPink (17)* are not provided by even one example image. Whereas such rare attributes are perfectly suited for identifying persons due to their uniqueness, it is a very difficult task to learn the weights of a deep neural network with so few training examples. Hence, it seems natural not to consider attributes for training, which do not appear in at least a certain amount of images. Table B.7 shows all 77

attributes which can be found in at least 1% of individuals. Furthermore, those which exceed the 5% threshold are marked in black. Already by taking only attributes available in $\frac{1}{100}$ of the dataset, the number of usable attributes shrinks by about 27% to in total 77 attributes. Raising the threshold to 5%, only 44 of the 105 attributes remain, which is about 42% of the original size.

## B.2 Composition of dataset

As stated by [Deng et al., 2014] the PETA dataset consists of ten different already existing datasets, containing 19,000 images of 8,705 persons from distinct perspectives and in different sizes. Each image is labeled with a subset of attributes, which are presented in Section B.1. An overview on the composition of the PETA dataset is given in Table B.6.

| **Dataset** | **Number of images** | **%** |
|---|---:|---:|
| 3DPeS | 1012 | 5.3263 |
| CAVIAR4REID | 1220 | 6.4211 |
| CUHK | 4563 | 24.0158 |
| GRID | 1275 | 6.7105 |
| i-LIDS | 477 | 2.5105 |
| MIT | 888 | 4.6737 |
| PRID | 1134 | 5.9684 |
| SARC3D | 200 | 1.0526 |
| TownCentre | 6967 | 36.6684 |
| VIPeR | 1264 | 6.6526 |

**Table B.6:** Composition of PETA dataset

*The PETA dataset contains many smaller datasets and two larger ones, which provide more than 60% of all available images.*

| | | |
|---|---|---|
| accessoryHairBand | accessoryHat | accessoryMuffler |
| accessoryNothing | accessorySunglasses | carryingBackpack |
| carryingFolder | carryingLuggageCase | carryingMessengerBag |
| carryingNothing | carryingOther | carryingPlasticBags |
| carryingSuitcase | footwearBlack | footwearBoots |
| footwearBrown | footwearGrey | footwearLeatherShoes |
| footwearRed | footwearSandals | footwearShoes |
| footwearSneakers | footwearStocking | footwearWhite |
| hairBald | hairBlack | hairBrown |
| hairGrey | hairLong | hairShort |
| hairWhite | hairYellow | lowerBodyBlack |
| lowerBodyBlue | lowerBodyBrown | lowerBodyCapri |
| lowerBodyCasual | lowerBodyFormal | lowerBodyGrey |
| lowerBodyJeans | lowerBodyLongSkirt | lowerBodyShortSkirt |
| lowerBodyShorts | lowerBodySuits | lowerBodyTrousers |
| lowerBodyWhite | personalFemale | personalLarger60 |
| personalLess30 | personalLess45 | personalLess60 |
| personalMale | upperBodyBlack | upperBodyBlue |
| upperBodyBrown | upperBodyCasual | upperBodyFormal |
| upperBodyGreen | upperBodyGrey | upperBodyJacket |
| upperBodyLogo | upperBodyLongSleeve | upperBodyNoSleeve |
| upperBodyOrange | upperBodyOther | upperBodyPink |
| upperBodyPlaid | upperBodyPurple | upperBodyRed |
| upperBodyShortSleeve | upperBodySuit | upperBodySweater |
| upperBodyThickStripes | upperBodyThinStripes | upperBodyTshirt |
| upperBodyWhite | upperBodyYellow | |

**Table B.7:** Subset of attributes with at least 1% frequency

*Alphabetically sorted list of all attributes from PETA dataset which can be found in at least 1% of person images. The black highlighted attributes indicate labels which are present in at least 5% of person images.*

