Master's Thesis

Practical Deep Learning for Person Re-Identification In Video Surveillance Systems

by
Gabriëlle E.H. Ras
s4628675

Supervisors
Dr. Marcel van Gerven   Radboud University Nijmegen
Dr. Gregor Pavlin   Thales Research & Technology, Thales Nederland

Radboud University Nijmegen

September 20, 2017

Contents

Abstract

In an existing particle filter tracking system at Thales Research & Technology it is required for a deep neural network to perform person re-identification on image pairs, maximizing the TPR and minimizing the FPR, in a situation where we have a relatively small gallery size. Current available literature on the use of deep learning in person re-identification focus on improving large dataset benchmarks with large gallery sizes, while improving the CMC ranking on individual benchmarks using very deep neural networks. While CMC ranking is a good test of the discriminative ability of a network, not much is known about the TPR and FPR performance of smaller deep neural networks in situations with small gallery sizes. In this study we found that as the gallery size increases, the ranking performance of the network goes down while the TPR and FPR remain similar. We show that a relatively small neural network can achieve similar performance on benchmark as very deep neural network, assuming that the gallery is small enough ($\leq$300 IDs). We achieve a performance of 50% on ViPER, 56% on PRID450 and 21% on Market1501. Most literature use simple distance metrics such as the L2 or Euclidean distance between the extracted image features to perform metric learning. However, neural metric learning has not been a topic of investigation, which if used, can results in a more natural approach to person re-identification by having a neural network learn the boundary between (mis)match classes. In this study we compare the Euclidean distance to various neural metric learners and found that the Euclidean distance consistently outperforms the neural metric learners. This study also investigates various training schemes for training a neural network on a mixture distribution consisting of multiple small datasets. However, no consistent improvements could be found compared to training on a single distribution. We did find that large and noisy datasets tend to generalize well to new environments and that the dataset mean can be used to gauge the ability for generalization. Finally we investigate the effectiveness of image data compared to video data, by comparing the performance of SCNN trained on images to the performance of S3DCNN trained on videos. We found that S3DCNN outperforms SCNN in all cases. We also found that when the dataset becomes larger ($\geq$ 1400 instances), the difference between the performances of SCNN and S3DCNN decreases.

Chapter 1

Introduction

Tracking vehicles or persons in urban and indoor environments is a vital functionality in security and safety applications. Knowing the whereabouts of subjects involved in law offenses is a critical capability that enables informed decision making. Due to the environmental constraints in each specific environment, tracking systems often rely on a myriad of independent sensors that assimilate various kind of data, which is later fused together. This requires suitable algorithms that can cope with diverse data, which is often of low quality and collected at low frequency. Thales Research & Technology Delft has recently developed a technique to cope with the challenges of fusing various data sources by combining particle filters with Bayesian networks [1, 2]. This system can incorporate an arbitrary number of and type of sensors in any given location and timespan. In this thesis we consider the context in which we want to track a person in an indoor, office-like environment using cameras. One of the challenges in this context is associating the camera data with the identity of the object being tracked. In other words: How do we know if the camera is observing the same person?

For the past 7 years, the use of artificial neural networks and deep learning has increasingly become more common, proving to be particularly successful in the area of computer vision. In this thesis we will investigate the use of deep neural network as a possible type of sensor model to incorporate in the particle filter. Specifically, a deep neural network will be used to perform person re-identification, which is the problem of recognizing a person who was previously observed.

In the next sections we will explain in detail the various subjects mentioned above.

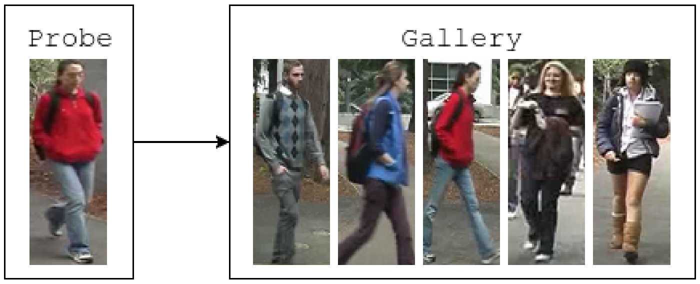


Figure 1.1: The person re-identification problem: given a probe, find the matching candidate in the gallery.

1.1 Background

1.1.1 Person Re-Identification

Person re-identification is the problem of recognizing a person who was previously observed. The *probe* is the person that we want to re-identify and the *gallery* contains a set of *candidates* that may match the probe. Given a gallery and a probe, the goal is to find the candidate that matches the probe. We will refer to the number of unique people in a dataset as the number of *identities (IDs)*. We illustrate the problem of person re-identification in Figure 1.1 with image samples taken from the VIPeR dataset [3], where the matching candidate in the gallery is the third image.

Probe image(s) are obtained from the videostream at the moment when the suspicious person is identified for the first time. It can also be that we have previous video footage of the probe, recorded at different scenario. But for the sake of simplicity we will assume that probe images and candidate images are taken in the same environment. Once the probe images are available we can start the process of re-identification. Candidate images are collected each time a person is detected and cropped to a specific height and width. A probe-candidate pair (p, c) is made and sent to the person re-identification algorithm, which returns a number between 0 and 1 indicating how similar the images are.

To be able to recognize the observed person, a re-identification algorithm needs to have two important components. The first component refers to the ability to identify the correct set of features that describe this person. In other words: What are the characteristics that discriminate this person from other people? This is known as feature learning. The second component refers to the ability to correctly match two sets of person features: Given two separate video footage, each containing a person, are these footage of the same person? This is known as similarity metric learning.

Until the rise of deep learning, both components were handled using a combination of classical computer vision methods [4]. These methods mainly utilize the information carried directly through pixels and lower level statistical information between (groups of) pixels. Intuitively speaking, they do not understand what is going on in the image at a higher level of abstraction. There is also a clear distinction between the focus on feature learning and the focus on similarity metric learning, even though both are equally important. However, since the re-discovery and rising popularity of deep learning algorithms, feature learning and similarity metric learning can be done simultaneously. Algorithms such as Convolutional Neural Networks (CNN) and three-dimensional CNNs (3DCNNs) have the ability to "interpret" spatio-temporal data in a high-level manner only from raw pixel input without any feature extraction done beforehand. Through CNNs we are able to accurately detect many different types of object in an image in various scenes [5]. By adding another dimension we arrive to 3DCNN, which are able to extract features through space as well as time. CNNs will be explained in more detail in Section 1.1.4.

The advantage of deep learning over classical computer vision is that deep learning can process information in an end-to-end manner. This means that instead of designing a system with many different components to detect hand-made features, we design one deep learning network that is able to do the same thing. This does however require a lot of labelled data. The developments in the field of deep learning have had an impact on the applications for person re-identification. Recent literature indicates that application of deep learning approaches for the purpose of person re-identification is promising. Novel deep learning architectures have been designed that outperform the state-of-the-art computer vision based methods on several benchmarks [6, 7, 8, 9, 10].

CMC Curve Rank-1 Score

Benchmarks on person re-identification aim to increase the rank-1 score of the Cumulative Match Characteristic (CMC) curve. This performance measure is also used in other re-identification settings, such as facial recognition. Given a re-identification task where there is a set of probes and a gallery, the CMC reflects how well the network can sort the candidates in the gallery based on their individual similarity with respect to the probe. More formally, given a set of n probes $P = p_1, p_2, ..., p_n$ and a gallery containing n candidates $G = c_1, c_2, ..., c_n$ the CMC algorithm outputs an ordering for each p_i, $(1 \leq i \leq n)$ in P, sorted in decreasing order on the similarity score computed by the re-identification algorithm for each probe-candidate pair (p_i, c_j). The procedure is denoted in Algorithm 1.

Usually only rank-1 of the CMC curve is compared. In our re-identification context we do not take CMC rank-1 score into account because this number is not relevant to the inner workings of the particle filter algorithm. Instead we only look at the True Positive Rate and the False Positive Rate. These two performance criteria will be explained later in Section 1.1.3. However, we will test our developed networks on existing benchmarks to compare with state of the art, but we will not base design decisions on the outcome of these tests.

```
input  : list probes =[probe_1, probe_2, ..., probe_n]
         list gallery = [candidate_1, candidate_2, ..., candidate_n]
output: list ranking=[rank_1, rank_2, ..., rank_n]
ranking ← [0, 0, ..., 0]                                  // has length n
foreach probe in probes do
    S ← empty list                                       // starting at index 1
    foreach candidate in gallery do
        s ← CalculateSimilarity(probe, candidate)
        c ← candidate
        insert (s, c) into S
    end
    S′ ← sort S on s                                     // in decreasing order
    r ← 1
    foreach (s, c) in S′ do
        candidate ← c
        if probe == candidate then
            ranking[r] ← ranking[r] + 1
            break
        end
        r ← r + 1
    end
end
ranking′ ← DivideBy(ranking, n)
return ranking′
```

Algorithm 1: Cumulative Match Characteristic (CMC). The CMC reflects how well the network can sort the images in the gallery based on their individual similarity with respect to the probe.

1.1.2 Intended Use-Case: Indoor Setup

We consider an indoor environment where there are up to 100 unique identities (persons) walking around. Each camera will be able to capture the full body of each identity. The

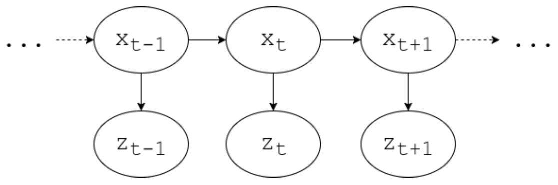


Figure 1.2: Caption

camera will be mounted at a height of about 2 meters in locations where only one identity can be seen on the image at a time. A face detection algorithm using Haar Cascades will detect if there is a face in the frame and a bounding box of 128×64 pixels will be formed around the body using the location of the face as reference point. The body image is cropped out and sent to the deep neural network for comparison. The neural network returns whether or not the captured identity matches the probe identity and this response gets passed on to the particle filter.

1.1.3 The Particle Filter

In this section we will only explain the components of the particle filter relevant to the deep neural network that will be developed for person re-identification. From the following explanation it should be clear what purpose the deep neural network serves when viewed in the context of the particle filter. For the full description of all the components of the particle filter see [2].

Particle filters are a type of Monte Carlo method for estimating the internal states in dynamical systems, when there are partial observations, noisy sensors and random perturbations in the system. The goal is to compute the posterior distribution over all the states. The filtering process is modeled as a dynamic Bayesian network as can be seen in Figure 1.2, where x_t represents a state at time t and z_t represents the measurement, capturing the observable phenomena, at time t. A state is a variable indicating if the person we are tracking has been present at some location at time t. A measurement z_t is the output of the neural network which receives input from the sensor at time t. The neural network returns *true* if the input of the sensor matches the probe and *false* otherwise. For each of the two cases we have a probability $p(z_t = true|x_t)$ or $p(z_t = false|x_t)$. The idea is to recursively calculate a degree of belief over state x_t by considering a set of collected measurements $z_{1:t}$ up to time t. Ultimately we want to estimate the probability density function (pdf) $p(x_t|z_{1:t})$, representing the probability of a certain state at time t given all measurements until time t. Assuming that $p(x_{t-1}|z_{1:t-1})$ from the previous time step is available we can

	PREDICTION		
	MATCH	MISMATCH	
GROUND TRUTH	True Positive	False Negative	MATCH′
	False Positive	True Negative	MISMATCH′

Table 1.1: A confusion matrix.

compute $p(x_t|z_{1:t})$ by first calculating $p(x_t|z_{1:t-1})$:

$$p(x_t|z_{1:t-1}) = \int p(x_{t-1}|z_{1:t-1})p(x_t|x_{t-1})dx_{t-1} \tag{1.1}$$

where $p(x_t|x_{t-1})$ is the transition model capturing the state evolution over time. At each time t a measurement z_t can be obtained and used to update $p(x_t|z_{1:t-1})$ with Bayes' rule, resulting in $p(x_t|z_{1:t})$:

$$p(x_t|z_{1:t}) = \frac{p(z_t|x_t)p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})} \tag{1.2}$$

where $p(z_t|x_t)$ is the measurement model and $p(z_t|z_{1:t-1})$ is the normalization constant. Since the exact computation of $p(x_t|z_{1:t})$ is often not tractable, we approximate it with a nonlinear Bayesian filter based on Monte Carlo simulations. A set of particles $\chi_t = x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}$ represent the uncertainty of $p(x_t|z_{1:t})$. Each particle $x_t^{[m]}(1 \leq m \leq M)$ is a possible instantiation of state x at time t. The particles in χ_{t-1} are updated by sampling from state transition probability $p(x_t|x_{t-1})$ to obtain particle set $\bar{\chi}_t$. Then, for each particle in $\bar{\chi}_t$ an importance weight $w_t^{[m]}$ is assigned, where

$$w_t^{[m]} = p(z_t|x_t) \tag{1.3}$$

Finally, samples are drawn, with replacement, from $\bar{\chi}_t$ using importance weight $w_t^{[m]}$ to obtain χ_t. It turns out that we can infer $w_t^{[m]}$ directly from the performance of our neural network. When we calculate $p(z_t|x_t)$ we are actually calculating the True Positive Rate

(TPR). True Positive Rate (TPR) is defined as

$$TPR = \frac{TP}{TP + FN} \tag{1.4}$$

where TP means True Positive and FN means False Negative. In Table 1.1 we can see when a prediction instance is considered TP or FN. Ideally we want $TPR = w_t^{[m]} = 1$, such that when there is a true probe-candidate match, the neural network makes the correct prediction 100% of the time. Note that $TPR = 1$ can be achieved when the neural network predicts a match for any probe-candidate pair. To prevent this from happening we also want the False Positive Rate (FPR) to be as low as possible. False Positive Rate (FPR) is defined as

$$FPR = \frac{FP}{FP + TN} \tag{1.5}$$

where FP means False Positive and TN means True Negative. In Table 1.1 we can see when a prediction instance is considered FP or TN.

