Re-identification with Recurrent Neural Networks

Johannes Skog

Master's thesis
2017:E54

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Lund University

Master's Thesis Report

Re-identification with Recurrent Neural Networks

Exploring spatio-temporal relationships in sequences of images

Author:

Johannes Skog

Supervisor:

Niclas Danielsson

Anton Jakobsson

Håkan Ardö

September 7, 2017

Lund University

Abstract

Today re-identification of persons in disjoint camera views demand large human effort and resources. There is a need of autonomous re-identification systems.
This thesis has been focused on the development of autonomous re-identification systems based on recurrent neural networks able to analyze sequences of images of persons, enabling the possibility to filter out outliers within sequences and make use of spatio-temporal features. Through the development and evaluation of multiple network architectures and recurrent units a broad examination of the viability of recurrent neural networks in re-identification systems have been made.
This thesis has shown, analysis of sequences of images can significantly increase the performance of re-identification systems compared to analysis of single images, with a doubling of the re-identification performance in multiple cases. Non-dynamic features tend to dominate the heap of information from sequences used during re-identification.
A realistic benchmark dataset was created as part of the thesis. Evaluation on the benchmark dataset has shown, to achieve satisfactory performance in real-world scenarios of autonomous re-identification systems, based on recurrent neural networks, larger training datasets with greater variety are needed.

Acknowledgements

I would like to thank Axis Communications for providing me the opportunity of doing my master thesis project at the analytics group. Special thanks to my supervisors at Axis Communications, Niclas Danielsson and Anton Jakobsson, for valuable guidance and support. For constructive feedback and ideas I would like to thank my supervisor at Lund University, Håkan Ardö.

Contents

Chapter 1

Introduction

Person re-identification is the method of re-identifing persons captured in disjoint camera views. Differences in person poses, camera viewpoints and illumination settings introduce significant complexity to the re-identification problem. There are two types of re-identification methods, single shot re-identification and multi shot re-identification. In the single shot case is a single image used to re-identity a person. In the multi shot case is a sequence of consecutive sampled images used to re-identify a person. This method enables temporal analysis of person specific temporal dependent features, such as the gait. A multi shot system compounds also non-temporal person specific features between the sampled images. Non-temporal information extraction in conjunction with the temporal analysis of a multi shot re-identification system leads to a system more complex and expressive than a single shot re-identification system.

Deep learning is an umbrella term for a set of machine learning algorithms that attempt to model high level abstractions in data. Included in this machine learning branch are neural networks, which try to mimic the highly intricate way in which the human brain processes information. In large due to increasingly powerful computing capabilities these type of networks have in the past years produced remarkable advances in a number of fields, including image classification [40], and video surveillance [48][1][44][30][43].

Video surveillance has got an increasingly prominent role in our contemporary world in order to ensure security and documentation of events. This upswing of video surveillance has resulted in the dire need of smart algorithms sifting through the massive amount of generated data. Deep learning based systems are predicted to fill this need in tomorrow's automatic surveillance systems. Person re-identification is a task mostly managed by human operators. In the past years there have been a boom of different autonomous re-identification systems based on deep learning [48][1][44][30][43]. These types of systems have introduced a significant leap forward in re-identification performance [48].
Many of today's state of the art deep learning based re-identification systems do not leverage the spatio-temporal relationship inherent in video feeds. Convolutional recurrent neural networks allow analysis of spatio-temporal information by analysis of sequences of consecutive sampled images.
Recent research articles have shown the successful inclusion of convolutional recurrent neural networks in multi shot re-identification systems [30][43]. These networks perform temporal analysis of sequences of images depicting person trajectories.

The thesis has been focused on the development and exploration of different types of autonomous person re-identification neural networks. Both single shot neural networks and multi shot neural networks have been developed as part of the thesis. These developed networks are able to re-identify a person present in different environments, both by analysis of single images and sequences of images. The main purpose of this thesis has been to examine in which way a neural network can leverage spatio-temporal features for re-identification purposes.

A realistic benchmark multi shot re-identification dataset has been created as part of the thesis. The realistic benchmark dataset has ensured comparative measurements of the re-identification performance of the developed multi shot re-identification networks in real-world scenarios.

The master's thesis report is divided into five chapters. In the first chapter, the theory behind neural networks is introduced. Following this chapter is an introduction of the different re-identification datasets used during training and evaluation of the developed networks. In this chapter, the realistic benchmark dataset is presented.
The following two chapters present the different developed networks, for single shot and multi shot re-identification. These chapters will also present the results from the two studies. The last chapter is dedicated to final conclusions.

Chapter 2

Neural Networks

The complexity of the cognitive process in the human brain has eluded researchers for hundreds of years. The process of storing, recollecting and bringing order to information is attributed to the highly complex way in which the brain cells i.e. neurons interact. These cells form an intricate web of intercommunication. The individual neuron receives information from its input connected neighbors, the input undergoes a highly nonlinear transformation and becomes the input to its output connected neighbors. This seemingly simple way in which the neurons convey information has given rise to the complex cognitive processes in which we form our understanding of the world. The human brain is regarded as plastic in the sense that new experiences form factual physiological changes in the brain, connections between neurons are continuously enforced or neglected by experiences. The concept of artificial neural networks is loosely based on the adaptive nature of the human brain, where the connections between artificial neuron are either reinforced or neglected as the artificial neural network learns, similarly to its biological counterpart.

2.1 Background

The concept of artificial neural networks can in large be attributed to an article written in the 1930's by a neurophysiologist Warren McCulloch and a mathematician Walter Pitts [42]. The article outlined how neurons in the brain might work. The two researcher proposed a highly simplified model of the interaction between neurons, each neuron was modeled either as on or off according to,

$$y = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{if} \quad \sum_{i=1}^{n} w_i x_i < T \end{cases} \tag{2.1}$$

where x_i is the input from neuron i and w_i the weight of neuron i, T is a threshold value. This simplistic model was implemented with electrical circuits and lacked machinery for learning.
The model was later extended in 1959 by Frank Rosenblat in his article *The perceptron: A probabilistic model for information storage and organization in the brain* to include a machinery for learning [33] and a bias term according to equation 2.2.

$$y = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{n} w_i x_i - b \geq 0 \\ 0 & \text{if} \quad \sum_{i=1}^{n} w_i x_i - b < 0 \end{cases} \tag{2.2}$$

This schematic representation of a human neuron was given the name perceptron. The learning algorithm accompanied by the perceptron was called *perceptron convergence algorithm.* The neuron weights and biases could now be learned in a deterministic fashion. A single perceptron is a linear classifier, it produces a class prediction based on external input data $x_1, x_2, ..., x_n$, the respective weights of each input are declared by $w_1, w_2, ..., w_n$. Figure 2.1 illustrates the class prediction of a perceptron, the two classes in the figure are represented by filled and outlined circles. Separating the two classes is a line, parametrized by the learned weights.

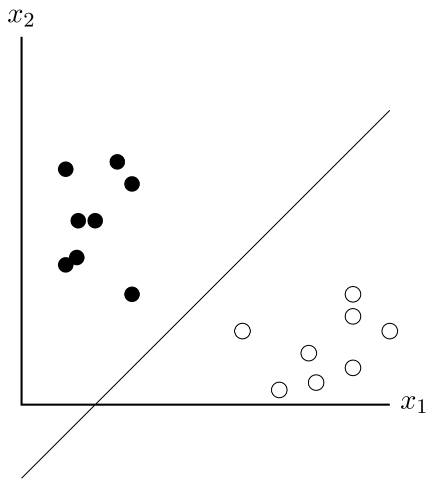

Figure 2.1: Class separation of two classes.

Despite the overall crudeness of the model, the perceptron was successfully applied to classification of simple shapes, represented by 20×20 images. The perceptron and its learning algorithm is in its essence the ground pillars to modern neural networks. As we will see in the following sections the ideas behind the perceptron and its precursors have given rise to the expanding field of deep learning.

2.2 Multilayer Fully Connected Networks

Figure 2.2 shows a schematic illustration of a single artificial neuron. The neuron receives input data x_1, x_2, x_3, where x_i is weighted by w_i and offset by a bias term b. The resulting summation of the weighted input is called the activation. Following the summation comes an activation function, which performs an element-wise transformations on the activation. The role of the activation function will be discussed in a following section.

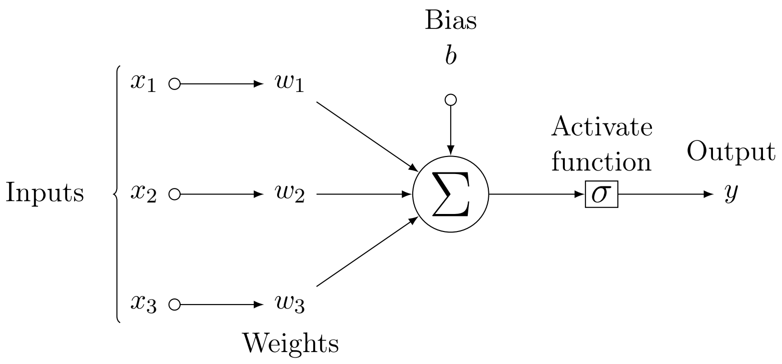

Figure 2.2: Illustration of a single neuron.

Multilayer fully connected networks are constituted by many fully connected layers, where each layer may contain multiple neurons. The input layer, i.e. the first layer, contains a number of neurons matching the number of elements of the input data. The input layer receives the external input and directs the data to the hidden part of the network. The hidden layers constitute the hidden part of a network. These layers are called hidden since they are in between the input and output layer of the network. The first hidden layer receives the input data from the input layer. Each consecutive hidden layer passes their output to the next layer in the network. This continues until the output layer is reached. The output layer concentrates the features extracted from the hidden part of the network into an output representation.
An artificial neuron i can mathematically be expressed as,

$$\begin{aligned} a_i &= \sum_{j=1}^{N} w_{i,j} x_j + b_i \\ y_i &= \sigma(a_i) = \sigma\left(\sum_{j=1}^{N} w_{i,j} x_j + b_i\right) \end{aligned} \tag{2.3}$$

where x_i is one of the N inputs, $w_{i,j}$ indicates the weight between neuron i and j, b_i is the bias term for neuron i. The neuron's activation is denoted a_i and the output from the activation function σ is denoted y_i. The neuron's output y_i becomes the input to other neurons in the succeeding layer. Figure 2.3 shows a schematic representation of a fully connected network where $x_1 ... x_D$ are input neurons, and $h^1_1 ... h^1_m, h^2_1 ... h^2_c$ are hidden neurons in the two hidden layer. The output layer is constituted by the output neurons $o_1 ... o_N$.

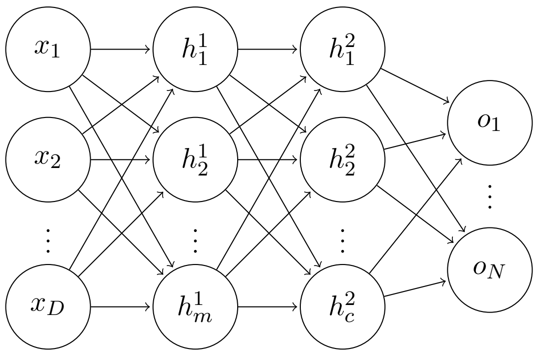

Figure 2.3: A two layer fully connected network.

Each layer's output can compactly be expressed in vector/matrix notation as,

$$\begin{aligned}\mathbf{a} &= \mathbf{W}\mathbf{x} + \mathbf{b} \\ \mathbf{y} &= \sigma(\mathbf{a}) = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)\end{aligned} \tag{2.4}$$

where $\mathbf{x}$ is the vector of outputs from the preceding layer, $\mathbf{W}$ is the weight matrix and $\mathbf{b}$ is the bias vector.

The universal approximation theorem states, a fully connected neural network with only one hidden layer, containing a finite number of neurons, can approximate any continuous function [5]. This capacity is directly attributed to the use of activation functions in the hidden layers. As we will see in the next section, a particular type of activation functions are needed, namely non-linear functions.

2.3 Activation Functions

An activation function receives an input and performs an element-wise transformation on it.

Let's examine a two-layer fully connected network, similarly to the one displayed in figure 2.3, with a linear activation function $\sigma_{\text{linear}}(\cdot)$ in each consecutive layer, meaning $\sigma_{\text{linear}}(\mathbf{a}) = \beta\mathbf{a} + \mathbf{b}$. Assuming $\beta = 1$ and $\mathbf{b} = \mathbf{0}$, the output of the fist hidden layer can be expressed as,

$$\mathbf{h}^1 = \sigma_{\text{linear}}\left(\mathbf{W}_{h^1}\mathbf{x} + \mathbf{b}_{h^1}\right) = \mathbf{W}_{h^1}\mathbf{x} + \mathbf{b}_{h^1} \tag{2.5}$$

subsequently, the output of the second layer can be expressed as $\mathbf{h}^2 = \mathbf{W}_{h^2}\mathbf{h}^1 + \mathbf{b}_{h^2}$. Finally, the output of the network can be expressed as a linear combination of the original input, according to,

$$\mathbf{o} = \mathbf{W}_o\mathbf{h}^2 + \mathbf{b}_o = \mathbf{W}_o\left(\mathbf{W}_{h^2}\mathbf{h}^1 + \mathbf{b}_{h^2}\right) + \mathbf{b}_o = \mathbf{W}_o\left(\mathbf{W}_{h^2}\left(\mathbf{W}_{h^1}\mathbf{x} + \mathbf{b}_{h^1}\right) + \mathbf{b}_{h_2}\right) + \mathbf{b}_o = \mathbf{W}\mathbf{x} + B \tag{2.6}$$

This can subsequently be shown to be true for an arbitrary number (> 0) of layers and choices of β and $\mathbf{b}$. A n-layer deep fully connected network without any non-linear activation functions can thereby always be reduced to a simple linear regression.

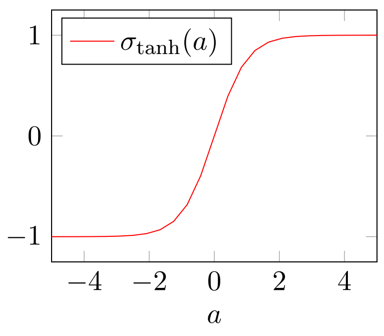

Figure 2.4: Tanh.

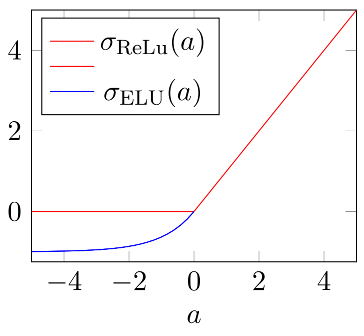

Figure 2.5: ReLU and ELU.

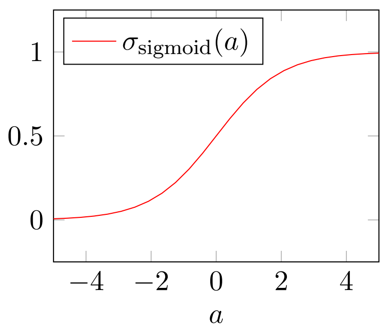

Figure 2.6: Sigmoid.

There exist many different activation functions, all with different properties and preferred uses. Some activation functions are illustrated by the figures 2.5, 2.4 and 2.6.

- **Sigmoid**
  The sigmoid activation function squashes the input to a limited range $[0, 1]$, both endpoints are reached asymptotically. The function is expressed by,

$$\sigma_{\text{sigmoid}}(a) = \frac{1}{1 + e^{-a}} \tag{2.7}$$

  The output becomes saturated as the input goes to either ends of the real axis. The sigmoid function has historically been the favored activation function, in large due to its biological analogy as a neuron which is either active $a = \infty$ or not active $a = -\infty$.

  The sigmoid activation function has an undesirable trait, as all activation functions with saturated regions. The gradient becomes close to zero and starts to vanish at the saturated regions. This introduces a significant problem when the network is under training and the gradient of the activation function is calculated.

  The sigmoid function is often used for binary-class logistic regression due to its relation to conditional probability measurements. The conditional probability of $c = 1$ given input data $a_1, a_2...a_n$ is expressed by $p(c = 1|\mathbf{a})$. The odds ratio is a measure of the ratio between the probability of a certain event e.g. $c = 1$ and the probability that the event does not accrues $c \neq 1$. The ratio is expressed by,

$$\text{odds}(p(c = 1|\mathbf{a})) = \frac{p(c = 1|\mathbf{a})}{1 - p(c = 1|\mathbf{a}))} \tag{2.8}$$

  If the log function is applied to the odds ratio the expression can be expressed as,

$$\log(\text{odds}(p(c = 1|\mathbf{a}))) = \log(p(c = 1|\mathbf{a})) - \log(1 - p(c = 1|\mathbf{a})) \tag{2.9}$$

  the new function is called the log-odds. Assuming the log-odds can be expressed as a linear combination of the n elements in $\mathbf{a}$, the odds ratio can be expressed by,

$$\frac{p(c = 1|\mathbf{a})}{1 - p(c = 1|\mathbf{a}))} = e^{\sum_{i=0}^{n} w_i a_i} \tag{2.10}$$

The conditional probability $p(c = 1|\mathbf{a})$ can now be re-formulated as the sigmoid function,

$$p(c = 1|\mathbf{a}) = \frac{1}{1 + e^{-\sum_{i=0}^{n} w_i a_i}} \tag{2.11}$$

The relationship between the sigmoid and the conditional probability is often exploited in the output layer of a network, when a binary class probability should be estimated.

The above theory can be generalized to the case of k-classes, such that the vector $p(c = i|\mathbf{a}), i \in [1, k]$ describes the probability for each class i. The softmax function is described by,

$$p(c = i|\mathbf{a}) = (\sigma(\mathbf{a})_{\text{softmax}})_i = \frac{e^{a_i}}{\sum_{j=1}^{k} e^{a_j}} \tag{2.12}$$

and is an extension of the sigmoid function to a multi class distribution [5][15].

- **The Hyperbolic Tangent**
  The tanh function squashes the input to a limited range $[-1, 1]$, the function is expressed by,

$$\sigma_{\text{tanh}}(a) = 2\sigma_{\text{sigmoid}}(2a) - 1 = \frac{2}{1 + e^{-2a}} - 1 \tag{2.13}$$

  The tanh function with its saturated regions share the undesirable trait of vanishing gradient with the sigmoid function.

- **Rectified Linear Unit**

  Because of ReLu's ability to significantly accelerate the convergence of neural network training methods, compared to tanh and the sigmoid activation function [22], the ReLu function has become increasingly popular in the past years. The function suppresses all negative values and lets all positive values through unhampered. The function is expressed by,

$$\sigma_{\text{ReLu}}(a) = \begin{cases} 0 & \text{if} \quad a < 0 \\ a & \text{if} \quad a \geq 0 \end{cases} \tag{2.14}$$

  The ReLu function has become one of the more prominent choices when constructing deep neural networks. Because of its linear representation on the positive real axis, the gradient flows unaffected from the loss-function to preceding layers in the network, in those cases $a > 0$. As a consequence of ReLu's ability to suppress all negative values, large parts of the network may become dead. During the training process some weights may be updated and become saturated on the negative real axis.

- **Exponential Linear Unit** In order to address the problem with the ReLu function, discussed above, other ReLu like activation functions have been introduced in the past years. One of those is the Exponential Linear Unit (ELU). The ELU can be seen in figure 2.5 and is expressed by,

$$\sigma_{\text{ELU}}(a) = \begin{cases} e^a - 1 & \text{if} \quad a < 0 \\ a & \text{if} \quad a \geq 0 \end{cases} \tag{2.15}$$

The ELU does not become saturated for all negative values as the ReLu. Negative values are however subdued. The ELU has shown to outperform ReLu in a number of cases, such as acceleration of the training process and increased performance during testing [11].

2.4 Convolutional Neural Networks

Inspired from the biological concept, convolutional neural networks are loosely based on the way in which the visual cortex in the brain processes visual information. The visual cortex has regions of cells dedicated to certain parts of the visual field, these specific regions are called receptive fields. The concept of receptive fields was introduced by the two neurophysiologists David Hubel and Torsten Wiese in 1959 [2]. Neighboring cells in the visual cortex have similar and overlapping receptive fields covering together the complete visual field. The two neurophysiologists went further and showed that the cells in the visual cortex are context sensitive, meaning, the cells become only active in response to certain specific visual stimuli, such as vertical/horizontal edges. The cells in the visual cortex are organized in layers which together produce the visual perception. In the following section we will see that convolutional neural networks are loosely based on the concepts behind receptive fields and hierarchical analysis of visual input.

2.4.1 Convolutional Layer

An image is affected by the *curse of dimensionality*, where each pixel correspond to one dimension. As the number of pixels increase, the space in which the image is expressed in grows exponentially. The *curse of dimensionality* is the term used to describe this undesirable relationship. The number of pixels in the image scale with the width w and height h of the image. A color image will also have three color channels. The image can thus be expressed as a tensor with dimensions $(w \times h \times 3)$. The first two dimensions are regarded as spatial and the last as a channel dimension. A fully connected layer containing 1000 neurons which receives an image $(160 \times 60 \times 3)$ as input will use approximately 29 million weights. Not only is the use of fully connected layers together with images often not computationally feasible, but each neuron in a fully connected layer lack knowledge of spatial relationships between pixels.

Convolutional layers introduce parameter-sharing by applying convolution operations on its input. A convolution kernel sweeps over the entire input, extracting features amongst spatially neighboring elements in a similar manner over the entire input. Similarly as the receptive fields, convolution kernels are able to extract context-specific features. Where some kernels are designed for the detection of edges and other for more complex relationships amongst spatially neighboring elements. Parameter-sharing in conjunction with the analysis of spatially neighboring elements of the convolution operation makes this type of layer especially adept in the analysis of images.

Convolutional neural networks are built by one or more convolutional layers, together producing a hierarchical analysis of the input image. Similarly as for fully connected layers, the output from a convolutional layer becomes the input to the following layer in the network. These types of networks have produced remarkable advances in fields spanning from image classification [38][40] to person re-identification [24][1][44].

Given a matrix $\mathbf{I}$ with dimensions $(w \times h)$ and a convolution kernel $\mathbf{K}$ $(k_1 \times k_2)$. The convolution operation is expressed as,

$$
\begin{aligned}
a_{i,j} &= (\mathbf{I} * \mathbf{K})\,(i,j) = \sum_{w}\sum_{h} I(w,h)K(i-w, j-h) \\
y_{i,j} &= \sigma(a_{i,j})
\end{aligned}
\tag{2.16}
$$

The kernel sweeps over the entire matrix **I**, resulting in an activation **a**. The activation is thereafter transformed according to the activation function σ, resulting in a feature map **y**. In order to ensure equal dimensions between **I** and **y**, **I** is often padded with zeros, such that the convolution operation 2.16 is extended outside the original borders of **I**. The matrix dimensions can thus be keep constant between convolution operations, in such a case the matrix **I** is said to be same-padded. Figure 2.7 shows a convolution operation with a (3×3) kernel performed on a same-padded matrix. The gray patches below represent the original matrix elements, the white patches represent zero padded elements. The (3×3) kernel sweeps over the matrix, resulting in the output matrix shown above.
Stride is the number of elements moved per step of the kernel when sweeping over the matrix. Convolution with stride larger than one is equivalent to convolution with stride one, followed by down sampling. In those cases the stride is two, the convolution kernel jump two elements as it sweeps over the matrix, resulting in an activation with half the spatial dimensions as the input, in those cases the matrix is same-padded.

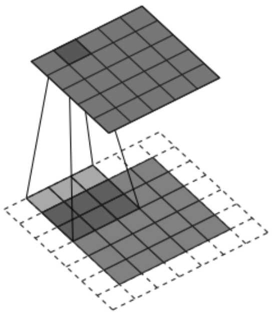

Figure 2.7: Convolution operation performed on a same padded matrix.

The convolution kernel size determines the spatial relationship amongst spatially neighboring elements in **I** that can be analyzed. A small kernel size analyze close local relationships amongst the spatially neighboring elements, whereas a larger kernel analyze larger local spatial relationships.
Applying two succeeding convolution operations with small kernels is equivalent to one convolution operation with a larger kernel [15]. Analysis of relationships between far away spatially neighboring elements can thereby be performed by applying multiple convolution operations with small kernels successively. However, a non-linear activation function applied in between the convolution operations makes the two cases no longer equivalent. Commonly used kernel sizes are (3×3) and (5×5) [38][5].

