Re-identification Using the Lower Body

Lasse Vork Borchert, Mathias Zacho Vestergaard

2016

Resumé (Danish Summery)

I denne rapport bliver forskellige metoder til re-identification kun ved hjælp af den nedre del af kroppen undersøgt. Der vil være fokus på metoder der bruger neurale netværk,hvor der vil blive designet netværk både til enkelt billeder og til sekvenser.

Alle netværk er trænet og testet på offentlige tilgængelige datasæt, for enkelt billede løsningerne bliver Market-1501 datasættet benyttet. Der er desuden lavet test af den bedst præsterende enkelt billede løsning, på både MARS og PRID2011 datasættene for at have en reference.

Det første forsøg på at implementere et eksisterende netværk i Keras, netværket er designet til enkelt billeder. Den nye implementering opnår ikke den rapporterede præcision. Det formodes at den lavere præcision er grundet manglende detaljer i artiklen som beskriver netværket. Der er derfor fundet en anden løsning som har åben kilde kode. Denne løsning er dog også genimplementeret i Keras for bedre at kunne udnytte de grafikkort der sider i de benyttede computere.

Denne implementerede løsning er den bedst præsterende implementerede løsning til enkelt billeder. Løsningen benytter ResNet-50 strukturen, og opnår en rank-1 nøjagtighed på 76.07 % hvor state of the art (som venter på publicering) i skrivende stund opnår en rank-1 nøjagtighed på 91.75 %.

Den bedst præsterende implementering til enkelt billeder er trænet og testet på MARS datasetet for at have en reference for de senere implementeringer.

Netværk designet til brug af sekvenser er trænet og testet på MARS, PRID2011 og ILIDS-VID. Hvor det første forsøg på at opnå en bedre præcision end referencen fra enkelt billede løsningen, er at udbygge denne løsning med en "RNN" blok efter ResNet-50 strukturen. Dette er dog ikke ligetil, da der er problemer med *"Timedistribution"* og *"batch normalization"*. Der er flere forsøg på at løse disse problemer, men der er ingen af dem der giver en god præcision. Det er derfor valgt at forsøge at udnytte den ekstra data fra sekvenser ved at tage gennemsnittet af resultatet fra de mange billeder fra samme sekvens. Dette er stadig gjort ved brug af ResNet-50 strukturen, dette resultere i den bedste implementerede løsning til sekvenser for MARS datasættet.

Den højeste rank-1 præcision opnået på MARS datasættet er 80.25 % hvor state of the art ligger på 81.21 %, og stammer fra samme artikel som levere state of the art på Market-1501 datasættet.

Denne løsning giver dog ikke gode resultater på hverken PRID2011 eller ILIDS-VID, så derfor er en anden løsning brugt til disse datasæt. Denne løsning benytter kun tre *convolutional layers* efterfulgt af en *recurrent unit*. Denne løsning benytter desuden *optical flow* som to ekstra kanaler til input billederne. Dette betyder at *temporal information* kan blive udnyttet, der er desuden et lag der tager gennemsnittet af de outputs der kommer fra hver billede i sekvensen. Dette er den bedste implementerede løsning til PRID2011 datasættet.

På PRID2011 opnår den implementerede løsning en præcision på 70.5 % rank-1, hvor state of the art ligger på 77.3 %.

De bedst præsterende netværk på de forskellige dataset trænet og testet med den nedre del af kroppen. De er derudover også tested og trænet med den øvre del af kroppen, for at kunne sammenligne hvilken del af kroppen der er mest beskrivende.

Testene viser at der ikke er nogen konsistent forskel på om den øvre eller nedre del af kroppen er mest beskrivende. Testene viser ydermere at der er et væsentlig lavere tab a præcision når sekvenser benyttes frem for enkelt billeder. Dette er tilfældet både ved brug af *temporal information* og ved brug af gennemsnit fra flere efterfølgende billeder.

AALBORG UNIVERSITY

Title:
Re-identification Using the Lower Body
Subject:
Computer Vision
Project group:
17gr1044
Participants:
Mathias Zacho Vestergaard
Lasse Vork Borchert

Supervisor:
Kamal Nasrollahi
Number of pages:
69

This report is focused on re-identification, where it is investigated how using lower body affects the performance.
Neural networks are used for the re-identification task, different structures are tested for both single images and sequences.

For single images the Market-1501 dataset is used, where the highest rank-1 accuracy reached is 76.07 % when using the ResNet-50 structure (SOTA is 91.75 %).
For sequences the networks are tested on both the MARS and PRID2011 dataset. Different networks are used to reach the highest accuracy on the respective dataset. The ResNet-50 structure is used to reached the highest accuracy on MARS at 80.25 % at rank-1 (SOTA is 81.21 %). For PRID2011 a siamese recurrent NN is used to reach 70.5 % rank-1 accuracy (SOTA is 77.3 %).

Results of using only the lower body compared to full body, shows that the drop in accuracy is a lot lower when using sequences compared to using single images. The test further shows that it is not only when using temporal information the accuracy drops less, it is also when using the average of multiple images.

Contents

1 Introduction

People re-identification is the task of matching people between cameras with non overlapping field of view. Re-identification is different from recognition, as recognition requires some sort of participation by the subject to be enrolled in the system. Whereas re-identification should be able to be applied without the subject actively participating. An example of recognition could be iris scanning as biometric access control. For the case of re-identification an example could be detecting people passing different cameras in an airport.

Re-identification can for example be used for tracking how people move through an airport or an amusement park. Where the camera views do not overlap. The information obtained can then be used for creating statistics describing how people move around. Another use could be searching multiple cameras for a person spotted in one camera view. This could for example help police locate a suspect using less man power.

For performing re-identification some basic steps have to be performed on a raw video fed. The first step is detecting where the person is located in the frame. Next, features have to be extracted from the image, and last the features have to be matched.

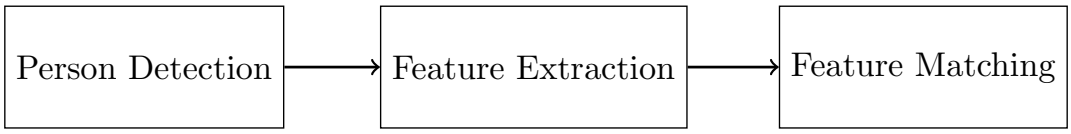


Figure 1.1: Re-identification steps.

When tracking people through multiple non overlapping camera views, it is possible that people will change cloth between the different cameras. The effect of people changing cloth between cameras, might be reduced by only focusing on the lower part of the body. As people often change only shirt, jacket or other clothing that is located on the upper part of the body. Whereas changing shoos or trousers while away from home is not done often. Disney have also considered this idea as they have described how such a system could be made [1].

Other methods for re-identification for queue measurement exists such as tracking people using Wi-Fi and Bluetooth [2]. Re-identification with multiple camera does however have the advantage that the users of the system does not depend on people having a mobile phone with them.

2 Problem Analysis

2.1 Problem Statement

The task of re-identification is hard to solve due to many challenges such as change in perspective and people being non rigid objects. Furthermore people are able to change cloth while walking around in an airport or other places where re-identification might be desired. To reduce the problem of people changing cloth between cameras, it might be possible to only focus on the lower body. As people rarely change their shoes or trousers, while walking in places such as airports.

This report will investigate the problem of re-identification, and evaluate the re-identification accuracy when only using lower body compared to using the entire body. There will furthermore bee tested how the accuracy will be effected by using the temporal information when performing re-identification on video sequences. This sums to a problem statement as follows:

How well does re-identification work when only the lower body is used compared to the entire body, and can using the temporal information increase the accuracy?