1.1.4 Neural Networks and Deep Learning

Artificial Neural Networks

A neural network is a set of units, called *neurons*, that are connected to each other by weighted edges. The weights represent the importance of the connections between the individual neurons. The neurons are arranged in layers. The most simple neural network is the single-layer perceptron [11] where there is only a single layer of neurons. This neural network is a function of its weights, input and bias and outputs either a 0 or 1 value:

$$f(\mathbf{x}, \mathbf{w}, b) = \begin{cases} 1, & \text{if } \sum_{i=1}^{m} w_i x_i + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1.6}$$

Here $\mathbf{x}$ represents the input data as a one-dimensional vector of length m, w_i is the weight corresponding to input x_i and b is the bias. Note that the perceptron is a linear classifier which means that it can only learn linear relationships between the input and output. The network trains on the data by making predictions and updating the weights if the prediction is wrong, until convergence is reached. The perceptron converges only if the data is linearly separable. The weights are updated using the perceptron learning rule:

$$w_i = w_i + \eta(t - o)x_i \tag{1.7}$$

where t is the target output, so the correct answer, o is the perceptron output and η is the learning rate, which determines the magnitude of the weight update.

When the network has an input layer, an output layer and at least one layer inbetween, we call it a multilayer perceptron (MLP). The inbetween layers are called hidden layers. An MLP has the advantage that it can learn non-linear relationships between the input and output due to the non-linear activation functions used when propagating information through the network: the output of each neuron is passed through a non-linear function before it is passed on to the next neuron. Without non-linear activation functions the network will behave just like a single-layer perceptron, no matter how many layers we add to it. Instead of using the perceptron rule to perform weight updates, the MLP uses backpropagation. Backpropagation computes the gradient of the loss function with respect to each weight. The gradient is then used in an optimization algortihm to update the weight such that the loss function is minimized. The weight update is performed as follows:

$$w_i = w_i - \eta \frac{\partial E(w_i, x_i, t_i)}{\partial w_i} \tag{1.8}$$

where loss E is a function of the weight w_i, the input x_i and the target output t_i. The loss function determines the penalty for the network making the wrong prediction.

Deep Learning

When we add multiple hidden layers to the network we can call the network a deep neural network. The term deep learning refers to the stacking of simple units to form a hierarchical structure. When all the neurons in one layer are connected to all the neurons in the next layer, we call that inbetween structure of connections (weights) a fully connected layer or an FC layer.

Siamese architecture

For the problem of person re-id we would like to compare extracted image features to learn about their similarity. A useful architecture for this problem is the Siamese architecture, where there are two identical feature extractors sharing the same weights. Siamese networks are used for the processing of pairs of data. Usually the extracted data feature is passed on to a distance learning algorithm that learns a metric defining the distance between the two extracted features.

Convolutional Neural Networks (CNN)

A convolutional layer is like a fully connected layer where each input gets processed by a convolutional operation before being passed on to the next layer. But instead of learning the weights of all the connections between the layers, a convolutional kernel is learned that is shared by all the connections between the two layers. Traditionally, convolutional layers are used to extract translation-invariant features from the input by using two-dimensional kernels. Pooling layers are often used in combination with convolutional layers to reduce the spatial dimensions of the input and to prevent overfitting. An often occurring combination is a convolutional layer followed by a pooling layer and then this pattern is repeated a couple of times until the spatial dimensions of the input have been reduced to one. This is how we can extract features from an image with a neural network. Usually this one-dimensional image feature is passed on to a series of FC layers that learn the high-level reasoning leading to the output. This setup is known as a convolutional neural network.

Three-Dimensional Convolutional Neural Networks (3DCNN)

A 3DCNN [12] is very similar to a CNN, with the addition of an extra dimension in the kernel: For example, instead of having a 3×3 kernel, which learns spatial features, we have a $3 \times 3 \times 3$ kernel, which learns temporal features in addition to spatial features.

1.2 Relevant Problems in Person Re-Identification

In this section we will look at some open problems in person re-identification and how they have been confronted in the literature. These problems are relevant to increasing the performance measures for the use-case.

1.2.1 Euclidean Distance Function

Many deep learning based person re-identification methods use a convolutional neural network (CNN) to extract features from the images. Most deep learning based person re-identification methods use a distance function to compute the similarity between the extracted probe-candidate pair features. A generalization of frequently used architectures can be seen in Figure 1.3. The most popular distance function used in current person re-identification literature [13, 14, 15, 8, 16, 6, 17, 9, 18] is the Euclidean distance function:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{1.9}$$

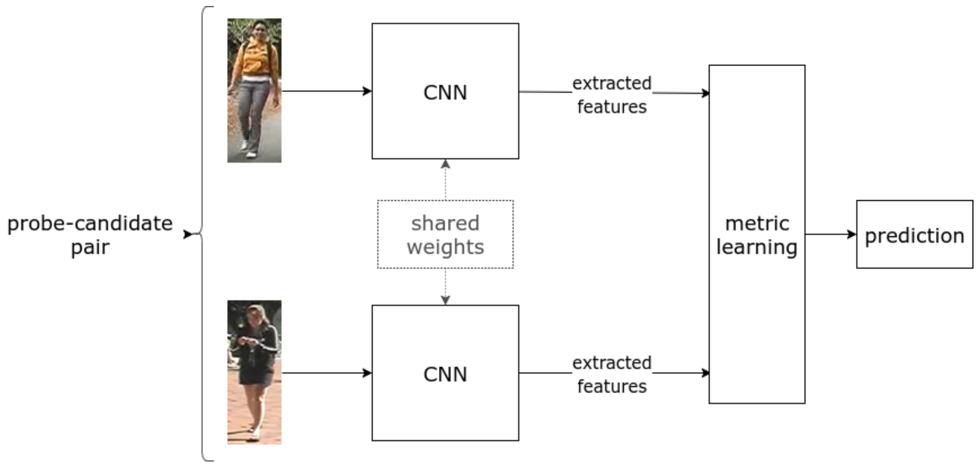


Figure 1.3: A frequently used architecture setup in deep learning for person re-identification literature. A siamese architecture is used, where weight-sharing CNNs extract features from a pair of images. The features are then fed to a distance function to compute the distance between the two images.

where $\mathbf{p}$ and $\mathbf{q}$ are the extracted probe and candidate feature vectors and $d(\mathbf{p}, \mathbf{q}) \geq 0$. Computing the distance between two large feature vectors reduces the detail of the image features to a single number. On one hand this can be desirable since it reduces the dimensionality of the data, making it less expensive to perform consecutive computations. But on the other hand this dimension reduction is undesirable because we essentially throw away important discriminative information. The Euclidean distance is also sensitive to scale and disregards correlations between features across dimensions [19]. When the Euclidean distance is used we need to set a threshold for when to classify a probe-candidate pair as matching or not matching. This threshold has to be set before training the network and can be considered a hyperparameter of the network. Before training, it is very difficult to know what kind of values the model will output as distance since Equation 1.9 only guarantees a value greater than zero, making it almost impossible to guess an appropriate value for the threshold.

Another approach is to preserve as much information as possible that is contained in the extracted features. For example by taking the element-wise subtraction between the two features or to concatenate the features. In a recent work the two feature vectors are fused by element-wise subtraction and then passed on to a fully connected (FC) layer for further processing [20]. In the results they show that state-of-the-art results can be obtained while avoiding the use of any distance function, simplifying the network and using the extracted information to a fuller extent. In another paper the element-wise difference between the two feature vectors is computed and then passed on to another convolutional layer followed by a set of FC layers [21]. In [22] features are extracted with a CNN and then concatenated with hand-crafted features and then passed on to a FC layer. [23] argue that while the Euclidean and cosine distances are simple to use, they have lower discriminative ability to measure the similarity between a CNN learning feature pair. Instead [23] uses a combination of element-wise absolute value and multiplication to perform feature fusion while achieving competitive results on several benchmarks. The use of FC layers after feature extraction suggest a more natural approach to person re-identification, where instead of imposing a constraint in the form of a distance function, the network can learn its own representation of a distance function. In this thesis we will adopt the method of using the full feature vectors and experiment with various element-wise feature operations, such as taking the (absolute) difference, addition and multiplication. In addition we will also perform experiments with the concatenation of the two feature vectors.

1.2.2 Small Datasets

We have many re-id datasets but most of them are quite small and none of these datasets even approach the size of image recognition benchmark datasets such as ImageNet. Ideally we also want to acquire a dataset from the environment in which we will use the trained network. However this dataset is likely to also be very small because of the time-consuming process of collecting and matching pairs of person images in a camera network. Since training deep neural networks require as much data as possible, the fact that no adequately big dataset of the desired environment available is an issue. Current literature deals with this issue by pre-training on existing datasets before training on the actual target dataset using [10, 20]. Others have attempted to augment existing datasets with exotic methods where the person is cut out of the original image and pasted on another background [24]. And another popular tactic is to extract as many features as possible from the available data by combining hand-crafted features with CNN features [22] or by learning image attributes [14]. Results obtained by [10] imply that by jointly training on multiple datasets, a CNN can learn a generic representation of the person re-identification classification task.

Batch normalization [25] is often used to increase performance in neural networks because it speeds up learning by reducing internal covariate shift and in addition provides weight regularization. Batch normalization involves taking the mean of a mini-batch:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \tag{1.10}$$

and the mini-batch varience:

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \tag{1.11}$$

for the normalization:

$$\hat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{1.12}$$

and then we scale and shift with the use of learnable parameters γ and β:

$$y_i \leftarrow \gamma \hat{x_i} + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \tag{1.13}$$

Consider the case where we train on multiple mixed datasets, each individual dataset having a different mean and variance. We visualize the mean image of some of the datasets that we use in 1.4. The mean images are visually not similar. When we train on a dataset with a mixture distribution, the mini-batch mean $\mu_{\mathcal{B}}$ will be representing the average of a subset of the mixture distribution, and not the average of a subset of the target distribution. The same will be the case for the mini-batch variance $\sigma_{\mathcal{B}}^2$. It is unknown what kind of effect this will have on network training and generalization. In this thesis we will investigate

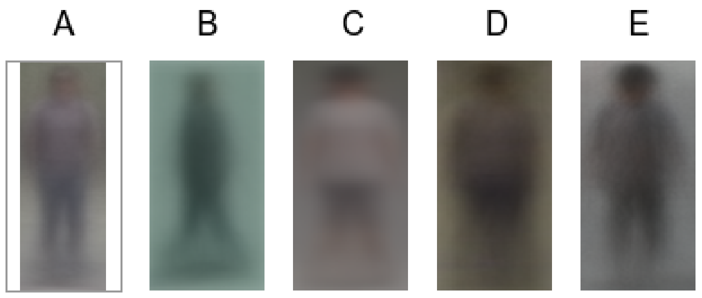
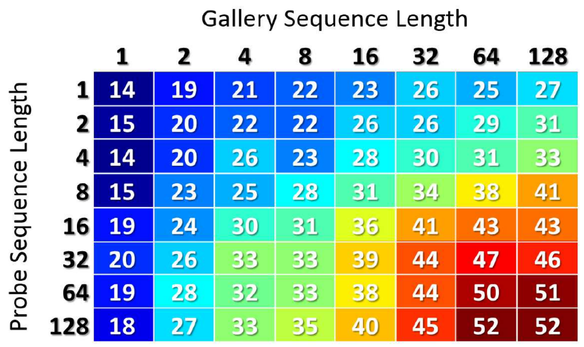


Figure 1.4: Dataset averages. (A) VIPeR (B) PRID450 (C) MARKET1501 (D) GRID (E) CUHK02. Best viewed in color.

various forms of mixing datasets together in various orders and mixing recipes. And we will find out how batch normalization behaves on a mixture distribution.



Gallery Sequence Length

Probe Sequence Length	1	2	4	8	16	32	64	128
1	14	19	21	22	23	26	25	27
2	15	20	22	22	26	26	29	31
4	14	20	26	23	28	30	31	33
8	15	23	25	28	31	34	38	41
16	19	24	30	31	36	41	43	43
32	20	26	33	33	39	44	47	46
64	19	28	32	33	38	44	50	51
128	18	27	33	35	40	45	52	52

Figure 1.5: iLIDS-VID rank 1 CMC re-identification accuracy as the lengths of the probe and gallery sequences are varied. Figure and caption taken from [9].