Convolutional layers extend often the convolution operation to the processing of a tensor **T** $(w \times h \times c)$. The convolutional kernel is extended to a volume with dimensions $(k_1 \times k_2 \times c)$, this volume is often called convolutional filter in neural network terminology. The convolutional filter sweeps over the width and height of the tensor $\mathbf{T}$. The dot product between the filter and the tensor is calculated as the filter sweeps over the tensor. A same-padded tensor inputted to a convolutional layer results in a feature map with dimensions $(w \times h)$. Added to the feature map is a bias term. By applying c^* filters to the same-padded tensor $\mathbf{T}$ and stacking the resulting feature maps along the channel dimension, $\mathbf{T}$ transforms into a new tensor $\mathbf{T}^*$, with dimensions $(w \times h \times c^*)$.

Cross channel parametric pooling is the method of aggregate cross channel information into a dense representation [27]. By applying a (1×1) convolutional filter, cross channel information is summed up and concentrated into a dense representation, serving to remove cross-channel dependencies. This pooling method is trainable, contrary to many other pooling methods such as max-pooling, discussed in the following section.

2.4.2 Max-pooling

The purpose of a spatial pooling method is to mitigate small distortions in the input, thereby introducing translations invariance of sought-after features within the input. The most common spatial pooling method is max-pooling. The max-pooling operation examines a small patch of a feature map and outputs the maximum value within this patch. The max-pooling patch sweeps over the entire input, similarly to a convolution kernel. The max-pooling operation is often applied with a stride larger than one, effectively reducing the spatial dimensions of the input. Max-pooling can thereby also serve as a means of reducing the computational burden in succeeding layer. Figure 2.8 shows the max-pool operation with a patch size (3×3) applied to the left matrix. The max-pooling is performed with stride one. The right matrix shows the result. The max-pooling operation is applied to each channel separately.

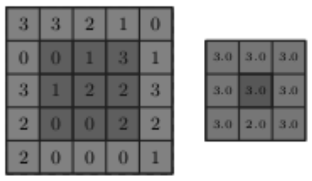

Figure 2.8: Max-pooling with stride one applied to the left non-padded matrix, the matrix to the right shows the result.

2.4.3 Convolutional Neural Network Architectures

The two different types of pre-trained convolutional neural networks used in the thesis will be introduced in this section.

InceptionV1

InceptionV1 introduced in 2014, was a state of the art convolutional neural network object classifier when it was introduced [40] . The network receives one image with spatial dimensions

(224 × 224) as input and output the class of the main object present in the image. InceptionV1 has the ability to classify 1000 different objects. By replacing large convolution kernels with multiple successive convolutional operations with small kernel sizes, the authors argue that a more effective way to analyze and extract features from images can be achieved. The InceptionV1 network outperformed all other competitors in the 2014 ImageNet classification challenge [12].

You Only Look Once V2 (YOLOv2)

YOLOv2 is a state of the art object detector [32]. YOLOv2 is a convolutional neural network which performs object detections on an input image. Associated with every detection is a bounding box, which marks the object's location in the image. The YOLOv2 has the ability to detect many different objects, spanning from chairs, tv monitors to persons. The network is designed for real time applications, on certain graphical processing units can as many as 40-90 images be processed per second [32]. Figure 2.9 shows detections made by YOLOv2.

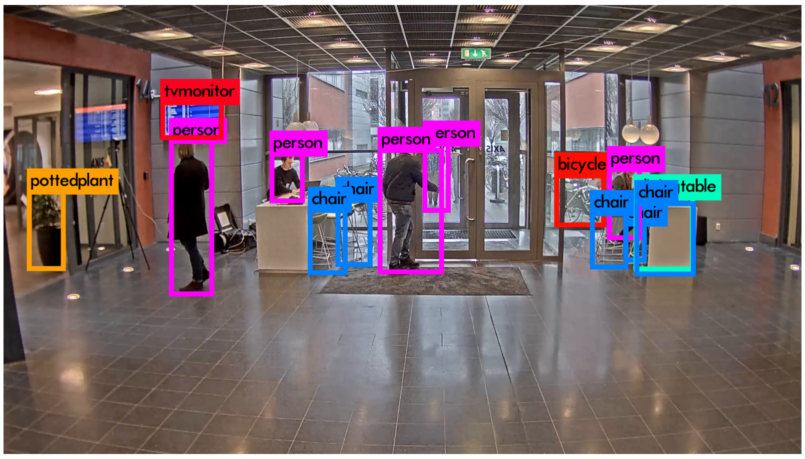

Figure 2.9: Detections made by YOLOv2 [32].

2.5 Training of Neural Networks

The loss function L describes the overall performance of a neural network given an input $\mathbf{x}$ and label $\mathbf{l}$. The label describes the sought-after network output given the input. A lower loss value indicates better performance. A property that is shared amongst all loss functions is differentiability, meaning, the gradient of the loss function can be calculated. In order to drive the network towards a state which yields lower loss value, the gradient of the loss function is calculated with respect to the weights of the network. During training the weights of the network are adjusted in the negative gradient direction, lowering the loss value. In order to calculate the gradient the backpropagation method is used. Two different steps are performed in this method, the forward pass and the backward pass. These steps will be presented in the following sections. Methods of updating the network's weights given the gradient calculations will also be examined.

2.5.1 Forward Pass

The forward pass calculates the output from one layer with respect to its input. The forward pass is recursively calculated from the input layer to the output layer in the network. If the α:th layer's output is denoted by the non-linear function $\mathbf{y}_\alpha = \phi_\alpha(\mathbf{y}_{\alpha-1}; \mathbf{\Theta})$, the final output from the last layer can be expressed as $\mathbf{y}_n = \phi_n(\mathbf{y}_{n-1}; \mathbf{\Theta})$. Where $\mathbf{\Theta}$ is the set of weights that parametrize the network. Given a loss function L and a label $\mathbf{l}$, the loss value can be expressed as $L(\mathbf{y}_n; \mathbf{\Theta}, \mathbf{l})$. The forward pass of a feedfoward layer is described by equation 2.4 and the forward pass of a convolutional layer is descried in the section dedicated to convolutional layers.

2.5.2 Backward Pass

The backward pass examines how different changes to the weights of the network affect the loss value. The backward pass is essentially a repeated action of chain rule calculations for partial derivatives. The gradient of the loss function with respect to $\mathbf{y_n}$ is passed backward from the output layer to succeeding layers in the network. The gradient of the loss function with respect to the weight between neuron i in layer $l-1$ and neuron j in layer l can be expressed as,

$$\frac{\partial L(\mathbf{y}_n; \mathbf{\Theta}, \mathbf{l})}{\partial w_{j,i}} = \frac{\partial L(\mathbf{y}_n; \mathbf{\Theta}, \mathbf{l})}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{j,i}} \tag{2.17}$$

Where y_j is the output from neuron j and a_j is the neuron's activation. The partial derivative of the output from neuron j with respect to its activation is expressed by,

$$\frac{\partial y_j}{\partial a_j} = \frac{\partial}{\partial a_j} \sigma(a_j) \tag{2.18}$$

The delta-rule introduces a compact notation for equation 2.17,

$$\delta_j = \frac{\partial L(\mathbf{y}_n; \mathbf{\Theta}, \mathbf{l})}{\partial y_j} \frac{\partial y_j}{\partial a_j} \tag{2.19}$$

the δ:s can be calculated successively for each layer, given δ:s for each neuron in layer $l+1$, according to [16],

$$\delta_j = \frac{\partial}{\partial a_j} \sigma(a_j) \sum_{k \in \Omega} \delta_k w_{k,j} \tag{2.20}$$

where Ω is the set of neurons in layer $l+1$ connected to neuron j in layer l. The equation 2.17 can thereby be re-expressed as [16],

$$\frac{\partial L(\mathbf{y}_n; \mathbf{\Theta}, \mathbf{l})}{\partial w_{j,i}} = \delta_j \frac{\partial a_j}{\partial w_{j,i}} = \delta_j y_i \tag{2.21}$$

These calculations continue in a recursive fashion from the output layer until the input layer is reached. Each weight's contribution to the loss value can subsequently be calculated by the use of the backward pass algorithm. Figure 2.10 illustrates schematically how δ:s are passed backward from one layer to another.

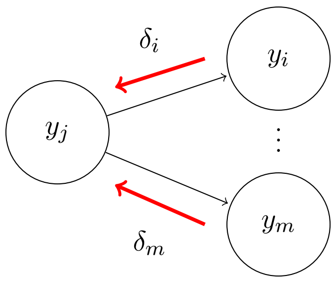

Figure 2.10: The figure illustrates schematically how δ:s are passed backward from neurons $i...m$ in layer $l+1$ to neuron j in layer l.

2.5.3 Gradient Descent

The gradient descent algorithm is a first order weight optimization method. The optimization method is applied iteratively, meaning, the set of trainable weights $\mathbf{\Theta}$ is updated incrementally. A first order optimization method examines only the gradient, contrary to higher order methods which examine higher order derivatives as well. The computational burden of higher order methods makes them often unfeasible for deep neural network training. The gradient descent is thereby the preferred method for deep neural networks training [15].

Given a set of trainable weights $\mathbf{\Theta}$, a weight $w_{j,i} \in \mathbf{\Theta}$ and a loss function $L(\mathbf{y}_n; \mathbf{\Theta}, \mathbf{l})$, the gradient descent algorithm updates the weight iteratively at each time step t according to [5],

$$w_{j,i}^{t} = w_{j,i}^{t-1} - \alpha \frac{\partial L(\mathbf{y}_n; \mathbf{\Theta}, \mathbf{l})}{\partial w_{j,i}^{t-1}} \tag{2.22}$$

where α is the learning rate. The learning rate expresses the rate in which the weights are updated. In order to achieve weight convergence, it is essential to decrease the learning rate depending on the number of past update steps t. Given that each update step move the weights to a more favorable representation, a non-decreasing learning rate may force the weights to lose some of their already conducted optimization. A way to address this problem is by linearly decreasing the learning rate according to,

$$\alpha_t = (1-\epsilon)\alpha_0 + \epsilon\alpha_\tau \tag{2.23}$$

where $\epsilon = \frac{t}{\tau}$, α_τ is the static learning rate and α_0 is the initial learning rate. The learning rate will linearly decrease from α_0 to α_τ as $t \rightarrow \tau$. Commonly, the learning rate is kept constant when $t > \tau$. Figure 2.11 illustrates schematically the concept behind gradient descent. The surface represents the loss function and the arrow represents the negative gradient direction.

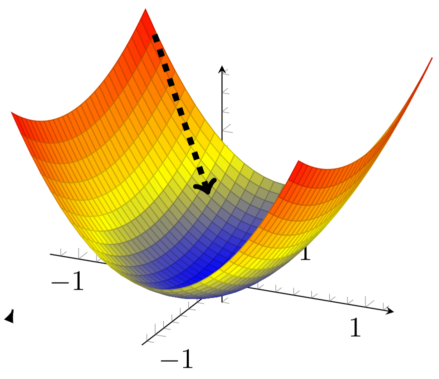

Figure 2.11: Schematic illustration of the gradient descent method, the surface represents the loss function and the arrow represents the negative gradient direction.

Calculation of the gradient in equation 2.22 for all the examples in the training dataset p_{training} is often computationally unfeasible. The gradient is instead calculated for a randomly chosen small subset of training examples, this subset is called a batch. The gradient is averaged over all examples in the batch, contrary to all examples in the training dataset. The gradient descent with batches introduces a stochastic approach to weight optimization. The stochastic approach assures somewhat robustness towards getting trapped in local minimas. If the whole training set is used to compute the gradient, the network will risk getting stuck in one of the many local minimas. The gradient descent algorithm will in such a case not be able to drive the network to escape the local minimia, since all gradients will be close to zero. If only one example is used to form the batch, the gradient estimates may be too noisy, resulting in bad weight updates.

It is necessary that the batches are formed randomly, such that two consecutive gradient estimates are independent, ensuring an unbiased estimate of the expected gradient from the set of samples [5].

2.5.4 Adaptive Moment Estimation Optimizer

The Adaptive Moment Estimation (ADAM) optimizer calculate adaptive learning rates for each trainable weight in the gradient descent optimization scheme. The ADAM optimizer keeps exponentially decaying estimates of the first moment $\boldsymbol{\mu}$ and the second uncentered moment $\mathbf{v}$ of the gradient, for each weight. The ADAM optimizer has shown to outperform numerous other optimization methods in terms of convergence speed and overall loss minimization performance [21].

The different steps in the ADAM optimization scheme are,

- Initialize $\boldsymbol{\mu}$ and $\mathbf{v}$ to zero for all trainable weights. This initialization makes the two estimates bias towards zero, however, corrections are made at each new time step to counteract this [21].

- Draw a batch from the training dataset p_{training} with N examples $\mathbf{x}_1, \mathbf{x}_2 ... \mathbf{x}_N$ and corresponding labels $\mathbf{l}_1, \mathbf{l}_2 ... \mathbf{l}_N$ at time t. Calculate the network's output $\mathbf{y}_{n,1}, \mathbf{y}_{n,2} ... \mathbf{y}_{n,N}$ given the input examples. Evaluate the expected loss given the batch,

$$\mathbb{E}\left(L(\mathbf{y}_n; \mathbf{\Theta}_{t-1}, \mathbf{l}) | (\mathbf{x}, \mathbf{l}) \sim p_{\text{training}}\right) = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{y}_{n,i}; \mathbf{\Theta}_{t-1}; l_i) \tag{2.24}$$

- From the expected loss, calculate the gradient with respect to the set of weights $\mathbf{\Theta}$,

$$\mathbf{g}_t = \mathbb{E}\left(\nabla_{\mathbf{\Theta}_{t-1}} L(\mathbf{y}_n; \mathbf{\Theta}_{t-1}, \mathbf{l}) | (\mathbf{x}, \mathbf{l}) \sim p_{\text{training}}\right) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mathbf{\Theta}_{t-1}} L(\mathbf{y}_{n,i}; \mathbf{\Theta}_{t-1}, l_i) \tag{2.25}$$

- Update the biased exponentially decaying first moment estimate, $\beta_1 \in [0, 1)$ determines the decay rate,

$$\boldsymbol{\mu}_t = \beta_1 \boldsymbol{\mu}_{t-1} + (1 - \beta_1)\mathbf{g}_t \tag{2.26}$$

- Update the biased uncentered exponentially decaying second moment estimate, $\beta_2 \in [0, 1)$ determines the decay rate,

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2 \tag{2.27}$$

- Correct the zero-bias estimates of $\boldsymbol{\mu}_t$ and $\mathbf{v}_t$ [21]. The bias corrected estimates converge to the exponentially decaying estimates as $t \to \infty$,

$$\begin{aligned} \hat{\boldsymbol{\mu}}_t &= \frac{\boldsymbol{\mu}_t}{1 - \beta_1^t} \\ \hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1 - \beta_2^t} \end{aligned} \tag{2.28}$$

- Update the set of weights $\mathbf{\Theta}$, where $1./\sqrt{\hat{\mathbf{v}}_t}$ adaptively scale the global learning rate α for each weight [21],

$$\mathbf{\Theta}_t = \mathbf{\Theta}_{t-1} - \alpha \hat{\boldsymbol{\mu}}_t./\sqrt{\hat{\mathbf{v}}_t} \tag{2.29}$$

  ./ indicates element-wise division.

The exponentially decaying first moment gradient estimates ensure that unwanted noise due to the stochastic approximations are subdued. The exponentially decaying estimations of the second uncentered moment are used to adaptively scale the learning rate for each weight in the update step in equation 2.29. Usual values for the hyper parameters in the ADAM optimizer scheme are, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\alpha = 0.001$ [21].

2.6 Transfer Learning

Transfer learning is the idea of applying a trained model/network on tasks different from its initially designed objective. The ability of a network to perform a certain task can often be transfered to another, i.e. knowledge can be transfered from one domain to another. A deep neural network extracts features in a hierarchical manner, where the feature extractions at each layer become increasingly task specific closer to the output layer [46]. Consequently, the early layers in the network extract general features applicable to numerous of tasks different from the initial objective of the network. Transfer learning also goes under the name *fine tuning*. One of the hardest problems to overcome when training a large network is the lack of sufficient data. Huge datasets for image classification can be used to pre-train a network [12]. The network can thereafter be fine-tuned on the actual task, often with a lower learning rate for layers close to the input layer. Deep neural networks can thereby be trained to perform tasks otherwise not possible without the concept of transfer learning.

2.7 Regularization

Deep neural networks are extremely prone to over-fitting. Over-fitting is a term used to describe a network which is not able to generalize past the training examples i.e. the network has learned the training dataset and not a generalized model. Given the complexity and the number of weights in a multilayer network, over fitting is easily achieved. Fully connected layers are especially prone to over fitting, due to their sheer amount of weights. Parameter sharing, adopted by convolutional and recurrent networks significantly mitigates the risk of overfitting, due to the reduction of weights.

Regularization is the method of imposing certain constraints on the ability of a network to adapt to the training dataset. There are a myriad of regularization methods often used to significantly increase the generalization ability of a network. Such methods will be presented in the following sections. Data augmentation, also a regularization method, will be presented in the section dedicated to the batch generation aspect.

2.7.1 Dropout

Dropout is a regularization method, closely resembling ensemble training. Ensemble training is the method of training multiple different networks separately on one objective. The outputs/-classifications are averaged in a test situation. The premise behind such an approach is, different networks together perform better than one alone.

Dropout remove neurons at random during training, subsequently creating a new network where the removed neurons are not present. Usual probabilities for the occurrence of random neuron removal span from 50-90 % [39]. Applying dropout during training serves as a method of removing co-adaptation amongst different neurons, enabling the network to learn a redundant representation.

Figure 2.12 shows a schematic representation of the multilayer fully connected network represented in figure 2.3 after dropout has been applied to the first hidden layer. Neuron h_2^1 and all connections to it have been removed, resulting in a similar but different network.

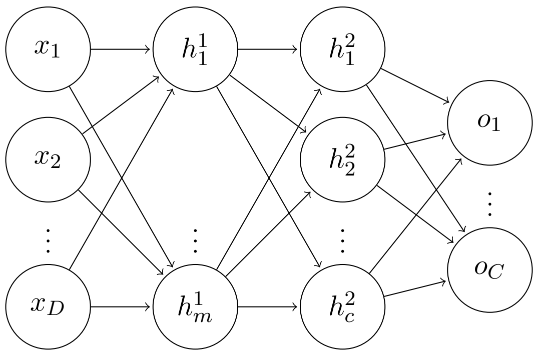

Figure 2.12: Dropout applied to the first hidden layer in the multilayer fully connected network seen in figure 2.3, neuron h_2^1 has been randomly removed.

2.7.2 Layer Normalization

Layer normalization, introduced in 2016, has shown significant promise in its ability to regularize a network [23]. The method normalizes the activation of layer l to have a trainable mean β and variance γ. Each output from layer l, for all examples in the batch, becomes subsequently drawn from a distribution with the same mean and variance. This normalization serves not only as a regularization method but also resulting in a speedup of the training process [23].

If the dynamics for layer l is described by $\mathbf{y}_l = \sigma(\mathbf{a}_l)$, the mean activation preceding the activation function is expressed by,

$$\mu_l = \frac{1}{N}\sum_{i=1}^{N} a_{l,i} \tag{2.30}$$

where $a_{l,i}$ is one of the N activations in layer l. The activation variance for layer l is expressed by,

$$\text{var}(\mathbf{a}_l) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mu_l - a_{l,i})^2} \tag{2.31}$$

Layer normalization normalize the activation $\mathbf{a}_l$ according to,

$$LN(\mathbf{a}_l) = \frac{\gamma}{\text{var}(\mathbf{a}_l)}(\mathbf{a}_l - \mu_l) + \beta \tag{2.32}$$

The output from layer l can thereby be expressed as,

$$\mathbf{y}_l = \sigma\left(\frac{\gamma}{\text{var}(\mathbf{a}_l)}(\mathbf{a}_l - \mu_l) + \beta\right) \tag{2.33}$$

The layer normalization method, applied before the activation function, mitigate undesired saturation, present for some activation functions.

As argued, the layer normalization scheme works exceptionally well when applied to recurrent units, especially for long sequences and small batch sizes [23]. Significant speedup and performance gain can be achieved by applying layer normalization preceding every non-linear activation function in a recurrent unit. Layer normalization can subsequently be applied in the LSTM and the GRU and their conv-versions in order to gain speedup and increased performance. The LSTM and the GRU will be presented in the following section. In those cases layer normalization is used in a convolutional layer, each channel is normalized separately.

2.8 Recurrent Neural Networks

Recurrent units extend neutral networks to the processing of sequential external input data $\mathbf{x}_0, \mathbf{x}_1, ...\mathbf{x}_t$. A recurrent unit mimics a dynamic system $g(\mathbf{x}_t, \mathbf{h}_{t-1}; \boldsymbol{\omega}_t)$, driven by external input $\mathbf{x}_t$ and past state values $\mathbf{h}_{t-1} = g(\mathbf{x}_{t-1}, \mathbf{h}_{t-2}; \boldsymbol{\omega}_{t-1})$. The dynamic system is parametrized at time step t by $\boldsymbol{\omega}_t$. The current state of the recurrent unit depends consequently on the current external input $\mathbf{x}_t$ and all past states in a recursive fashion $\mathbf{h}_t = g(\mathbf{x}_t, g(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}); \boldsymbol{\omega}_{t-2})$. Figure 2.13 illustrates temporal dependencies between the states of the recurrent unit at each time step.

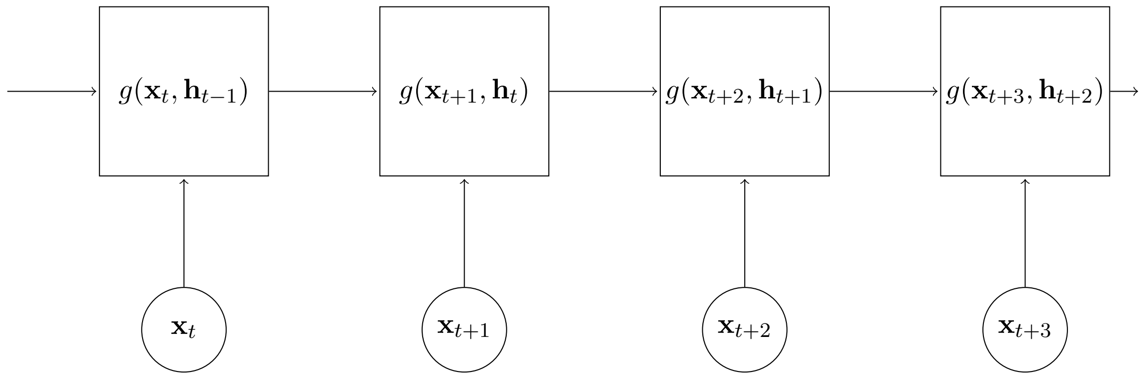
$g(\mathbf{x}_t, \mathbf{h}_{t-1})$
$g(\mathbf{x}_{t+1}, \mathbf{h}_t)$
$g(\mathbf{x}_{t+2}, \mathbf{h}_{t+1})$
$g(\mathbf{x}_{t+3}, \mathbf{h}_{t+2})$
$\mathbf{x}_t$
$\mathbf{x}_{t+1}$
$\mathbf{x}_{t+2}$
$\mathbf{x}_{t+3}$

Figure 2.13: An unfolded recurrent unit.

A network with one or more recurrent units is often called a recurrent neural network. Recurrent neural networks borrow the concept of parameter sharing from convolutional neural networks. The weight vector which parameterize the network is constant for all time steps, meaning $\boldsymbol{\omega}_t = \boldsymbol{\omega}$. Parameter sharing amongst the different time steps ensures a network which is able to operate on theoretically all sequence lengths. Consequently, the resulting network also becomes invariant to temporal shifts present in the input sequence.