The success criteria for this project are:

- Try to implement a still image CNN that achieves close to state-of-the-art performance.
- Try to implement a video CNN that achieves close to state-of-the-art performance.
- Try to make a combination of the still image and video based methods.
- Test the body parts(full body, lower body and upper body) influence on the accuracy of the different re-identification solutions.

2.2 Related Works

In this section describes how others have tried to solve the problem of re-identification using deep learning. The methods are separated into three categories which are based on the method for training the networks. The category for the siamese structure is separated into two parts, one for single images and one for sequences, as it is the most popular methods for training.

2.2.1 Survey Article

The survey article [3] outlines the state of re-identification. It shows that the vast majority of articles deals with the problems after a bounding box of the person have been found. It furthermore outlines the two main different ways of doing re-identification. This is either done by using single images or videos for re-identification.

The most used method for evaluation is the Cumulative Matching Characteristics(CMC) curve. This does however not take into account if there are multiple correct candidates in the gallery. The mean average precision(mAP) can be used in this case.

The datasets for single image based re-identification can be seen in Table 2.1.

Dataset	Time	IDs	Images	Cameras	Label	Evaluation
VIPeR	2007	632	1,264	2	hand	CMC
iLIDS	2009	119	476	2	hand	CMC
GRID	2009	250	1,275	8	hand	CMC
CAVIAR	2011	72	610	2	hand	CMC
PRID2011	2011	200	1,134	2	hand	CMC
WARD	2012	70	4,786	3	hand	CMC
CUHK01	2012	971	3,884	2	hand	CMC
CUHK02	2013	1,816	7,264	10(5 pairs)	hand	CMC
CUHK03	2014	1,467	13,164	2	hand/DPM	CMC
RAiD	2014	43	1,264	4	hand	CMC
PRID 450S	2014	450	900	2	hand	CMC
Market-1501	2015	1,501	32,668	6	hand/DPM	CMC/mAP

Table 2.1: Single images re-identification datasets, table from [3].

The datasets have been made at various places like a underground station for GRID, a airport for iLIDS and university campuses for CUHK01, CUHK02, CUHK03 and Market-1501. The newer datasets improves by having more identities and cameras. In the newer datasets, bounding boxes are also created automatically by using a pedestrian detector, such as the Deformable Part Model(DPM). This is interesting to use since the misalignment may cause degradation of re-identification accuracy and is more similar to a real world scenario.

The article's evaluation of the performance of the datasets VIPeR, CUHK01, i-LIDS, PRID 450S, CUHK03 and Market-1501 shows that there is a increase in performance over time. It also show that the state of the art on all the datasets except for VIPeR is deep learning. The article therefore further splits the methods into two groups, called hand crafted systems and deep learning systems. The articles concerning deep learning systems will be described more detailed in the following. The hand crafted system mostly use color features and only few uses texture features.

It is also noted that despite a high rank-1 of 65.88% accuracy for the Market-1501 dataset the mAP is 39.55%.

The video based re-identification datasets can be seen in Table 2.2.

Dataset	Time	IDs	Tracks	Bbox	Cameras	Label	Evalution
ETHZ	2007	148	148	8,580	1	hand	CMC
3DPES	2011	200	1000	200k	8	hand	CMC
PRID-2011	2011	200	400	40k	2	hand	CMC
iLIDS-VID	2014	300	600	44k	2	hand	CMC
MARS	2016	1261	20,715	1M	6	DPM/GMMCP	mAP/CMC

Table 2.2: Video re-identification datasets, table from [3].

The accuracy on the ETHZ dataset have currently reached close to 100% rank-1 accuracy. This is however due to a single moving camera which causes less image variance than multiple cameras. Few persons is furthermore used which also makes it less challenging. The state of the art methods for all the video datasets use deep learning.

2.2.2 Triplet Methods

This article [4] is the currently best performing on both the Market-1501 and MARS datasets. Two different Convolutional Neural Networks (CNN) were tested, one which was pre trained and the other was not. The pre trained network delivered the highest accuracy. The networks were trained using a triplet structure, meaning the networks were shown one reference image, with a matching and non matching image as one example.

The article implemented two different network structures. The first described called TriNet were inspired by the ResNet-50 structure, where the last fully connected layer were replaced by two fully connected. The first consisting of 1024 units and the last having 128 units. The TriNet structure uses the pre trained weights from ResNet-50 for the identical layers. The second structure described is called LuNet which were inspired by the ResNet-v2 structure. LuNet is different from ResNet-v2 as it uses Leaky ReLU activation functions, and max-pooling layers of 3×3 with a stride of 2, instead of convolutions with stride.

For determining which triplet the network should see, a hard negative and hard positive mining were implemented. This was done by choosing a given number of random triplets, for a given number of persons. Then the triplets with the largest in class variation were chosen as hard positives, and the triplets with the lowest between class variation were chosen as hard negatives.

The networks were tested on both the Market-1501 and MARS datasets. Where TriNet delivered the highest accuracy on both dataset, at 91.75 % rank-1 for Market-1501 and 81.21 % rank-1 on MARS. The LuNet structure delivered 89.31 % rank-1 accuracy for Market-1501 and 78.48 % rank-1 on MARS.

Another article that uses a triplet structure is [5].

This article [5], focus on a triplet structure for training the network with a LSTM at the end.

The network consist of both a CNN and a LSTM to find descriptive features. The CNN structure was inspired by AlexNet, the output from the CNN was fed to the LSTM together with a location map that describes where in the image to look for discriminative features.

Training the network was done by optimizing as usually for triplets, where the distance between the matching pair plus a margin should be smaller then the distance between the non matching pair. The location map was updated for every time step and was based on what the last frame contained of information. Choosing how to create the triplets were done based on how well they hold the criteria for the triplet distances. Where triplets that holds the criteria worst were chosen as the examples for training the network in the next batch.

The LSTM cell should remember and forget information based on the location map and how it was trained, furthermore the output of the LSTM will effect the next time steps location map.

The system was tested on the CHUK01, CHUK03 and Market-1501 datasets. The highest rank-1 accuracies reached on these datasets were: 65.65 % on CHUK03, 81.04 % on CHUK01 and 48.24 % on the Market-1501 dataset.

Another method for training the neural network could be using a siamese structure, which focus on only two images at the time compared to three images for the triplet structure. This means that there have to be shown pairs of matching and non matching images. This approach is used more often then the triplet structure for the case of re-identification. The article [23] uses a NN in order to make a hashing function which can be used both for image retrieval and person re-identification. Binary hash codes are not differentiable and an approximation of the sign function based on tanh is therefore used. It can be seen in equation 2.1 where β is the smoothness of the function. β is increased from 2 to 1000 during the iteration of training.

$$o(v) = \frac{1 - e^{-\beta v}}{1 + e^{-\beta v}} \tag{2.1}$$

The entire NN can be seen in table 2.3. It first consists of some normal CNN layers with average pooling. It was followed by two fully connected layers. The last fully connected layer used equation 2.1 as a activation function. This was done to convert the values into binary being either -1 or 1. The last layer was an element wise multiplication in order to weight the elements. The ReLU activation function was used for all layers except of the last fully connected layer.