1.2.3 Video vs. Image Person Re-Identification

While deep learning based image person re-identification has been studied extensively, deep learning based video person re-identification has not received the same attention. There are some papers that handle video for use in person re-identification using deep learning [26, 9, 7, 27, 28, 29, 30], most of them very recent. The trend in video based re-identification is to also use a distance metric [26, 9, 27, 28, 30]. All of them use a recurrent neural network to extract the features based in time. It is known that to have good performance with recurrent neural network a lot of data is needed. However in the case with person re-identification often we only have very small datasets. Instead of a recurrent neural network, we will use 3D convolutions to learn weight kernels, not only in space but also through time [12]. Another problem that has not received attention is how image based re-identification compares to video based re-identification. It is assumed that because video holds more information this is better for telling people apart. But it has not been objectively tested. [9] implicitly tested this, by varying the number of images in a sequence, the results of which are shown in Figure 1.5. A sequence with length 1 is just an image, therefore [9] tested the effectiveness of images against video. However, the neural network used was one trained on sequences of 16 images, resulting in a model that relies on at least 16 images to achieve a good performance. The underperformance of sequences of length 1 can be attributed to the network architecture used and the resulting model. In this thesis we will build on this work by comparing the data types in a more appropriate way, by comparing a 2D CNN with a 3D CNN, trained on images and videos respectively. Both image and video data will come from the same video source and will have a similar distribution.

1.3 Research Questions

The end goal of this thesis is to develop a deep neural network that achieves suitable performance for use in the particle filter. Aside from that we research several existing problems, which the solutions of may lead to increased performance in our network. We can identify the following questions that need answering:

1. What is the effect of replacing the Euclidean distance with a neural network for metric learning?
2. How can we best make use of all available datasets?
   (a) What are the effects of different training schemes on the performance?
   (b) How effective is pre-training on auxiliary person re-identification datasets?
   (c) How well does a learned model generalize to a different environment?
3. How can we compare the effectiveness of the use of image data versus video data?
   (a) How can we modify the image network minimally to work on video data?
   (b) Is video data more informative than image data?

Chapter 2

Methods

To answer the questions posed in Section 1.3 we perform a series of experiments by making modifications to the neural network and observing the effects of the modification. Since the main goal is to use the developed network in the particle filter system described in Section 1.1.3 the aim is to optimize the TPR and the FPR. First the formulation of the person re-identification problem will be explained. Then the overall architecture of the model that we will use and the choice of hyperparameters are described. We use the same base siamese neural network in each experiment, with variations in some of the components. Finally the choice of datasets is clarified and a description of each dataset used is given.

2.1 Problem Formulation

The particle filter has to receive input from the neural network that says whether there was a specific person identified in the imagery obtained at a specific location. The formulation that fits this problem is that of classification: does the observed person match the probe or not? The network receives a pair of images as input and outputs whether the images match or not.

2.2 Models

2.2.1 Siamese Convolutional Neural Network (SCNN)

This network is used to conduct experiments with pairs of images. Like almost all of the deep learning models in person re-identification literature, we use the siamese architecture for our network because it can accept probe-candidate pairs of images as input. To extract the features from the images we use a series of convolutional layers, followed by max pooling, then passed through an activation function and normalized using batch normalization. After that the extracted pair of feature vectors gets merged to one feature vector and processed by the similarity learning component. The individual images are quite small, so we don't need many convolutional layers to extract the features. The CNN branches are based on AlexNet and then through iterative experiments we arrived at our specific composition as illustrated in Figure 2.1.

The raw input images are first resized to 128×64 pixels. If the image is smaller than 128×64 pixels then zero padding is added to fill the gaps. If the image is larger than 128×64 pixels then the image is re-scaled. Since the width-height ratio in most person re-identification is similar, re-scaling the larger images will not lead to strange body proportions. The images are passed through a batch normalization layer to make sure that it is normalized to have 0 mean and a standard deviation of 1. We use the Exponential Linear Unit (ELU) [31] as our nonlinearity because we found that empirically it outperforms the more commonly used Rectified Linear Unit (ReLU). We now refer to the components described in Figure 2.1. A *convolutional unit* consists of a convolutional layer with a stride of 1. Since the images are small, a small 3×3 kernel is used to capture more detail. This is followed by a max pooling operation with a 2×2 kernel, followed by the ELU and finally batch normalization is applied to reduce internal covariate shift as the features are propagated through the branch. Notice that in the first convolutional unit a max pooling operation of 4×2 is used. This was done to have a one-dimensional feature vector as the output of convolutional unit 6. The two branches of the siamese network share the same weights. Each branch outputs a one-dimensional 512 element feature vector. These are merged together using an element-wise subtraction and then by taking the absolute value per element, resulting in the absolute difference feature vector. This vector is then passed on to the *similarity learning* block. The similarity learning block consists of a set of FC layers combined with a dropout of 0.5 between the layers to provide regularization. At the end we have an output layer consisting of two one-hot-encoded outputs: $[0, 1]$ for match and $[1, 0]$ for a mismatch. The predictions are passed through the softmax function to normalize the output: Prediction $P = [p_1, p_2]$ and if $p_2 > 0.5$ then the network predicts that the pair is a match.

2.2.2 Euclidean Siamese Convolutional Neural Network

In Figure 2.2 we can see the architecture of a our network when we use the Euclidean distance. We can see that the branches of our network are the same as the branches in the SCNN in Figure 2.1. The difference lies in the components after feature extraction has

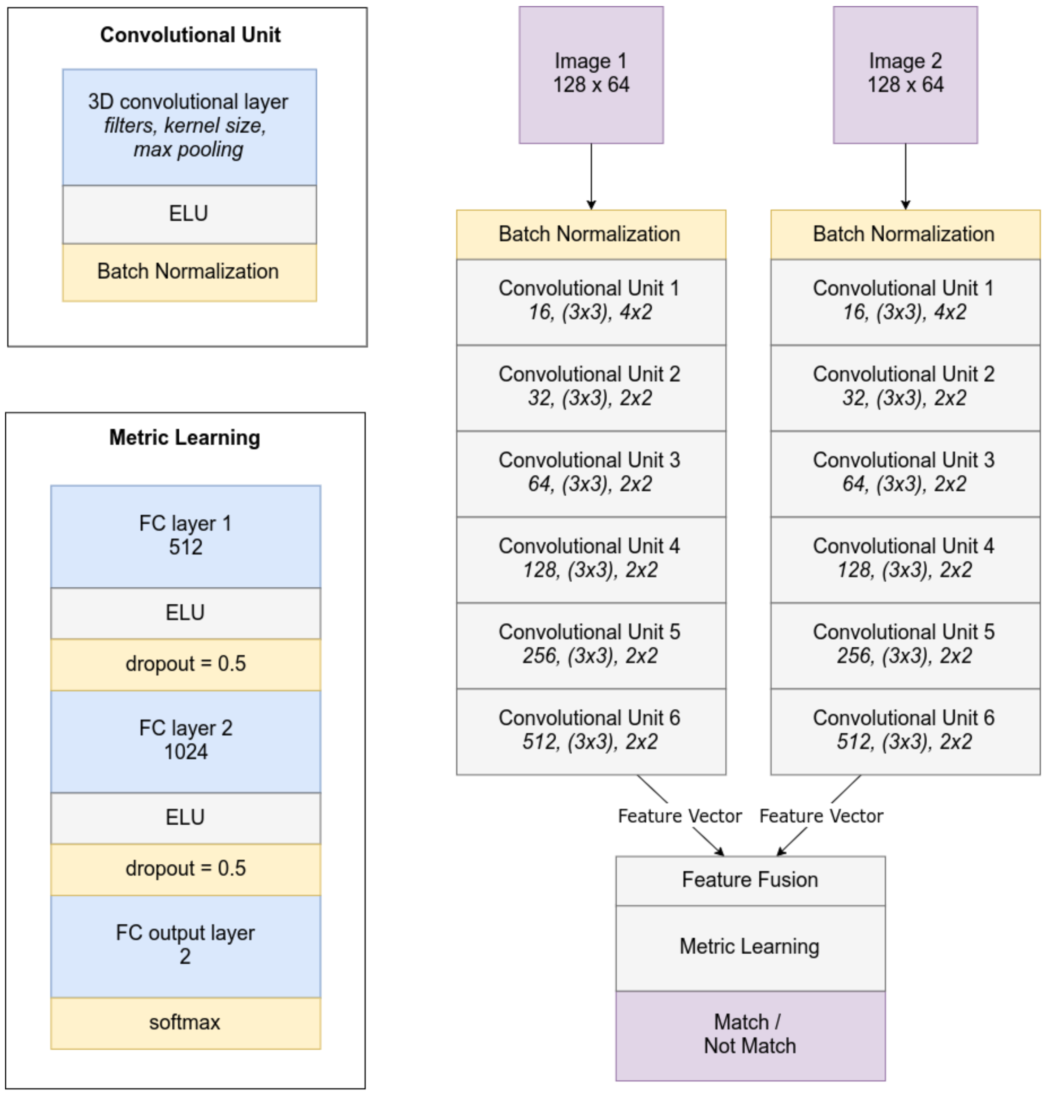


Figure 2.1: The Siamese Convolutional Neural Network architecture.

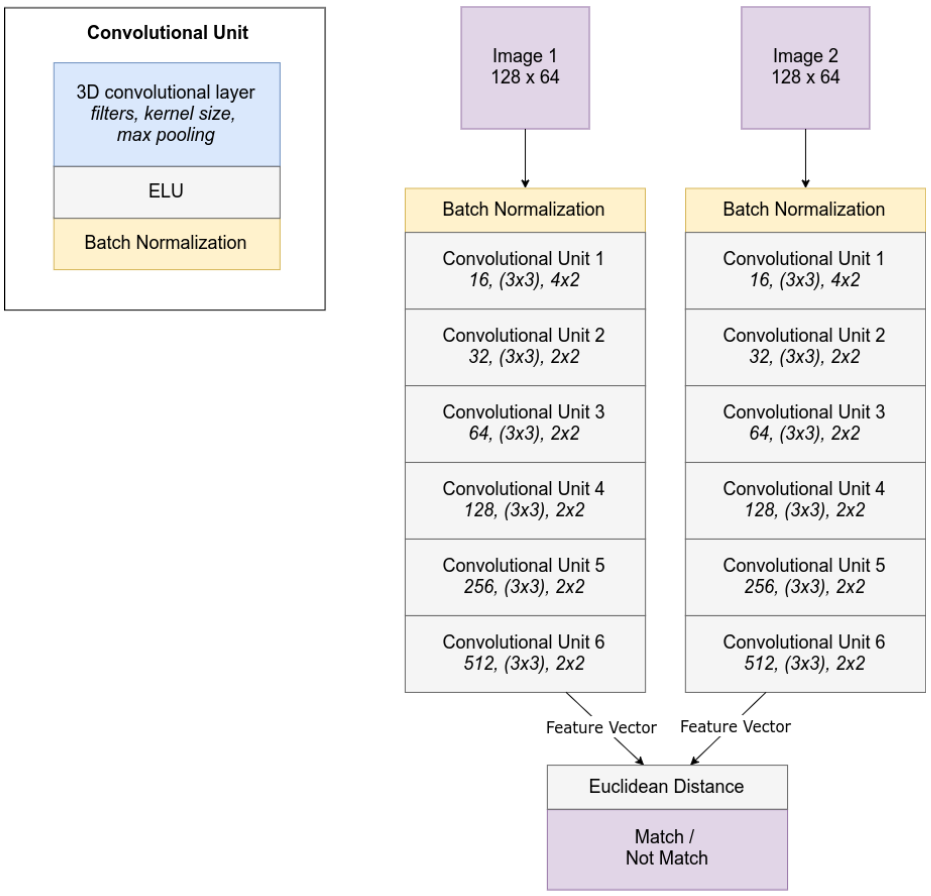


Figure 2.2: The Euclidean Siamese Convolutional Neural Network.

taken place. The extracted features are passed directly to the Euclidean distance for metric computation.

2.2.3 Siamese 3D Convolutional Neural Network (S3DCNN)

This network is used to perform experiments with pairs of image sequences (video pairs). In Figure 2.3 the architecture can be viewed. We can see immediately that the network architecture bears a striking resemblance to the network architecture in Figure 2.1. The only discrepancy is the use of 3D convolutions instead of 2D convolutions. The goal that we wanted to achieve with this network is to see if we can develop a network as similar as possible to the network in Section 2.2.1 in order to objectively assess whether or not images work better than videos.