# References

# C

| Conferences | | | | | | |
|---|---|---|---|---|---|---|
| Year | '11 | '12 | '13 | '14 | '15 | '16 |
| **CVPR** - Conference on Computer Vision and Pattern Recognition | 0 | 1 | 1 | 1 | 2 | 0 |
| **ICCV** - International Conference on Computer Vision | 0 | 0 | 0 | 0 | 2 | 0 |
| **ECCV** - European Conference on Computer Vision | 0 | 0 | 0 | 2 | 0 | 1 |
| **ICLR** - International Conference on Intelligent Autonomous Systems | 0 | 0 | 0 | 0 | 0 | 1 |
| **BMVC** - British Machine Vision Conference | 1 | 1 | 0 | 0 | 0 | 0 |
| **NIPS** - Conference on Neural Information Processing Systems | 0 | 1 | 0 | 0 | 0 | 0 |
| Entry: Number of relevant publications | | | | | | |

| Journals | | | | | |
|---|---|---|---|---|---|
| Year | 2012 | 2013 | 2014 | 2015 | 2016 |
| Pattern Recognition | 0 | 0 | 0 | 1 | 0 |
| Computer Vision and Image Understanding | 0 | 1 | 0 | 0 | 0 |
| Nature | 0 | 0 | 0 | 1 | 0 |
| Entry: Number of relevant publications | | | | | |

# List of Figures

# List of Tables

# Bibliography

[Ahmed et al., 2015] Ahmed, E., Jones, M., and Marks, T. K. (2015). An improved deep learning architecture for person re-identification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Bak et al., 2010] Bak, S., Corvee, E., Bremond, F., and Thonnat, M. (2010). Person re-identification using spatial covariance regions of human body parts. In *IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 435–440.

[Cheng et al., 2011] Cheng, D. S., Cristani, M., Stoppa, M., Bazzani, L., and Murino, V. (2011). Custom pictorial structures for re-identification. In *British Machine Vision Conference (BMVC)*.

[Chollet, 2015] Chollet, F. (2015). Keras. `https://github.com/fchollet/keras`.

[Clevert et al., 2016] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). In *Proceedings International Conference on Learning Representations (ICLR)*.

[Davis et al., 2007] Davis, J. V., Kulis, B., Jain, P., Sra, S., and Dhillon, I. S. (2007). Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 209–216, New York, NY, USA. ACM.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Deng et al., 2014] Deng, Y., Luo, P., Loy, C. C., and Tang, X. (2014). Pedestrian attribute recognition at far distance. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM, pages 789–792, New York, NY, USA. ACM.

[Ding et al., 2015] Ding, S., Lin, L., Wang, G., and Chao, H. (2015). Deep feature learning with relative distance comparison for person re-identification. *Pattern Recognition*, 48(10):2993–3003.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 12:2121–2159.

[Eisenbach et al., 2016] Eisenbach, M., Seichter, D., Wengefeld, T., and Gross, H.-M. (2016). Cooperative multi-scale convolutional neural networks for person detection. In *World Congress on Computational Intelligence (WCCI)*, pages 267–276. IEEE.

[Eisenbach et al., 2015] Eisenbach, M., Vorndran, A., Sorge, S., and Gross, H. M. (2015). User recognition for guiding and following people with a mobile robot in a clinical environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3600–3607.

[Farenzena et al., 2010] Farenzena, M., Bazzani, L., Perina, A., Murino, V., and Cristani, M. (2010). Person re-identification by symmetry-driven accumulation of local features. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2360 –2367.

[Gross et al., 2016] Gross, H.-M., Scheidig, A., Debes, K., Einhorn, E., Eisenbach, M., Mueller, S., Schmiedel, T., Trinh, T. Q., Weinrich, C., Wengefeld, T., Bley, A., and Martin, C. (2016). Roreas: robot coach for walking and orientation training in clinical post-stroke rehabilitation—prototype implementation and evaluation in field trials. *Autonomous Robots*, pages 1–20.

[Jobson et al., 1997] Jobson, D. J., Rahman, Z., and Woodell, G. A. (1997). A multi-scale retinex for bridging the gap between color images and the human observation of scenes. *IEEE Transactions on Image Processing*, 6(7):965–976.

[Karpathy, 2015] Karpathy, A. (2015). Cs231n convolutional neural networks for visual recognition. `http://cs231n.github.io/convolutional-networks/#case`. (Accessed on 06/27/2016).

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Köstinger et al., 2012] Köstinger, M., Hirzer, M., Wohlhart, P., Roth, P. M., and Bischof, H. (2012). Large scale metric learning from equivalence constraints. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2288–2295.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[Kulis, 2013] Kulis, B. (2013). Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364.