Type	Output Dim	FS	Stride
Convolutional	$* \times * \times 32$	3×3	2
Average Pool	$* \times * \times 32$	2×2	1
Convolutional	$* \times * \times 64$	3×3	2
Average Pool	$* \times * \times 64$	2×2	1
Convolutional	$* \times * \times 128$	3×3	2
Average Pool	$* \times * \times 128$	2×2	1
Fully Connected	512	-	-
Fully Connected	Hash length	-	-
Element wise	Hash length	-	-

Table 2.3: CNN Layers.

The network was trained with a triplet function.

The system was tested on the CUHK03 dataset. Where a hash length of 64 gives a rank-1 score of 21.96 % and rank-5 score of 46.66 %. A hash length of 128 gave a rank-1 score of 18.74 % and rank-5 score of 48.39 %.

2.2.3 Siamese Method

This article [6] have the best performing network trained using a siamese structure on many of the larger re-identification dataset, such as CHUK01 and Market-1501. The network structure was inspired by GoogLeNet, which was used to produce the feature vectors for comparing and classification.

The structure of the network was separated into four parts, where the base network used the GoogLeNet structure for extracting features from two input images, the feature vectors were then processed by a loss specific dropout unit. The feature vectors with reduced information was then used in either the classification layer or in the verification layer.

The classification layer tries to classify which ID the two input images belong two, and the verification layer was used to evaluate if the two input images were of the same person or not. The loss specific dropout unit changes the dropout rate depending on the classification loss and the verification loss.

The verification was built by subtracting the two feature vectors element wise, the difference vector was then passed through the ReLU activation function and then through a fully connected layer. The output from the fully connected layer was used in the classification layer with a softmax activation function.

The loss specific dropout unit depends on what the feature vector should be used for, if it was for classification a usual dropout layer is used. Where each of the two feature vectors had elements dropped with a certain probability, this mean that both vectors might not have the same elements dropped. Whereas when the feature vectors should be used for verification they have two identical dropout, meaning that the same elements should be dropped. Otherwise the dropout could influence the result of the matching.

When the network had to be trained for re-identification, the pre-trained weights from the ImageNet challenge were used as the network was to large to train from scratch. For making sure that the randomly initialized classification layer did not change the network randomly, all other weights in the network was

locked for some time. When the trained classification layer delivered a good performance, the rest of the network could be fine-tuned to the new dataset.

Test of the system showed that the network performs well on the large datasets for re-identification but not as well on smaller datasets. The highest rank-1 accuracy on Market-1501 was 83.7 % for single query and 89.6 % for multi query, on CUHK03 it was 85.6 % for manually annotated and 84.1 % for automatic annotated images. Where this networks outperforms all other networks and methods on the CUHK03 dataset by a large margin.

The article [7] also uses a siamese structure with a new type of layer called a matching gate layer. It reaches a rank-1 accuracy of 62.3 % using only the siamese nn on the market-1501 and 65.9 % if the matching gate layers are used.

The siamese network can be seen in 2.4. It consists of small filters but with many layers instead of bigger filters with fewer layers due to the conclusion from [8]. A fully connected layer is added after the last convolutional layer. The activation function for the layers is the Parametric rectified linear unit (PReLU). Batch normalization is also used to speed up the training.

Type	Output Dim	FS	Stride
Input	$128 \times 64 \times 3$	-	-
Convolutional	$* \times * \times 32$	5×5	2
Max Pool	$* \times * \times 32$	2×2	2
Convolutional	$* \times * \times 50$	3×3	1
Max Pool	$* \times * \times 50$	2×2	2
Convolutional	$* \times * \times 32$	3×3	1
Max Pool	$* \times * \times 32$	2×2	2
Convolutional	$* \times * \times 32$	1×4	1
Convolutional	$* \times * \times 32$	1×3	1
Convolutional	$* \times * \times 32$	1×3	1
Convolutional	$* \times * \times 150$	16×1	1

Table 2.4: Base CNN for siamese structure.

A contrastive loss function was used which can be seen in (2.2) where Y describes if the two feature vectors X_1 and X_2 should be similar or not. Y should be 0 if the vectors X_1 and X_2 are similar and 1 if they are different. m is a margin which is set to 1 in this case.

$$L(Y, X_1, X_2) = (1 - Y)\frac{1}{2}(\|X_1 - X_2\|)^2 + Y\frac{1}{2}\max(0, m - \|X_1, X_2\|)^2 \tag{2.2}$$

The article contributes with a layer called matching gates that compares the responses from both convolutional layers in the siamese network and creates a mask for each response. This layer was inserted between the 4-5, 5-6 and 6-7 convolutional layers.

The matching gates uses a row wise convolution with each response from the two previous layers. The weight for this convolution is shared. A Gaussian activation function is used to get a response between 0 and 1. The common pattern is then found from the gated values, next the input is added to the gated values. The result of this is normalized using the L2 Norm.

There is a bias in the data due to more dissimilar people than similar. Augmentation in the form of flipping and random translation is used combined with using 5 times more samples of similar people than dissimilar people.

Training is done with a batch size of 100 for 20 epochs. The NN is trained from scratch with the weights initialized based on [9]. RMSProp is used for learning with a learning rate of 0.002 and is reduced by a scaler of 0.1 for each epoch. The RMSProp decay parameter is set to 0.95. The mean image made from the training data is subtracted from training and test images.

Testing a image is done by pairing it with each of the gallery images and calculating the euclidean distance of the pairs. The distances are sorted in ascending order to get the most similar people.

The article [10] use a deep CNN for a siamese structure, they further contribute with a new type of layer called *difference layer*. The network structure can be seen in table 2.5. Two input images are used which are passed through two identical networks before they are passed to the difference layer. The difference layer then combines the two inputs and give one output.

The activation function used was tanh except for the last layer which uses a softmax function in order to do classification. The tanh function was scaled between -1 and 1 ($tanh(\frac{3x}{2})$) in order to have the training spread uniformly over all the layers. The last layer outputs whether or not the two images are of the same person. The reason for using two 3×3 filter layers instead of one 5×5 filter in the first layer was to add one more non-linear activation function without having more parameters which can make it more discriminative.

Type	Output Dim	FS	Stride
Input	$160 \times 60 \times 3$	-	-
Convolutional	$157 \times 57 \times 32$	3×3	1
Pool	$79 \times 29 \times 32$	2×2	2
Convolutional	$76 \times 26 \times 32$	3×3	1
Pool	$38 \times 13 \times 32$	2×2	2
Convolutional	$35 \times 10 \times 32$	3×3	1
Convolutional	$32 \times 7 \times 32$	3×3	1
Difference	$32 \times 7 \times 32$	3×3	3
Convolutional	$29 \times 4 \times 32$	3×3	1
Pool	$15 \times 2 \times 32$	2×2	2
Fully Connected	4096	-	-
Dropout(0.5%)	4096	-	-
Fully Connected	4096	-	-
Dropout(0.5%)	4096	-	-
Fully Connected	512	-	-
Fully Connected	2	-	-

Table 2.5: Layers in PersonNet.

The difference layer gives the difference in the filter responses for the two images. This is calculated based on Equation 2.3.

$$K_i(x, y) = f_I(x, y)\mathbb{I}(3, 3) - N[h_J(x, y)] \tag{2.3}$$

Where $f_I(x, y)$ is the pixel value located at (x,y), $I(3, 3)$ is a 3×3 matrix with 1 in all places and $N[h_J(x, y)]$ is a 3×3 neighbourhood located around (x,y). RMSProp is used as the optimizer for back propagation with a mini-batch size of 2 and is regularised using a L2 norm. The initial learning rate is 0.05 and decreased with a factor of 10 when the validation accuracy converged.