In those cases a convolutional neural network is used in conjunction with one or more recurrent units the network is often called a convolutional recurrent neural networks. Convolutional recurrent neural networks have the ability to extract spatio-temporal information from sequences of images. These types of networks have produced remarkable advances in a number of fields in the past years, spanning from video sequence classification [6][3][13] to the field of multi shot re-identification [43][30].

2.8.1 Back Propagation Through Time

In order to train a recurrent neural network the standard back propagation algorithm is extended to Back Propagation Through Time (BPTT). The idea behind BPTT is to unfold the recurrent neural network through time. Figure 2.13 illustrates an unfolded recurrent unit. Given that the sequence length of the external input data is $t_{\max}$, the unfolding process will create $t_{\max}$ copies of the network, all with identical weights. The idea behind BPTT algorithm is to let the δ:s in equation 2.21 propagate from the loss function through the entire unfolded graph, in a similar fashion as in the standard back propagation algorithm. Because the weights are shared amongst the different network copies, the gradients for each weight in equation 2.21 are summed up for all time steps, resulting in a single gradient $\frac{\partial L(\mathbf{y}_n; \boldsymbol{\Theta}, \mathbf{l})}{\partial w_{i,j}}$ for each weight $w_{i,j}$ [16]. Recurrent neural networks can thereby be trained in a similar fashion as a non-recurrent network.

2.8.2 Recurrent Units

This section will introduce three different types of recurrent units that can be used to construct a recurrent neural network. The difficulty of training recurrent neural networks will also be discussed. Methods to mitigate certain shortcomings of the standard recurrent unit will be presented and analyzed.

2.8.2.1 Standard Recurrent Unit

The standard recurrent unit with a linear activation function, $\sigma_{\text{linear}}(\mathbf{a}) = \beta\mathbf{a} + \mathbf{b} = \mathbf{a}$, can be expressed by,

$$\mathbf{h}_t = \sigma_{\text{linear}}\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}\right) = \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b} \tag{2.34}$$

where $\mathbf{W}$ and $\mathbf{U}$ are weight matrices associated with the state $\mathbf{h}$ and the external input $\mathbf{x}$, $\mathbf{b}$ is a bias term. The standard recurrent unit is troubled by two unwanted problems, exploding gradients and vanishing gradients. These problems can easily be understood by further examination of equation 2.34. If the external input $\mathbf{x}$ and $\mathbf{b}$ are neglected (or thought of as zero) from the equation, the recurrent unit can be re-expressed as $\mathbf{h}_t = \mathbf{W}\mathbf{h}_{t-1}$. By expanding the recurrent relationship, the state at a given time step t is simplified to [5],

$$\mathbf{h}_t = \mathbf{W}^t\mathbf{h}_0 \tag{2.35}$$

If the matrix $\mathbf{W}$ is square ($N \times N$), with N linearly independent eigenvectors an eigendecomposition can be performed, such that $\mathbf{W}$ is expressed as [5],

$$\mathbf{W} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\text{-}1} \tag{2.36}$$

where the columns in $\mathbf{Q}$ represent eigenvectors and the diagonal matrix $\mathbf{\Lambda}$ represents the eigenvalues. Equation 2.35 can thus be re-expressed as [5],

$$\mathbf{h}_t = \mathbf{Q}\mathbf{\Lambda}^t\mathbf{Q}^{-1}\mathbf{h}_0 \tag{2.37}$$

Each eigenvalue λ_i is consequently multiplied with itself t times. The gradient $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_0} = \mathbf{Q}\mathbf{\Lambda}^t\mathbf{Q}^{-1}$ will thereby decay towards zero or explode as t becomes large, depending on the magnitude of the eigenvalues.

The commonly used activation function of the standard recurrent unit in equation 2.34 is tanh. The vanishing and exploding gradient problem is also present for tanh and other non-linear activation functions with external inputs and biases different from zero [31]. The exploding gradient problem can be redeemed by thresholding the norm of the gradient. It has been shown that such a constraint does not severely affect the network's ability to learn temporal relationships within the external data [31][19]. The vanishing gradient problem is a harder problem to be solved. The ability of a network to propagate information from one time step to another is hampered by short time fluctuations which tend to dominate the contributions to the hidden state [5]. The vanishing gradient problem imposes thus a severe problem when long-term dependencies should be learned. In order to address the discussed shortcomings of the standard recurrent unit, more advanced recurrent units have been introduced. Two of those units will be presented in the following sections.

2.8.2.2 Long Short Term Memory

Long Short Term Memory (LSTM) was first introduced in 1997 in the attempt to address the vanishing gradient problem [35]. The LSTM is especially adept in the analysis of long term dependencies in sequences of data. This ability is largely attributed to the way in which new information is added to the internal states. Different types of gates, modeled by sigmoid functions, control the flow of information between the external input $\mathbf{x}_t$, the hidden state $\mathbf{h}_t$ and the cell state $\mathbf{c}_t$. Three different gates are used by the LSTM, the forget gate $\mathbf{f}_t$, the input gate $\mathbf{i}_t$, and the output gate $\mathbf{o}_t$. Through the use of gates, which control the flow of information, the LSTM has the possibility to add or remove data from the internal states in a contextual manner.

The cell state $\mathbf{c}$ is at time step t expressed by,

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{2.38}$$

The cell state has a linear self-loop, $\mathbf{f}_t \odot \mathbf{c}_{t-1}$, which enables the gradient to flow freely from one time step to another. The LSTM is thereby not severely affected by the vanishing gradient problem. The candidate hidden state $\tilde{\mathbf{c}}_t$ will be introduced further down. The linear self-loop is controlled by element-wise multiplications, denoted $\odot$, with the forget gate. The forget gate is expressed by,

$$\mathbf{f}_t = \sigma_{\text{sigmoid}} \left(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f \right) \tag{2.39}$$

The forget gate examines the current external input $\mathbf{x}_t$ and the past hidden state $\mathbf{h}_{t-1}$ in order to express how much of past cell state information $\mathbf{c}_{t-1}$ that should pass on to the current cell state $\mathbf{c}_t$. The current external input $\mathbf{x}_t$ $(n \times 1)$, is multiplied with the weight matrix $\mathbf{W}_f$ $(m \times n)$. Where m is the length of the cell state vector. The hidden state $\mathbf{h}_t$ $(m \times 1)$ undergoes a similar transformation by multiplication with the weight matrix $\mathbf{U}_f$ $(m \times m)$. Added to the summations of the two transformations is a bias term $\mathbf{b}_f$. If the bias term would be initialized to zero (or drawn from a zero-mean distribution) the forget gate would be initialized around 0.5. As a result, the gradient would diminish by a factor ≈ 0.5 at each time step [19]. The vanishing gradient problem returns. If however $\mathbf{b}_f$ is initialized to a large value, the forget gate will be initialized close to one, thereby causing the vanishing gradient problem to go away. The special initialization of $\mathbf{b}_f$ is only valid for the forget gate, which controls the important self loop of the cell state.

New information $\tilde{\mathbf{c}}_t$ is also added to the cell state at each time step, $\tilde{\mathbf{c}}_t$ is called the candidate cell state and is expressed by,

$$\tilde{\mathbf{c}}_t = \sigma_{\tanh} \left(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c \right) \tag{2.40}$$

The candidate state gathers information from $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ by means of vector/matrix multiplications with the weight matrices $\mathbf{W}_c$ and $\mathbf{U}_c$, added to the summation is a bias term $\mathbf{b}_c$. The result undergoes a non-linear transformation specified by $\sigma_{\tanh}$. The tanh function is often used as the candidate cell state activation function [28][19].

The input gate $\mathbf{i}_t$ controls the flow of new information from the candidate cell state to the cell state in equation 2.38. The input gate operates in a similar fashion as the forget gate. The input gate is expressed by,

$$\mathbf{i}_t = \sigma_{\text{sigmoid}}\left(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i\right) \tag{2.41}$$

The updated cell state $\mathbf{c}_t$ is thereafter used to produce the new hidden state $\mathbf{h}_t$,

$$\mathbf{h}_t = \mathbf{o}_t \odot \sigma_{\tanh}\left(\mathbf{c}_t\right) \tag{2.42}$$

where the cell state is firstly passed through an activation function $\sigma_{\tanh}$ and thereafter element-wise multiplied with the output gate. The output gate is defined according to,

$$\mathbf{o}_t = \sigma_{\text{sigmoid}}\left(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o\right) \tag{2.43}$$

The hidden state $\mathbf{h}_t$ is thereafter passed on to the succeeding layer in the recurrent neural network.

When dealing with sequences of images, the use of the LSTM becomes problematic. The LSTM is derived for the use of vectors as input, not feature maps. A feature map can be flattened into a single vector, making it a valid input. However, the flattened vector does not share the same obvious spatial relationship between elements as the feature map. The large dimension of the flattened feature map makes the computational burden often unfeasible. One way to remedy this shortcoming of the standard LSTM is the use of convolution operations instead of vector/matrix multiplications in the LSTM equations [36]. If $\mathbf{x}_t$,$\mathbf{h}_t$ and $\mathbf{c}_t$ are thought of as feature maps, and all weight matrices as convolutional filters. All vector/matrix multiplications in the LSTM equations can be replaced by convolutional operations. This re-defined version of the LSTM is called convolutional-LSTM (conv-LSTM). The convolutional operations in the conv-LSTM make this re-defined version aware of spatial relationships, inherent in feature maps. This in conjunction with its ability to extract temporal information makes the conv-LSTM a good choice when sequences of images should be analyzed and spatio-temporal information extracted. The LSTM and the conv-LSTM have been applied to a myriad of different applications, spanning from scene labeling to weather forecasting [3][6][36].

2.8.2.3 Gated Recurrent Unit

The Gated Recurrent Unit (GRU) was introduced in 2014 [9], in an attempt to simplify the LSTM [19]. Similarly as the LSTM, the GRU have gates modeled by sigmoid functions. The GRU has only one internal state $\mathbf{h}_t$ and two gates, the reset gate $\mathbf{r}_t$ and the update gate $\mathbf{z}_t$. The GRU is subsequently a less complex version of the LSTM. However, despite fewer parameters and overall complexity, the GRU has shown to outperform the LSTM on most tasks [19].

In order to overcome the vanishing gradient problem the GRU borrows the idea of a linear self loop between time steps from the LSTM. The hidden unit $\mathbf{h}_t$ is updated at each time step according to,

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (\mathbf{1} - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t \tag{2.44}$$

The update gate $\mathbf{z}_t$ controls the linear self loop between past hidden states $\mathbf{h}_{t-1}$ and the current state $\mathbf{h}_t$ $(m \times 1)$ in a contextual manner. This is achieved by observing the past hidden state and current external input $\mathbf{x}_t$ $(n \times 1)$. The two weight matrices $\mathbf{W}_z$ $(m \times n)$, $\mathbf{U}_z$ $(m \times m)$ and the bias term $\mathbf{b}_z$ $(m \times 1)$ parameterize the update gate. The update gate is expressed by,

$$\mathbf{z}_t = \sigma_{\text{sigmoid}}\left(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z\right) \tag{2.45}$$

All values in $\mathbf{z}_t$ are always in the range $[0,1]$ for all inputs, all values in $\mathbf{1}-\mathbf{z}_t$ become thus also limited to the same interval. The update gate $\mathbf{z}_t$ measures the importance of past information, whereas $\mathbf{1}-\mathbf{z}_t$ measures the importance of new information receding in the candidate hidden state $\tilde{\mathbf{h}}_t$. The update gate performs thereby similar tasks as the forget gate and the input gate in the LSTM. The candidate hidden state is expressed by,

$$\tilde{\mathbf{h}}_t = \sigma_{\tanh}\left(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) + \mathbf{b}_h\right) \tag{2.46}$$

the term $\mathbf{W}_h\mathbf{x}_t$ incorporates new information from the external input. Whereas, the reset gate $\mathbf{r}_t$ in $\mathbf{U}_h\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right)$ determines the importance of the past hidden state. A bias term $\mathbf{b}_h$ is added to the resulting transformation before passed through an activation function $\sigma_{\tanh}$. The reset gate is expressed in a similar fashion as the update gate,

$$\mathbf{r}_t = \sigma_{\text{sigmoid}}\left(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r\right) \tag{2.47}$$

The updated hidden state $\mathbf{h}_t$ in equation 2.44 is used as input to the succeeding layer in the network.

The GRU can be modified, similarly how the LSTM was modified into a conv-LSTM [36], into a convolutional Gated Recurrent Unit (conv-GRU). The conv-GRU has successfully been applied in a multi shot re-identification system [43].

2.8.3 Temporal Pooling

From a sequence of n external inputs $\mathbf{x}^{1..n} = \mathbf{x}^1, \mathbf{x}^2..\mathbf{x}^n$ a recurrent unit produces n outputs $\mathbf{h}^{1..n} = \mathbf{h}^1, \mathbf{h}^2...\mathbf{h}^n$. If the entire sequence $\mathbf{x}^{1..n}$ is used to perform a single classification or embedding then $\mathbf{h}^{1..n}$ must be reduced into a single representation, such that sequences of different lengths can be compared. A temporal pooling function $T(\cdot)$ takes a sequence as inputs and performs a pooling operation, such that the sequence representation is reduced from $\mathbf{h}^{1..n}$ to $T(\mathbf{h}^{1..n}) = \mathbf{h}^*$. Two main temporal pooling methods exist,

- A last output pooling function returns the last output $\mathbf{h}^n$. The idea behind such an approach is, given a sequence the output of the recurrent unit at time step n has converged into a representation which retains the essential information from past time steps.
- The temporal mean pooling function takes the sequence of outputs $\mathbf{h}^{1..n}$ from the recurrent unit and average them according to,

$$T(\mathbf{h}^{1..n}) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}^i \tag{2.48}$$

Temporal mean pooling is frequently used in multi shot re-identification systems [30][43].

2.9 Similarity Measurements

A common nontrivial task in image analysis is the examination of the similarity between two images $\mathbf{I}_i$ and $\mathbf{I}_j$. The naive way to express the similarity would be to apply a similarity metric directly on the unprocessed images i.e. examine the similarity between corresponding pixels. However, if the objective of the similarity measurement is the detection of similar objects between two images one realizes quickly that such a method is pointless. The pixel-wise similarity measurement between the two images is not shift/scale invariant i.e. the position and scale of the sought-after object/objects affects the similarity measurement. Such a method can not put the single pixel in a context i.e. associate a certain attribute to a group of pixels with a certain intensity.
Clearly, a more complex approach is needed when similarity between images depicting people is analyzed. A common approach to mitigate the *curse of dimensionality* for images is to project them onto a lower-dimensional space using an unsupervised or supervised dimensional reduction method. Not only is the image features concentrated into a lower dimensional representation but unwanted features associated with the background can be removed. Succeeding such a step a clustering or classifier algorithm is often applied, such that similar images are grouped together or classified as belonging to the same class.

Deep convolutional neural networks introduce an end-to-end approach for similarity measurements between images. A deep convolutional neural network can produce a rich hierarchical analysis of its input image, extracting contextual information about the relative position of objects and other high order features. The convolutional and spatial pooling layers introduce scale and translation invariance of sought-after features within images. The network performs also mappings to a lower dimensional space using different types of spatial pooling operations. The dense feature representation outputted from the network can later be used to express the similarity between different images. Such networks are often called encoder networks, whose purpose are to extract dense feature representations of their input.

In the case of similarity measurements between sequences of images $\mathbf{I}_i^1, \mathbf{I}_i^2 ... \mathbf{I}_i^n$, the deep convolutional neural network can be extended to encompass the temporal aspect inherent in sequences. By adding recurrent unit/units to the network the similarity between sequences of images can be examined. Temporal pooling can thereafter be performed on sequences of feature representations $\mathbf{f}_i^1, \mathbf{f}_i^2 ... \mathbf{f}_i^n$ in order to reduce the sequences to one representation $\mathbf{f}_i^*$, used for similarity measurements between sequences.

This section will introduce two different network architectures often used in re-identification and face recognition applications, the siamese network [30][10][24] and the triplet network [8][34]. Two sections will describe the network architectures from the viewpoint of similarity measurements between sequences of images depicting persons. A sequence of n images $\mathbf{I}_i^1, \mathbf{I}_i^2 .. \mathbf{I}_i^n$ will hereafter be denoted $\mathbf{I}_i^{1..n}$. Two images/sequences will in this report be expressed as matching or non-matching, corresponding to the similarity between captured person/persons in the images/sequences.

2.9.1 Siamese Network

The siamese network architecture makes use of two identical copies of one encoder network.

2.9.1.1 Embedding

Each encoder network produces a feature representation $\mathbf{f}_i^*$ associated with the sequence of images $\mathbf{I}_i^{1..n}$ as output. The goal of the siamese network is the generation of similar feature representations corresponding to matching sequences and dissimilar feature representations corresponding to non-matching sequences. Similarity between the feature representations can be expressed in numerous ways. One of those ways is the euclidean distance.
Each feature representation $\mathbf{f}_i^*$ with dimensionality $(n \times 1)$ corresponds to a point in $\mathbb{R}^n$. The euclidean distance between two feature representations $\mathbf{f}_i^*$ and $\mathbf{f}_j^*$ in the high dimensional room can be regarded as a measurement of the similarity between the representations. Often the squared euclidean distance $\|\mathbf{f}_i^* - \mathbf{f}_j^*\|^2$ is used as a measurement of the similarity between the two input sequences $\mathbf{I}_i^{1..n}$ and $\mathbf{I}_j^{1..n}$ [4]. However, the non-squared euclidean distance can also be used as the similarity measurement [10]. A commonly used energy function which describes the overall performance of the mapping from $\mathbf{I}_i^{1...n}$ to $\mathbf{f}_i^*$ in terms of the objective stated above is called the contrastive hinge energy function. The function is expressed by,

$$E(\mathbf{f}_i^*, \mathbf{f}_j^*) = \begin{cases} \frac{1}{2}\|\mathbf{f}_i^* - \mathbf{f}_j^*\|^2 & \text{matching sequences} \\ \frac{1}{2}\max(m - \|\mathbf{f}_i^* - \mathbf{f}j^*\|, 0)^2 & \text{non-matching sequences} \end{cases} \tag{2.49}$$

The equation states that feature representations yielded from matching sequences should have minimal distance in $\mathbb{R}^n$. The energy function inflicts also that feature representation derived from non-matching sequences should have an euclidean distance larger than a chosen margin m. The hard cut-off imposed by $\max(\cdot, 0)$ ensures that distances $\|\mathbf{f}_i^* - \mathbf{f}_j^*\|$, from non-matching sequences, larger than m do not contribute to a lower energy.

In order to train a network towards producing feature representations which yield low contrastive energy, the following loss function can be used,

$$L = \frac{1}{N}\sum_{k=1}^{N} E(\mathbf{f}_{i,k}^*, \mathbf{f}_{j,k}^*) \tag{2.50}$$

where N is the number of examples within the batch. The loss function is called contrastive hinge loss and expresses the average contrastive hinge energy for all the examples in the batch. Contrastive hinge loss has successfully been used in multi shot re-identification network training [30].

2.9.1.2 Classification

Classification of the two feature representations, outputted from the two encoder networks, have successfully been applied in re-identification applications [1][44]. An additional network, on top of the siamese encoder networks, can be used to perform a classification of the two input sequences, as either matching or not matching.

The classifier network transforms the difference between $\mathbf{f}_1^*$ and $\mathbf{f}_2^*$ into a probability distribution, expressed by a vector with two elements. Feeding the network directly with $\mathbf{f}_i^* - \mathbf{f}_j^*$ results in a network not invariant to the order of subtraction. In order to remedy this, the element-wise squared difference can be used $(\mathbf{f}_i^* - \mathbf{f}_j^*).^2$.

Assuming the output of the classifier network is denoted $\mathbf{z}$, the probability distribution estimate of the similarity between two feature representations can be expressed by,

$$\hat{y}_i = \sigma_{\text{softmax}}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{2} e^{z_k}} \tag{2.51}$$

The cross entropy function measures the similarity between the true probability distribution $\mathbf{y}$ and its estimate $\hat{\mathbf{y}}$ [16]. By averaging over the N examples in the batch, the cross entropy loss function can be expressed as [16],

$$L = -\frac{1}{N}\sum_{k=1}^{N}\sum_{i=1}^{2} y_{k,i}\log\left(\hat{y}_{k,i}\right) \tag{2.52}$$

Through minimization of the cross entropy loss, the encoder network is driven to a representation which extracts the necessary features from the input sequences, such that a classification of the inputs can be made.

Figure 2.14 depicts a siamese network. Two identical copies of a convolutional encoder network performs an embedding of the two input sequences $\mathbf{I}_1^{1..n}$ and $\mathbf{I}_2^{1..m}$, yielding two feature sequences $\mathbf{f}_1^{1..n}$ and $\mathbf{f}_2^{1..m}$. Each white block in the encoder network is a convolutional layer, the number above each block indicates the number of convolutional filters and the numbers below indicate the convolutional filter size. The feature sequences are passed on to recurrent units (RU), which perform temporal analysis on the input. A temporal pooling function $T(\cdot)$ reduces the two output sequences from the recurrent units to two individual representations $\mathbf{f}_1^*$ and $\mathbf{f}_2^*$. The figure depicts also a classifier network, which receives the squared element-wise difference $(\mathbf{f}_1^* - \mathbf{f}_2^*).^2$ between the two feature representations and performs a classification of the person/persons in the input sequences as either matching or not matching. The layers marked by FC are fully connected layers.

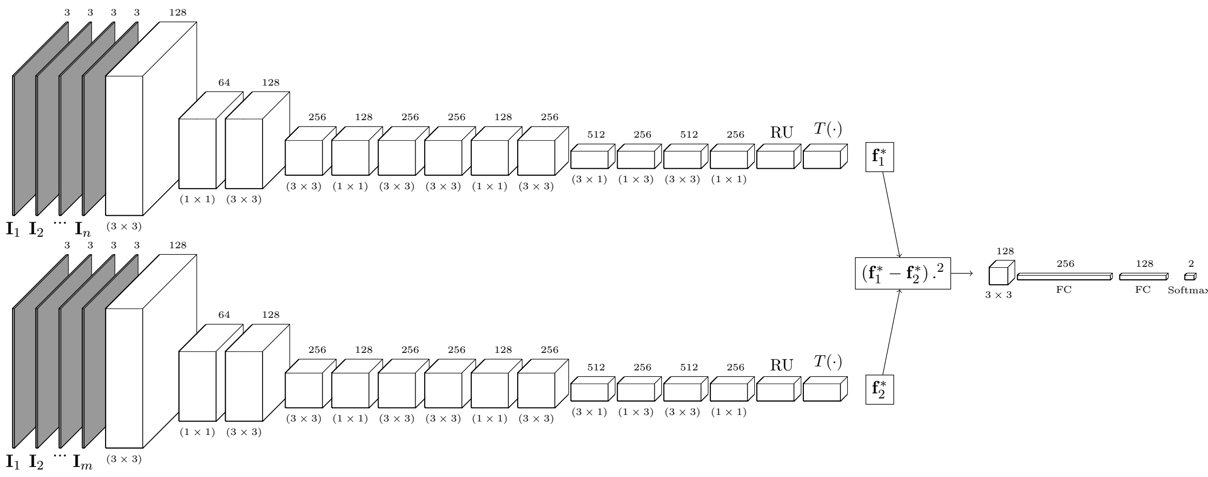

Figure 2.14: A siamese network receives two sequences of images as input $\mathbf{I}_1^{1..n}$ *and* $\mathbf{I}_2^{1..m}$ *and classifies the person/persons in the sequences as either matching or not matching.*

2.9.2 Triplet Network

The triplet network architecture can be seen as an extension of the siamese network architecture. Three encoder networks are used to generate feature representations from three different input sequences.
The three sequences are denoted *anchor* $\mathbf{I}_{anc}^{1..n}$, *positive* $\mathbf{I}_{pos}^{1..m}$ and *negative* $\mathbf{I}_{neg}^{1..k}$. The *anchor* together with the *positive* and *negative* is called a triplet. The *anchor* together with *positive* and *negative* are used to generate two sequence pairs, a positive pair $\{\mathbf{I}_{anc}^{1..n}, \mathbf{I}_{pos}^{1..m}\}$ and a negative pair $\{\mathbf{I}_{anc}^{1..n}, \mathbf{I}_{neg}^{1..k}\}$. The positive pair corresponds to a pair of matching sequences, whereas a negative pair corresponds to a pair of non-matching sequences.