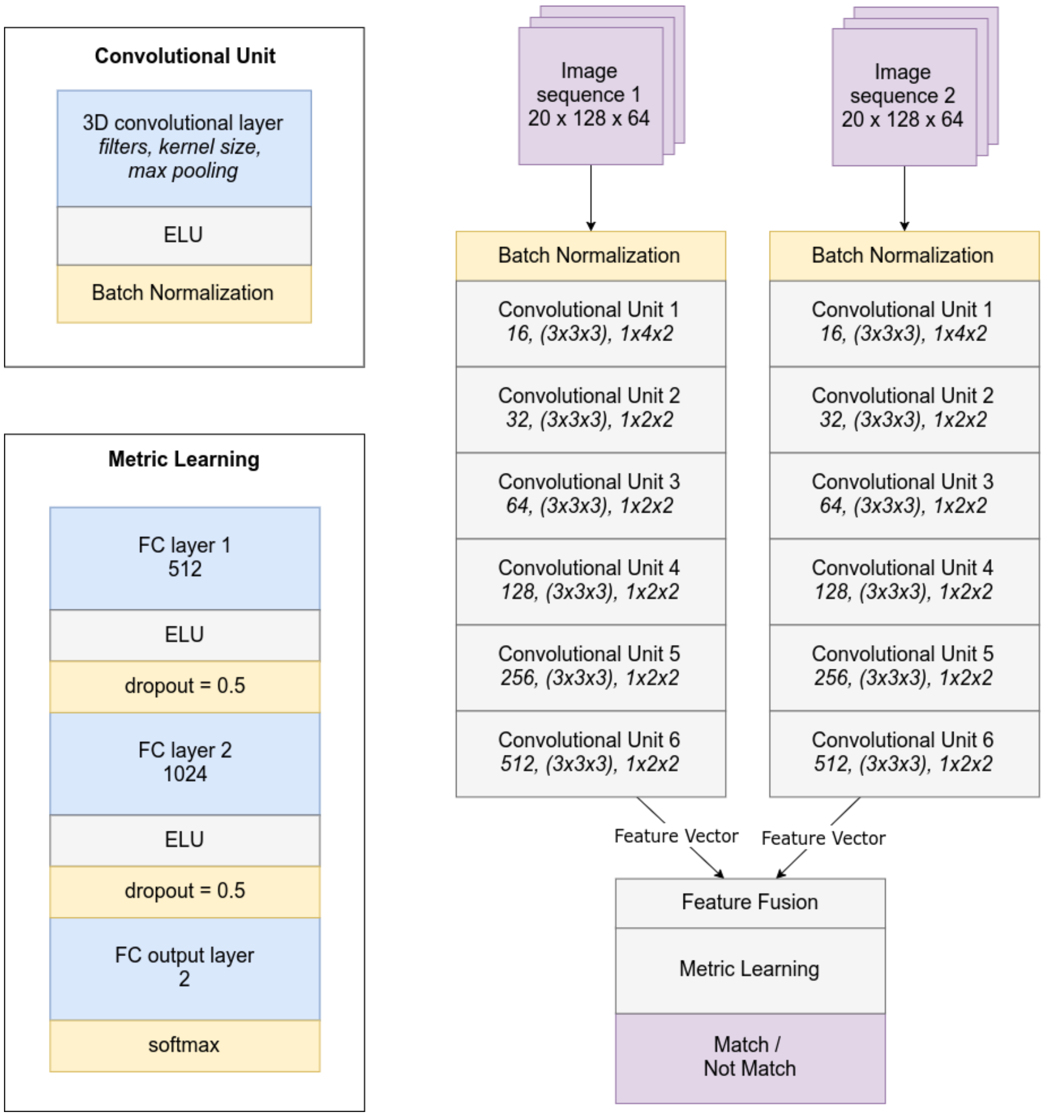


Figure 2.3: The Siamese 3D-Convolutional Neural Network architecture.

2.3 Training

Our training set consists of labeled image pairs, indicating whether a pair of images belong to the same identity. A label of $[0, 1]$ indicating that the given pair is a match, or a $[1, 0]$ indicating that the given pair is a mismatch. By doing this we cast the re-identification problem as a classification problem consisting of two classes: *match* or *mismatch*. All networks are trained for 100 epochs.

Neural Metric Learner

We train the network using a Cyclical Learning Rate (CLR) [32]. We set the lower boundary at 0.00001 and the upper boundary at 0.005 and use the triangular policy. These settings gave the best performance empirically. Most DL papers train using a learning rate that decreases as the network learns (decay), while CLR cycles periodically through the lower and upper boundaries, as the name implies it. This speeds up training time, requiring less steps to reach convergence. We found that it also increases the generalization capabilities of the network in comparison with using a learning rate with decay. For the loss function we minimize the categorical cross-entropy. We optimize the model using Nadam with a decay of 0.95 and we set a batch size of 32. We use dropout with a dropout rate of 0.5 to provide for regularization.

Euclidean Distance

When we use the Euclidean as distance learner we do not use CLR because we found that it destabilizes the learning process and the network does not converge. Instead we use a learning rate of 0.00001. The contrastive loss is used as in [33], which was developed for use with simple distance functions such as the Euclidean distance. Empirically we found that when we use the categorical cross-entropy loss the network does not converge when using the Euclidean distance. When optimizing the model we use Nadam with a decay of 0.95.

2.4 Datasets

We select a variety of datasets to assess the performance of our network across several benchmarks. The datasets chosen are commonly used in literature and are quite clean in terms that there are no occlusions and all boundary boxes are assigned by humans.

2.4.1 Image Datasets

VIPeR

This dataset consists of 632 IDs where each ID is present twice in the dataset, resulting in 1264 images in total [3]. The images are taken from various angles under varying lighting conditions in an academic setting. Each image has been extracted from a full frame image and scaled to 128×48 pixels. Before the images are fed to our network we pad them with zeros to a size of 128×64 pixels.

CUHK02

This dataset consists of 1816 IDs where each ID is present four times in the dataset, resulting in 7264 images in total [34]. The images are taken from 10 different cameras on a university campus. Each image has been extracted from a full frame image and scaled to 160×60 pixels. Before the images are fed to our network we scale them down and pad the width with zeros to a size of 128×64 pixels.

Market-1501

This dataset consists of 1501 IDs where each ID is present at least two times in the dataset, resulting in 25259 images in total [35]. The images are taken from six different cameras near a supermarket. Each image has been extracted from a full frame image and scaled to 160×60 pixels. Before the images are fed to our network we scale them down and pad the width with zeros to a size of 128×64 pixels.

QMUL-GRID

This dataset consists of 250 IDs where each ID is present twice in the dataset, resulting in 500 images in total [36]. The images are taken from 8 different cameras in an underground sation. Each image has been extracted from a full frame image and scaled to 291×106 pixels. Before the images are fed to our network we scale them down to a size of 128×64 pixels.

PRID450

This dataset consists of 450 IDs where each ID is present twice in the dataset, resulting in 900 images in total [37]. The images are taken from two different cameras near a crosswalk. Each image has been extracted from a full frame image and scaled to 155×90 pixels. Before the images are fed to our network we scale them down to a size of 128×64 pixels. This dataset is based on the PRID2011 dataset, that we will mention next.

2.4.2 Video Datasets

PRID2011

PRID2011 dataset was captured in the same scenario as PRID450 from the same camera network consisting of two cameras [38]. However PRID2011 consists of the entire sequences of images and not just single-shot images. Camera view A shows 385 IDs and camera view B shows 749 IDs. In total there are 200 IDs that appear on both cameras. Each sequence is between 5 and 675 images long. For each image in each sequence we scale the image to 128×64 pixels. Instead of following the protocol that is recommended for training and testing, which is to use the sequences as is, we decided to only select the IDs that appear on both cameras and have a sequence length of at least 20 images. In case that the ID does not appear on both cameras but it does have a sequence length of 40 images or more, these IDs are also selected. The rest of the IDs are discarded. For each ID we split the sequences in non-overlapping subsequences of 20 images each. With this technique we now have 615 unique IDs instead of 200. We choose a sequence length of 20 images because that is approximately the number of frame it takes on average for an ID to make two steps. We assume that gait information is encoded in the sequences and that two steps should provide sufficiently discriminative information. As another result of this technique we also have more sequences in total than before, because we decreased the length of the individual sequences. We think that a length of 20 and the increase in unique identities is a good compromise, since [9] shows in Figure 1.5 that their network gave a rank-1 of about 40 given sequences of length 20, compared to a rank-1 of 52 with sequences of length 128.

iLIDS-VID

The iLIDS-VID dataset [27] was captured in an airport arrival hall from two non-overlapping camera views. In total there are 300 unique IDs that are present in both camera views. Each sequence is 23 to 192 frames long. The images are 128 by 64 pixels so no resizing is necessary. To be consistent with with the choices made in the PRID2011 dataset we also split each sequence in several non-overlapping parts of 20 images per sequence. What we lose in sequence length we gain back in the number of samples available from each sequence. This means that we now have more positive matches to train with, increasing the total size of the training set. Just like it was the case with PRID2011, we now consider both views from both cameras. In other words, each sequence in a sequence pair can belong to the same view or a different view. Say we have cameras A and B and we denote a sequence pair with the camera that they belong to. For example, the pair (A, A) means that both probe and candidate are coming from camera A. In the original procedure we only have (A, B) whereas in our modified version of the procedure the network has to

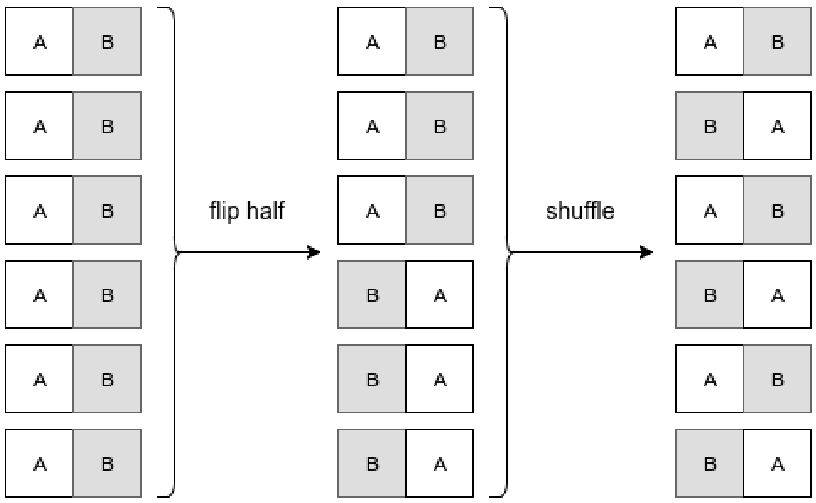


Figure 2.4: Sideways shuffle

learn from instances that could belong to: (A, B), (B, A), (A, A) and (B, B). This makes the re-identification task more difficult, but ultimately makes for a more general model.

2.4.3 Benchmarking vs. Real World Use

The recommended protocol with all of the datasets when doing benchmarking is to use the datasets as is. This means that, assuming we have cameras A and B, that the probe is always from camera A and the candidates are from camera B. So to get a good performance on the benchmark we would have to train always in pairs of (A, B). However this also means that the network will learn a view-specific representation, which is not useful if we want to transfer the learned model to another camera in another view. For benchmarking this is acceptable because we are not dealing with a real world scenario. But in real life, such as in our particle filter system, such view specific models are inconvenient. The cameras used in the custom real world scenario will be different than the ones used for capturing the dataset and the angles will be different. This is why we perform a technique that we call *sideways shuffling* on the training data, such that the network will not learn a view specific representation. Say we have a dataset with two camera views A and B. Instead of passing pair (A, B) to the camera each time we pass (A, B) half of the time and (B, A) the other half of the time. This is illustrated in Figure 2.4.

2.4.4 The Class Imbalance Problem

A natural class imbalance arises in the data because we work with pairs of images to train the model. This results in a much higher number of negative (mismatching) image pairs

Table 2.1: The increase of the class imbalance when the number of IDs increase.

cameras = 2				
IDs	**total pairs**	**positive pairs**	**negative pairs**	$r_{p,n}$ **%**
10	190	10	180	5.56
50	4950	50	4900	1.02
100	19900	100	19800	0.51
500	499500	500	499000	0.10
1000	1999000	1000	1998000	0.05

compared to the number of positive (matching) image pairs. When we make pairs of images we get the following:

$$p = \frac{(IC)^2 - IC}{2} \tag{2.1}$$

$$p_{pos} = I\frac{C^2 - C}{2} \tag{2.2}$$

$$p_{neg} = p - p_{pos} \tag{2.3}$$

where p is the total number of pairs possible, C is the number of cameras in a network, I is the number of IDs in the dataset visible on all cameras, p_{pos} is the number of positive pairs possible and p_{neg} is the number of negative pairs possible. When we compare the ratio between positive and negative pairs we can see clearly that there is a huge imbalance, see Table 2.1 which was computed from Equations 2.1 - 2.4.

$$r_{p,n} = \frac{C - 1}{C(I - 1)} \tag{2.4}$$

The common way to tackle this problem is to use data augmentation techniques to augment the number of positive pairs, such as mirroring, rotating and zooming. However we wanted to see how well the datasets would perform on their own, without using augmentation so we did not use data augmentation.