Augmentation was used in order to alleviate the problem of data imbalance between positive and

negative pairs. Five images were made by translating them by $[-0.05H, 0.05H] \times [-0.05W, 0.05W]$ in a uniform distribution where H and W are the image height and width. Flipping them horizontally was also used for the smaller dataset such as CHUK01.

The rank-1 accuracy was 71.14% for CUHK01 and 64.8% for CUHK03. The market-1501 rank-1 score for single shot and single query was 37.21% with a mAP score of 18.57%.

Another article that also use the Siamese structure for still images as well is [11]. It focus on training the network to produce a feature vector for each image. These feature vectors can then be used to compare the input images using Euclidean distance. The Euclidean distance is used for computing the loss function, which was a combination of two loss functions: One for the case of matching images and one for the non matching images. Where the loss for the matching images becomes larger together with the distance between the image pairs. The loss function of the non matching image pairs decreases as the distance between the feature vectors increase, but is set to be 0 when the distance is higher then a given threshold. These two functions are combined into one, using the information about if the image pairs are matching or not, the resulting formula can be seen in (2.2). Where Y is either 1 or 0 as follows:

$$Y = \begin{cases} 0 & \text{if the images are matching.} \\ 1 & \text{if the images are not matching.} \end{cases} \tag{2.4}$$

The total loss of one batch is found by summing the loss for each image pair that is used. This summed loss is then used to update the parameters in the network by using gradient descent.

To prevent overfitting and make the system generalize better, multi-task learning is used. This means that the system will not only be trained to match two images, but will also learn to find attributes from the two images. These attributes could fore example be the pose of a person or the color of the cloth. The idea of training the system to be able to classify these attributes, should help the system generalize better and thereby increase the accuracy.

Multi-task learning is implemented by adding an extra loss function for every task the system have to do, and the system thereby have to optimize for all the different tasks.

The network structure consist of three convolutional layers and the structure can be seen in Table 2.6.

Input	Conv	Max Pool	Conv	Max Pool	Conv
$64 \times 64 \times 3$	$5 \times 5 \times 3 \times 16$	2×2	$5 \times 5 \times 16 \times 64$	2×2	$5 \times 5 \times 64 \times 64$

Table 2.6: The CNN structure used for extracting the feature vectors.

For testing the system both the VIPeR and PETA dataset are used, each dataset is separated into two parts, one for training and one for testing, they are shared approximately equal for test and training.

The test is performed using the network without any multi-task learning as a benchmark, and with different kinds of multi-task learning to determined which to use. The test shows that the extra task of identification for each input image, increased the accuracy of how well the system re-identified people. It increased the accuracy at rank-1 from around 10 % to 30 %, on the VIPeR dataset. Multi-task learning also increased the accuracy on the PETA dataset.

The test furthermore shows that the extra task that increased the accuracy most was trying to identify the different training subjects. Another part of the test focused on determining how many layers the network should consist of, where it was found that the accuracy increased until the network reached 3 layers and did not improve with 4 layers.

Another article that describes how to use a neural network for performing re-identification, using a siamese structure is [12].

The network used, consist of two convolutional layers which are both followed by a max pooling layer, The last layer is a fully connected layer. The output vector from the fully connected layer consist of 500 elements. This vector is used to compare how similar two input images are, where the result is either a match or not. The network structure can be seen in Table 2.7.

Input	Conv	Max pooling	Conv	Max pooling	Fully connected
$128 \times 48 \times 3$	$7 \times 7 \times 64$	Stride 2	$5 \times 5 \times 64$	Stride 2	500

Table 2.7: The CNN structure used for extracting feature vectors for comparison.

The ReLU activation function was used after the two convolutional layers.

The input images are split into three overlapping parts, which all have a size of 48×48 pixels. The first convolutional layer have shared weights for all the three parts, whereas the last layer have separate weights for the three different parts. The results of the three processed parts are combined in the fully connected layer using the sum rule.

When the feature vectors are extracted for both input images the distance between them are found using the Cosine similarity, as it is in the range -1 to 1, and does not depend on the magnitude of the samples. Furthermore the loss function used in this paper is chosen to be Binomial deviance as it is thought that this will result in a better trained network.

The network is tested on both the VIPeR and PRID dataset, where it is tested for two different scenarios: One where it is trained on the same dataset (intra dataset) and the other case is when it is trained on another dataset (Cross dataset). Where the highest rank-1 accuracy for the VIPeR dataset are 34.49 % for the case of intra dataset, and 17.72 % for the case of cross dataset. The highest rank-1 accuracies on the PRID dataset is 17.9 % for intra dataset and 13.8 % for cross dataset.

This article uses a siamese structure to match image pairs [13], they focus on determining whether two input images are a match or not. This is done using the siamese structure where the first individual part is shown in Table 2.8. Where the two first convolutional layers are implemented as tied-convolution as they have to process the two input images the same way.

Input	Conv	Max Pool	Conv	Max Pool
$160 \times 60 \times 3$	$5 \times 5 \times 3 \times 20$	2×2	$3 \times 3 \times 20 \times 25$	2×2

Table 2.8: First part of the siamese structure: Separate processing of the input images.

After the images have been convolved with the same kernels, the differences between the images are found using cross-input neighbourhood differences. Where the neighbourhoods are 5×5 pixels. This is then propagated through a combined convolutional network, which is done for comparing the images.

As the datasets can be small, data augmentation is used to prevent the system from overfitting. This is done by translating the images in both the horizontal and vertical axis. There was further one challenge, that there were many more negative then positive pairs. This could result in the system only learning to classify as non matching as these were a lot more common then matches. Therefore some of the negative matches were randomly thrown away, such there were only twice as many negative then positive matches.

The settings for the system while training were as follows: Base learning rate 0.01, momentum 0.9 and a decay at 5×10^{-4}. Where the learning rate is updated for each mini-batch using formula (2.5).

$$\eta^{(1)} = \eta^{(0)}(1 + \gamma * i)^{-p} \tag{2.5}$$

Where $\gamma = 10^{-4}$, i is the mini-batch number, η is the learning rate and $p = 0.75$.

For determining when the network converged, a validation set was used, which was also used to find the best performing model. It took the system about 12-14 hours to converge using a NVIDIA GTX780 graphics card. The system was implemented in Caffe.

When testing the system on the CHUK01 dataset, the data was separated into training and test data. Where the normal distribution was around half for training and testing, in this case the network was first trained on the CHUK03 dataset and then fine-tuned on the training part of the CHUK01 dataset. This resulted in a test accuracy of 47.5 % at rank-1. Another way the dataset was used in this article was by separating the dataset into 100 persons for test and the remaining for training, meaning there was around 90 % of the dataset for training. This gave a test accuracy of 65 % at rank-1.

They had further trained the system to only use one body part for classifying the subjects, where the image was separated into 5 parts. This test was performed on the CHUK01 dataset, it indicated that the upper most part of a person is the most discriminative part of the persons. As the rank-1 accuracy became lower while moving down the different body parts.

In this article [14] the overall idea of how to build a system for re-identification is described. Where the focus of the article is on the last three steps of the four step process, using a siamese neural network.

The article describes the four steps of re-identification as: Pedestrian detection, feature extraction, transformation and feature matching. Where the transformation is how the images are transformed between cameras, such as lighting changes and perspective change.