2.9.2.1 Embedding

The goal of the triplet network is to produce feature representations $\mathbf{f}_{anc}^{*}$,$\mathbf{f}_{pos}^{*}$,$\mathbf{f}_{neg}^{*}$, such that the euclidean distance of the positive pair $\|\mathbf{f}_{anc}^{*} - \mathbf{f}_{pos}^{*}\|$ is smaller than the distance of the negative pair $\|\mathbf{f}_{anc}^{*} - \mathbf{f}_{neg}^{*}\|$. This ensures not only that positive pairs are grouped together but also that negative pairs are separated in the high dimensional space. The loss function used to achieve such an objective is expressed by [18],

$$L = \frac{1}{N}\sum_{k=1}^{N} \max\left(0, \left[\|\mathbf{f}_{anc,k}^{*} - \mathbf{f}_{pos,k}^{*}\| - \|\mathbf{f}_{anc,k}^{*} - \mathbf{f}_{neg,k}^{*}\| + m\right]\right) \tag{2.53}$$

where m is the margin and N is the number of triplets in the batch. Triplets with differences $\|\mathbf{f}_{anc,k}^{*} - \mathbf{f}_{neg,k}^{*}\| - \|\mathbf{f}_{anc,k}^{*} - \mathbf{f}_{pos,k}^{*}\|$ larger than the margin m does not contribute to the loss. In order for a triplet to be valid, the loss for the triplet must be larger than zero.
All triplets can be made valid by unit length normalization and scaling of the feature representations [34]. Each normalized feature representation will in such a case reside on the surface of a hyper sphere. However, reduction of the available embedding space may not be the best solution [18].

The triplet loss function in equation 2.53 with unnormalized feature representations is not well suited to use when the batch size is small. Only few triplets in the batch may be valid, and contribute to the loss. The hard cut-off in equation 2.53 can be replaced by its smooth approximation, called the softplus function. The softplus function is expressed by $\log(1+e^{x})$ [15]. Figure 2.15 shows the hard-cut-off function and its approximation. As argued, the triplet loss function can be rewritten as [18],

$$L = \frac{1}{N}\sum_{k=1}^{N} \log(1 + e^{\|\mathbf{f}_{anc,k}^{*} - \mathbf{f}_{pos,k}^{*}\| - \|\mathbf{f}_{anc,k}^{*} - \mathbf{f}_{neg,k}^{*}\|}) \tag{2.54}$$

The triplet softplus loss function in equation 2.54 ensures that all triplet pairs will be valid and the whole $\mathbb{R}^{n}$ can be used as embedding space, where n is the number of elements in the feature representation. The softplus triplet embedding will in effect have an infinite margin, however, the softplus function decays towards zero when $\|\mathbf{f}_{anc,k}^{*} - \mathbf{f}_{neg,k}^{*}\| \gg \|\mathbf{f}_{anc,k}^{*} - \mathbf{f}_{pos,k}^{*}\|$. Different types of triplet networks together with the triplet loss are frequently used in training of re-identification neural networks [8][18][7].

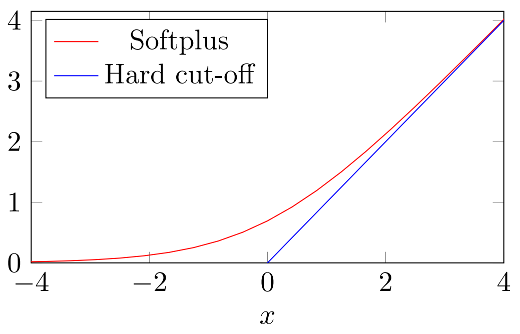

Figure 2.15: The hard-cut-off function and its approximation, the softplus function.

Chapter 3

Datasets

This chapter presents the different datasets used for network training, validation and evaluation. The batch generation procedure, used during training, will also be discussed.

3.1 Re-identification Datasets

In this section the different datasets used for network training, validation and testing will be presented. Two types of datasets will be introduced, one for single shot re-identification and one for multi shot re-identification. Special emphasis will be on the Axis multi shot re-identification dataset, developed as part of the thesis, for the purpose of multi shot re-identification network evaluation.

3.1.1 Single Shot Datasets

The single shot datasets contain non-sequential images of persons captured in different camera views.

CUHK01

The CUHK01 dataset, developed at the Chinese University of Hong Kong [25], contains 971 persons. Each person is captured in two different camera views. The dataset contains in total 3,384 manually cropped and labeled images. One of the two disjoint views captures mainly the frontal view of persons whereas the second view exhibit larger interclass pose and viewpoint variations amongst the captured images. Figure 7.1 shows example images from the dataset.

CUHK03

The CUHK03 dataset, also developed at the The Chinese University of Hong Kong [26] was one of the first re-identification datasets large enough for deep learning, 1,467 different persons are captured in 13,164 manually labeled images. Six different cameras with two different camera views were used in the dataset development. The multi-camera setup ensured significant intraclass pose/viewpoint variations of the different persons. Each person is captured in the two camera views. The dataset was obtained from a series of video sequences recorded over several months, resulting in a large span of illumination settings, caused by different weather conditions. Figure 7.2 shows example images from the dataset.

Market-1501

The Market-1501 dataset, developed at the Tsinghua University [47], is the largest public re-identification dataset with over 1,501 persons captured by six different cameras. The dataset contains in total 32,668 images. Each person is captured by at least two different cameras. It should be noted that overlap exists amongst the different cameras views. The multi-camera setup ensured significant intraclass pose/viewpoint variations of the different persons in the dataset. The automatic pedestrian detector used to produce cropped images of detected persons exhibited large performance variations, resulting in sporadic non-centered cropped images of persons. Figure 7.7 shows example images from the dataset.

3.1.2 Multi Shot Datasets

The multi shot dataset contains sequential images of persons captured in different camera views. Each sequence of consecutive images is often called a trajectory.

iLIDS-VID

The iLIDS-VID dataset contains 300 different persons captured in two disjoint cameras views, at an airport arrival terminal [41]. Each person is represented by two trajectories captured in two disjoint camera views. Due to the busy nature of the airport terminal, sporadic occlusions are present in many of the trajectories. The trajectory lengths varies from 23 sequential images to 192, with an average number of 73 images. The total number of images in the dataset are 43,800. Figure 7.5 shows example images from the dataset.

PRID-2011

The PRID-2011 dataset was jointly developed by the Austrian Institute of Technology and the Graz University of Technology [29]. The dataset contains manually cropped images extracted from trajectories. Two cameras with disjoint views were used in the dataset development. The dataset contains 934 different persons, however, only 200 appear in both camera views. The trajectory lengths span from five to 675 images, with an average of 100. The total number of images in the dataset are 96,123. The intraclass viewpoint variations are low for the different persons, no frontal or back views are present in the dataset. The dataset exhibits however considerable illumination variations between the two views, observable in figure 7.4.

Axis Multi Shot Re-identification Dataset (AMSR-id)

The Axis multi shot re-identification dataset (AMSR-id), developed at Axis Communication as part of the thesis, contains 120 different persons with a total of 20,362 cropped images. Five different cameras were used in the development, three Axis cameras and two handheld cameras. The three Axis cameras were placed in the lobby of Axis Communication headquarter. The cameras were placed in such a way that persons entering the lobby simultaneously would be captured from three different views. The dataset was constructed to closely resemble a live test with varying illumination, background and viewpoints settings. The two handheld cameras were placed such that a person entering the entrance from outside was captured from different viewpoints. The cameras placed in the lobby continuously recorded as persons walked by, the two cameras outside recorded only when a person/persons passed by.
In order to extract cropped sequential images of different persons the following steps were taken,

1. Sample the videos with 10 frames per second.
2. Pass each sampled frame to YOLOv2 [32] in order to detect persons and their bounding boxes. Figure 2.9 shows detections made by YOLOv2 during the development of the dataset.
3. Create cropped images from the bounding boxes generated by YOLOv2.
4. Match each cropped image of a person manually to all other persons in the dataset.
5. Create trajectories for each person, where one trajectory is constituted by a series of consecutive cropped images.

Each person is on average captured in two different trajectories, with an average of 169 images for all available trajectories. Figure 3.2 shows two trajectories captured from different viewpoints. Figure 3.1 shows five different persons captured from two separate viewpoints. The dataset exhibits large image quality variations, depending on the person's distance to the camera. The object detector exhibited large performance variations, resulting in sporadic non-centered cropped images of persons and images where the whole person is not captured. This introduces significant hardships for re-identification of persons present in the dataset.

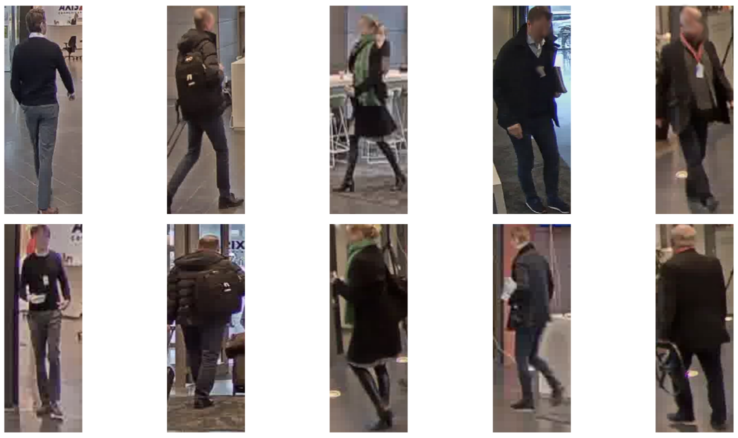

Figure 3.1: Examples of images from AMSR-id. Each column shows one person captured in two different disjoint camera views.

Figure 3.2 shows two sub-trajectories of a person in the dataset captured in two different camera views.

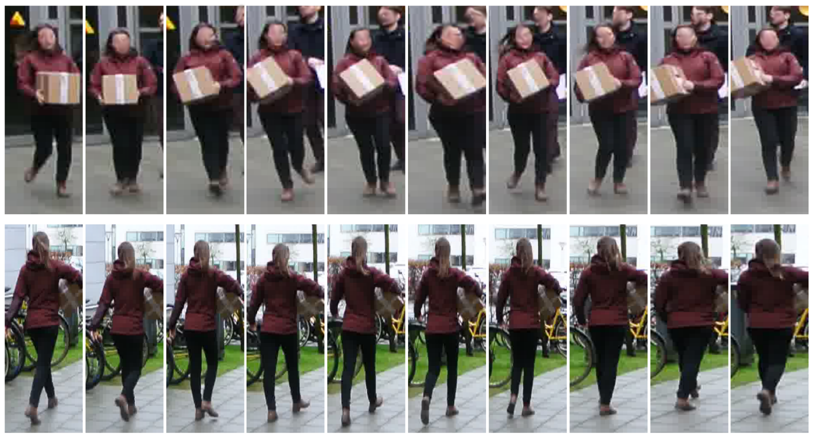

Figure 3.2: Two sub-trajectories of the same person captured in different camera views from AMSR-id.

3.2 Batch Generation

The batch generation aspect is crucial for the network's ability to learn a general model. In order to achieve a generalized network representation, each batch is sampled from a distribution of images taken from a multitude of different datasets. The cross-dataset batch generation ensures a trained network which is somewhat invariant to differences in illumination, background and image quality.
Each non-matching pair of images/sequences depicting different persons is drawn from the same dataset. If however, cross-dataset non-matching pair generation is used, the network's ability to extract relevant features is affected. Differences in illumination/background associated with the different datasets may be easier for the network to use when dissimilar persons should be classified/separated. Therein hindering relevant feature extraction of attributes associated with the persons in images/sequences.

In those cases a batch is formed for siamese network training, the distribution of matching pairs and non-matching pairs within the batch is enforced to be equal. Without this control the batches would almost entirely consist of non-matching pairs.

3.2.1 Multi Shot Re-identification

Three different datasets were used in the training of multi shot re-identification networks, iLIDS-VID, PRID-2011 and AMSR-id.

Figure 3.3 shows the tree structure from which a trajectory is drawn. Initially, one person p_i is drawn at random from all datasets. One trajectory v_k associated with p_i is thereafter chosen at random. A sub-trajectory with m images is randomly chosen $(i_t ... i_{t+m}) \in v_k$ and added to the batch.

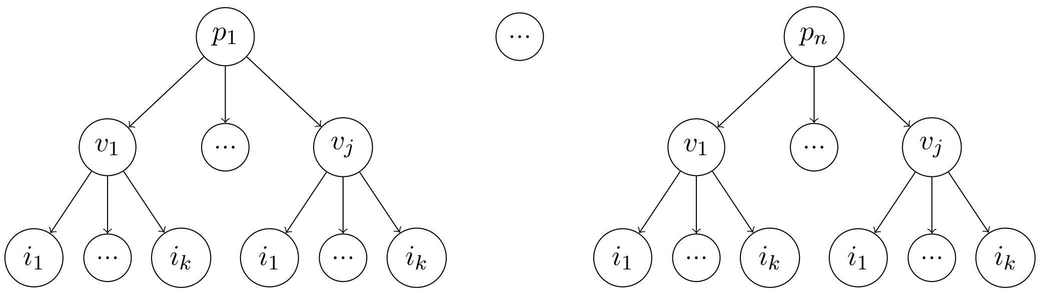

Figure 3.3: The tree structure in which the images of the different persons were sorted, p denotes a person, v a view and i an image.

A matching sequence pair is created by randomly choosing two trajectories v_k and v_i, from the same person, such that $v_k \neq v_i$. A non-matching sequence pair is created by randomly choosing two persons from the same dataset, and thereafter randomly choosing one trajectory for each person.

3.2.2 Single Shot Re-identification

Five different datasets were used in the training of the single shot re-identification networks, CUHK03, CUHK01,Market-1501, PRID-2011 and a reduced version of iLIDS-VID. The reduced iLIDS-VID dataset was created manually by removing images affected by occlusions. The reduced dataset contains 30,496 images. The whole PRID-2011 dataset was used, as no occlusions are present in any of the different trajectories.

The images used to form the batches were not drawn directly from the distribution of all images. Large datasets, in terms of number of images, such as PRID and iLIDS-VID contain many more images of a single person compared to smaller datasets, such as CUHK03 and Market-1501. In such a case would the occurrence of a single person in the batch be proportional to the number of images depicting the person, which is not favorable when a general model should be learned.

The single shot batches are formed in a similar manner as in the multi shot batch generation procedure, however, the chosen sub-trajectories contain only one image.

3.3 Data Augmentation

Data augmentation, is the method of artificially enlarging a training dataset. Data augmentation, often used with convolutional neural networks [22][38][5], is a powerful method to regularize a network. The ability of a network to generalize is directly affected by the size and variation of the training dataset. A network trained with a large and varying training dataset often learns a more generalized model compared to a network trained with a smaller dataset. The idea behind data augmentation is built on that premise.

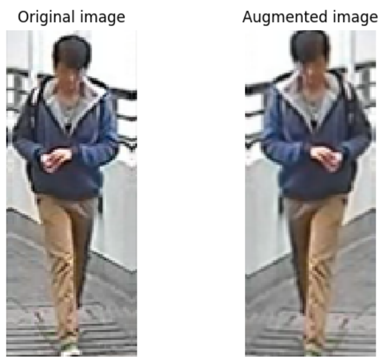

Figure 3.4: Left: Original image
Right: Augmented image.

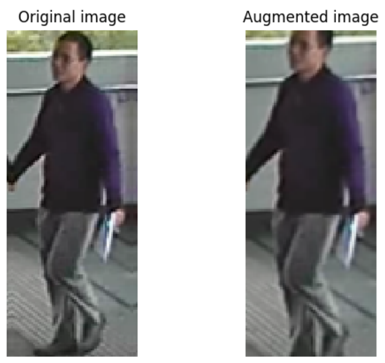

Figure 3.5: Left: Original image
Right: Augmented image.

Center cropping and horizontal flipping applied to each original image, resulting in an augmented image.

At each iteration of the network training scheme, each example in the batch undergoes a series of adjustments. Consequently, the training dataset is enlarged. Some often used adjustments to augment datasets of images for re-identification are center cropping and horizontal flipping [44][30]. Figure 3.4 and 3.5 show an image from one of the re-identification datasets and its augmented counterpart.

Chapter 4

Single Shot Re-identification

Single shot re-identification is the method of re-identifing a person between disjoint camera views by analysis of single images depicting persons. A single shot re-identification system interconnects images taken in different environments depicting the same person. Differences in person poses, camera viewpoints and illumination settings introduce significant complexity to the re-identification problem.

In large due to advancements in deep learning research single shot re-identification performance has been significantly increased in the past years [48]. Early re-identification systems have all been based on low-level hand-crafted features, spanning from color histograms, gradient histograms, local binary patterns to SIFT descriptors [48]. The hand-crafted features have in a classification step been used to re-identify persons, trainable distance metrics have often been used in this step [48]. However, systems based on low-level features can not sufficiently capture the complexity inherent in the re-identification problem. Given the constraints of such systems persons are seldom correctly re-identified when large camera viewpoint and illumination variations exist amongst the images. Deep learning introduces an end-to-end approach for person specific feature extraction and person re-identification. Neural networks have the possibility to learn complex feature extractions otherwise unconceivable to a human designer. The lack of training data has previously made the idea of deep neural networks for re-identification purposes unfeasible. However, the emergence of large datasets, such as CUHK03 and Market-1501, have introduced a boom of re-identification systems based on deep neural networks.
This part of the thesis project continues on that track and introduces different types of single shot re-identification systems based on convolutional neural networks.

Convolutional neural networks of varying depths and sizes have been evaluated together with three different loss functions, contrastive hinge loss in equation 2.50, triplet softplus loss in equation 2.54 and cross entropy loss in equation 2.52. The different loss functions correspond to two different re-identification approaches, embedding and classification. In the classification case are two images classified as either matching or not, depending if the images depict the same person or not. In the embedding case, the euclidean distance between low-dimensional feature representations of images are measured, where a small distance equals high similarity between the persons in the images.

The thesis work is a continuation of many contributions to the field of single shot re-identification, to name a few [1][44][24][8]. The thesis work's contribution to single shot re-identification research can in large be attributed to the broad exploration of different networks and loss functions.

Contrary to many published research articles the developed networks have been trained on multiple different dataset [1][44]. The multi dataset training procedure has enabled the development of larger networks, compared to many previously developed networks, introduced in [1][44][24].

This chapter will present the developed single shot re-identification networks. Methods adopted for training and evaluation will also be discussed. Results obtained from the single shot re-identification study will be presented and analyzed. In the section dedicated to future work will ideas outside the scope of the thesis be presented.

4.1 Implementation Details

This section introduces implementation details regarding the training of the developed networks.

The different datasets used during training can be seen in the chapter dedicated to the datasets. CUHK01 was split into three subsets, with approximately the same size. The three subsets were used separately for validation, testing and training. Because each image is used in a matching or non-matching pair, the number of examples the network can be trained on is vastly more than the total number of images in the datasets.

Each image was preprocessed before it was passed on to the networks. The image was firstly resized by bilinear interpolation into a standardized size 160×60, often used in similar applications [1][44]. In order to transform the pixel intensity values into a predefined range each image $\mathbf{I}_i$ was transformed according to,

$$\mathbf{I}_i^* = \frac{\mathbf{I}_i}{\max(\mathbf{I}_i)} \tag{4.1}$$

The rescaled image $\mathbf{I}_i^*$ becomes transformed into the often used interval $[0, 1]$ [5].

The ADAM optimizer was used to optimize the weights of the networks. A learning rate of 10^{-3} was initially used. The learning rate was decreased linearly according to equation 2.23, where τ was set to be equal half the total number of training iterations. The final learning rate was a factor 0.008 smaller than the initial. The batch size was chosen quite low, this was mainly done to speed up the training process and prevent over fitting on the training datasets [20]. The batch size was chosen to be 20.

The weights were initialized according to the Xavier weight initialization scheme [14].

The different networks were trained for a fixed amount of iterations t_{MAX}, depending on the size of the network. The number of iterations for the different networks are introduced further down. In order to mitigate the risk of over-fitting, the performance on the validation dataset was monitored, the network was saved continuously as the performance increased on this dataset. The network was validated in steps of 200 iterations during training.
The networks were also saved automatically in increments of $t_{\text{MAX}}/5$. At the end of the training, the re-identification performance was measured on the whole validation dataset, the best saved network was chosen.

The best networks for each choice of loss function were also trained on augmented images, resulting in enlarged training datasets and increased number of training iterations. Data augmentation adjustments were applied to all images in the training datasets. Two different image augmentation operations were used, center cropping and horizontal flipping of the image. The cropping was conducted through reduction of the images, a center image with size spanning from (144×54) to (160×60) was chosen for each example in the batch. The reduced image was thereafter rescaled to the standard size (160×60). Figure 3.4 and 3.5 show the results after the different data augmentation adjustments have been applied.

4.2 Evaluation Method

The different networks were evaluated on the CUHK01 test dataset. The common way of evaluating the performance of a re-identification system is by examination of the Cumulative Match Characteristic (CMC) curve [1][44]. Given a test image depicting a person p_t and a gallery (a set of images), the CMC-curve yields information about the cumulative recognition match as a function of the rank outputted from the re-identification system is a similarity score for each image in the gallery, corresponding to the certainty that the gallery image depicts p_t. If the image with the highest similarity score amongst all images in the gallery set would depict p_t, the rank would be one. However, if the correct image would have the fifth highest similarity score, the rank would be five. The rank yields thus information about where in the sorted list of similarity scores the correct match is found. The CMC-curve expresses the cumulative recognition percentage. A CMC value of 70 % at rank five means that a correct match will be present somewhere amongst the five images with highest similarity scores 70 % of the time. The gallery set was chosen to include 100 images, commonly used in most articles [1][44][24]. The whole gallery set contains images from a camera view different from the view which the test image was captured in.

The similarity score has been calculated in different ways depending on the re-identification system. If the re-identification system performs an embedding and output a feature representation, the euclidean distance has been used as the similarity score. Smaller distances equals higher scores in such cases. If the developed system instead performs a classification of two input images, either as matching or not matching, the probability of similarity has been used as the similarity score.

In order to properly evaluate the different networks 100 persons were drawn at random from the test dataset. Two images per person were randomly selected amongst the different views and placed in the test and gallery set. For each image in the test set the rank with respect to the gallery set was calculated. These steps were repeated 10 times, such that an average CMC-curve could be calculated.

4.3 Proposed Single Shot Re-identification Networks

In this section the different networks will be presented. Two types of network architectures were tested, the siamese and the triplet. Associated with each of the two architectures is an encoder network, which extracts a feature representation associated with the input image. Three different encoder networks with varying sizes (number of convolutional filters) and depth (number of layers) were developed and evaluated.

4.3.1 Encoder Networks

The role of the encoder network is to extract a person specific low-dimensional feature representation, invariant to undesirable variations, from an image depicting a person.

Aggressive reduction of the image's spatial dimensions is performed in early layers for all developed encoder networks. At the fourth layer the original spatial dimensions have been reduced by a factor six. The spatial dimensions are reduced by two methods, down sampling and max pooling. Following the first convolutional layer is down sampling, serving as a means of reducing the computational burden in succeeding layers. The down sampling is achieved by convolution with stride two in the first convolutional layer. All convolutional and max-pooling layers use same-padding. The remaining spatial dimension reduction steps are performed by (3×3) max-pooling with stride 3. Following each max-pooling layer is a doubling in the number of convolutional filters, conventionally used in order to aggregate essential information passing through the max-pooling layer [32][38]. The spatial dimensions of the final output are approximately 1/18 (9×4) of the original image, for all the developed encoder networks.

Table 4.1: Large sized encoder network, RE-ID-3.