To overcome this imbalance we perform undersampling in the negative class: For each training epoch, the set of negative pairs is shuffled and a random subset is sampled from the set of negative image pairs. This sampled subset has the same number of pairs as the number of pairs in the set of positive image pairs, making the final training set equally balanced for both classes.

Note that balancing the data means that we ignore the prior belief that the occurrence of a match is a rare event. Instead, the resulting model assumes that a match occurs with equal frequency as a mismatch. If we were using the neural network as a standalone detector, then it would make sense to not balance the data 50-50, since in the limit, we would obtain a sensor model $P(z_t|x_t)$ imposed with the prior belief that an occurrence of a match is rare, optimal for the real world use case. However, the neural network is used as part of a larger estimation process, where the priors of a specific person being present at the sensor is computed. $P(x_t|z_{1:t-1})$ can be viewed as the prior belief that the neural network has detected a match at the location of the sensor at time t. Since the prior is explicitly computed, the likelihood given by the sensor model $P(z_t|x_t)$ should assume that a match occurs with equal frequency as a mismatch.

Chapter 3

Experiments

In this chapter we will explain the setup of the various experiments performed to find answers to the research questions presented in Section 1.3. Each experiment section will address each research question respectively.

3.1 Experiments 1: Feature fusion

The first research question states: What is the effect of replacing the Euclidean distance with a neural network for metric learning? To answer this question we run experiments with the SCNN by replacing the similarity learning component with the Euclidean distance and we compare them to the following feature fusion methods: concatenation, addition, subtraction, multiplication and absolute difference. All the fusion methods with the exception of concatenation happen element-wise. For concatenation the extracted feature vectors get concatenated, and a new feature vector with twice the length is formed. In the feature fusion methods fusion is performed and then passed on to a set of FC layers as depicted in Figure 2.1. We use three small person re-identification datasets for this experiment: VIPeR, GRID and PRID450. We measure the results on the TPR and FPR performance measures. The goal is to maximize the TPR and minimize the FPR. The perfect result would be to have a TPR of 1 and a FPR of 0. In Table 3.1 we can see the results of this experiment. The Euclidean distance gave a low FPR across most datasets, which is desirable. The Euclidean distance resulted in an TPR over 0.55 on most datasets, however on the GRID dataset it gave very poor performance on the TPR with 0.15. Considering that GRID is the noisiest dataset of the three this is not surprising. It is interesting to see that as the datasets get larger the Euclidean distance performs better. The results for the fusion methods are unclear. In terms of FPR, the absolute fusion method outperforms the rest of

Table 3.1: Experiments 1 results: Feature Fusion

feature fusion	**TPR**					**FPR**				
	VIPeR	GRID	PRID 450	CUHK 02	Market 1501	VIPeR	GRID	PRID 450	CUHK 02	Market 1501
Euclidean	0.61	0.15	0.55	0.77	0.78	0.15	0.04	0.09	0.03	0.03
concatenate	0.54	0.70	0.61	0.76	0.80	0.21	0.18	0.20	0.07	0.05
add	0.51	0.52	0.79	0.78	0.78	0.22	0.25	0.36	0.11	0.07
subtract	0.47	0.55	0.45	0.78	0.75	0.14	0.10	0.10	0.06	0.04
multiply	0.26	0.22	0.28	0.60	0.66	0.07	0.04	0.06	0.03	0.03
absolute	0.19	0.10	0.16	0.71	0.67	0.04	0.01	0.02	0.05	0.03

the fusion methods, but scores low on TPR on the smaller datasets. In terms of TPR the concatenate method outperforms the rest of the fusion methods, but scores relatively high on the FPR for the smaller datasets. However both of these methods have a reasonable TPR-FPR balance on the larger datasets. In fact, all the fusion methods have a reasonable TPR-FPR balance when the datasets are large. This suggests that for larger datasets it is not so important what is used for metric learning. But for small datasets the choice of metric learner is crucial.

3.2 Experiments 2: Strategies for training

The overall second research question states: How can we make use of all the available datasets? We break down this question in subquestions to properly answer this overall question. We identified three subquestions. For each subquestion we run a series of experiments. The aim of these experiments is to find out if there is a configuration that improves the performance of the base network, which is trained on a single dataset. In these experiments the distiction is made between the auxiliary datasets and the target dataset. Auxiliary datasets are labeled person re-identification datasets that are available but that will not be used to test the network on. As the name suggests, these datasets serve to provide extra data to train the network with. The target dataset is the dataset on which we actually want to test the performance of the network on. This dataset will be split into a training subset and a testing subset. In the following experiments we look at the effects of data mixing when using the absolute and concatenate fusion methods because, depending on which performance measure we look at, these two fusion methods performed best. We do not include the Euclidean distance in the experiments because several components are different from a neural network using a neural metric learner: the network architecture is different, as indicated in Section 2.2.2, the loss is different, the optimizer used is different and CLR is not used, as indicated in Section 2.3. This makes it hard to determine what the cause in performance difference could be. Furthermore, we believe that batch normalization will cause a decrease in performance when mixing different data distributions, as mentioned in Section 1.2.2, and we will repeat the experiments without batch normalization and compare to another normalization method, in order to be able to assess whether or not batch normalization is causing a decrease in performance. For Experiments 2.1.1, 2.1.2, 2.2.1 and 2.2.2 we use the VIPeR, GRID and PRID450 datasets because we want to emphasize the effects of mixing together small datasets. Previous empirical results indicated that mixing with larger datasets tends to give the same results as training only on the large dataset, which is logical since a dataset, consisting of a large dataset with several smaller datasets, is comprised of at least 70% the larger dataset (70% for Market1501 and 74% for CUHK02). In Experiments 2.2.3 experiment with CUHK02, Market1501 and GRID because these datasets have a similar appearance as assessed by the dataset mean in Figure 1.4.

3.2.1 Experiments 2.1: Training on Auxiliary Datasets

Experiment 2.1.1: Randomly mixing multiple datasets

In the first experiment we consider three datasets VIPeR, GRID and PRID450. There are three experiments in total, each experiment targeting each dataset in the considered set respectively. The other two datasets that are not the target dataset are considered auxiliary datasets. The training subset belonging to the target dataset is mixed in randomly with the auxiliary datasets and at the end of the training phase the network is tested for its performance on the test subset of the target dataset. For example, let us consider the case where the PRID450 dataset is our target dataset P_T, making VIPeR and GRID the auxiliary datasets V_A, G_A respectively. We split PRID450 into a training subset P_{train} and a test subset P_{test}. The total training set becomes $Train = shuffle(V_A + G_A + P_{train})$. This experiment is visualized in Figure 3.1. Because we believe that batch normalization might cause a problem, as explained in Section 1.2.2, we also run the three experiments without batch normalization and compare the results. Also, the experiments are run with the use of SELUs to replace the normalization that was previously provided by batch normalization, so that we can compare to batch normalization.

The results of these experiments can be viewed in Table 3.2. First of all, when using the absolute feature fusion, we can see that batch normalization did not lead to worse performance as we expected (except for the GRID dataset) when compared to not using any batch

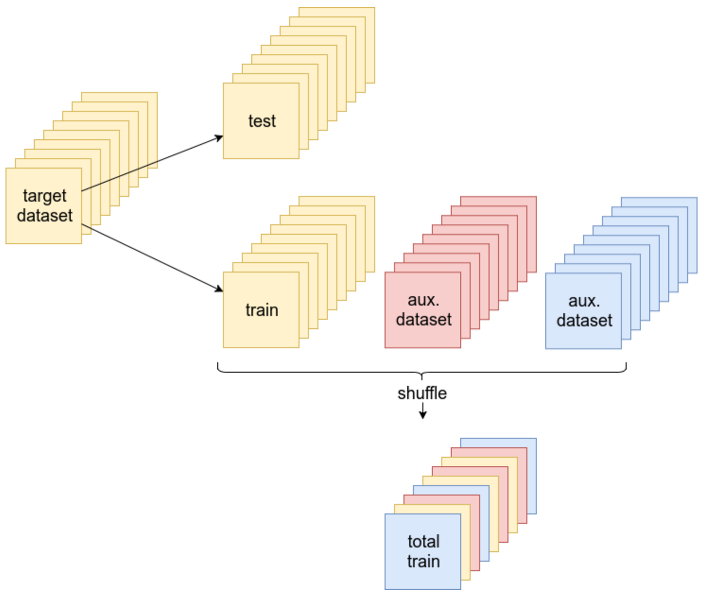


Figure 3.1: Experiment 2.1.1: Randomly mixing multiple datasets

normalization. In fact the performance when using batch normalization is slightly better than the use of no normalization method and slightly better than the use of SELUs. This means that batch normalizing a network that is trained on data from multiple networks does not affect performance in a negative way. However, batch normalizing a network trained on multiple datasets does not improve the performance of the network by much as compared to batch normalizing a network that is trained on a single dataset. When concatenation is used as feature fusion method, we see that the TPR for each dataset is higher compared to the absolute feature fusion. However this comes at a cost as the FPR is also increased. Now we look at the results of this experiment compared to the results in Experiment 1, when only a single dataset is used for training. This comparison is given in Table 3.3. We can see that there is TPR increase across datasets with both fusion methods. But the FPR with absolute fusion is increased, while the FPR with concatenation is decreased. Since increase in FPR is worse for performance than decrease in TPR we can say that, overall, (1) there are small performance gains with randomly mixing datasets + using concatenation and (2) random mixing is not beneficial to the absolute fusion method.

Experiment 2.1.2: Semi-randomly mixed datasets

In this experiment we consider the datasets mentioned in experiment 2.1.1. In this case there are also three experiments in which we target each dataset in the considered set respectively. The two other datasets that are considered auxiliary datasets are randomly

Table 3.2: Experiment 2.1.1 results: Randomly mixing multiple datasets

feature fusion	**normalization**	**TPR**			**FPR**		
		VIPeR	GRID	PRID450	VIPeR	GRID	PRID450
absolute	nothing	0.27	0.48	0.41	0.04	0.05	0.06
absolute	batchnorm	0.34	0.59	0.48	0.05	0.09	0.06
absolute	selu+AD	0.28	0.49	0.49	0.03	0.06	0.07
concatenate	batchnorm	0.52	0.75	0.70	0.16	0.17	0.18

Table 3.3: Experiment 1 vs. 2.1.1 results comparison.

feature fusion	**mix method**	**TPR**			**FPR**		
		VIPeR	GRID	PRID450	VIPeR	GRID	PRID450
absolute	single	0.19	0.10	0.16	0.04	0.01	0.02
absolute	multiple	0.34	0.59	0.48	0.05	0.09	0.06
concatenate	single	0.54	0.70	0.61	0.21	0.18	0.20
concatenate	multiple	0.52	0.75	0.70	0.16	0.17	0.18

mixed. Instead of mixing in the training subset of the target dataset, we append it to the end of the mixed auxiliary datasets. This means that the network is trained on a mix of the auxiliary datasets first and then on the training subset of the target dataset, all in one epoch. This repeats for all the following epochs. For example, consider PRID450 as the target dataset P_T consisting of training subset P_{train} and test subset P_{test}. Consider also the auxiliary datasets VIPeR V_A and GRID G_A. The total training set per epoch will be $Train = shuffle(V_A + G_A) + P_{train}$. A visualization of this procedure can be seen in Figure 3.2. As was the case in experiment 2.1.1, in this setting we will also repeat the experiments comparing the use of batch normalization to not using batch normalization and to using SELU and AlphaDropout for providing normalization. In addition we will also repeat the experiment when using concatenation as feature fusion method.

The results can be seen in Table 3.4. Again we see that batch normalization did not negatively impact the results compared to the results not using batch normalization. On the PRID450 batch normalization did score 0.02 lower on TPR compared to when SELU + AD was used. However the FPR was also 0.02 lower when using batch normalization compared to using SELU + AD. We can also say again that batch normalizing a network trained on multiple datasets does not improve the performance of the network by much as compared to batch normalizing a network that is trained on a single dataset. When comparing concatenate to absolute we can see that the TPR is increased by 0.16 to 0.25 and the FPR is increased by 0.07 to 0.09. In Table 3.5 we compare the results from this experi-

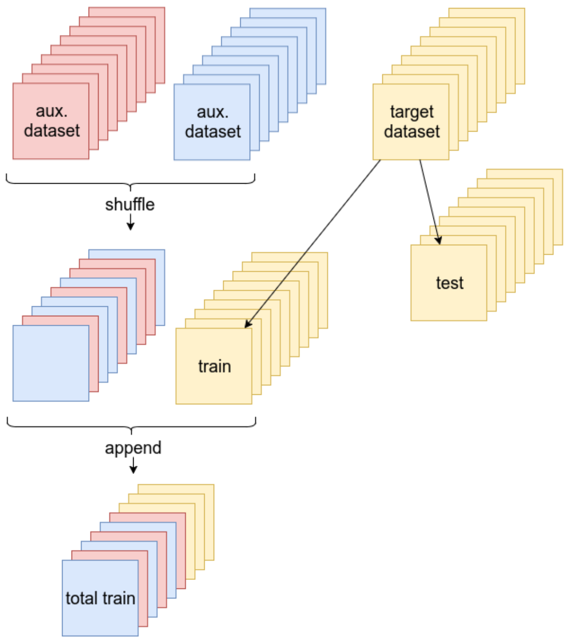


Figure 3.2: Experiment 2.1.2: Semi-randomly mixed datasets

ment to the results of experiment 1. As was the case in Table 3.3 we see that the benefit of semi-randomly mixing the data has more benefits to concatenate than to absolute.