To solve the last three parts of the problem, a neural network is chosen with a siamese structure. The network consist of six layers which should be able to handle most of the transformations between two different camera views. The first layer is a convolutional and max pooling layer, which extract local features and reduces local misalignment by pooling them. The second layer is patch matching, which matches the patches between images only in the horizontal direction. This is done for matching the local features. The third layer splits the matching results into a fixed number of groups, and for each group only the maximum response is propagated. The fourth layer is a convolutional and max pooling layer, which is used to find the displacement of body parts between the images. The fifth layer is a fully connected layer which is used to combine all the different displacements from the previous layer, this is done to model how the transformation is between the two camera views. The sixth and last layer is a softmax classification layer used to match the two input images, and have two different classes 1 and 0, 1 for matching images and 0 for non matching images.

When training the network different techniques are used to improve the accuracy of the system. There are introduced a dropout layer and data augmentation to make the network generalize better, data balancing is further used to make the system more general. The last technique used is bootstrapping, where the network is presented with the examples that it have the hardest time classifying, when the network have converged using only the other techniques.

The network is tested on both the CUHK01 and CUHK03 dataset, where the highest rank-1 accuracy reached on the CUHK01 dataset is 27.87 %. When testing on the CUHK03 dataset there are two different scenarios, one with automatic detected bounding boxes and one with manually labelled bounding boxes. Where the highest rank-1 accuracies are: 19.89 % for automatic and 20.65 % for manual bounding boxes.

In this article [15] the focus is on creating a siamese NN where a linear Support Vector Machine (SVM) is used to replace the softmax activation layer for classifying. The input images are resized to 32×32 pixel images. Before every convolutional layer in the network, zero padding is used to keep the output the same size as the input.

The structure of the network is split into two parts, one that is performed individual for both input images and one that uses a combination of the two processed images. The first part of the structure can be seen in Table 2.9.

Input	Conv	Sub sampling
$32 \times 32 \times 3$	$5 \times 5 \times 3 \times 32$	3×3, stride 2

Table 2.9: The CNN structure used to extract the feature vectors.

When the input images have been through this part of the structure, they are combined and processed in the structure seen in Table 2.10.

Conv	Sub sampling	Conv	Sub sampling	Fully connected
$16 \times 16 \times 2 \times 32$	3×3, stride 2	$8 \times 8 \times 32 \times 128$	3×3	2048

Table 2.10: The CNN structure used to extract the feature vectors.

The output from the fully connected layer was then used to classify if the two input images were a match using a linear SVM. The loss function for training the network was based on L2-SVM, which was chosen as it punishes wrong guesses harder and is differentiable.

As many of the dataset designed for re-identification are small, the networks are prone to overfit. To prevent the network from overfitting two methods were used in this article. The first method was to initialize the weights of the network using unsupervised learning, and the second was using two dropout layers. The layers were placed right before the last convolutional layer and the fully connected layer. The dropout rate was set to 0.5, which means that there was 50 % probability for a neuron to be dropped.

Testing the network was done on the VIPeR and CAVIAR4REID datasets. Where each dataset was separated into test and training set, with half of the persons for test and the other half for training. For training the network image pairs were produced. This was done by taking all images from one camera view where a matching and non matching pair was produced for these. The matching pair was the matching image from the other camera view, and the non matching examples was created by choosing a random image from the other camera view that does not match.

The rank-1 accuracy on the VIPeR dataset ended at 12.5 %, and 7.2 % on the CAVIAR4REID dataset. The article [19] also uses a siamese structure to perform re-identification, but with a LSTM to determine the important features extracted from the different parts of the image.

The idea was that each image was split into a fixed amount of rows. For each of these rows, features were then extracted and fed to the neural network. The features used were Local Maximal Occurrence (LOMO) features and Color Names (CN)[20]. These features were concatenated to form one large feature vector for each part of the image.

The reason for splitting the image into rows was that the LSTM should then be able to learn which features to keep and which to discard. Furthermore it should be able to combine features across the different rows do to the memory cell in the LSTM. This was thought to result in the LSTM finding some good key features, as an LSTM have been used to find keywords in speech.

The two feature vectors produced were then matched to determine if they were matching or not. A match should give 0 and non matching should result in 1. The loss used for training the siamese structure was the contrastive loss function. The training was done using stochastic gradient descent with a mini-batch of 100 image pairs per batch. The network was trained for a maximum of 20 epochs and furthermore uses an early stop if the validation performance was saturating.

Testing the network was done on the three dataset: Market-1501, CHUK03 and VIPeR. The highest reached rank-1 accuracy of the network on these dataset were: 61.6 % on Market-1501, 57.3 % on CHUK03

and 42.4 % on the VIPeR dataset.

2.2.4 Siamese Method on Video

The article [16] uses video to do re-identification. The structure was a siamese NN which consist of some CNN layers followed by some special CNN layers called ConvGRU.

ConvGRU is a special version of Gated Recurrent Unit(GRU). The equations for a standard GRU can be seen in (2.6). The value z_t is a update gate and r_t is the reset gate. $\odot$ is a element wise multiplication.

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1}), \\ r_t &= \sigma(W_r x_t + U_r h_{t-1}), \\ \hat{h}_t &= \tanh(W x_t + U(r_t \odot h_{t-1})), \\ h_t &= (1 - z_t) h_{t-1} + z_t \hat{h}_t, \end{aligned} \tag{2.6}$$

The article modified the GRU to be able to use the structure of the convolutional maps. The equations for ConvGRU can be seen in (2.7) where $*$ is the a convolution operation. Zero padding is used to avoid the output from the convolution to get smaller over time. The ConvGRU furthermore uses the output of the previous layer hidden unit seen in $W^l_{z^l} * h^{l-1}_t$ and $W^l_{r^l} * h^{l-1}_t$. This makes it possible to use different spatial resolutions.

$$\begin{aligned} z^l_t &= \sigma(W^l_z * x^l_t + W^l_{z^l} * h^{l-1}_t + U^l_z * h^l_{t-1}), \\ r^l_t &= \sigma(W^l_r * x^l_t + W^l_{r^l} * h^{l-1}_t + U^l_r * h^l_{t-1}), \\ \hat{h}^l_t &= \tanh(W^l * x_t + U^l * (r^l_t \odot h^l_{t-1})), \\ h^l_t &= (1 - z^l_t) h^l_{t-1} + z^l_t \hat{h}^l_t, \end{aligned} \tag{2.7}$$

The entire NN can be seen in table 2.11.

Type	Output Dim	FS	Stride
Input	$160 \times 60 \times 3$	-	-
Convolutional	$157 \times 57 \times 32$	3×3	1
Pool	$79 \times 29 \times 32$	2×2	2
Convolutional	$76 \times 26 \times 32$	3×3	1
Pool	$38 \times 13 \times 32$	2×2	2
Convolutional	$35 \times 10 \times 32$	3×3	1
Pool	$18 \times 5 \times 32$	2×2	2
Convolutional	$15 \times 2 \times 32$	3×3	1
ConvGRU	$- \times - \times 128$	-	-
ConvGRU	$- \times - \times 256$	-	-
ConvGRU	$- \times - \times 256$	-	-
Fully Connected	4096	-	-
Fully Connected	4096	-	-
Fully Connected	512	-	-

Table 2.11: CNN Layers.