Type	Output Dim.	Filter Size	Stride	Act. func.	Normalization
Input	$160 \times 60 \times 3$	-	-	-	-
Convolutional	$80 \times 30 \times 128$	3×3	2	ReLu	Layer Normalization
Convolutional	$80 \times 30 \times 64$	1×1	1	ReLu	Layer Normalization
Convolutional	$80 \times 30 \times 128$	3×3	1	ReLu	Layer Normalization
Max Pooling	$27 \times 10 \times 128$	3×3	3	-	-
Convolutional	$27 \times 10 \times 256$	3×3	1	ReLu	Layer Normalization
Convolutional	$27 \times 10 \times 128$	1×1	1	ReLu	Layer Normalization
Convolutional	$27 \times 10 \times 256$	3×3	1	ReLu	Layer Normalization
Convolutional	$27 \times 10 \times 256$	3×3	1	ReLu	Layer Normalization
Convolutional	$27 \times 10 \times 128$	1×1	1	ReLu	Layer Normalization
Convolutional	$27 \times 10 \times 256$	3×3	1	ReLu	Layer Normalization
Max Pooling	$9 \times 4 \times 256$	3×3	3	-	-
Convolutional	$9 \times 4 \times 512$	3×3	1	ReLu	Layer Normalization
Convolutional	$9 \times 4 \times 256$	1×1	1	ReLu	Layer Normalization
Convolutional	$9 \times 4 \times 512$	3×3	1	-	Layer Normalization

The different encoder networks make use of two different convolutional filter sizes, (3×3) and (1×1). Convolutional layers with 1×1 filter sizes are used in between layers with (3×3) filter sizes, serving to compress the feature maps into dense representations [32][27]. Such convolutional layers are equivalent to cross channel parametric pooling layers [27]. Commonly, large filter sizes are used in those convolutional layers which employ stride larger than one. However, initial tests showed no significant performance gain with (5×5) filters at the first layer. In the attempt to reduce the number of parameters in the first layer (3×3) filters were chosen for this layer as well.

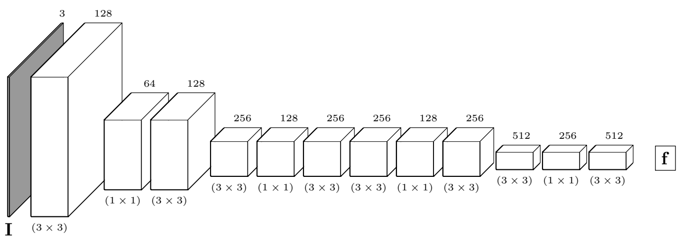

Figure 4.1: Illustration of the large encoder network RE-ID-3, where **I** *is the input image and* **f** *is the outputted feature representation.*

Three different encoder networks were developed, the different networks can be seen in the appendix. The first network, denoted RE-ID-1 can be seen in table 7.1, this network is 9 layers deep. The second network, called RE-ID-2 is similar to RE-ID-1, however, twice as many filters are used in each layer. The network can be seen in table 7.3. The last encoder network extends the networks of the other by adding additional layers. Three extra layers are added and the number of filters at each layer significantly increased. Table 4.1 and figure 4.1 show the RE-ID-3 encoder network. Each white block in the figure is a convolutional layer, the number above each block indicates the number of filters and the numbers below describe the filter sizes.

Layer normalization was used for all layers. Initial tests without layer normalization were carried out, however, with lower performance and lower convergence speed. All results presented in this chapter for the developed encoder networks were obtained by applying layer normalization in all layers.

The networks were trained for a fix number of iterations. The networks were saved according to the procedure discussed in the section dedicated to the implementation details. The smaller networks have fewer weights and need therefore fewer update steps, compared to the larger networks. All the developed networks together with the different encoder networks were trained for,

- RE-ID-3 - 200,000 iterations
- RE-ID-2 - 150,000 iterations
- RE-ID-1 - 100,000 iterations

The number of training iterations for the different networks were chosen heuristically by observing the convergence of the loss value. Those networks trained on the augmented datasets were trained for the double amount of iterations, and later fine-tuned for 30,000 iterations on the CUHK01 training dataset. Without fine-tuning on the CUHK01 training dataset the networks trained on the augmented datasets seemed not to be able to produce results near those obtained on networks trained on the non-augmented datasets.

4.3.2 Embedding - Contrastive Hinge Loss

In order to perform an embedding of the outputs yielded by the encoder networks in the siamese network architecture, two fully connected layers, each with 1000 neurons, were added to the networks. The output of the first fully connected layer is layer normalized and passed through a non-linear activation function (ReLu) and thereafter passed on to the output layer. The final output layer is not followed by any activation function. The contrastive hinge loss in equation 2.50 with margin $m = 1$ receives thereafter the two outputs from the siamese network. Table 4.2 shows the results obtained by the networks trained with contrastive hinge loss. The results show increased performance on the larger models. The rank-1 accuracy is lower on the largest network compared to the middle sized network, however the remaining rank-accuracies show increased performance on the largest network. The network trained on the augmented datasets performs overall worse compared to the network trained on the non-augmented dataset.

Table 4.2: Results on CUHK01 obtained by the networks trained with contrastive hinge loss. The performance is measured in cumulative match characteristic.

Network	Rank 1	Rank 5	Rank 10	Rank 20
RE-ID-1-Contrastive	29.1	64.4	80.0	92.9
RE-ID-2-Contrastive	33.2	69.0	82.6	94.4
RE-ID-3-Contrastive	32.1	72.5	85.1	95.6
RE-ID-3-Contrastive Augmented	32.3	71.2	85.3	94.0

4.3.3 Embedding - Triplet Softplus Loss

Two fully connected layers were added to the three encoder networks in the triplet network architecture, similarly to the procedure in the siamese network architecture with contrastive hinge loss. The two fully connected layers perform the embedding of the encoder networks' outputs. The embeddings are passed on to the triplet softplus loss function seen in equation 2.54. Table 4.2 shows the results obtained by the networks trained with triplet softplus loss. The results show a significant increase in rank accuracies for the largest network compared to the other. The network trained with the augmented datasets performs worse compared to the network trained on the non-augmented datasets.

Table 4.3: Results on CUHK01 obtained by the networks trained with triplet softplus loss. The performance is measured in cumulative match characteristic.

Network	Rank 1	Rank 5	Rank 10	Rank 20
RE-ID-1-Triplet	41.2	75.7	86.6	95.0
RE-ID-2-Triplet	44.2	78.6	90.5	96.4
RE-ID-3-Triplet	58.8	86.8	93.7	97.7
RE-ID-3-Triplet Augmented	56.2	86.4	93.5	98.6

4.3.4 Classification

Classifier networks were used together with the different encoder networks. The classifier networks receive the squared element-wise difference between the two feature maps outputted from the encoder networks in the siamese network architecture. The input undergoes a number of transformations. The first layer is convolutional, which analyze the squared difference between spatially neighboring elements of the feature maps. Following the convolutional layer are fully connected layers which reduce the convolutional layer's output into a vector with two elements. A dropout layer was placed in between the fully connected layers in the classifier networks. The dropout percentage was chosen to be 50%. The output is passed on to the softmax function in equation 2.51, which estimates the binary classification distribution. The cross entropy loss function in equation 2.52 is thereafter applied to the distribution estimate. Different types of classifier networks with varying sizes were used together with the different encoder networks. A larger classifier network was used together with the largest encoder network, RE-ID-3. The classifier networks can be seen in table 7.2, 7.4 and 7.5. Table 4.5 shows results obtained by the different classifier networks. The results show significant increase in rank-1 accuracy for the RE-ID-2-Classifier network compared to the smallest network. The rank-1 accuracy difference between RE-ID-2-Classifier and RE-ID-3-Classifier is more subtle. The network trained on the augmented datasets performs worse compared to the network trained on the non-augmented datasets.

Table 4.4: The table shows the results obtained on CUHK01 by the classifier networks. The performance is measured in cumulative match characteristic.

Network	Rank 1	Rank 5	Rank 10	Rank 20
RE-ID-1-Classifier	54.1	83.9	92.6	97.0
RE-ID-2-Classifier	63.9	87.5	93.2	96.5
RE-ID-3-Classifier	65.2	90.6	95.5	98.8
RE-ID-3-Classifier Augmented	62.2	88.1	95.1	98.2

All results presented below were obtained by the RE-ID-3-Classifier network.

Figures 4.2,4.3 and 4.4 show one test image of one test person and the ten most probable matches amongst 100 persons in the gallery. The list is sorted in descending order, from the most probable to less likely. Above each image is a similarity score, scores marked in green indicate correct matches. In the case seen in figure 4.2, the correct person is re-identified with the largest similarity score. In the second case, illustrated by figure 4.3, the correct person has the second largest similarity score. Respectively, the two persons are re-identified with rank one and two.

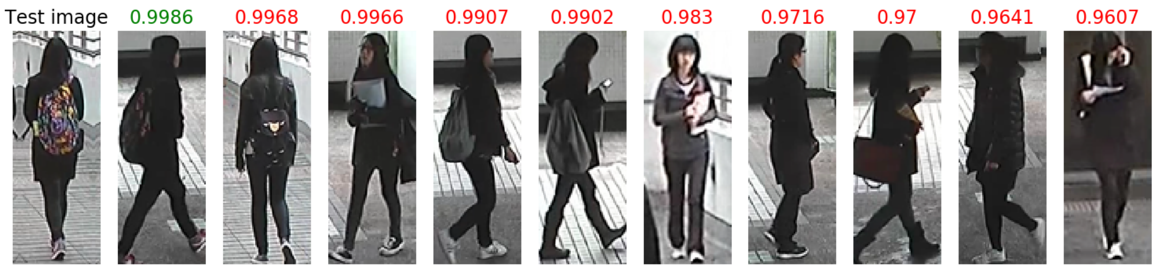

Figure 4.2: To the far left: a test image depicting a test person. To the right: a collection of the ten most probable re-identification matches. Above each image is a similarity score.

One can clearly observe in figure 4.2 and 4.3 that the network has learned to find similar looking people. Clearly, clothing is one essential attribute the network makes use of when a person is re-identified. In the case seen in figure 4.2 all persons amongst the ten most likely have similar black clothing.

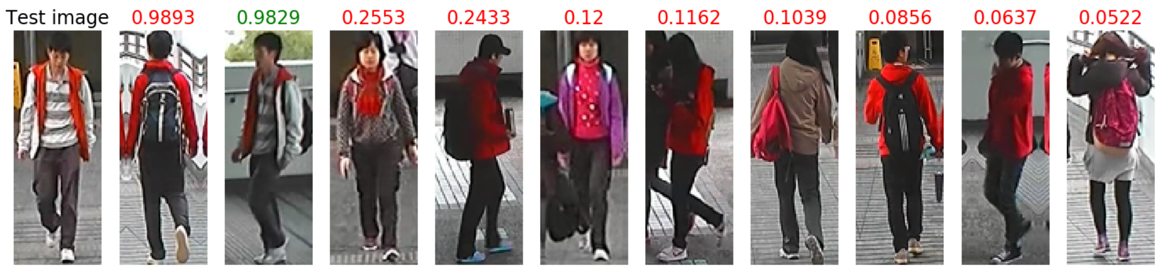

Figure 4.3: To the far left: a test image depicting a test person. To the right: a collection of the ten most probable re-identification matches. Above each image is a similarity score.

In the case seen in figure 4.3 one can see that distinct colors are something the network makes use of when a person is re-identified. All persons amongst the ten most likely have something distinctly red in their clothing, similarly as the test person.

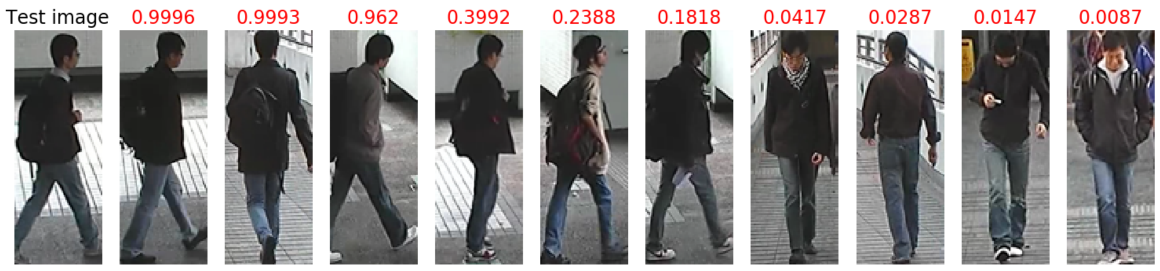

Figure 4.4: To the far left: a test image depicting a test person. To the right: a collection of the ten most probable re-identification matches. Above each image is a similarity score.

Figure 4.4 shows one re-identification case when the correct person is not amongst the ten most probable. However, the most probable persons all have similar attributes as the test person. The test person was re-identified with rank 15.

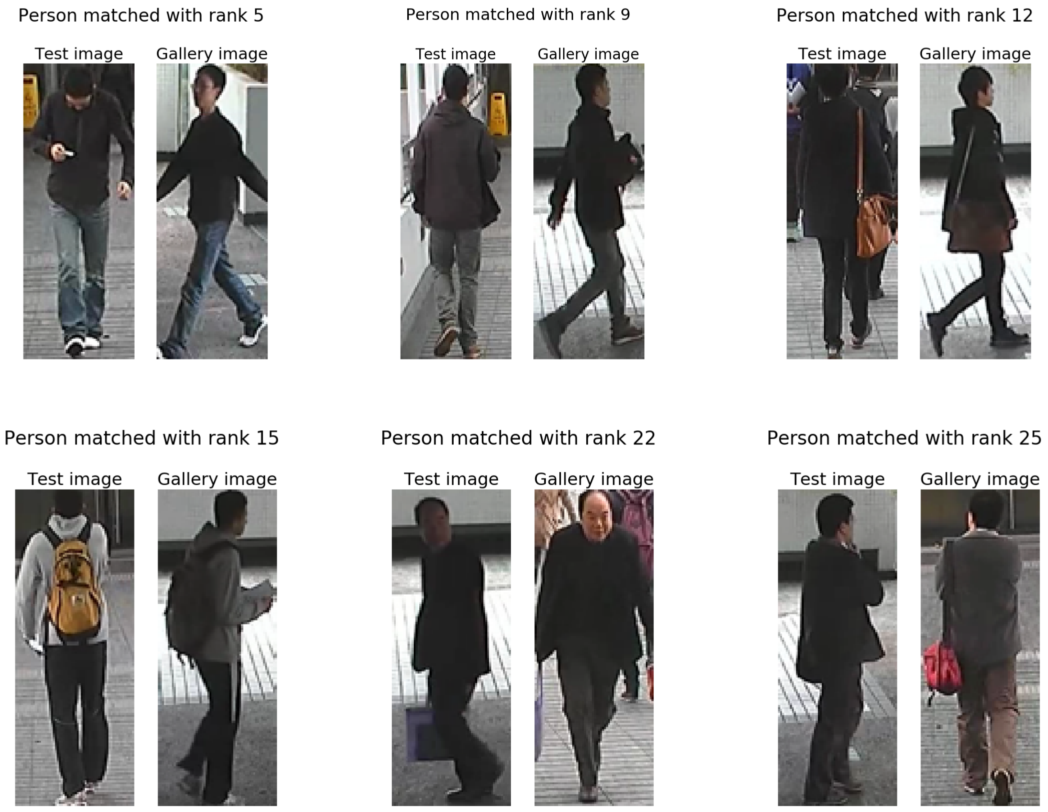
Person matched with rank 5
Test image
Gallery image
Person matched with rank 9
Test image
Gallery image
Person matched with rank 12
Test image
Gallery image
Person matched with rank 15
Test image
Gallery image
Person matched with rank 22
Test image
Gallery image
Person matched with rank 25
Test image
Gallery image

Figure 4.5: Examples of test images and their correct matches in the gallery for different difficult cases. Above each matching pair is the rank in which the person was re-identified with.

Figure 4.5 shows certain cases where the network had trouble re-identifying the persons. The re-identification cases give an idea of the conditions the network find difficult. Hardships for the developed network are when large pose/illumination variations exists amongst the test image and the correct image in the gallery.

Figure 4.6 shows non-linear 2-D mappings of feature representations obtained from images depicting five different persons. The 2-D mappings were obtained by mapping the feature representations outputted from the trained RE-ID-3 encoder network onto two dimensions. This was achieved by applying two fully connected layers, with 1000 and 2 neurons respectively, following the original representation of the RE-ID-3 encoder network. The two fully connected layers were trained for 50,000 iterations with the triplet softplus loss function. The encoder network was kept constant during the training. Each dot in the figure corresponds to one image and every color to one unique person. Even in this low dimensional representation the different persons are separated.

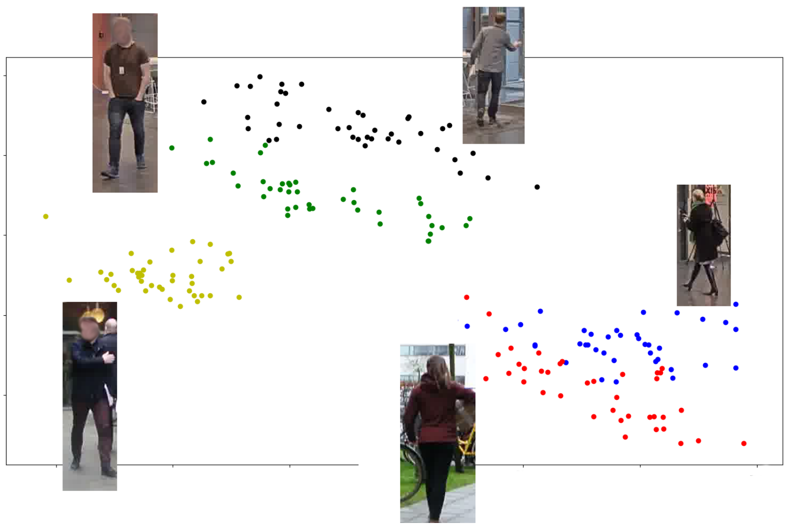

Figure 4.6: Feature representations of images corresponding to five persons mapped onto a plane.

4.4 Transfer Learning

In order to evaluate transfer learning for re-identification, InceptionV1 [40], pretrained on ImageNet [1], was tested as an encoder network. The tests with the different developed networks had shown that classification of two persons as either matching or not matching outperform the attempts of embedding. The InceptionV1 network was thereby used as an encoder network for a classifier network. The classifier network is the same used for the RE-ID-3-Classifier network.

The InceptionV1 network is originally designed for object classification. The output layer in the InceptionV1 network concentrates the features extracted from the earlier layers and output a class prediction. The role of the InceptionV1 network in the re-identification system is to extract features applicable for re-identification and not class predictions. The output layer was removed such that the classifier network would receive more general features. InceptionV1 requires 224×224 images as input, consequently each image was resized into the required size. Given that the InceptionV1 network was pre-trained, a lower learning rate was used for this network, one tenth of the standard learning rate used for the other networks. The learning rates were linearly decreased, similarly as the procedure discussed in the section dedicated to the implementation details.
Initial tests showed problems, the loss value would not converge. In order to address this problem, the weights in the classifier network were adjusted to be in line with the output from the InceptionV1 network. This was achieved by keeping the InceptionV1 network constant and only train the added layers. The added layers were trained for 30,000 iterations with batch size 50. The whole network was thereafter trained for additionally 50,000 iterations with batch size 10. In addition to evaluation on CUHK01, the network was also evaluated on CUHK03, CUHK03 was divided into a training and test dataset. Only CUHK03 and CUHK01 were used during the training. The datasets were not augmented. The results obtained using InceptionV1 as encoder

[1]https://github.com/tensorflow/models/tree/master/slim

network can be observed in table 4.5. The rank-1 accuracy on CUHK01 is the highest amongst all networks.

Table 4.5: The table shows the results obtained on CUHK01 and CUHK03 by the classifier network using InceptionV1 as an encoder network. The performance is measured in cumulative match characteristic.

Network	Rank 1	Rank 5	Rank 10	Rank 20
InceptionV1-Classifier CUHK01	67.0	90.8	96.4	98.9
InceptionV1-Classifier CUHK03	63.9	91.3	95.8	98.3

4.5 Conclusion & Discussion

By examination of the results obtained in this study overall conclusions can be drawn, those conclusions are presented below.

Classification of two persons as either matching or not matching outperformed all attempts of embedding in the examined cases. Significant performance gain was achieved by utilization of a classifier network, which examines the element-wise difference between feature representations. The performance variations amongst the two objectives, classification and embedding, may possibly be understood by examination of how the different network types analyze differences in feature representations. The classifier network analyze the contextual relationship between element-wise differences in the two feature maps, which may extract information not yielded by direct examination of the euclidean distance. The complex analysis comes however at a cost. Performing a classification with a classifier network is considerably more computationally demanding than direct examination of the euclidean distance, and in a live system such a classifier system is unfeasible. In hindsight, the classifier networks should have been keep constant for all the different encoder networks, this would have enabled a fairer estimation of the performance increase of the different networks were dependent on the changes to the encoder network or to the changes to the classifier network.
During training, the encoder and classifier networks were continuously saved as the loss value on the validation set decreased. The networks were also saved at the end of the training. In almost all cases, the networks saved at the end of training performed better than those saved during the training, when the validation loss decreased. The average loss value for single example pairs within the validation batch seems to be a bad measurement of the actual performance when the networks are used to distinguish between many persons, contrary to only one person. In a matter of fact, the validation loss increased after some time, but the re-identification performance rose. It is not enough that the networks can classify two images depicting person/persons as either matching or not-matching, the networks must yield probability estimates close to the actual probability distribution in order for the networks to correctly re-identify a person amongst many others. The classifier network produced worse classification accuracies for single examples at the end of the training. However, in those cases the network classified two persons correctly, the similarity scores were larger, yielding better performance when the network was used to distinguish between many persons. Both the average classification accuracy and the average loss value for pairs within the batch is thereby a bad measurement of the actual re-identification performance of the classifier networks. Instead, each example pair should have been fed to the classifier network, together with every other example in the validation batch, such that an actual measurement

of the re-identification performance could have been measured during the validation step. This was however not done. Even better performance could almost certainly be obtained if such a validation method was used.

The networks trained with triplet softplus loss outperform those networks trained with contrastive hinge loss. The triplet softplus loss introduces a two class classification problem, where the *anchor* is forced to be closer (in euclidean distance) to the *positive* than the *negative*. In the contrastive hinge loss case each example is either pulled closer to a matching example or pushed away from a non-matching example. The triplet softplus loss introduces thereby a two fold push-pull objective for examples within the batch. The triplet softplus loss has in the tested cases driven the networks towards representations which produce embeddings more suitable for re-identification purposes compared to those produced by a network trained with the contrastive hinge loss.
During validation the average re-identification rank value was calculated for all the examples within the validation batch. The networks were saved as the average rank value decreased. This validation method enabled a direct measurement of the re-identification performance of the developed embedding networks, contrary to the method used in the classification case.

The large and complex pre-trained InceptionV1 network together with the classifier network produced results better than those obtained by the other developed networks. The InceptionV1 network was only fine tuned, and better results could almost certainly be achieved by longer training. The method of transfer learning has shown to be applicable for re-identification purposes. Further tests with other pre-trained networks is an interesting way forward.

It has become evident from this study, large networks, both in terms of depth and number of weights, introduce significant performance gain, compared to shallow networks with fewer weights. A more complex hierarchal analysis of person specific traits can be made by a deeper network. A network with more weights corresponds to a network which is able to retain and extract more information from its input. A more complex extraction of person specific features can thereby be learned by a larger network. However, larger networks must be matched with larger training datasets as overfitting is easily achieved on large networks trained with small datasets.

The results obtained on CUHK01, by the best performing networks, are comparable to state of the art results obtained in multiple articles [1][44][24].

- PersonNet [44] (2016), rank-1 accuracy of 71 %, with 100 persons in the gallery.
- JointRe-id [1] (2015)
  - rank-1 accuracy of 65 % , with 100 persons in the gallery.
  - rank-1 accuracy of 40.5 % , with 486 persons in the gallery.
- FPNN [24] (2016), rank-1 accuracy of 28 % , with 100 persons in the gallery.