Conclusion to Mixing Experiments

In both Experiments 2.1.1 and 2.1.2 we see that there are small benefits to dataset mixing to increase the number of training instances, but only when using concatenation as the feature fusion method, with an increase in TPR and a decrease in FPR in most cases. We also find that using batch normalization has no adverse effects on the performance as previously hypothesized. Overall there is no clear benefit to using mixing methods as the FPR in both concatenate and absolute remain high or increase respectively.

3.2.2 Experiments 2.2: Pre-Training and Transfer Learning

Experiment 2.2.1: Training on Randomly Mixed Auxiliary Datasets + Retraining on the Target Dataset

Here we consider the performance when a network completely retrains on the target training subset from a network that was trained on the mix of multiple auxiliary datasets. For

Table 3.4: Experiment 2.1.2 results: Semi-randomly mixing multiple datasets.

feature fusion	**normalization**	**TPR**			**FPR**		
		VIPeR	GRID	PRID450	VIPeR	GRID	PRID450
absolute	nothing	0.30	0.39	0.39	0.05	0.04	0.05
absolute	batchnorm	0.38	0.39	0.42	0.06	0.04	0.05
absolute	selu+AD	0.26	0.34	0.44	0.04	0.03	0.07
concatenate	batchnorm	0.54	0.64	0.61	0.15	0.11	0.16

Table 3.5: Experiment 1 vs. 2.1.2 result comparison.

feature fusion	**mix method**	**TPR**			**FPR**		
		VIPeR	GRID	PRID450	VIPeR	GRID	PRID450
absolute	single	0.19	0.10	0.16	0.04	0.01	0.02
absolute	multiple	0.38	0.39	0.42	0.06	0.04	0.05
concatenate	single	0.54	0.70	0.61	0.21	0.18	0.20
concatenate	multiple	0.54	0.64	0.61	0.15	0.11	0.16

example, consider again PRID450 to be the target dataset P_T with subsets P_{train} and P_{test}. We take a network that was initialized with Glorot weights and we train it on a mix of the auxiliary datasets VIPeR V_A and GRID G_A for a number of epochs. After the network is finished training we retrain the network on P_{train} for a number of epochs. This procedure is visualized in Figure 3.3. The results are given in Table 3.6. Here we can see that, when using absolute fusion, the batch normalized network performs worse across the three datasets compared to when no normalization is used and when SELU + AD is used. It seems that, when retraining a network on the target dataset that using SELU + AD is more beneficial for performance than using batch normalization. When looking at the concatenate fusion method compared to the absolute fusion method results we can see that the TPR is increased by 0.24 to 0.43 but that the FPR is also increased by 0.12. In Table 3.6 we can see the results of this experiment compared with the results from experiment 1. Looking at the results when the absolute fusion is used there is a slight increase in TPR for two datasets and a decrease in TPR for PRID450. When using concatenate fusion, TPR across all datasets is decreased by 0.06 to 0.11 while FPR is also decreased by 0.04 to 0.07. This experiment shows that pre-training on a mixture of auxiliary datasets and then retraining on the target dataset leads to poorer results than just training on the target dataset from scratch. The results are somewhat boosted when using SELU + AD.

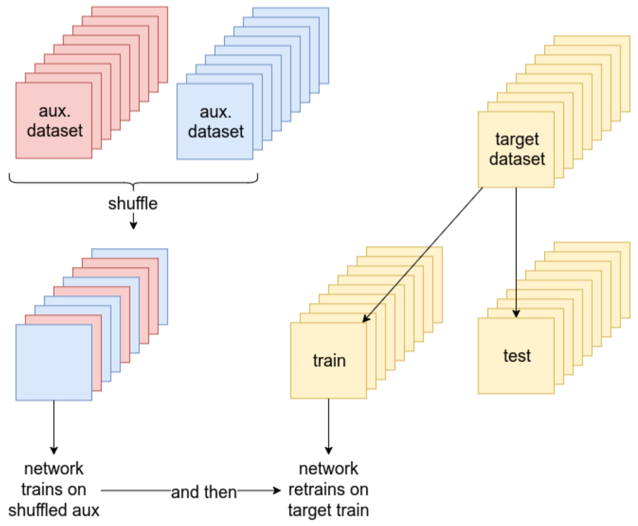

aux. dataset
aux. dataset
shuffle
target dataset
train
test
network trains on shuffled aux
and then
network retrains on target train

Figure 3.3: Experiment 2.2.1: Pre-train on randomly mixed datasets

Table 3.6: Experiment 2.2.1: Pre-train on randomly mixed datasets

feature fusion	**normalization**	**TPR**			**FPR**		
		VIPeR	GRID	PRID450	VIPeR	GRID	PRID450
absolute	nothing	0.27	0.20	0.25	0.05	0.03	0.04
absolute	batchnorm	0.23	0.16	0.10	0.05	0.03	0.01
absolute	selu+AD	0.31	0.18	0.25	0.06	0.02	0.04
concatenate	batchnorm	0.47	0.59	0.52	0.17	0.11	0.13

Table 3.7: Experiment 1 vs. 2.2.1 result comparison.

feature fusion	**mix method**	**TPR**			**FPR**		
		VIPeR	GRID	PRID450	VIPeR	GRID	PRID450
absolute	single	0.19	0.10	0.16	0.04	0.01	0.02
absolute	multiple	0.23	0.16	0.10	0.05	0.03	0.01
concatenate	single	0.54	0.70	0.61	0.21	0.18	0.20
concatenate	multiple	0.47	0.59	0.52	0.17	0.11	0.13

Experiment 2.2.2: Pre-train Consecutively on Multiple Datasets + Retraining on the Target Dataset

In this case we train the network on the first auxiliary dataset, then retrain on the second auxiliary dataset and finally retrain on the target train subset. This procedure is visualized in Figure 3.4. The order in which the network trains and re-trains can have an effect on the outcome, since different weights are learned depending on which dataset is trained on first. Therefore we make 6 experiments, considering all the possible orderings. For these experiments we use the absolute fusion method, since this was the baseline at the moment of running the experiment. The results are given in Table 3.8. In the leftmost column the training order is given in the form of *aux1, aux2* $\rightarrow$ *target*, *aux1* and *aux2* being the auxiliary datasets and *target* being the target dataset. First we look at the differences in ordering per dataset. Per each two rows in Table 3.8 we vary the target dataset, so we compare the rows that belong to the same target dataset. We can see that there is no big difference when training in a different order, across all the datasets, as the TPR results differ by 0.06 at most. When comparing different normalization techniques, we can see that the FPR is very stable, always between 0.02 and 0.06. SELU + AD give the overall best TPR performance and batch normalization gives the overall worst TPR performance. To compare the performance of pre-training consecutively to the performance of training only on the target dataset we look at the average result per dataset when using batch normalization and compare these to the absolute fusion results of experiment 1. This comparison can be seen in Table 3.9. The TPR is decreased by 0.05-0.06 on the GRID and PRID450 datasets and increased by 0.01 on the VIPeR dataset when using the sequential pre-training. And there is a 0.1 increase in FPR on the GRID dataset when using sequential pre-training. The results suggest that there is no clear performance increase when performing this transfer scheme with batch normalization, and that small performance increase can be achieved when using SELU + AD.

Table 3.8: Experiment 2.2.2 results: Pre-train consecutively on multiple datasets

order	nothing		batchnorm		selu+AD	
	TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
GRID, PRID450 $\rightarrow$ VIPeR	0.22	0.05	0.21	0.04	0.26	0.05
PRID450, GRID $\rightarrow$ VIPeR	0.28	0.06	0.18	0.03	0.26	0.05
VIPeR, PRID450 $\rightarrow$ GRID	0.18	0.02	0.10	0.02	0.23	0.03
PRID450, VIPeR $\rightarrow$ GRID	0.19	0.02	0.12	0.02	0.17	0.02
GRID, VIPeR $\rightarrow$ PRID450	0.21	0.03	0.12	0.02	0.26	0.04
VIPeR, GRID $\rightarrow$ PRID450	0.23	0.04	0.10	0.01	0.25	0.03

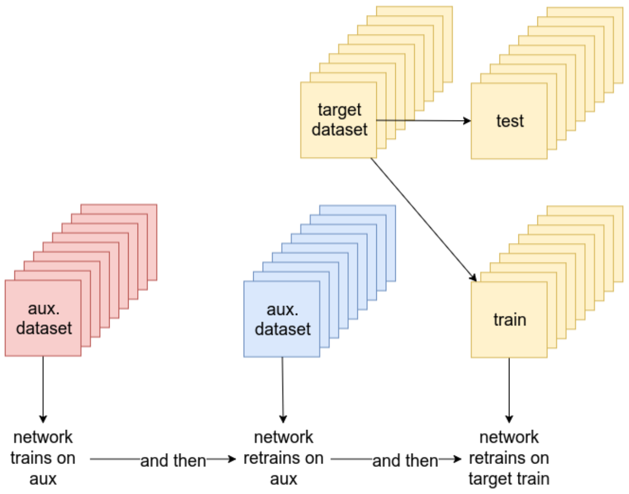


Figure 3.4: Experiment 2.2.2: Pre-train consecutively on multiple datasets

Table 3.9: Experiment 1 vs. 2.2.2 results comparison.

feature fusion	**mix method**	**TPR**			**FPR**		
		VIPeR	GRID	PRID450	VIPeR	GRID	PRID450
absolute	single	0.19	0.10	0.16	0.04	0.01	0.02
absolute	multiple BN	0.20	0.11	0.11	0.02	0.02	0.02
absolute	multiple SELU+AD	0.26	0.26	0.20	0.03	0.05	0.04

Experiment 2.2.3: Model Transfer to Other Environments

In the following experiments we mimic a real world situation where a network is trained on a dataset which is not the dataset that matches the target environment. This will be a common situation since the process of creating a dataset of the target situation is often cumbersome and resource intensive. With these experiments we want to get an idea of how learned weights from one re-id environment transfer to another environment. We think that the more similar the scenarios are, the better the weights will transfer. For this experiment we use three person re-identification datasets: GRID, CUHK02 and Market. GRID dataset has been augmented with affine transformations to the original images. We judge dataset similarity by looking at the dataset average that is shown in Figure 1.4. We can see that the datasets are alike in the sense that the mean image appears to display an either front or back facing person. The datasets are different from the VIPeR dataset because in VIPeR the person appears smaller on the image due to the use of padding on the

Table 3.10: Experiment 2.2.3 results: comparing model transfer for use in different environments. Each number in a cell represents the TPR and FPR respectively.

Trained on	Tested on		
	GRID	CUHK02	Market 1501
GRID	0.78 0.06	0.78 0.08	0.79 0.06
CUHK02	0.59 0.09	0.76 0.07	0.77 0.05
Market 1501	0.59 0.09	0.74 0.06	0.80 0.05

sides. Even though zero padding does not introduce new information to the image, it can imply to the network that there is no information on the sides of the images. The datasets are also different from the PRID450 dataset because the person in PRID450 is displayed from the side such that the gait can clearly be seen. When we compare the chosen datasets to each other we can see that Market and CUHK02 are more similar to each other than Market and GRID or CUHK02 and GRID. This means that the learned weights should be transferable between Market and CUHK02. And it means that the learned weights should be less transferable between GRID and Market or GRID and CUHK02. To measure how transferable the weights are we compare the results from transfer with the performance results from training on the target datasets only. We say that the weights are transferable when the results from transfer are similar to the results when training on the target datasets only. In Table 3.10 we can see the results of this experiment. Each cell shows the TPR and FPR respectively. When referring to a specific cell in Table 3.10 we refer to it as (dataset trained on, dataset tested on) = [TPR, FPR]. For example when addressing results about a model trained on GRID and tested on CUHK02, we refer to this with (GRID, CUHK02) = [0.78, 0.08]. The results indicate that a the model trained on GRID is transferable to both CUHK02 and Market: (GRID, CUHK02) = [0.78, 0.08] vs. (CUHK02, CUHK02) = [0.76, 0.07] and (GRID, Market) = [0.79, 0.06] vs. (Market, Market) = [0.80, 0.05]. The results also indicate that models trained on CUHK02 or Market do not transfer well to GRID: (CUHK02, GRID) = [0.59, 0.09] vs. (GRID, GRID) = [0.78, 0.06] and (Market, GRID) = [0.59, 0.09] vs. (GRID, GRID) = [0.78, 0.06]. There is a TPR performance decrease of about 0.2. Lastly, the results indicate that, indeed, Market and CUHK02 are very similar datasets because models trained on Market or CUHK02 transfer well to CUHK02 or Market respectively: (CUHK02, Market) = [0.77, 0.05] vs. (Market, Market) = [0.80, 0.05] and (Market, CUHK02) = [0.74, 0.06] vs. (CUHK02, CUHK02) = [0.76, 0.07].