The similarity was calculated based on equation (2.8) where $\acute{h}^a$ and $\acute{h}^b$ are the two outputs from the siamese NN and $*$ is the element-wise inner product.

$$s(X^a, X^b) = \frac{1}{1 + e^{-v^T[diag(\acute{h}^a * \acute{h}^b)]+c}} \tag{2.8}$$

The loss function was calculated using equation (2.9). Where ϑ is the parameters of the network and Z is the list of similar(S) and dissimilar(D) images.

$$\mathcal{L}(\Theta; Z) = -\left[\sum_{(a,b)\epsilon S} \log s(X^a, X^b) + \sum_{(a,b)\epsilon D} \log(1 - s(X^a, X^b)) \right] \tag{2.9}$$

RMSProp was used for gradient descent and the weights were initialized with a uniform distribution between -1 and 1. A batch size of 10 was used and the gradients were clipped to be between -5 and 5. The sequence length used was 20 which were a randomly selected subset of all frames in each video sequence. They were randomly selected at training time. The NN combined with KISSME get 46.1% for iLIDS-VID and 69% PRID2011.

The article [17] also uses a siamese structure for re-identification on video. Where a recurrent layer was placed after the convolutional layers. The structure of the CNN and recurrent layer can be seen in table 2.12. The activation function used for all the layers was tanh. The input was a RGB image combined with the horizontal and vertical optical flow as two additional channels. The optical flow channels were normalized to be between 1 and -1 and the image was converted to the YUV colorspace and the mean of the image itself was subtracted.

Images from the sequences were passed through the CNN layer and connected through the recurrent layer. Temporal pooling is used on the outputs of all the recurrent layers. An experiment with max and average pooling, resulted in the average pooling delivering the highest accuracy.

Type	Output Dim	FS	Stride
Input	$* \times * \times 5$	-	-
Convolutional	$* \times * \times 16$	5×5	1
Max Pool	$* \times * \times 16$	2×2	2
Convolutional	$* \times * \times 32$	5×5	1
Max Pool	$* \times * \times 32$	2×2	2
Convolutional	$* \times * \times 32$	5×5	1
Dropout	$*$	-	-
Fully Connected	128	-	-
Recurrent	$*$	-	-
Temporal Pooling	$*$	-	-
Fully Connected	Number of persons	-	-

Table 2.12: CNN and Recurrent layers.

The siamese NN have two instances of this structure which shares the weights. The loss function was a combination of the two classification loss from the last layer and the contrastive loss between the two vectors from the second last layer. The margin for the contrastive loss was set to 2.

Cropping and flipping was used for data augmentation. Training was done by using a batch size of one and a learning rate of 1e-3. Each batch alternated between showing a similar and dissimilar example.

Random dissimilar examples were used to shown the same amount of similar and dissimilar examples. A sequence length of 16 was used for training.

The datasets were split into two with one half being used for training and the other half being used for testing. The rank-1 accuracy is 70% for the PRID-2011 dataset and 58% for the iLIDS-VID dataset.

2.2.5 Classification

The article [18] uses a CNN as the previous articles, but does not use the siamese structure. This article does instead train the network using normal classification. The network was a pre trained version of AlexNet. The network was trained to perform classification of 1000 different classes.

The pre trained network was then fine tuned on a re-identification dataset called PETA. Multi-task learning was used when the network was fine tuned, as this made the network generalize better. The PETA dataset consists of 10 different datasets. One dataset was used for testing and was therefore removed during training. The network was implemented in Caffe.

The system was tested on four different dataset: VIPeR, CHUK01, PRID450S and GRID where the best rank-1 accuracies reached in this paper were: 52.1 %, 62.3 %, 71.5 % and 29.1 %, respectively. These results were reached by combining the feature vectors extracted using the CNN with some hand crafted features.

It was further tested if the accuracy increased when using multi-task learning, which was found to be the case. Tests showed an increased rank-1 accuracy between 1 and 4 %. Another test was made extracting the feature vectors from the different layers in the NN. The test showed that the last fully connected layer gives the feature vector that produces the highest test accuracy. Another way of using neural networks could be as in article [21]. Where the focus was on training a LSTM layer which was fed with hand crafted features from each image in a sequence.

The network was fed with both color and texture features. This gave a vector with 58950 elements for each frame, the most important features were then stored in the LSTM memory cell, with a size of 512. This meant there were stored a vector with 512 elements in the LSTM, this vector was then updated for every new frame in the sequence. The network was trained as a classification problem, and therefore have a classification layer after the LSTM with as many elements as subjects in the training set.

There was a fixed sequence length for the network which was 10 frames for one sub-sequence, and for every frame a feature vector was extracted and concatenated. This gives a feature vector of 5120 elements, for every sub-sequence. For every sequence there were extracted multiple sub-sequences, the resulting feature vectors were then averaged to make a more general descriptor for each sequence.

For matching the feature vectors a combination of RankSVM and the cosine distance metric was used, where the RankSVM was inspired by [22].

The network was tested on the two datasets: PRID2011 and iLIDS-VID, where it was found that the highest accuracy is reached when fusing features from 10 frames. It was furthermore found that the number of sub-sequences resulting in the highest accuracy was 10 sub-sequences. The highest overall rank-1 accuracy reached on the iLIDS-VID dataset was 49.3 % and 58.2 % on the PRID2011 dataset, these accuracies were reached when using the ranked SVM classifier.

The article [24] was focused on using a CNN for extracting gait features from optical flow images.

This was done by first computing the optical flow for one sequence, next these were grouped together to form one data structure containing all optical flow images from that sequence. Then the network was trained using the collected data structure and the ID of the person it belongs to, as it was trained as a classification problem.

The extraction of optical flow images was done by resizing the images to 80×60 pixels, and then extracting the optical flow between images using OpenCV. At the same time, people were located on the images by using background subtraction. This was used to crop the images to 60×60 pixels, where the image keeps the original width. The cropping was performed to remove some of the background.

When the optical flow images were extracted, they were put together in sequences of 25 frames with a 80 % overlap. The length of 25 frames was chosen as it should cover a whole walking cycle.

Furthermore data augmentation was used to produce more examples for the network to see. This was done by translating the images by 5 pixels in each direction, resulting in 8 different translation of the images. Then the images were also flipped around the vertical axis to produce even more examples.

The neural network consisted of four convolutional layers, two fully connected layers and a softmax activation layer, which is used for classification. The structure of the network can be seen in Table 2.13.

Input	Conv	Conv	Conv	Conv	Fully connected	Fully connected
$60 \times 60 \times 50$	$7 \times 7 \times 96$	$5 \times 5 \times 192$, stride 2	$3 \times 3 \times 512$	$2 \times 2 \times 4096$	4096	2048

Table 2.13: The CNN structure used to extract gait features.

Each convolutional layer in the structure was followed by a max pooling layer of 2×2 pixels, for reducing the sensitivity to translations.

When training the network a dropout layer was placed after each of the fully connected layers to reduce overfitting. The dropout rate was set at 0.4.

Testing of the network was done in a way that can be directly compared with re-identification, as they make a gallery for matching against in the case where SVM or nearest neighbour was used. The dataset used have different scenarios, where the results for the different scenarios were ranging from 56.3 % to 99.7 % accuracy at rank-1. These results were promising compared to other solution for gait re-identification such as [25], though they use another dataset which might be more difficult.

3 Background Theory

This chapter will contain some of the required background knowledge for designing the neural networks to perform re-identification.

Fully Connected Layer

Fully connected layers take inputs from all the previous notes and combine them for each note in the current layer. For every neurone in the current layer the outputs from the previous layer is multiplied by a weight, then all the weighted outputs are summed and passed through an activation function [26]. An illustration of this can be seen in Figure 3.1. This can for example be useful to produce a feature vector that finds good features depending on more then one pixel.