In order to employ a fairer evaluation procedure, the authors behind JointRe-id [1] tested cross dataset evaluation, with training on CUHK03 and evaluation on CUHK01.
They come to the conclusion:
"This is unlikely to work well since the network does not know the statistics of the test data

set, and in fact, our model trained on CUHK03 and tested on CUHK01 gave rank-1 accuracy of around 6%, which is far below the state of the art."[1].
It should be noted that the evaluation in this case was carried out with 486 persons in the gallery.

Cross dataset evaluation was also tested with the RE-ID-3-Classifier network. The network was re-trained without CUHK01. A rank-1 accuracy of 35 % was obtained on CUHK01 in this case, resulting in a significant decrease in performance compared to the case when a subset of CUHK01 was used during trained. The performance decrease was however not as drastic as the one described for JointRe-id [1].
Cross dataset evaluation is probably the preferred approach for proper evaluation of networks' performance in real-world situations. Using the same dataset for training and evaluation tends to lead to dataset bias [30], where the network has adapted to certain settings inherent in the training dataset. That may explain the decrease in performance in the two cases.
Cross dataset training has probably enabled developed networks with generalization capabilities exceeding those of other developed networks which only have been trained on a single dataset. Importantly, non-matching pairs in the batch should be created by examples from within the same dataset, as previously discussed in the chapter dedicated to the datasets. Significant performance decrease was observed when non-matching pairs were drawn from different datasets, this was especially evident for the classifier networks.

The developed networks tend to focus on the color of clothing when a person is re-identified. In most cases all the persons amongst the 10 most likely in the gallery set have similar colors on their trousers and jacket/sweater. The networks tend also to focus on distinct color features, as can be seen in figure 4.3. When those distinct features are not observable in a different camera view the networks tend to miss-re-identify the person.

Almost all networks trained on augmented datasets performed worse during testing compared to those networks trained on the non-augmented datasets. The decrease in performance was significant for RE-ID-Triplet and RE-ID-Classifier. Intuitively, the networks should have learned a more general representation when trained on the larger and more diverse datasets, this was however not the case. Possible explanations why the networks performed worse might be, the networks are not able to generalize well, the adjusted images may depict scenarios not presented in the test dataset. The number of iterations during training may have been chosen too low, the networks might have performed better if they were trained for a longer time. One interesting aspect to examine is which augmentation operation that caused the degradation in performance.

There have not been enough time to test all combinations of hyper parameters, it is possible that other choices of learning rate, batch size, margin etc. would have resulted in a different outcome. Further tests with different augmentation parameters and adjustments may be a possible way to improve the performance of the networks. The number of training iterations may have been too low, however the loss value showed to have reached convergence at the end of training. Because the batches were drawn randomly, the data augmentation adjustments randomly chosen and the weights randomly initialized, the results for the same network tend to be different from training to training. Large performance variations were observed amongst networks with the same design for different training sessions. This illustrates the hardship of proper evaluation of the performance of different network designs. In order to perform a fairer estimation of the developed networks' performance, the networks should have been trained 10 times and the results averaged, similarly to the method adopted in most research articles [44][1]. Due to lack of time this was not done.

4.6 Future Work

This section will introduce ideas for further exploration of single shot re-identification networks. The ideas presented in this section were out of the scope for this thesis. Hopefully, the ideas presented in this section can inspire further improvements of single shot re-identification networks.

4.6.1 Embedding with a Trainable Metric

Expressing the similarity between images as inversely proportional to the euclidean distance of their feature representations have successfully been applied to numerous applications [10][30]. However, a more advanced method where the network derives its own similarity metric can be used in re-identification applications, as discussed in this article [7]. By utilization of a second network, a trainable similarity metric can be learned. In the case of a siamese network architecture, the similarity metric network transforms the difference between $\mathbf{f}_1^*$ and $\mathbf{f}_2^*$ into a dense similarity measurement.
Feeding the similarity metric network directly with $\mathbf{f}_i^* - \mathbf{f}_j^*$ results in a network not invariant to the order of subtraction. In order to remedy this, the squared element-wise difference can be used [7]. Let's assume the similarity metric network is only one layer deep and the output is a scalar $d(\mathbf{f}_i^*, \mathbf{f}_j^*)$. The output can be express as,

$$d(\mathbf{f}_i^*, \mathbf{f}_j^*) = \alpha_1(f_{i,1}^* - f_{j,1}^*)^2 + \alpha_2(f_{i,2}^* - f_{j,2}^*)^2 + ... + \alpha_n(f_{i,n}^* - f_{j,n}^*)^2 \tag{4.2}$$

If $\sum_{i=1}^{n} \alpha_i = 1$ and $\alpha_i \geq 0 \quad \forall i$, the scalar output from the similarity metric network is equaled to the squared weighted euclidean distance. If the network's depth is increased, and non-linear activation functions are used in the layers, the mapping from $(\mathbf{f}_i^* - \mathbf{f}_j^*).^2$ to the output becomes highly non-linear. The similarity metric network yields thereby directly a similarity measurement between the two inputs. By using a similarity metric network the difference between $\mathbf{f}_i^*$ and $\mathbf{f}_j^*$ can be concentrated onto a lower dimensional space, extracting information not yielded by direct examination of the euclidean distance.

The contrastive hinge loss in equation 2.50 can directly be applied to the output of the similarity metric network, replacing $\|\mathbf{f}_i^* - \mathbf{f}_j^*\|$ in equation 2.49. The different versions of the triplet loss function can also make use of the trainable similarity metric output, replacing $\|\mathbf{f}_{anc,k}^* - \mathbf{f}_{pos,k}^*\|$ and $\|\mathbf{f}_{anc,k}^* - \mathbf{f}_{neg,k}^*\|$ in equation 2.54 with the trainable metric outputs.

4.6.2 Hard Mining

Exposing the network to a non-matching pair of images depicting people with completely different appearances does not contribute to the networks ability to generalize. If however non-matching pair of images depicting similarly looking people are used, the network might learn to extract features not dependent on obvious traits such as color of clothing [34][18]. Batch hard mining is the method of constructing batches such that only sufficiently hard examples are included. It has been shown that such a method can significantly increase the convergence speed and overall generalization ability of a triplet network [34][18]. Batch hard mining can be applied to all the loss functions covered in this thesis. Batch hard mining requires multiple forward passes through the network, such that each example's contribution to the loss value is examined.

4.6.3 Unsupervised Learning

In order to train deeper and more complex models than those developed in this thesis project more data is needed. However, manual labeling of data is not a viable way forward when large datasets are created. Instead, performing unsupervised learning on large unlabeled datasets is a cheap and easily achieved way to pre-train deep complex networks. Fine tuning can later be performed on smaller labeled datasets.

Large unlabeled datasets can easily be created by the use of an object detector on sampled video sequences, similarity to the procedure used in the AMSR-id dataset development. There are numerous unsupervised learning methods applicable for pre-training of large complex re-identification networks. Two such methods are autoencoder and deep for embedding cluster analysis. These methods will briefly be presented in the following sections.

4.6.3.1 Autoencoder

The goal of an autoencoder is to learn a dense representation $\mathbf{f}$ which captures the essential information of the input $\mathbf{I}$. The dense representation is created by an encoder network. A decoder network receives thereafter $\mathbf{f}$ and tries to recreate the original input. If the decoder network is able to recreate the original input, the compact representation can be regarded as having captured all essential information from the input. Figure 4.7 shows an autoencoder network, constituted by an encoder and a decoder network.

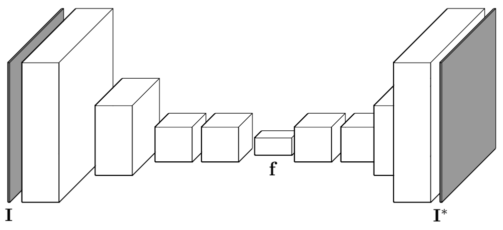

Figure 4.7: An auto encoder network, where $\mathbf{f}$ *is a dense representation of* $\mathbf{I}$ *and* $\mathbf{I}^*$ *is the recreated input.*

The autoencoder architecture can be used to pre-train the encoder network in a re-identification system on large unlabeled datasets of persons. The encoder network is in such a case trained to extract essential features from images depicting persons.

4.6.3.2 Deep Embedding for Cluster Analysis

Deep embedding for cluster analysis of images is the idea of grouping similar images together. This grouping can be achieved by performing an embedding of an input image $\mathbf{I}_i$ to a feature representation $\mathbf{f}_i$. The embedding can be performed by an encoder network, similar to those developed in this thesis project. Associated with a clustering scheme are n different clusters, where each cluster have a specific centroid $\boldsymbol{\mu}_k$. The goal of the clustering scheme is minimization of the distance $\min_k \|\boldsymbol{\mu}_k - \mathbf{f}_i\|$, which ensures that similar feature representations are grouped together. Deep embedding for cluster analysis has successfully been applied to image clustering [45], where the encoder network and cluster's centroids are trained in an end-to-end fashion.

Deep embedding for cluster analysis can be applied for pre-training of encoder networks on large unlabeled person datasets. The pre-trained networks can later be fine-tuned on smaller labeled re-identification datasets.

Chapter 5

Multi Shot Re-identification

Multi shot re-identification is the method of re-identifing a person between disjoint camera views by analysis of sequences of consecutive sampled images of persons.
The sequence of consecutive sampled images depicts a person trajectory. Behind this type of re-identification systems is the idea that more person specific information is present in a sequence of images compared to a single image. The hope is, performing temporal analysis of sequences, person specific movements i.e. the gait can be used as an attribute during re-identification. By examination of multiple images depicting the same person, non-temporal dependent features is compounded between the sampled images, features such as color of clothing, hair color etc.

Deep recurrent neural networks which perform a temporal analysis of person trajectories have been introduced in the past years [43][30]. Multi shot re-identification with recurrent units is a relative new concept, the lack of sufficient amount of training data have made the developed networks rather shallow and small [43][30].
This thesis project introduces the idea of transfer learning from single shot re-identification to multi shot re-identification. By applying pre-training on large single shot re-identification datasets, large and complex networks have been developed for multi shot re-identification. Figure 2.14 shows one such network. This chapter is built on the knowledge gained in the past chapter, where single shot re-identification was studied. Recurrent units together with a pre-trained network, similar to RE-ID-3-Classifier, have been used to perform extensive examinations of different recurrent units for multi shot re-identification purposes. Four different types of recurrent units have been evaluated. The different recurrent units have been trained on different sequence lengths. Two different temporal pooling methods have been tested, for all the different recurrent units.

The best performing recurrent unit has also been tested in a smaller network. This network was trained directly on multi shot re-identification, without the concept of transfer learning.

Certain chosen networks have also been trained on augmented datasets, which have enabled performance comparisons between networks trained on augmented and non-augmented datasets.

In order to evaluate if temporal analysis of sequences of images introduces increased performance, results obtained by networks with recurrent units have been compared to results obtained by a baseline network. The baseline network does not perform a temporal analysis on person trajectories.

In order to properly evaluate the developed network's re-identification performance in real-world situations a subset of the realistic Axis multi shot re-identification dataset (AMSR-id) has been used during evaluation.

This chapter presents the developed multi shot re-identification networks. The training and evaluation methods of the developed networks will also be presented. Results obtained from the multi shot re-identification study will be presented and analyzed. In the section dedicated to future work, ideas outside the scope of the thesis project will be presented.

5.1 Implementation Details

This section introduces implementation details regarding the training of the developed networks.

In order to stop the risk of exploding gradients during training all gradients in equation 2.17 were clipped to lie in the interval $[-5, 5]$, similarly as described in [43].

The ADAM optimizer was used to optimize the weights of the networks. The initial learning rate was chosen to be 5×10^{-4}, the learning rate was linearly decreased, similarly as the method applied during the single shot re-identification network training. The network's weights were initialized according to the Xavier weight initialization scheme [14]. All biases were initialized to zero, except the LSTM's forget gate bias in equation 2.39, which was initialized to 1, according to already stated theory.

Each image was resized, similarly as in the single shot re-identification training procedure. The images were preprocessed according to equation 4.1 before they were passed on to the networks.

Because of limiting amount of multi shot re-identification training data, a validation set was not used.

Approximately two thirds of the iLIDS-VID dataset was used during the network training procedure. Sequences containing 98 persons were used during testing and the remaining were used during training. All sequences depicting persons captured in two different camera views were used from the PRID-2011 dataset during training, totaling 200 persons. From AMSR-id were sequences depicting 55 persons chosen and used during testing, the remaining persons were used during training. In total were 467 persons included in the training datasets.

Each network was trained on non-augmented multi shot datasets. Some of the best performing networks were also trained on augmented images, resulting in enlarged training datasets and longer training time. The same augmentation adjustments were used as in the single shot re-identification training procedure. Data augmentation adjustments were applied similarly to all images within a sequence.

5.2 Evaluation Method

The developed networks were evaluated on two different datasets, AMSR-id and iLIDS-VID. The same evaluation procedure was employed as in the single shot re-identification case, however, with sequences instead of images. The different networks were evaluated on different sequence lengths. The test sequence length describes the sequence length the networks were permitted to use during testing. A test sequence length of 160 does not mean that all sequences used during testing include 160 images. That means only that the networks were permitted to use 160 images, if they existed in the sequence. In the best situation, all sequences would have equal an number of images, but this was not the case. The evaluation method used in this study gives however an idea of how increased test sequence lengths affect the re-identification performance. The average sequence length for the different datasets is presented in the chapter dedicated to the datasets.

During the evaluation procedure a sub-sequence was chosen. In order to obtain comparable results, the different sub-sequences were chosen similarly for all evaluated networks. Because the two test sets did not have 100 persons, 98 persons from iLIDS-VID and 55 persons from AMSR-id were used during the evaluation procedure.
Because of the sheer amount of evaluated networks, only the rank-1 accuracies will be presented in the result sections.

5.3 Proposed Multi Shot Re-identification Networks

This section will present the different networks. Firstly, the baseline network will be introduced, this network does not perform any temporal analysis on the images in the sequences. The results obtained by the baseline network are regarded as the baseline when recurrent units are examined. The baseline network is similar to the RE-ID-3-Classifier network, with an encoder and a classifier network.

In the second part, multiple different recurrent units and configurations have been evaluated, together with the trained encoder network from the baseline network. This part makes use of transfer learning from single shot to multi shot re-identification, enabling faster evaluation of the different recurrent units and configurations, compared to complete training of the networks, from scratch.
In the third part, the best performing recurrent unit has been used in a smaller network, trained directly on the different multi shot datasets, without the concept of transfer learning.

5.3.1 Baseline

The baseline network is similar to the RE-ID-3-Classifier network, seen in tables 4.1 and 7.5. The network classifies two images/sequences as either matching or not matching.

The different multi shot datasets contain fewer examples compared to the larger single shot datasets. The classifier network was thus significantly reduced, compared to the RE-ID-3-classifier network. The encoder network was kept the same. The classifier network was reduced in the following way. The number of filters in the convolutional layer was reduced from 512 to 128. The first fully connected layer, following the convolutional layer, was reduced from 1000 neurons to 256 neurons. A dropout layer was added following the first fully connected layer, enabling regularization of the network. The probability of random node removal was chosen to be 50 %. After the dropout layer, a fully connected layer with 128 neurons was added. This layer was followed by the output layer with two neurons. All layers, except the output layer, uses layer normalization together with the ReLu activation function.

The baseline network was trained for single shot re-identification together with multiple single shot datasets seen in the chapter dedicated to the datasets. All datasets, except the reduced iLIDS-VID dataset, previously used during the single shot re-identification network training were used in baseline network training. The reduced iLIDS-VID dataset includes images from both the iLIDS-VID test and training dataset. The iLIDS-VID test dataset was instead used. The AMSR-id test dataset was also included during the training. The network was trained for 200,000 iterations, similarly as the large classifier network in the single shot study. The batch size was chosen to be 50.

Initial tests showed problems, the loss value did not converge during training. In order to address this problem, different changes were carried out to the network. The ReLu activation function in the first fully connected layer, in the classifier network, was removed, making the layer's output linear. If a large part of the output from the ReLu activation function would be zero. The amount of information passed on to the next layer would be small, due to the dropout layer placed directly after. This was previously not a severe problem when the fully connected layer had 1,000 neurons. Due to the problem inherent with the ReLu activation function, large parts of the network may have become dead during training. All ReLu activation functions were exchanged to ELU functions. Table 5.1 shows the classifier network in the baseline network. The network showed increased performance after the changes, and the loss value converged during training.

Table 5.1: The classifier network in the baseline network.

Type	Output Dim.	Filter Size	Stride	Act. func.	Normalization
Convolutional	$9 \times 4 \times 128$	3×3	1	ELU	Layer Normalization
Fully Connected	$1 \times 1 \times 256$	-	-	-	Layer Normalization
Dropout	-	-	-	-	-
Fully Connected	$1 \times 1 \times 128$	-	-	ELU	Layer Normalization
Fully Connected	$1 \times 1 \times 2$	-	-	-	-

The trained network was also fine tuned for single shot re-identification on the multi shot training datasets, this was done for 30,000 iterations.

In addition, a separate baseline network was trained on augmented datasets, the number of training iterations were increased to 400,000. The network was also fine-tuned for 50,000 iterations on the different augmented multi shot datasets.

Present in different sequences are outliers, caused by occlusion, bad crop, motion blur etc. Outliers are heavily present in sequences from AMSR-id. One interesting aspect to study is if outliers in images correspond to outliers in their feature representations, outputted from the encoder network. Figure 5.1 and 5.2 show 20 images from two sequences, ten of those are regarded as inliers and the others as outliers.

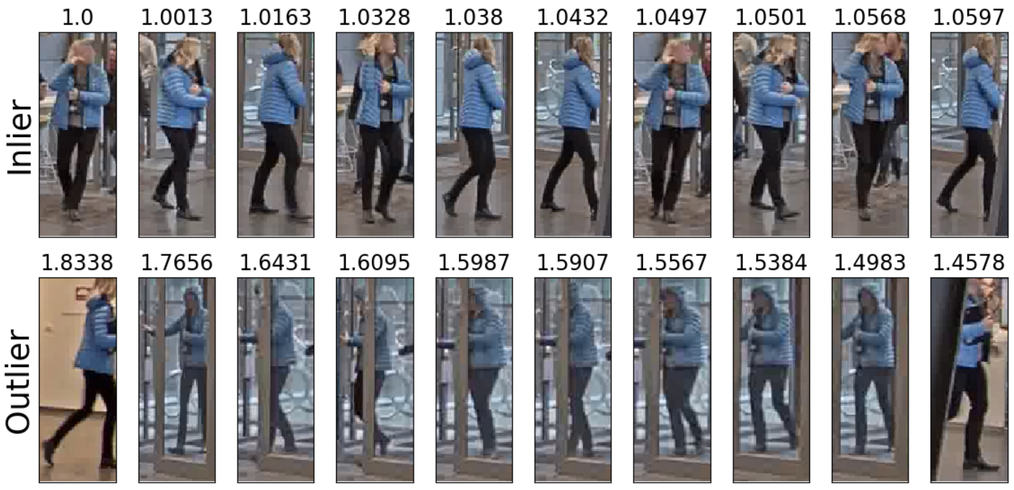

Figure 5.1: Inliers and outliers in one sequence from AMSR-id. Above each image is the normalized euclidean distance between the mean feature representation for all images in the sequence and the image's feature representation.

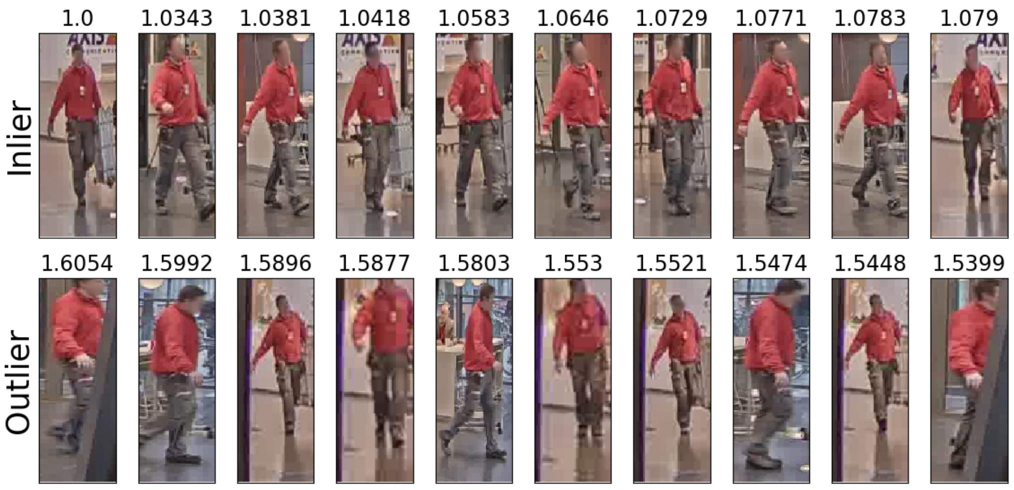

Figure 5.2: Inliers and outliers in one sequence from AMSR-id. Above each image is the normalized euclidean distance between the mean feature representation for all images in the sequence and the image's feature representation.

Above each image is the normalized euclidean distance from the image's feature representation to the mean feature representation for all the images in the sequence. The ten images with feature representations closest to the mean feature representation are regarded as inliers in the two figures 5.1 and 5.2. The ten images with feature representations furthest away from the mean feature representation are regarded as outliers in the two figures 5.1 and 5.2. Each distance has been normalized by division with the smallest distance. In the two cases, outliers in the image sequences correspond to outliers in their feature representation. This was observable for images with bad crop, bad quality and images affected by occlusion.

The baseline network was extended to the processing of sequences. A temporal mean pooling layer was added after the encoder network and before the classifier network. This extended network examines sequences by applying temporal mean pooling directly on sequences of feature representations, outputted from the encoder network. The result is passed on to the classifier network.
This network does not perform any temporal analysis on sequences, i.e. each image is separately processed by the encoder network, as in a single shot re-identification network.
The results obtained by this network are regarded as the baseline when recurrent units are examined. The results can be seen in table 5.2. The results obtained on AMSR-id are seen inside parentheses (·). Results in bold indicates the best rank-1 accuracy for a particular network.

Table 5.2: Rank-1 accuracies on iLIDS-VID and AMSR-id (·) with the baseline networks together with temporal mean pooling.

Train\Test seq. l.	1	5	10	20	40	80	160
1	31.8 (17.5)	33.7 (21.8)	39.8 (21.8)	48.0 (27.3)	56.1 (36.4)	62.2 (**36.4**)	**64.3**(34.5)
1 Augmented	32.9 (24.5)	36.7 (40.0)	48.0 (40.0)	55.1 (43.6)	65.3 (45.5)	68.4 (**52.7**)	**68.4**(50.9)

The networks perform significantly better when sequences are examined, compared to the cases when only a single image is used. Longer test sequence lengths yield overall better rank-1 accuracies, compared to shorter test sequence lengths.
The network trained on the augmented datasets performs significantly better than the network trained on the non-augmented datasets, especially on AMSR-id. The different augmentation adjustments might have made the other training datasets to closer resemble AMSR-id. Bad image crops are often present in sequences from AMSR-id. The network trained on the augmented datasets might thereby have learned to be somewhat tolerant towards bad crops.

5.3.2 Temporal Analysis with Recurrent Units

Recurrent units may increase the re-identification performance in two different ways. The recurrent units may compound valuable non-temporal information and neglect bad information from sequences i.e. act as outlier filters. Valuable non-temporal information is, color of clothing, hair color etc. Present in some sequences are occlusions, images affected by occlusion are considered bad information. The second way in which a recurrent unit may increase the performance is by temporal analysis of certain person specific movements.

Different recurrent units have been tested together with the pre-trained baseline encoder network. Four different types of recurrent units have been evaluated, two convolutional based recurrent units (conv-GRU and conv-LSTM) and two vector/matrix based recurrent units (GRU and LSTM). Usual training sequence lengths for multi shot re-identification networks are 16 and 20 [43][30]. The recurrent networks in this study have been trained on sequence lengths of 10 and

20. The best performing recurrent unit has also been trained with a sequence length of 40. Figure 2.14 illustrates the whole network in those cases a convolutional recurrent unit is used.