Conclusion to Pre-Training and Transfer

Results of Experiments 2.2.1 and 2.2.2 show that there is no clear performance increase when pre-training on auxiliary datasets followed by re-training on the target dataset. Both experiments show that there is a small performance increase when, instead of batch normalization, SELU + AD are used. These results suggest that, for small datasets, the network does not learn a general model of the person re-identification model that can be used in another environment, even when the network has trained been re-trained on data from this environment. Results in Experiment 2.2.3 show that this does not hold for networks trained on larger datasets. These results show that when the dataset is large and noisy, it transfers well to environments that are more clean, without having to be re-trained on a dataset from said environment.

3.3 Experiments 3: Video Data vs. Image Data

In this section we conduct experiments by comparing the use of video data to the use of image data. To do this we conduct experiments on the Siamese 3D Convolutional Neural Network (S3DCNN) using the iLIDS-VID and the PRID2011 datasets. The two datasets differ in several aspects: (1) The images in the iLIDS-VID dataset are captured from two cameras where the view from each camera is dissimilar to the other in terms of image resolution, lighting and graphical distortions. In comparison to PRID2011, the images from both cameras are very similar to each other in terms of the previously mentioned properties. (2) The images in iLIDS-VID are captured inside an airport, which, due to the high density of people with luggage, results in a cluttered background. In contrast to the images in PRID2011, which are captured at a crosswalk, resulting in relatively uniform and uncluttered backgrounds. We make a general distinction between the datasets by saying that PRID2011 is of higher quality than iLIDS-VID. We compare the results of the experiments with results from running the image versions of these datasets on the SCNN. Image versions of these datasets are obtained by extracting images from the original footages. By doing this we ensure that both datasets are from the same distribution. The only difference between the SCNN and the S3DCNN is that the S3DCNN uses 3D convolutions and the SCNN uses 2D convolutions. Except for this difference, everything else is the same. We think that the experiments with the video datasets should result in better performance, because video data encodes more information than static image data, which the S3DCNN should be able to extract.
In Table 3.11 we can view the results of the experiment. We can see that, in all cases, the S3DCNN trained on video outperforms the SCNN trained on images, consistently in terms

Table 3.11: Experiments 3 results: Image Data vs. Video Data.

dataset	**training instances**	**image TPR, FPR**	**video TPR, FPR**
PRID2011	700	0.61 0.20	0.69 0.08
	1400	0.74 0.09	0.74 0.08
iLIDS-VID	400	0.54 0.21	0.65 0.20
	800	0.64 0.16	0.74 0.14

of TPR increase and FPR decrease when video is used. When the number of training instances is doubled, we see improvement in performance for both datasets. We also see that, for the PRID2011 dataset, as the number of instances increase, the difference in performance between the SCNN and the S3DCNN becomes very small. This suggests that, as data becomes abundant, the advantage of temporal information will decrease.

Chapter 4

Discussion

In this chapter we discuss the results obtained in the previous chapter, where each section addressed the specific research question(s). In this chapter we maintain this structure where each section discusses the results of a set of experiments that addressed a research question. Subsequently we give an answer to the research question(s) posed.

4.1 Neural Metric Learners

In Section 1.3 we asked: What is the effect of replacing the Euclidean distance with a neural network for metric learning? This section answers this question.

The results regarding metric learning using neural networks (neural metric learners) have shown that neural networks are less efficient as metric learners than the Euclidean distance, see Table 3.1, requiring more data to reach the same level of performance. We also learned that, as the datasets become larger (CUHK02 and Market1501), the difference in performance between the neural metric learners and the Euclidean distance decreases, especially when considering the concatenate fusion method. This trend indicates that as the dataset becomes larger, the neural metric learner rivals the performance of the Euclidean distance. However, the difference between the Euclidean distance and the neural metric learners became much more apparent when we compared CMC rank-1 scores in Table 4.1, with Euclidean distance outperforming the concatenate and absolute neural learners by a large margin. A reason for this can be that, as the contrastive loss function drives the convolutional feature extractors to develop such that the Euclidean distance is between [0, 1], information about the distance between images is preserved, in the sense that the network prediction is a measure of how similar an image is from another image. For example, if a network using the Euclidean distance predicts 0.1 and 0.4 for two different image pairs respectively, both will get classified as a match. It means that the image pair with 0.1 prediction is more similar than the image pair with a 0.4 prediction. Thus the network output is a distance. In contrast, when we use a neural metric learner, we use the categorical crossentropy loss function which only cares about the two classes: The network learns the boundary between these classes. Therefore the predictions that the network makes when using a neural metric learner can only be interpreted with respect to the boundary between the classes. The predicted numbers themselves do not indicate similarity between one image and another. Using the same example as before, both image pairs will get classified as a mismatch. However, we cannot say that the image pair with a 0.1 prediction is more different than the image pair with the 0.4 prediction.

4.2 The Use of Auxiliary Datasets

In Section 1.3 we asked:

- How can we best make use of all available datasets?
    1. What are the effects of different training schemes on the performance?
    2. How effective is pre-training on auxiliary person re-identification datasets?
    3. How well does a learned model generalize to a different environment?

4.2.1 Data Mixtures

In this section we answer the first question: What are the effects of different training schemes on the performance? In Table 3.2 and 3.4 we saw the results for training a neural network on mixtures of data. We saw that batch normalization does not lead to a deterioration of performance as we had first hypothesized. A reason for this could be that the learnable scale and shift parameters, in Equation 1.13 γ and β respectively, in the batch normalization function are adapted to the joint distribution mean and variance, such that the resulting activation is not distorted beyond usefulness. Thus the activations are jointly distributed and as a result the network learns a more general representation of the classification task.

We also saw that mixing data yields small improvement in performance in a network that uses concatenation as feature fusion method, while it did not lead to improvements when using the absolute fusion method. An obvious difference is that when using concatenation we pass on all the extracted features to the metric learner, which then serves as a bottleneck since the concatenated feature vector is larger than the number of units in the first layer of the metric learner. This forces the network learn to use only the important extracted information. When using the absolute method, the bottleneck becomes wider, diminishing the bottleneck effect sincce the absolute feature vector is half the size of the concatenated vector. Thus when dealing with a mixture of data, which has a joint distribution, the absolute feature vector contains more distilled information, that could not be optimal for the network, while the concatenated feature vector simply contains all the information.

4.2.2 Transfer Learning

In this section we answer the second question: How effective is pre-training on auxiliary person re-identification datasets? We also answer the third question: How well does a learned model generalize to a different environment? The results of Experiment 2.2.1 and 2.2.2 in Table 3.6 and 3.8 indicate that pre-training and then re-training on small datasets do not yield improved results compared to only training on the target dataset. Our results are contrary to the results achieved in [20], where via a two-step fine-tuning scheme (ImageNet-> CUHK03+Market1501 -> Target dataset) their network achieved improved results over several benchmarks They claim that their models can transfer features from auxiliary data sources because it was trained on multiple objectives (classification + verification). These results are consistent with our findings, that if attempting to transfer features from auxiliary datasets only using the classification objective, it will be unsuccessful. [20] uses GoogLeNet as their base network for feature extraction, which is a much deeper network than our CNN. This can also explain why our network was not able to transfer features from auxiliary datasets. Furthermore, our findings about the transfer of small person re-identification datasets are consistent with findings in [39], which show that when training on small datasets, the features don't transfer to a new environment. In contrast, larger datasets tend to transfer much better. This is also consistent with our findings. Another factor to the transfer of data as documented in [40] is that, as the discrepancy between datasets grow, the transfer ability of the features is diminished. In this thesis in Table 3.10 we can see that our results contradict this fact. Even though Market1501 and CUHK02 are more similar to each other than each to GRID, we get better transfer when training on GRID. A reason can be that GRID can be viewed as a noisy superset containing CUHK02 and Market1501. Thus our network learns a more general model from the noisy data that can better be transferred to datasets in the superset.

4.3 Image vs. Video

In Section 1.3 we asked:

- How can we compare the effectiveness of the use of image data versus video data?
    1. How can we modify the image network minimally to work on video data?
    2. Is video data more informative than image data?

In this section we answer both these questions. In Section 3.3 we modified the SCNN by replacing the 2D convolutional layers with 3D convolutional layers, resulting in a S3DCNN, see Figure 2.3. We learned from the results that the S3DCNN trained on videos outperforms the SCNN trained on images. We also saw that doubling the amount of training data boosted performance across all variables. When we take a closer look at the performance increase across datasets for both image and video data types, we see that the amount to which the performance is increased is inconsistent: We expect that doubling the number of training videos would contribute to more performance increase in the S3DCNN than doubling the number of training images used for training the SCNN, because video data contains information in the temporal domain in addition to information stored in the spatial domain. For PRID2011 image, the performance boost (as a result of doubling) is more prominent than PRID2011 video and iLIDS-VID image. The difference with iLIDS-VID image can be explained because with iLIDS-VID image (1) the base number of training instances is lower compared to PRID2011 image (400 vs. 700) and (2) the base number of training instances is increased with a lower number of instances compared to PRID2011 image (+400 vs. +700). The difference between PRID2011 image and video is more difficult to explain because we expect there to be a similar performance boost as in PRID2011 image. One reason that could explain this difference is that the S3DCNN reaches its performance limit earlier than the SCNN. With performance limit we mean the point when the performance boost per increase in training instance starts to decline.

4.4 Comparison to Literature

We run our models on three benchmarks in order to compare to other methods, see Section 1.1.1. The results are given in Table 4.1. We augment the VIPeR and the PRID450 dataset by applying affine transformations (rotation, zooming, horizontal mirroring, horizontal mirroring + rotation, horizontal mirroring + zooming) to each image in the datasets. We see that our relatively simple SCNN is competitive on at least two small datasets while not being specifically trained on the CMC ranking test. In Table 4.1 we can compare the

Table 4.1: Comparison to State of The Art.

network	VIPeR rank-1	PRID450 rank-1	Market rank-1
SCNN concatenate	37	45	-
SCNN absolute	43	46	-
SCNN Euclidean	50	56	21
JLML [16]	50.2	-	85.1
CRAFT [15]	54.2	-	71.8
DHSL [23]	44.9	-	-
PIE [13]	54.5	-	79.3
DTL [20]	56.3	-	83.7
DFR [22]	51.1	66.6	-
MTDnet [8]	47.5	31	-
CDML [19]	40.9	-	-
APR [14]	-	-	84.3
CAN [6]	-	-	48.2

results of our SCNN using Euclidean distance, concatenation and absolute feature fusion with DHSL [23], which uses a combination of the element-wise absolute difference and the element-wise multiplication to fuse the extracted feature vectors into a single vector. Specifically we compare on the ViPER benchmark. Both DHSL and SCNN formulate the person re-identification problem as a match-mismatch classification problem. We can see that the SCNN using absolute fusion performs on about the same level as DHSL and that the SCNN using a Euclidean distance outperforms DHSL by a margin of 5.1%. Our results show that in contrast to the assessment made by [23] (Euclidean distance has lower discriminative ability to measure the similarity between a CNN learning feature pair), we find that, clearly, the Euclidean distance has sufficient discriminative ability to outperform at least three different neural metric learners on the CMC rank-1 ViPER benchmark, which is inherently a measure of discriminative ability.

When we compare our SCNN to JLML [16] in Table 4.1 we find that we achieve similar performance on the ViPER benchmark with Euclidean (50%) as JLML (50.2%). This is remarkable considering our network consists of 9 layers in total, 6 layers which serve for feature extraction, while the JLML uses a 39 layered ResNet to extract features from a single image. Furthermore, there are major differences between SCNN and JLML, the most important difference being that the network is specifically optimized for the CMC ranking task by taking the L2 distance between extracted probe-gallery feature pairs. This strategy is highly effective for large datasets, when the benchmark gallery becomes much larger. We can see the clear difference in performance between SCNN (21%) and the JLML (85.1%) on the Market1501 benchmark. This thesis shows that, when the gallery is small enough (≤300 IDs), a relatively small neural network, trained on a classification problem, can achieve similar performance on benchmark as very deep neural networks trained specifically on the CMC ranking problem. A more general statement can be derived: This thesis shows that, when the gallery is small enough (≤300 IDs), a relatively small neural network, trained on a classification problem, can achieve similar performance on benchmark as very deep neural networks. This holds for comparing to PIE [13] (54.5%), a large network which was trained on the match-mismatch classification problem, in which, among other things, a ResNet50 was used for feature extraction. We can see that our small network rivals its performance on ViPER. The same can be said when comparing to [20] which uses GoogLeNet (22 layers) for feature extraction. Our statement holds again when comparing our SCNN to MTDnet [8], where we outperform MTDnet on both ViPER and PRID450 benchmarks. MTDnet is a 24 layer multi-task neural network training on both the ranking and classification task.