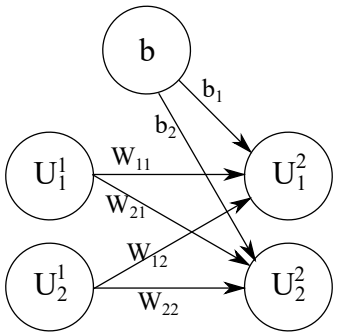


Figure 3.1: Illustration of a fully connected layer.

The training of the fully connected layers consist of updating the weights. This is done using backpropagation.

Convolutional Layer

Convolutional layers are another form of layers for extracting features from the input, they do so by convolving the input with a kernel. Therefore they are very popular for image processing, as convolution is often used when trying to find edges, this could for example be done by convolving the image with a Sobel kernel. The kernels in the convolutional layers can have different sizes and there can be different amount of kernels in the layers, the weights in the kernels are then updated using backpropagation as for the fully connected layers. [27]

There is further one parameter for convolutional layers called stride, it tells how many pixels the kernel should be moved between each convolution. This can be used to down sample the amount of data fed to the next layer. An example of a kernel of size 3×3 applied to an image with a stride of 1 can be seen in Figure 3.2. Where the red square marks the first place the kernel is applied to, and the blue square marks the second place the kernel is applied to.

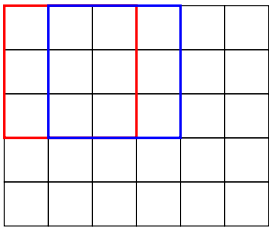


Figure 3.2: Illustration of a 3×3 kernel applied to the input image

Using only convolutional layers could result in the system being very sensitive to placement changes, to reduce this sensitivity a pooling layer such as max pooling could be used, which is also a kernel based operation. However it chooses the largest value within the kernel, this help emphasize the maximum response (for example steepest edges).

Recurrent Unit

Recurrent Neural Network(RNN) are used to give the neural network memory of what it have seen in the past, such that the network is able to understand the connection between different time steps. This is done by transferring the information from the previous step to the current step and so forth. An example of a recurrent unit can be seen in Figure 3.3. Where the previous output and the current input is concatenated and fed to a fully connected layer followed by a tanh activation function. Using a recurrent units mostly add short term memory as the information from the previous steps will vanish slowly, this is caused by the simple structure of the recurrent unit.

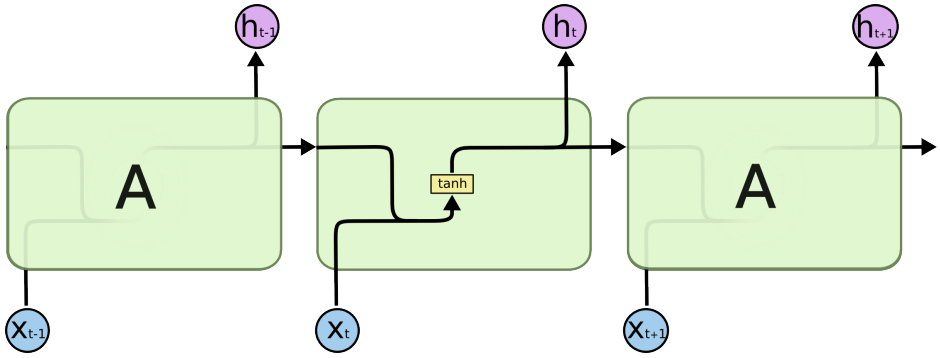


Figure 3.3: Illustration of a recurrent unit. Figure from [28].

Long Short Term Memory

Long Short Term Memory (LSTM)is a type of RNN that differs from others by having a both a memory cell and a hidden state. The LSTM can remember long time dependencies due to memory cell, which is updated for every time step depending on a forget gate and an update gate. These two gates both depends on the current input and the previous output. An illustration of a LSTM can be seen in Figure 3.4.

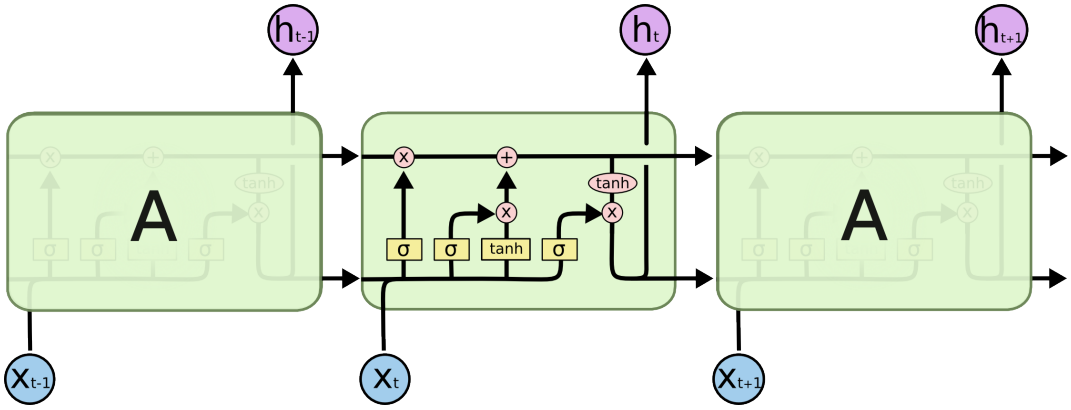


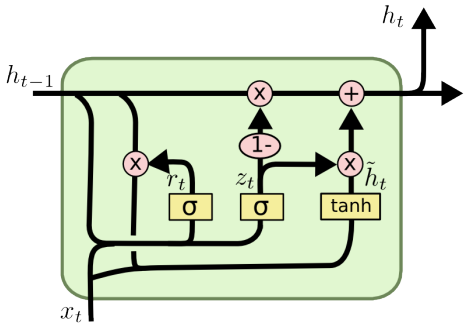
Figure 3.4: Illustration of a LSTM. Figure from [28].

Where the line going through all blocks is the memory cell, and the first change to the memory cell in each block is the forget gate, followed by the update gate. These two stages can be seen as usually fully connected layers, where they sum up the weighted inputs and weighted bias. Then transform it through the activation function. [29]

Gated Recurrent Unit

Gated Recurrent Unit (GRU) is also a type of RNN, this type of unit only have a hidden state compared to the LSTM which have both a hidden state and a memory cell.

The idea of the GRU is that it updates the hidden state at every time step using three gates. The updated data in the hidden state is then used as output for that time step and as part of the input to the next time step. The GRU structure further consist of one less gate then the LSTM. An illustration of a single GRU cell can be seen in Figure 3.5. Where the formulas, for the different point in the GRU cell, can also be found.[28] [30]



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3.5: Illustration of a GRU, and the formulas for the unit. Figure from [28].

Where W_r, W_z and W are the trainable weights of the GRU.