The steps in the temporal analysis of sequences are,

1. All images in a sequence pass through the baseline encoder network, such that a feature representation for each image is extracted.
2. The output from the encoder network is layer normalized.
3. A cross channel parametric pooling layer with 256 filters reduces the output from the baseline encoder network into a dense representation. The layer's activation is layer normalized and passed through an ELU activation function.
4. For vector/matrix based recurrent units, the feature map must be reduced to a vector, before it is passed on to the recurrent unit. In those cases, a fully connected layer with 512 neurons is placed directly after the cross channel parametric pooling layer. The layer's activation is layer normalized and passed through an ELU activation function.
5. A recurrent unit performs temporal analysis.
   - The added vector/matrix based recurrent units use,
     - LSTM - vectors with dimensions (1×512) for the hidden state and cell state.
     - GRU - a vector with dimensions (1×512) for the hidden state.
   - The added convolutional recurrent units use 256 filters with (3×3) convolutional filter sizes on,
     - conv-LSTM - feature maps with dimensions $(9 \times 4 \times 256)$ for the hidden state and cell state.
     - conv-GRU - a feature map with dimensions $(9 \times 4 \times 256)$ for the hidden state.
6. A temporal pooling function reduces the output for all the time steps from the recurrent unit to one feature representation. Two different temporal pooling methods were tested,
   - Temporal mean pooling.
   - Last output pooling.
7. The classifier network receives the squared element-wise difference between the outputs from the siamese network architecture. The classifier network classifies the person/persons in the sequences as either matching or not matching.
   In those cases a convolutional recurrent unit is used, the classifier network is the same as the one used in the baseline network.
   In those cases a vector/matrix based recurrent unit is used, the first layer in the classifier network is replaced by a fully connected layer with 512 neurons. The layer's activation is layer normalized and passed though an ELU activation function.

The different networks with recurrent units were trained in two steps.

1. The pre-trained baseline encoder networks were kept constant for,
   - non-augmented datasets - 30,000 iterations

   - augmented datasets - 30,000 iterations

   The added weights are adjusted to be in line with the pre-trained encoder network's output during this step.

2. The whole networks were trained for,
   - non-augmented datasets - 20,000 iterations
   - augmented datasets - 50,000 iterations

   with one 10th the learning rate for the pre-trained encoder network.

The number of training iterations for the different networks were chosen heuristically by observing the convergence of the loss value.

Initial tests showed that recurrent units with layer normalization, preceding every non-linear activation function, outperformed those recurrent units without layer normalization. More extensive tests were carried out with layer normalization inside the recurrent units.

The results obtained by the different recurrent neural networks can be seen in the tables below. The tables show results obtained by the networks in different cases. Cases marked by (M) indicate that temporal mean pooling was used and cases marked by (L) indicate that last output pooling was used. A star "*" indicates that layer normalization was not used inside the recurrent unit. All networks not marked by a star use layer normalization inside the recurrent unit, preceding every non-linear activation function. The results obtained on AMSR-id are seen inside parentheses $(\cdot)$. Results in bold indicate the best rank-1 accuracies obtained by a particular network.

5.3.2.1 Convolutional Gated Recurrent Unit

Table 5.3 shows results obtained by networks which use the conv-GRU.

Temporal Mean Pooling The overall rank-1 accuracies on iLIDS-VID increase as the test sequence length becomes longer. Those networks trained on non-augmented datasets produce the best results on AMSR-id when the test sequence length is 40.

Longer training sequence length seems to increase the rank-1 accuracies on iLIDS-VID for long test sequences, the same is not observable on AMSR-id. In those cases the test sequence length is short, the networks trained with sequence length of 10 perform better than the network trained with sequence length of 20.

Layer normalization, applied inside the recurrent units, increases the performance significantly on both datasets, especially for long test sequence lengths.

The network with the conv-GRU, trained with sequence length of 20, performed the best on iLIDS-VID, amongst all evaluated networks with a recurrent unit. This network was also trained on augmented datasets. The results yielded by the network trained on the augmented dataset show substantial increase in rank-1 accuracies on AMSR-id, compared to the same network trained on the non-augmented datasets. No significant difference is observable on iLIDS-VID.

The network with the conv-GRU was also trained with sequence length of 40. This network performs worse for short sequence lengths compared to those networks trained with shorter sequence lengths. However, when the test sequence length is long (≥ 80), the network produces results comparable with the network trained with sequence length of 20.

Last Output Pooling The networks trained with last output pooling seem to perform worse on test sequence lengths significantly shorter or longer than the one used during training. This becomes especially evident when one study the performance difference between the networks trained with sequence lengths of 10 and 20, in those cases when the test sequence lengths are 5 and 160.
Layer normalization, applied inside the recurrent unit, increases the performance for some test sequence lengths, the increase is however not as significant as in the temporal mean pooling case.

*Table 5.3: Rank-1 accuracies on iLIDS-VID and AMSR-id (·) with the conv-GRU network for different test and training sequence lengths. Temporal mean pooling (M). Last output (L). Networks marked by a star * do not use layer normalization inside the recurrent unit.*

Train\Test seq. l.	5	10	20	40	80	160
10 (M)	34.7 (23.6)	44.9 (32.7)	57.1 (32.7)	69.4 (**40.0**)	70.4 (34.5)	**71.4** (32.7)
10* (M)	35.7 (18.2)	42.8 (18.2)	**53.0** (27.3)	51.0 (**30.9**)	42.9 (16.4)	42.9 (18.2)
20 (M)	29.6 (23.6)	41.8 (27.2)	56.1 (36.4)	66.3 (**38.2**)	73.5 (34.5)	**75.5** (34.5)
20 (M) Augmented	27.6 (30.9)	39.8 (38.2)	55.1 (38.2)	66.3 (41.8)	73.5 (**47.3**)	**73.5** (41.8)
40 (M)	27.6 (18.2)	34.7 (25.5)	50.0 (25.5)	59.2 (**34.5**)	73.5 (34.5)	**74.5** (30.9)
10 (L)	38.8 (21.8)	51.0 (29.1)	**59.2** (27.3)	49.0 (**30.9**)	43.8 (20.0)	40.8 (21.8)
10* (L)	36.7 (23.6)	43.9 (23.6)	48.0 (30.9)	**55.1** (**30.9**)	44.9 (18.2)	44.9 (18.2)
20 (L)	29.6 (20.0)	45.9 (**27.3**)	54.1 (29.1)	**59.2** (23.6)	46.2 (23.6)	51.0 (18.2)

The networks with temporal mean pooling perform overall better than those with last output pooling. Figure 5.3 shows rank-1 accuracies on iLIDS-VID, for different sequence lengths and temporal pooling methods. Both networks were trained with sequence length of 20.
The two pooling methods produce comparable results on short test sequence lengths. The results start however to diverge as the test sequence length becomes significantly longer than the training sequence length.

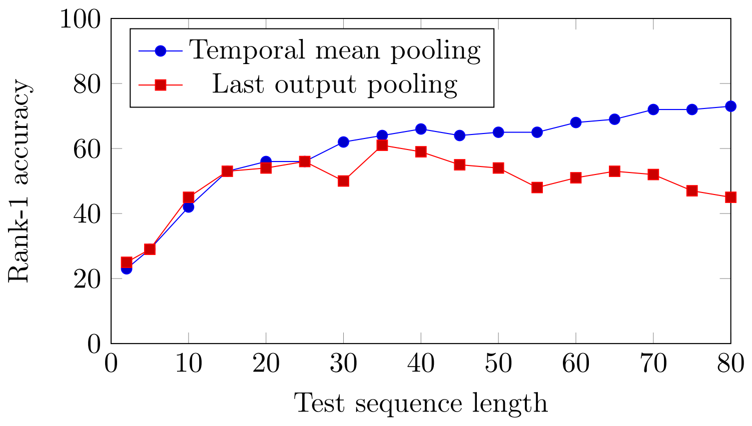

Figure 5.3: Rank-1 accuracy on iLIDS-VID, for different test sequence lengths and temporal pooling methods, by the conv-GRU network trained on the non-augmented datasets with a sequence length of 20.

All results presented below were obtained by the network with the conv-GRU, trained with sequence length of 20 and with temporal mean pooling. The network was trained on non-augmented datasets.

Figure 5.4,5.5 and 5.6 show different re-identification cases from AMSR-id.
Each sequence is represented by their first image. The sequence length used during testing was 80. To the left is the test sequences represented. To the right are the ten most probable matches amongst 55 persons in the gallery. The list is sorted in descending order, from the most probable to less likely. Above each image is a similarity score, scores marked in green indicate correct matches.

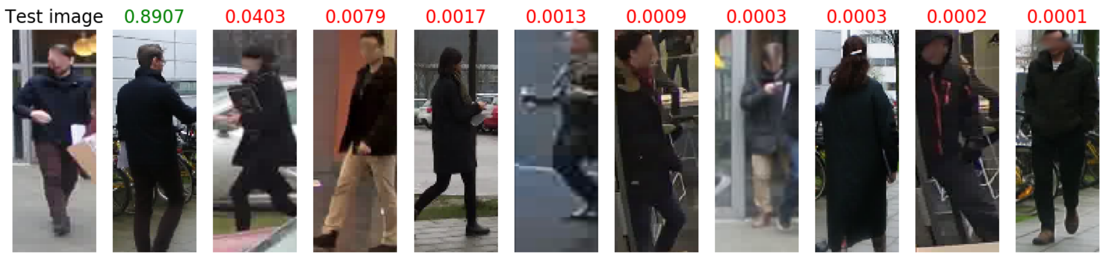

Figure 5.4: Each sequence is represented by their first image. To the far left: a test sequence depicting a test person. To the right: a collection of the ten most probable re-identification matches.

In the first case, illustrated by figure 5.4, the network has found persons with similar black clothing as the test person. The test person was re-identified with rank 1. In the second case, illustrated by figure 5.5, the four most likely persons have similar blue jackets as the test person. The test person was re-identified with rank 3. Similarly as in the single shot re-identification case, the multi shot networks make use of color of clothing when persons are re-identified.

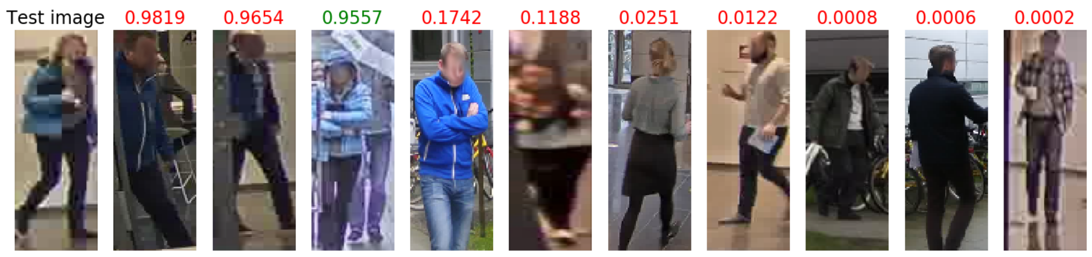

Figure 5.5: Each sequence is represented by their first image. To the far left: a test sequence depicting a test person. To the right: a collection of the ten most probable re-identification matches.

The case illustrated by figure 5.6 shows an example when the test person was not re-identified amongst the ten most likely persons. All persons amongst the ten most likely have low similarity scores.

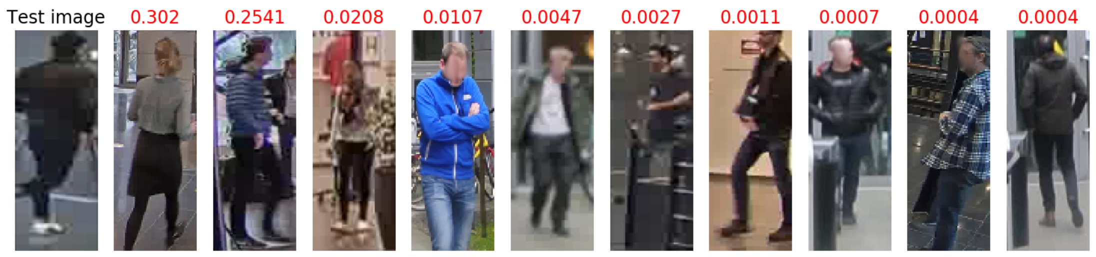

Figure 5.6: Each sequence is represented by their first image. To the far left: a test sequence depicting a test person. To the right: a collection of the ten most probable re-identification matches.

5.3.2.2 Convolutional Long Short Term Memory

Table 5.4 shows results obtained by networks which use the conv-LSTM.

Temporal Mean Pooling Similarly as those networks with the conv-GRU, the overall rank-1 accuracies on iLIDS-VID increase as the test sequence length becomes longer. The same trend is not as distinct on AMSR-id.

The network trained with sequence length of 20 outperforms the networks trained with sequence length of 10 on the iLIDS-VID dataset.
Layer normalization, applied inside the conv-LSTM, increases the performance on the iLIDS-VID dataset for long test sequence lengths, the increase is not however as substantial as in the conv-GRU case with temporal mean pooling. On AMSR-id is the highest rank-1 accuracy obtained by the network without layer normalization inside the conv-LSTM. Layer normalization inside the conv-LSTM does not seem to increase the overall performance as significantly as in the conv-GRU case with temporal mean pooling.

Last Output Pooling Similarly as for the networks with the conv-GRU the networks trained with last output pooling perform worse on test sequence lengths significantly longer than the one used during training. Layer normalization, applied inside the conv-LSTM, increases the overall performance for long test sequences (≥ 80) on iLIDS-VID, the same is not observable on AMSR-id. The overall tendency is however, layer normalization inside the conv-LSTM increase the performance on the two datasets.

*Table 5.4: Rank-1 accuracies on iLIDS-VID and AMSR-id (·) with the conv-LSTM network for different test and training sequence lengths. Temporal mean pooling (M). Last output (L). Networks marked by a star * do not use layer normalization inside the recurrent unit.*

Train\Test seq. l.	5	10	20	40	80	160
10 (M)	30.6 (25.5)	43.9 (30.9)	55.1 (25.5)	66.3 (34.5)	69.4 (**38.2**)	**70.4** (34.5)
10* (M)	41.8 (21.8)	49.0 (30.9)	59.2 (30.9)	59.2 (**40.0**)	**67.3** (32.7)	63.3 (34.5)
20 (M)	32.6 (27.7)	48.0 (29.1)	60.2 (**38.2**)	66.3 (27.3)	71.4 (21.8)	**72.4** (18.2)
10 (L)	33.7 (20.0)	**51.0** (29.1)	49.0 (**34.5**)	51.0 (32.7)	45.9 (20.0)	41.8 (16.4)
10* (L)	35.7 (25.5)	33.7 (**36.4**)	49.0 (29.1)	**51.0**(29.1)	40.8 (20.0)	29.6 (16.4)
20 (L)	40.8 (27.7)	49.0 (29.1)	**54.1** (**38.2**)	51.0 (27.3)	40.8 (21.8)	36.7 (18.2)

5.3.2.3 Gated Recurrent Unit

Table 5.5 shows results obtained by networks which use the GRU.

Temporal Mean Pooling Similarly as for the other recurrent units, evaluation on long test sequence lengths produce overall better results, compared to shorter sequences. Layer normalization inside the GRU increases the performance for almost all test sequence lengths.

Last Output Pooling Similarly as in the other cases with last output pooling, the performance decreases as the test sequence lengths become significantly longer than the training sequence length. Layer normalization, applied inside the GRU introduces overall performance gains.

*Table 5.5: Rank-1 accuracies on iLIDS-VID and AMSR-id (·) with the GRU network for different test and training sequence lengths. Temporal mean pooling (M). Last output (L). Networks marked by a star * do not use layer normalization inside the recurrent unit.*

Train\Test seq. l.	5	10	20	40	80	160
10 (M)	25.5 (21.8)	39.8 (23.6)	41.8 (29.1)	50.0 (40.0)	53.1(34.5)	**53.1** (**34.5**)
10* (M)	30.6 (14.5)	35.7 (20.0)	38.8 (18.2)	43.9 (**23.6**)	39.8 (21.8)	**44.9** (20.0)
20 (M)	29.6 (23.6)	35.7 (20.0)	41.8 (29.1)	49.0 (27.3)	58.2 (29.1)	**59.2** (**29.1**)
10 (L)	32.7 (21.8)	39.8 (21.8)	43.9 (**27.3**)	**50.0** (20.0)	39.8 (21.8)	39.8 (20.0)
10* (L)	30.6 (14.5)	35.7 (20.0)	38.8 (18.2)	**43.9** (**23.6**)	39.8 (21.8)	44.9 (20.0)
20 (L)	27.5 (18.2)	35.7 (29.1)	41.8 (**29.1**)	**53.0** (25.5)	45.9 (18.2)	49.9 (20.0)

5.3.2.4 Long Short Term Memory

Table 5.6 shows results obtained by networks which use the LSTM.

Temporal Mean Pooling The networks with the LSTM do not seem to be able to produce the same overall increase in performance for longer test sequence lengths as the other network with recurrent units. Interestingly, layer normalization inside the LSTM does not seem to affect the results as with the other networks with other recurrent units. The network without layer normalization inside the LSTM outperforms the networks with layer normalization inside the LSTM on the iLIDS-VID dataset, for long test sequence lengths (≥ 40).

Last Output Pooling Similarly as the other networks with last output pooling, the overall performance decreases when the test sequence lengths become significantly longer than the training sequence length. Layer normalization, applied inside the LSTM introduces performance gain for almost all test sequence lengths.

*Table 5.6: Rank-1 accuracies on iLIDS-VID and AMSR-id (·) with the LSTM network for different test and training sequence lengths. Temporal mean pooling (M). Last output (L). Networks marked by a star * do not use layer normalization inside the recurrent unit.*

Train\Test seq. l.	5	10	20	40	80	160
10 (M)	31.6 (12.7)	36.7 (21.8)	48.0 (25.5)	**52.0** (**34.5**)	50.0 (30.9)	50.0 (32.7)
10* (M)	30.6 (21.8)	36.7 (23.6)	45.9 (27.3)	**53.1** (**29.1**)	53.1 (25.5)	53.1 (23.6)
20 (M)	21.4 (12.7)	25.5 (18.2)	41.8 (30.9)	**52.0** (**34.5**)	49.0 (27.3)	48.0 (27.3)
10 (L)	21.4 (18.2)	32.6 (**27.3**)	36.7 (23.6)	**41.8** (14.5)	27.6 (20.0)	31.6 (14.5)
10* (L)	27.6 (27.3)	27.6 (29.1)	**36.7** (**29.1**)	32.7 (29.1)	24.5 (14.5)	27.6 (12.7)
20 (L)	25.5 (12.7)	34.7 (16.4)	42.9 (21.8)	**45.9** (**27.3**)	41.8 (21.2)	39.8 (18.2)

5.3.3 Complete Training of Small Recurrent Network

The best performing recurrent unit, conv-GRU, was tested in a smaller network. The smaller network architecture is similar to the baseline network. However, significantly fewer weights are used, which enabled direct training on the different multi shot datasets. The network can be seen in table 7.6 in the appendix.

The encoder network uses six convolutional layers. Cross channel parametric pooling layers with (1×1) filter sizes are placed in between convolutional layers with (3×3) filter sizes. The spatial dimensions of the input image are reduced by two (3×3) max-pooling layers with stride 3, similarly as in the baseline encoder network. Following the convolutional encoder network comes a conv-GRU. The conv-GRU uses 64 filters with (3×3) filter sizes on the hidden state. Layer normalization is used inside the conv-GRU.
The outputs from the conv-GRU, for all the time steps, are passed through a temporal pooling function. The classifier network examines the squared element-wise difference between the outputs from the siamese network. The classifier network has a similar design as the baseline network, fewer convolutional filters are however used in the first layer.

The network was trained directly on the different multi shot datasets, with a sequence length of 20. The network was trained with two different temporal pooling methods, mean pooling and last output. In both cases, the networks were trained on augmented datasets and non-augmented datasets. The networks trained on the non-augmented datasets were trained for

100,000 iterations and the networks trained on the augmented datasets were trained for 120,000 iterations. The number of training iterations for the different networks were chosen heuristically by observing the convergence of the loss value.

The results obtained by the networks can be seen in table 5.7.

The table shows also the results when the sequences were shuffled during evaluation, the networks were still trained on non-shuffled sequences. Shuffling makes the images in the sequences non-sequential, thereby removing temporal dependencies between the images.

Table 5.7: Rank-1 accuracies on iLIDS-VID and AMSR-id (·) for different test sequence lengths. Temporal mean pooling (M). Last output (L).

Train\Test seq. l.	5	10	20	40	80	160
20 (M)	12.2 (10.9)	23.5 (18.2)	43.9 (25.5)	58.2 (**27.7**)	59.2 (25.5)	**65.3** (25.5)
20 (M) Shuffled	15.1 (14.5)	30.8 (18.4)	44.9 (24.4)	54.5 (**28.7**)	53.7 (24.0)	**58.4** (22.4)
20 (M) Augmented	15.3 (9.1)	27.6 (18.2)	44.9 (20.0)	61.2 (**25.5**)	66.3 (23.6)	**69.4** (23.6)
20 (L)	25.5 (12.7)	34.7 (16.4)	**56.1** (**32.7**)	55.1 (18.2)	49.0 (25.5)	48.0 (21.8)
20 (L) Shuffled	24.6 (17.1)	35.2 (18.5)	52.7 (**26.9**)	**60.3** (24.1)	60.2 (21.3)	59.2 (18.9)
20 (L) Augmented	26.5 (7.3)	43.9 (14.5)	**54.1** (20.0)	51.0 (23.6)	54.1 (**30.9**)	54.1 (27.3)

Interestingly, evaluation on shuffled sequences yields better results in multiple cases compared to evaluation on non-shuffled sequences.
In the temporal mean pooling case, the performance is better when short sequences are shuffled compared to the case when the sequences are not shuffled. The performance on the iLIDS-VID dataset becomes however worse when the test sequence length is longer or equal to 40, compared to the results on the non-shuffled sequences.
The performance on shuffled sequences does not show the same substantial increase in performance for long test sequence lengths compared to the performance on non-shuffled sequences. The network seems to have learned to depend on some temporal dependencies in long sequences.

The network with temporal mean pooling, trained on the augmented datasets, produces better performance on the iLIDS-VID dataset for all sequence lengths compared to the network trained with the non-augmented datasets. However, the overall performance is worse on AMSR-id. The overall performance on the two datasets is thereby the opposite compared to the pre-trained network with the conv-GRU which was trained on the augmented datasets. The trained network is considerably smaller and less complex compared to the previous network with the conv-GRU. The network might not be able to learn a representation which is able to generalize and produce good results on the difficult AMSR-id.

Surprisingly, the network with last output pooling performs better on shuffled sequences from the iLIDS-VID dataset, for almost all test sequence lengths. The performance on AMSR-id tends to vary.
The performance on non-shuffled sequences is however significantly better on both datasets when the test and training sequence lengths are equal. The network seems to be dependent on some temporal dependencies when the test sequence length is the same as the training sequence length. Training on the augmented datasets seems to improve the performance in certain cases when last

output pooling is used, especially for long sequences (≥ 80) from the iLIDS-VID dataset. The overall performance on AMSR-id is however worse on short sequences, compared to the case when non-augmented datasets were used, similarly as in the temporal mean pooling case.

The networks with temporal mean pooling produce overall better results than the networks with last output pooling.

5.4 Previous Work & Comparisons

Niall McLaughlin, Jesus Martinez del Rincon and Paul Miller used a standard recurrent unit with a convolution encoder network in their developed multi shot re-identification system [30]. The convolutional encoder network was firstly fine tuned on a small single shot dataset. The network was trained using a siamese network architecture with temporal mean pooling.
A rank-1 accuracy of 58 %, with 150 persons in the gallery, was obtained on the iLIDS-VID dataset. The network was also trained and tested on PRID-2011 with 100 persons in the gallery, a rank-1 accuracy of 70 % was obtained in this case.
In order to estimate the real world performance of their developed network, cross dataset evaluation was tested. Their network was trained on iLIDS-VID and evaluated on PRID-2011. A rank-1 accuracy of 28 % was obtained in this case, significantly lower than the result obtained by the same network trained on a subset of PRID-2011.