Our Euclidean SCNN and absolute SCNN outperform CDML [19] on ViPER by a margin of 9.1% and 2.1% respectively. CDML is a small neural network with 7 layers, of which five are used for feature extraction from an image pair. This network used subtract feature fusion to fuse extracted features and then a Mahalanobis metric is learned from the fused feature. In this case we show that both the Euclidean SCNN and the absolute SCNN outperform a hybrid method of metric learning which combines feature fusion with Mahalanobis metric learning.

Chapter 5

Conclusion

The goal of this thesis was to build a deep neural network detector for a particle filter tracking system, where we maximize TPR and minimize FPR. The deep neural network serves to solve the problem of person re-identification in an indoor environment with no more than 100 identities present in total. The particle filter requires the detector to return a true or false statement indicating if the image captured by the sensor represents the person we are looking for or not respectively. Therefore we adopt the siamese network architecture, accepting probe-candidate image pairs as input and returning a true or false if the image pair is a match or not. Furthermore we have investigated multiple problems that we face when doing person re-identification in practice. Overall:

- There are indications that siamese neural networks perform sufficiently well as detectors, see Table 3.10 where we measure generalization to new scenarios.
- We describe conditions under which the network can perform well and investigate the impact of architectural variations and data preparation methods on TPR and FPR performance.
- We extract best practice rules from the experiment results considering architecture design and data preparation.
- This thesis shows that, a relatively small neural network can achieve similar performance on benchmark as very deep neural network, assuming that the gallery is small enough ($\leq$300 IDs).

More specifically: Assuming that the target scenario can be considered a subset of the GRID dataset, a performance TPR and FPR of 0.78 and 0.08 can be expected respectively. We compare the use of the Euclidean distance to the use of a neural network for metric learning, with various methods of feature fusion. We found that using a neural network for metric learning is less effective than using the Euclidean distance, when looking at the TPR, FPR and CMC rank-1 score across five different datasets. Overall it is difficult to draw a decisive conclusion as to which feature fusion method is the best because there are two performance measures to evaluate. We find that increases and decreases in TPR and FPR are inconsistent across the five datasets experimented with. The only noticeable trend is that as datasets become larger, the difference between performance on individual datasets diminishes. The conclusion that we can draw from this is that the convolutional branches in the SCNN are flexible enough to adapt to any form of fusion method, given that we have enough data. We also compare several schemes of using auxiliary data to train the network on and found that in the absence of sufficient desirable target environment data, the use of auxiliary data is beneficial to the extent that the concatenate feature fusion is used. Overall no consistent performance increase was measured when using mixtures of small datasets. We also found that pre-training our network on (mixtures of) small datasets was ineffective, but that large and noisy datasets tend to generalize well to new environments. Finally we found that video data has an informational advantage over image data. When using the Euclidean distance we find that our model is competitive on the VIPeR (50%) and the PRID450 (56%) datasets without being specifically trained to perform on the ranking task. However, as the ranking task becomes more difficult (with bigger datasets) we can clearly see the limitations of our network as performed on the Market1501 dataset (21%).

5.1 Practical Recommendations

To practitioners we have a few recommendations based on the research conducted in this thesis. The recommendations are specifically useful in a situation such as our particle filter scenario, when we do not have many IDs ($\leq$100). In this scenario we saw that small neural networks perform on a similar level compared to a very deep neural network. Smaller neural networks have the benefits of:

1. Less data necessary to achieve sufficiently well level of performance.
2. Shorter training times.
3. Low memory requirements, cheaper GPUs can be used.
4. Resulting model has a smaller digital footprint, ideal for mobile devices.

Our recommendations assuming that no dataset of the target environment is available:

1. Find large noisy image dataset(s) that appears similar to the target environment. The dataset mean can be used for visually estimating how similar the dataset is to the target environment. In general the more noisy the dataset, the better. Our research shows that the noisier the dataset, the better it generalizes to a novel scenario.

2. Augment said noisy dataset with transformations such as, but not restricted to mirroring, zooming and rotation. This way we get more training data. The more training data, the better.

3. If the datasets found are similar enough, mix these together in a random order. Also perform sideways shuffling as explained in Section 2.4.3.

4. Initialize the network weights using Xavier initialization [41].

5. Assuming we have a lot of noisy training data (see step 2 and 3), use a network with the Euclidean distance as a metric, with a learning rate of 0.00001, a contrastive loss function using either the Nadam optimizer with a decay of 0.95 or the RMS optimizer.

If a (small) target environment dataset is available:

1. Do previous step 1 and 2.

2. Semi-randomly mix the training data, as illustrated in Figure 3.2.

3. Do previous step 4 and 5.

5.2 Future Work

The overall conclusion is that very deep neural networks are unnecessary for a small scale person re-identification problem, since small deep neural networks are capable of achieving a similar results. Therefore more research is needed in order to properly exploit the capabilities of smaller deep neural networks and how to scale up their performance as the gallery size increases. Alternative loss functions should be considered, such as the contrastive loss function, when using neural metric learners. The contrastive loss function may be more appropriate than the categorical crossentropy loss because the contrastive loss takes into account the similarity of the instances with respect to each other, and not just with respect to the decision boundary. This way, like with the Euclidean SCNN, we learn the decision boundary between classes while the network prediction can be interpreted as a distance, enhancing the discriminative ability of a network using a neural metric learner, potentially lowering the FPR rate as was the case in Euclidean SCNN. In Table 3.1 we see differences between performance when different feature fusion methods are used on small datasets. It is difficult to explain what exactly is the cause of these differences. A visualization of the kernels in the convolutional layers, for example using deconvolutional networks [42], can help us understand which features are being considered as important when different fusion methods are used. Finally, more experiments should be done on image vs. video data to determine if there really is a difference between the performance limit of the SCNN vs. S3DCNN. This can have implications on the use of video for person re-id in practice since it is more cumbersome to create video datasets than it is to create image datasets, networks take longer to train on video datasets and resulting video network models have a bigger digital footprint. Furthermore, adaptations [43] can be made to the S3DCNN in order to boost performance significantly while reducing the digital footprint.

Bibliography

[1] Rik Claessens, Gregor Pavlin, and Patrick de Oude. Context driven tracking using particle filters. In *Information Fusion (Fusion), 2015 18th International Conference on*, pages 1128–1135. IEEE, 2015.

[2] Patrick de Oude, Gregor Pavlin, and Rik Claessens. Attractor-directed particle filtering with potential fields. In *Information Fusion (FUSION), 2016 19th International Conference on*, pages 1079–1086. IEEE, 2016.

[3] Douglas Gray and Hai Tao. Viewpoint invariant pedestrian recognition with an ensemble of localized features. *Computer Vision–ECCV 2008*, pages 262–275, 2008.

[4] Srikrishna Karanam, Mengran Gou, Ziyan Wu, Angels Rates-Borras, Octavia Camps, and Richard J Radke. A comprehensive evaluation and benchmark for person re-identification: Features, metrics, and datasets. *arXiv preprint arXiv:1605.09653*, 2016.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[6] Hao Liu, Jiashi Feng, Meibin Qi, Jianguo Jiang, and Shuicheng Yan. End-to-end comparative attention networks for person re-identification. *arXiv preprint arXiv:1606.04404*, 2016.

[7] Lin Wu, Chunhua Shen, and Anton van den Hengel. Deep recurrent convolutional networks for video-based person re-identification: An end-to-end approach. *arXiv preprint arXiv:1606.01609*, 2016.

[8] Weihua Chen, Xiaotang Chen, Jianguo Zhang, and Kaiqi Huang. A multi-task deep network for person re-identification. *arXiv preprint arXiv:1607.05369*, 2016.

[9] N McLaughlin, J Martinez del Rincon, and P Miller. Recurrent convolutional network for video-based person re-identification. *CVPR*, 2016.

[10] Tong Xiao, Hongsheng Li, Wanli Ouyang, and Xiaogang Wang. Learning deep feature representations with domain guided dropout for person re-identification. *arXiv preprint arXiv:1604.07528*, 2016.

[11] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[12] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[13] Liang Zheng, Yujia Huang, Huchuan Lu, and Yi Yang. Pose invariant embedding for deep person re-identification. *arXiv preprint arXiv:1701.07732*, 2017.

[14] Yutian Lin, Liang Zheng, Zhedong Zheng, Yu Wu, and Yi Yang. Improving person re-identification by attribute and identity learning. *arXiv preprint arXiv:1703.07220*, 2017.

[15] Ying-Cong Chen, Xiatian Zhu, Wei-Shi Zheng, and Jian-Huang Lai. Person re-identification by camera correlation aware feature augmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[16] Wei Li, Xiatian Zhu, and Shaogang Gong. Person re-identification by deep joint learning of multi-loss classification. *arXiv preprint arXiv:1705.04724*, 2017.

[17] Shengyong Ding, Liang Lin, Guangrun Wang, and Hongyang Chao. Deep feature learning with relative distance comparison for person re-identification. *Pattern Recognition*, 48(10):2993–3003, 2015.

[18] Rahul Rama Varior, Mrinal Haloi, and Gang Wang. Gated siamese convolutional neural network architecture for human re-identification. In *European Conference on Computer Vision*, pages 791–808. Springer, 2016.

[19] Hailin Shi, Xiangyu Zhu, Shengcai Liao, Zhen Lei, Yang Yang, and Stan Z Li. Constrained deep metric learning for person re-identification. *arXiv preprint arXiv:1511.07545*, 2015.

[20] Mengyue Geng, Yaowei Wang, Tao Xiang, and Yonghong Tian. Deep transfer learning for person re-identification. *arXiv preprint arXiv:1611.05244*, 2016.

[21] Lin Wu, Chunhua Shen, and Anton van den Hengel. Personnet: Person re-identification with deep convolutional neural networks. *arXiv preprint arXiv:1601.07255*, 2016.

[22] Shangxuan Wu, Ying-Cong Chen, Xiang Li, An-Cong Wu, Jin-Jie You, and Wei-Shi Zheng. An enhanced deep feature representation for person re-identification. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*, pages 1–8. IEEE, 2016.

[23] Jianqing Zhu, Huanqiang Zeng, Shengcai Liao, Zhen Lei, Canhui Cai, and LiXin Zheng. Deep hybrid similarity learning for person re-identification. *arXiv preprint arXiv:1702.04858*, 2017.

[24] Niall McLaughlin, Jesus Martinez Del Rincon, and Paul Miller. Data-augmentation for reducing dataset bias in person re-identification. In *Advanced Video and Signal Based Surveillance (AVSS), 2015 12th IEEE International Conference on*, pages 1–6. IEEE, 2015.

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

[26] Yichao Yan, Bingbing Ni, Zhichao Song, Chao Ma, Yan Yan, and Xiaokang Yang. Person re-identification via recurrent feature aggregation. In *European Conference on Computer Vision*, pages 701–716. Springer, 2016.

[27] Xiaolong Ma, Xiatian Zhu, Shaogang Gong, Xudong Xie, Jianming Hu, Kin-Man Lam, and Yisheng Zhong. Person re-identification by unsupervised video matching. *Pattern Recognition*, 65:197–210, 2017.

[28] Shuangjie Xu, Yu Cheng, Kang Gu, Yang Yang, Shiyu Chang, and Pan Zhou. Jointly attentive spatial-temporal pooling networks for video-based person re-identification. *arXiv preprint arXiv:1708.02286*, 2017.

[29] Hao Liu, Zequn Jie, Karlekar Jayashree, Meibin Qi, Jianguo Jiang, Shuicheng Yan, and Jiashi Feng. Video-based person re-identification with accumulative motion context. *arXiv preprint arXiv:1701.00193*, 2017.

[30] Wei Zhang, Shengnan Hu, and Kan Liu. Learning compact appearance representation for video-based person re-identification. *arXiv preprint arXiv:1702.06294*, 2017.

[31] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

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

[33] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

[34] Wei Li and Xiaogang Wang. Locally aligned feature transforms across views. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3594–3601, 2013.

[35] Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. Scalable person re-identification: A benchmark. In *Computer Vision, IEEE International Conference on*, 2015.

[36] Chen Change Loy, Tao Xiang, and Shaogang Gong. Multi-camera activity correlation analysis. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1988–1995. IEEE, 2009.

[37] Peter M. Roth, Martin Hirzer, Martin Köstinger, Csaba Beleznai, and Horst Bischof. Mahalanobis Distance Learning for Person Re-Identification. In Shaogang Gong, Marco Cristani, Shuicheng Yan, and Chen C. Loy, editors, *Person Re-Identification*, Advances in Computer Vision and Pattern Recognition, pages 247–267. Springer, London, United Kingdom, 2014.

[38] Martin Hirzer, Csaba Beleznai, Peter M. Roth, and Horst Bischof. Person re-identification by descriptive and discriminative classification. In *Proc. Scandinavian Conference on Image Analysis (SCIA)*, 2011.

[39] Dong Yi, Zhen Lei, and Stan Z Li. Deep metric learning for practical person re-identification. *arXiv preprint arXiv:1407.4979*, 2014.

[40] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[41] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[42] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.

[43] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *arXiv preprint arXiv:1705.07750*, 2017.