[Layne et al., 2012] Layne, R., Hospedales, T. M., and Gong, S. (2012). Person re-identification by attributes. In Bowden, R., Collomosse, J. P., and Mikolajczyk, K., editors, *British Machine Vision Conference, BMVC, Surrey, UK, September 3-7, 2012*, pages 1–11. BMVA Press.

[Layne et al., 2014] Layne, R., Hospedales, T. M., and Gong, S. (2014). *Attributes-Based Re-identification*, pages 93–117. Springer London, London.

[Lecun et al., 2015] Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Li et al., 2014] Li, W., Zhao, R., Xiao, T., and Wang, X. (2014). Deepreid: Deep filter pairing neural network for person re-identification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 152–159.

[Liao et al., 2015] Liao, S., Hu, Y., Zhu, X., and Li, S. Z. (2015). Person re-identification by local maximal occurrence representation and metric learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[Liao et al., 2010] Liao, S., Zhao, G., Kellokumpu, V., Pietikäinen, M., and Li, S. Z. (2010). Modeling pixel process with scale invariant local patterns for background subtraction in complex scenes. In *IEEE Conference on Computer Vision and Pattern Recognition CVPR*, pages 1301–1306.

[Nielsen, 2015] Nielsen, M. A. (2015). *Neural Networks and Deep Learning.* Determination Press.

[Pala, 2016] Pala, F. (2016). Re-identification and semantic retrieval of pedestrians in video surveillance scenarios.

[Patrice Y. Simard, 2003] Patrice Y. Simard, Dave Steinkraus, J. P. (2003). Best practices for convolutional neural networks applied to visual document analysis. Institute of Electrical and Electronics Engineers, Inc.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain.

[Schroff et al., 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

[Sorge, 2013] Sorge, S. (2013). Wiedererkennung von Personen durch symmetriegetriebene Extraktion von Merkmalen. Master's thesis.

[Su et al., 2015] Su, C., Yang, F., Zhang, S., Tian, Q., Davis, L. S., and Gao, W. (2015). Multi-task learning with low rank attribute embedding for person re-identification. In *IEEE International Conference on Computer Vision (ICCV)*.

[Su et al., 2016] Su, C., Zhang, S., Xing, J., Gao, W., and Tian, Q. (2016). Deep attributes driven multi-camera person re-identification. *CoRR*, abs/1605.03259.

[Szegedy et al., 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842.

[Theano Development Team, 2016] Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

[Vorndran, 2015] Vorndran, A. (2015). Evaluation von Distance-Metric-Learning für die Kleidungs-basierte Wiedererkennung von Personen im klinischen Einsatzfeld. Bachelor's thesis.

[Wang et al., 2015] Wang, S., Shang, Y., Wang, J., Mei, L., and Hu, C. (2015). Deep features for person re-identification. In *2015 11th International Conference on Semantics, Knowledge and Grids (SKG)*, pages 244–247.

[Weinberger et al., 2006] Weinberger, K. Q., Blitzer, J., and Saul, L. K. (2006). Distance metric learning for large margin nearest neighbor classification. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *Conference on Neural Information Processing Systems (NIPS)*, pages 1473–1480. MIT Press.

[Xiong et al., 2014] Xiong, F., Gou, M., Camps, O., and Sznaier, M. (2014). Person re-identification using kernel-based metric learning methods. In *European Conference on Computer Vision (ECCV)*, pages 1–16. Springer.

[Yi et al., 2014] Yi, D., Lei, Z., Liao, S., and Li, S. Z. (2014). Deep metric learning for person re-identification. In *International Conference on Pattern Recognition (ICPR)*.

[Zeiler, 2012] Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

[Zhao et al., 2013] Zhao, R., Ouyang, W., and Wang, X. (2013). Unsupervised salience learning for person re-identification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3586–3593.

[Zheng et al., 2015] Zheng, L., Shen, L., Tian, L., Wang, S., Wang, J., and Tian, Q. (2015). Scalable person re-identification: A benchmark. In *IEEE International Conference on Computer Vision (ICCV)*.

[Zhu et al., 2015] Zhu, J., Liao, S., Yi, D., Lei, Z., and Li, S. Z. (2015). Multi-label cnn based pedestrian attribute learning for soft biometrics. In *International Conference on Biometrics (ICB)*, pages 535–540.