Lin Wu, Chunhua Shen and Anton van den Hengel used a conv-GRU together with a convolutional encoder network in their multi shot re-identification network [43]. The network was trained using a siamese network architecture.
Their system was evaluated on the iLIDS-VID dataset with 150 different persons in the gallery. Different temporal pooling methods were tested in their system, Fisher vector pooling [37] yielded slightly better results compared to the temporal mean pooling approach [43]. A rank-1 accuracy of 43.2 % was obtained using Fisher vector pooling, and a rank-1 accuracy of 42.6 % was obtained using temporal mean pooling.

In order to ensure comparable performance measurements between the best developed multi shot re-identification network and those introduced in this section, the best network was re-trained and evaluated on subsets, containing 150 persons, of the iLIDS-VID dataset. The best network i.e. the large recurrent network with the conv-GRU and temporal mean pooling was trained in a similar fashion as described before, except the changes to the iLIDS-VID dataset. A rank-1 accuracy of 50 % was obtained on iLIDS-VID, with test sequence lengths of 80 and 160 .

5.5 Conclusion & Discussion

In order to properly evaluate the different recurrent units in the multi shot re-identification networks, the networks should have been trained multiple times and the results averaged. Because of limiting time and resources, this was not done. The results obtained by the different networks tend to vary from different test sequence lengths and evaluation datasets. In order to properly evaluate why the performance decreases for some test sequence lengths and increases for other, extensive tests are needed. Such tests were however out of the scope of this thesis project. Sporadic outliers within some sequences may explain the performance variations.

The broad examination of different recurrent units and configurations have yielded results which should be interpreted as a first step in a deeper examination. Although, overall conclusions can be drawn. Those conclusions will be presented below.

Images from AMSR-id are heavily affected by different undesirable artifacts, such as bad crop, low quality and motion blur. The different undesirable artifacts can be seen in figure 5.4, 5.5, 5.1,5.2 and 5.6. These artifacts might explain the inferior performance obtained on AMSR-id, compared to the iLIDS-VID dataset.

The results obtained by the baseline network has shown the successful extension of a single shot re-identification network to multi shot re-identification. The baseline network outperforms multiple networks with recurrent units. Significant increase in performance was observed on AMSR-id when the baseline network was trained on augmented images, compared to the case when the network was trained on non-augmented datasets.
Interestingly, the baseline network, trained on the augmented datasets, outperforms all recurrent networks on AMSR-id. The classifier network in the baseline network was trained on many more datasets than those used during the multi shot re-identification training. The baseline network has possibly learned a more generalized representation compared to the recurrent networks.

The networks with vector/matrix based recurrent units perform overall worse compared to those networks with convolutional recurrent units. The performance difference between those networks which use convolutional recurrent units and those networks which use vector/matrix based recurrent units are dependent on multiple factors, besides the use of convolutional operations inside the recurrent unit. One such factor is the use of another classifier network. No real conclusion can be drawn regarding if the performance difference is dependent on the different recurrent unit or other changes to the network architecture.
The large recurrent networks with the conv-GRU outperforms all other recurrent networks. Similarly as for their convolutional versions, the GRU outperforms the LSTM in most cases.

The large networks with convolutional recurrent units (conv-GRU and conv-LSTM) outperform the baseline network on iLIDS-VID. Besides temporal analysis of the sequences, the recurrent units may have learned to filter out outliers, present in the sequences.
The large recurrent network with the conv-GRU, trained on the augmented datasets, perform significant better on AMSR-id, compared to the same network trained on non-augmented datasets. The large recurrent network, trained on the augmented datasets, has possibly learned to be somewhat tolerant towards bad crops, heavily present in AMSR-id.

A recurrent unit with temporal mean pooling is encouraged to compound valuable external information between the different time steps. Because the outputs from the recurrent unit are averaged between the different time steps, outliers within sequnces tend to be subdued.
Those recurrent units with last output pooling are encouraged to only pass on the essential information from their state/states and external input from one time step to another. Intuitively, recurrent units with last output pooling are more sensitive towards multiple consecutive outliers within sequences. Multiple consecutive outliers, especially occlusions, within sequences may well cause the internal state/states to be updated in such a way that previous information is lost.
Mean pooling outperforms last output as the temporal pooling method for long sequences. For short sequences the difference in performance tends to vary, and in some cases outperform last output pooling temporal mean pooling. The networks with last output pooling perform worse

on sequence lengths significantly shorter or longer than the one used during training. Possibly, the recurrent units have learned to depend on a fix sequence length during training. This may explain why the performance declines when the test sequence lengths are significantly longer than the training sequence length.

Layer normalization, applied inside the conv-GRU, conv-LSTM and the GRU increases the overall performance, in those cases temporal mean pooling is used. Layer normalization normalize all activations inside the recurrent units to have similar distributions, for all time steps. If layer normalization is not used, the activation distributions become different from time step to time step, which may be undesirable when temporal mean pooling is performed.
Applying layer normalization inside those recurrent units which use last output pooling showed some performance gain, however the overall performance gain was not as significant as in the temporal mean pooling case.

Those networks trained with sequence length of 20 produce overall better rank-1 accuracies on long test sequence lengths, compared to those networks trained with sequence length of 10. The network trained with sequence length of 40 showed however no real increase in performance, compared to the same network trained with sequence length of 20.
Lack of sufficient amount of data makes training on long sequence lengths hard. Longer training sequence lengths reduce the number of unique training examples. The training datasets may thereby have been too small for the network to gain increased performance when the training sequence length was 40.

Cross dataset evaluation was carried out with the large recurrent network with the conv-GRU. The network was trained with sequence lengths of 20. The AMSR-id dataset was excluded from the training datasets. A rank-1 accuracy of 21.8 % was obtained on AMSR-id. The performance is significantly worse compared to the case when the network was trained on a subset from AMSR-id. As discussed in the single shot re-identification chapter, dataset bias is possible to blame. The network has certainly been over-fitted to particular scenarios in the training datasets, not presented in the excluded evaluation dataset.
The authors behind [30] came to the same conclusion in their cross dataset evaluation case:
"One cause of this problem is that any given dataset represents only a small fraction of all real-world data, making it difficult for the system to learn which aspects of the training data are essential to the problem, and which are just artifacts of the dataset."
The AMSR-id dataset depicts scenarios not present in the other datasets, with bad crops and large differences of quality from image to image within a sequence.

The large pre-trained baseline encoder network may well have been made invariant to differences in person poses during the single shot re-identification training. When person specific movements are analyzed by the added recurrent units this might have introduced a severe problem.
However, the results obtained on shuffled sequences, by the networks trained directly on the multi shot datasets, showed that temporal dependent information about person specific movements has not been used as the primary feature when persons are re-identified. Even though some results on shuffled sequences indicate that temporal dependencies amongst images are used for long sequences when temporal mean pooling is used. In those cases last output pooling is used, and the test and training sequence lengths are equal, the recurrent units seem to depend on some temporal dependencies within the sequences, as indicated by the results obtained by the small

recurrent network.
The performance difference between shuffled test sequence and non-shuffled test sequences is hard to explain. Shuffling decorrelate the sequences, which may be favorable when multiple consecutive images are affected by occlusion. That might explain why the small recurrent network with last output pooling performs better on long shuffled sequences, compared to non-shuffled sequences. However, further tests are needed in order to draw any real conclusions why the networks perform so differently in those cases when the test sequences are shuffled. None of the cited articles have tested their developed networks on shuffled sequences, so no comparison can be made.

The network trained directly on the multi shot datasets is considerably less complex than the larger recurrent networks. This might explain why the larger recurrent networks with the conv-GRU and the conv-LSTM outperform the small network which was trained from scratch. However, the small network performs better than the other large recurrent networks with vector/-matrix based recurrent units.

The previously cited multi shot re-identification network [30], in the previous work & comparisons section, outperforms the multi shot re-identification networks developed in this study, with a rank-1 performance difference of 8 percentage points. The best multi shot re-identification network outperforms however the other cited multi shot re-identification network [43], with a rank-1 performance difference of 7 percentage points .

As a final remark, non-temporal features have certainly been more prominent and easier for the networks to use when persons are re-identified, compared to temporal dependent features such as the gait. Longer training sequence lengths and more training data may enable the recurrent units to learn and depend more on temporal dependent features.

5.6 Future Work

This section will introduce ideas for further improvements of multi shot re-identification networks. The ideas presented in this section were out of the scope for this thesis. Many ideas for future work presented in the single shot chapter are applicable in multi shot re-identification networks as well, such as hard mining, deep embedding for cluster analysis and embedding with a trainable metric. Hopefully, the ideas presented in this section can inspire further improvements of multi shot re-identification networks.

5.6.1 Data Augmentation

Interestingly, training on augmented images introduced a significant performance gain in multiple cases, compared to training on non-augmented images . Further tests with different data augmentation adjustments is a possible way to increase the performance of multi shot re-identification networks. Importantly, the training datasets should closely depict the scenarios in which the system is going to be evaluated/tested on. In those cases the evaluation datasets are different from the training datasets, different data augmentation adjustments can bring the training datasets to closer resemble the evaluation datasets. Cropping of images may introduce tolerance towards bad crops, resolution down sampling of images may introduce tolerance towards low quality images. Other possible image augmentation adjustments are, saturation adjustments, hue adjustments and intensity re-scaling.

5.6.2 Dataset Development

In order to increase the performance of multi shot re-identification systems with recurrent units more training data is surely needed. The AMSR-id dataset development procedure is in need of improvements. In order to improve the quality of the cropped images from YOLOv2, a tracker may be used. Where the bounding box coordinates, outputted from YOLOv2, are filtered, such that smoother person trajectories can be created.

5.6.3 Other Network Architectures

Because of the sheer amount of different recurrent units that were evaluated during this study, other network architectures besides the siamese classifier network architecture were not tested. Further tests with different networks is an interesting way forward. The triplet network together with the triplet softplus loss showed promise in the single shot case. Possibly would such a network yield comparable or even better performance compared to the networks used in this study.

Chapter 6

Final Conclusions

This master's thesis has examined a broad variety of network architectures for single shot re-identification. The single shot re-identification study showed, a classifier network which classifies person/persons as either matching or not matching performed the best. This knowledge was thereafter applied when different recurrent units were evaluated for multi shot re-identification.

The conv-GRU outperformed all other tested recurrent units. In almost all cases temporal mean pooling performed better than last output pooling. Long test sequence lengths introduce, in the temporal mean pooling case, significant performance gain, compared to shorter sequences. Layer normalization, inside multiple recurrent units, introduced significant performance gain in multiple cases. The baseline network yielded comparable results, in some cases, to the best performing recurrent network. The recurrent networks seem to make use of different temporal dependent features in certain cases. However, non-temporal dependent features have certainly been more prominent and easier for the network to use when persons are re-identified.

This study has shown, analysis of sequences of images can significantly increase the performance of a re-identification system compared to analysis of single images.

Cross dataset evaluation has shown, re-identification networks tend to over-fit to particular scenarios within the training datasets. Dataset bias is a severe problem to overcome. The development of larger training datasets with wider variations is a viable way forward to increase the generalization ability of single and multi shot re-identification networks.

Chapter 7

Appendix

Table 7.1: RE-ID-1 encoder network.

Type	Output Dim.	Filter Size	Stride	Act. func.	Normalization
Input	$160 \times 60 \times 3$	-	-	-	-
Convolutional	$80 \times 30 \times 32$	3×3	2	ReLu	Layer Normalization
Convolutional	$80 \times 30 \times 32$	1×1	1	ReLu	Layer Normalization
Convolutional	$80 \times 30 \times 32$	3×3	1	ReLu	Layer Normalization
Max Pooling	$27 \times 10 \times 32$	3×3	3	-	-
Convolutional	$27 \times 10 \times 64$	3×3	1	ReLu	Layer Normalization
Convolutional	$27 \times 10 \times 64$	1×1	1	ReLu	Layer Normalization
Convolutional	$27 \times 10 \times 64$	3×3	1	ReLu	Layer Normalization
Max Pooling	$9 \times 4 \times 64$	3×3	3	-	-
Convolutional	$9 \times 4 \times 128$	3×3	1	ReLu	Layer Normalization
Convolutional	$9 \times 4 \times 128$	1×1	1	ReLu	Layer Normalization
Convolutional	$9 \times 4 \times 128$	3×3	1	-	Layer Normalization

Table 7.2: The classifier network in RE-ID-1-Classifier.

Type	Output Dim.	Filter Size	Stride	Act. func.	Normalization
Convolutional	$9 \times 4 \times 128$	3×3	1	ReLu	Layer Normalization
Fully Connected	$1 \times 1 \times 1000$	-	-	ReLu	Layer Normalization
Dropout	-	-	-	-	-
Fully Connected	$1 \times 1 \times 2$	-	-	-	-

Table 7.3: RE-ID-2 encoder network.

Type	Output Dim.	Filter Size	Stride	Act. func.	Normalization
Input	$160 \times 60 \times 3$	-	-	-	-
Convolutional	$80 \times 30 \times 64$	3×3	2	ReLu	Layer Normalization
Convolutional	$80 \times 30 \times 64$	1×1	1	ReLu	Layer Normalization
Convolutional	$80 \times 30 \times 64$	3×3	1	ReLu	Layer Normalization
Max Pooling	$27 \times 10 \times 64$	3×3	3	-	-
Convolutional	$27 \times 10 \times 128$	3×3	1	ReLu	Layer Normalization
Convolutional	$27 \times 10 \times 128$	1×1	1	ReLu	Layer Normalization
Convolutional	$27 \times 10 \times 128$	3×3	1	ReLu	Layer Normalization
Max Pooling	$9 \times 4 \times 128$	3×3	3	-	-
Convolutional	$9 \times 4 \times 256$	3×3	1	ReLu	Layer Normalization
Convolutional	$9 \times 4 \times 256$	1×1	1	ReLu	Layer Normalization
Convolutional	$9 \times 4 \times 256$	3×3	1	-	Layer Normalization

Table 7.4: The classifier network in RE-ID-2-Classifier.

Type	Output Dim.	Filter Size	Stride	Act. func.	Normalization
Convolutional	$9 \times 4 \times 256$	3×3	1	ReLu	Layer Normalization
Fully Connected	$1 \times 1 \times 1000$	-	-	ReLu	Layer Normalization
Dropout	-	-	-	-	-
Fully Connected	$1 \times 1 \times 2$	-	-	-	-

Table 7.5: The classifier network in RE-ID-3-Classifier.

Type	Output Dim.	Filter Size	Stride	Act. func.	Normalization
Convolutional	$9 \times 4 \times 512$	3×3	1	ReLu	Layer Normalization
Fully Connected	$1 \times 1 \times 1000$	-	-	ReLu	Layer Normalization
Dropout	-	-	-	-	-
Fully Connected	$1 \times 1 \times 100$	-	-	ReLu	Layer Normalization
Fully Connected	$1 \times 1 \times 2$	-	-	-	-

Table 7.6: Network with the conv-GRU, trained directly on the multi shot datasets, t_{max} indicates the sequence length.

Type	Output Dim.	Filter Size	Stride	Act. func.	Normalization
Input	$t_{\max} \times 160 \times 60 \times 3$	-	-	-	-
Convolutional	$t_{\max} \times 160 \times 60 \times 32$	3×3	1	ELU	Layer Normalization
Convolutional	$t_{\max} \times 160 \times 60 \times 16$	1×1	1	ELU	Layer Normalization
Convolutional	$t_{\max} \times 160 \times 60 \times 32$	3×3	1	ELU	Layer Normalization
Max Pooling	$t_{\max} \times 54 \times 20 \times 32$	3×3	3	-	-
Convolutional	$t_{\max} \times 54 \times 20 \times 64$	3×3	1	ELU	Layer Normalization
Convolutional	$t_{\max} \times 54 \times 20 \times 32$	1×1	1	ELU	Layer Normalization
Convolutional	$t_{\max} \times 54 \times 20 \times 64$	3×3	1	ELU	Layer Normalization
Max Pooling	$t_{\max} \times 18 \times 7 \times 64$	3×3	3	-	-
conv-GRU	$t_{\max} \times 18 \times 7 \times 64$	3×3	1	tanh	Layer Normalization
Temporal Pooling	$1 \times 18 \times 7 \times 64$	-	-	-	-
Squared element-wise difference	$1 \times 18 \times 7 \times 64$	-	-	-	-
Convolutional	$1 \times 18 \times 7 \times 32$	3×3	1	ELU	Layer Normalization
Fully Connected	$1 \times 1 \times 256$	-	-	-	Layer Normalization
Dropout	$1 \times 1 \times 256$	-	-	-	-
Fully Connected	$1 \times 1 \times 128$	-	-	ELU	Layer Normalization
Fully Connected	$1 \times 1 \times 2$	-	-	-	-

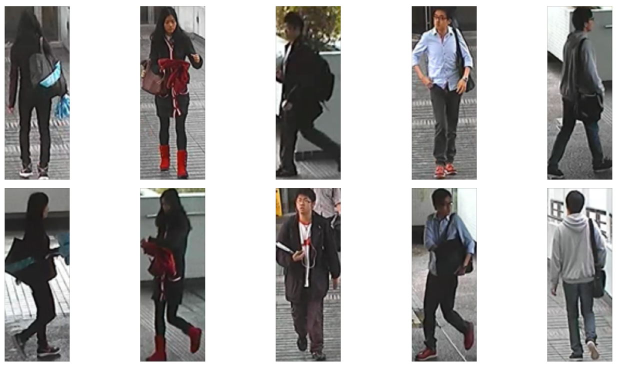
Figure 7.1: CUHK01

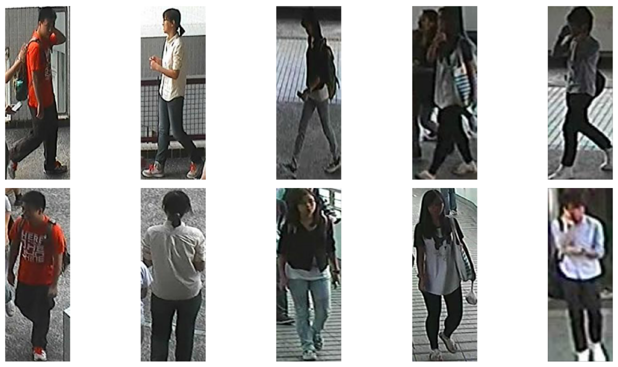
Figure 7.2: CUHK03

Examples of images from CUHK01 and CUHK03. Each column shows one person captured in two different disjoint camera views.

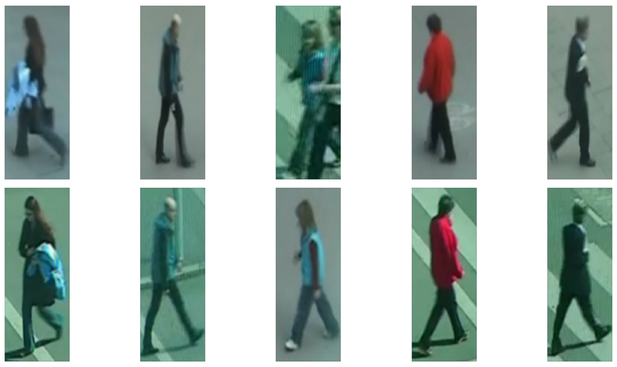

Figure 7.4: PRID-2011

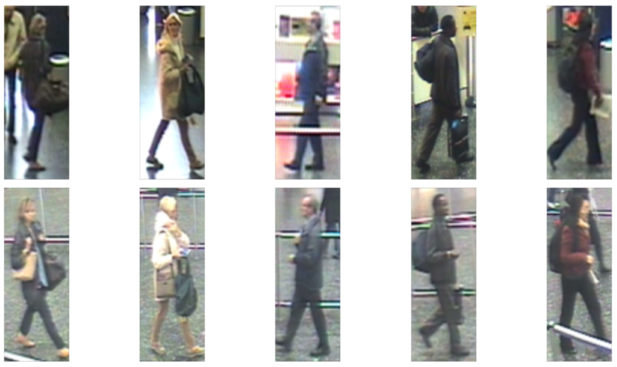

Figure 7.5: iLIDS-VID

Examples of images from PRID-2011 and iLIDS-VID. Each column show one person captured in two different disjoint camera views.

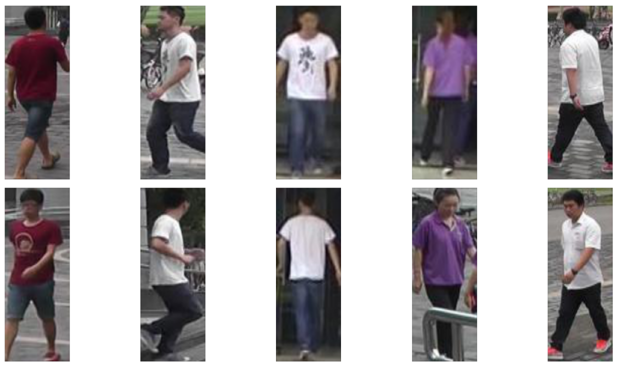

Figure 7.7: Examples of images from Market-1501. Each column shows one person captured in two different disjoint camera views.

Bibliography

[1] E. Ahmed, M. Jones, and T. K. Marks. An improved deep learning architecture for person re-identification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3908–3916, June 2015.

[2] T. N. Wiesel B. H. Hubel. Receptive fields of single neurons in the cat's striate cortexs. Vol 65(6), 1959.

[3] Nicolas Ballas, Li Yao, Chris Pal, and Aaron C. Courville. Delving deeper into convolutional networks for learning video representations. *CoRR*, abs/1511.06432, 2015.

[4] Sean Bell and Kavita Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Trans. on Graphics (SIGGRAPH)*, 34(4), 2015.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki. Scene labeling with lstm recurrent neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3547–3555, June 2015.

[7] W. Chen, X. Chen, J. Zhang, and K. Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. *ArXiv e-prints*, April 2017.

[8] De Cheng, Yihong Gong, Sanping Zhou, Jinjun Wang, and Nanning Zheng. Person re-identification by multi-channel parts-based cnn with improved triplet loss function. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[9] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *ArXiv e-prints*, June 2014.

[10] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1, June 2005.

[11] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[13] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[16] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks.* Springer, 2012.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[18] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *CoRR*, abs/1703.07737, 2017.

[19] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. *Journal of Machine Learning Research*, 2015.

[20] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

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

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

[23] J. Lei Ba, J. R. Kiros, and G. E. Hinton. Layer Normalization. *ArXiv e-prints*, July 2016.

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

[25] Wei Li, Rui Zhao, and Xiaogang Wang. Human reidentification with transferred metric learning. 2012.

[26] Wei Li, Rui Zhao, Tong Xiao, and Xiaogang Wang. Deepreid: Deep filter pairing neural network for person re-identification. 2014.

[27] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[28] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[29] P. M. Roth H. Bischof M. Hirzer, C. Beleznai. Person reidentification by descriptive and discriminative classification. 2011.

[30] N. McLaughlin, J. M. d. Rincon, and P. Miller. Recurrent convolutional network for video-based person re-identification. pages 1325–1334, June 2016.

[31] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org, 2013.

[32] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[33] F Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, Vol 65(6), 1958.

[34] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[35] Paolo Frasconi Sepp Hochreiter, Yoshua Bengio and Jrgen Schmidhuber. Gra- dient flow in recurrent nets: the difficulty of learning long-term dependencies. 1997.

[36] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

[37] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Fisher networks for large-scale image classification. In *Advances in Neural Information Processing Systems*, 2013.

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[40] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[41] X. Zhu S. Wang T. Wang, S. Gong. Person re-identification by video ranking. 2014.

[42] Walter Pitts Warren S. McCulloch. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 1943.

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

[44] Lin Wu, Chunhua Shen, and Anton van den Hengel. Personnet: Person re-identification with deep convolutional neural networks. *CoRR*, abs/1601.07255, 2016.

[45] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. *CoRR*, abs/1511.06335, 2015.

[46] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *ArXiv e-prints*, November 2014.

[47] Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. Scalable person re-identification: A benchmark. 2015.

[48] Liang Zheng, Yi Yang, and Alexander G. Hauptmann. Person re-identification: Past, present and future. *CoRR*, abs/1610.02984, 2016.

Master’s Theses in Mathematical Sciences 2017:E54
ISSN 1404-6342

LUTFMA-3328-2017

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/