Backpropagation

Backpropagation is used to update the weights in the network to give the desired output. This is done by calculating the error of the network (the distance between the desired and actual output). When the

overall error is calculated, the contribution to the error of the different weights are calculated, by finding the partial derivatives. An example can be seen in equation (3.1), where the chain rule is used to find the influence of weight (w_{11}) from Figure 3.1. First the contribution from unit (U_1^2) have to be found from the total error, then the contribution through the activation function (act_1^2), and last the contribution of w_{11} on the input to the activation function.

$$\frac{\partial E_{total}}{\partial w_{11}} = \frac{\partial E_{total}}{\partial U_1^2} \cdot \frac{\partial U_1^2}{\partial act_1^2} \cdot \frac{\partial act_1^2}{\partial w_{11}} \tag{3.1}$$

This is done for every weight in the network, when the contribution of all weight to the error have been found. They are then used for updating the weights to reduce the error for the next batch. The change in the weights also depends on the learning rate (ϵ), this dependency can be seen in equation (3.2). [31]

$$w_{11}^{t+1} = w_{11}^t - \epsilon \cdot \frac{\partial E_{total}}{\partial w_{11}^t} \tag{3.2}$$

Momentum

The idea of momentum is to use the previous updates to the weights for speeding up convergence. It adds the term $p\Delta w_{t-1}$ to the gradient descent calculation where p is a factor usually 0.9 and w_{t-1} is the last updates to the weights. The full equation can be seen in equation 3.3. [32]

$$\Delta w_t = -\epsilon \nabla_w E(w) + p\Delta w_{t-1} \tag{3.3}$$

Where ∇ represents the partial derivatives, this means the first part of the equation is the same as is equation (3.2). With the difference that in (3.3) all weights are considered. An extended example of equation (3.2) with momentum can be seen in equation (3.4).

$$w_{11}^{t+1} = w_{11}^t - \epsilon \cdot \frac{\partial E_{total}}{\partial w_{11}^t} + p\Delta w_{11}^{t-1} \tag{3.4}$$

Batch Normalization

Batch normalization is used to normalize the outputs leading to the next layers. Where there are two methods that are normally used for calculating the normalization, one only depending on the mini-batch and one with running average. Where the method for mini-batch normalization uses mean and standard deviation calculated based on the mini-batch $B = (x_1 ... x_m)$. The steps for normalizing the outputs are shown in equation (3.5). [33]

$$\begin{aligned} \mu_B &\leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma_B^2 &\leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \\ \hat{x_i} &\leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \end{aligned} \tag{3.5}$$

Where $\hat{x_i}$ is the normalized output of x_i and ϵ is used for stability (do not divide by 0). The normalization uses moving average, calculates the mean and standard deviation depending on both the current and previous batches.

4 Setup

4.1 Setup

The networks will be trained and tested on two desktop computers, they are both equipped with NVIDIA GPUs for faster training of the networks. The specifications of the computers can be seen below:

- Computer 1:
  - CPU: Intel I7-3770k
  - RAM: 24 GB
  - GPU 1: NVIDIA GTX 1070 8 GB
  - GPU 2: NVIDIA GTX 1060 6 GB
- Computer 2:
  - CPU: Intel I5-6500
  - RAM: 32 GB
  - GPU 1: NVIDIA GTX 1060 6 GB
  - GPU 2: NVIDIA GTX 1060 3 GB

The framework used, for designing the networks, is Keras [34] with TensorFlow [35] as backend.

4.2 Framework for NN Testing

A framework for handling the data for the different NNs was made in order to make it easier to test each step and reuse as much code as possible.

An overview of the final framework can be seen in 4.1.

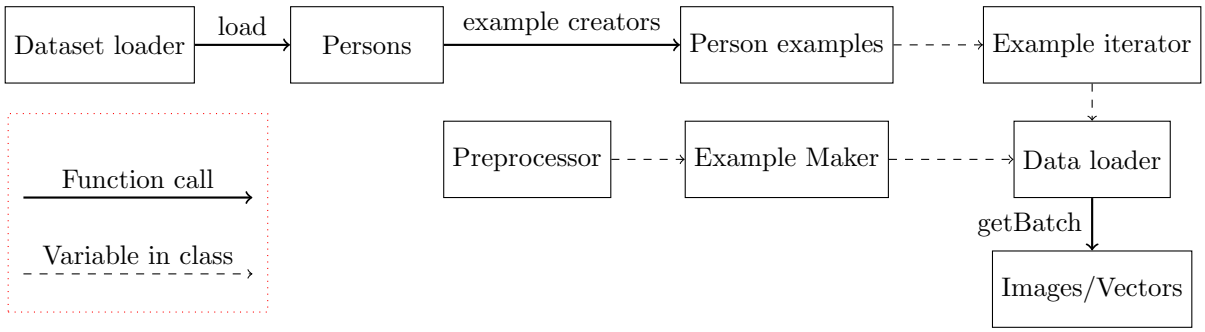


Figure 4.1: Overview of the framework.

The framework splits loading and making examples into several smaller parts. The first part of the process is to load the dataset. This is not done by loading the images directly but instead storing the information about it in a class called `Person`. A list of `Person` class instances is the output for all dataset loaders. The `Person` class can both store a dataset with single images or sequences of images.

The next step is the `example creators` where several functions can be used to create a number of different example types like single images, sequence or siamese examples. A complimentary `Example Maker` must be added for the `Person example` class in order to be able to understand how to transform the

information into the images and vectors required by the NN. The `Example Maker` furthermore uses the `Preprocessor` class to apply preprocessing to the images.

The `Example iterator` handles going through the example list. The most normal one simply iterates through a list. A more complex iterator balances the examples based on some criteria.

The `Data loader` uses multiple threads to load the data using the example maker and example iterator. The output can then be fed to the NN.

The `Example Makers` are required to have two functions due to the multi threaded loader. The first function called `batchdata`, create the data structures required for the examples given a certain batch size. The next function called `loadexamples` is called multiple times using different threads. This function receives the examples it should load and a offset in the data structure were it should start inserting the loaded data. The loader also applies preprocessing before inserting the data. This is done by calling the `apply` function with the image and a variable called `properties` which is used to turn on or off certain preprocessing steps depending on the example. The `properties` variable is a dictionary which means that several different values can be set. The `Person examples` shown here includes a `properties` variable which are used by the `Example Makers`. Certain `Example Makers` does however also add to this `properties` variable, e.g. for sequences which requires the same preprocessing steps to be applied for multiple images.

5 Images

This section will describe the NNs made for re-identification used for dataset containing only images. The Market-1501 dataset was used for training and testing on single images. The dataset consist of 6 camera views and 1501 persons. The persons are already split in half when downloading the dataset, where 751 persons are used for training and the other 750 for testing.

In the following table state of the art performance, using neural networks, for different datasets are listed:

Dataset	Results
Market-1501	91.75 % [4]
MARS	81.21 % [4]
PRID-2011	77.3 % [44]

Table 5.1: State of the art performance for still image dataset (rank-1 accuracy).

5.1 Gated Siamese

The first CNN structure tested with still images were the base CNN from the gated siamese CNN[7]. The code for the CNN was not available, therefore an attempt was made to recreate it in Keras.

Euclidean distance and contrastive loss are not implemented in Keras but the code for making it can be found in the examples [36].

Code 5.1 Euclidean distance implemented in Keras.

```
def euclidean_distance(vects):
    x,y = vects
    return B.sqrt(B.sum(B.square(B.abs(x - y)), axis=1, ↙
      keepdims=True))

def eucl_dist_output_shape(shapes):
    shape1, _ = shapes
    return (shape1[0], 1)
```

The first function called `euclidean_distance` calculates the euclidean distance using the functions from the Keras backend which is `B`. The second function called `eucl_dist_output_shape` calculates the shape of the output. This is necessary in order to make a custom layer in Keras using the Lambda layer which will be showed later.

Code 5.2 Contrastive loss implemented in Keras.

```
def contrastive_loss(y_true, y_pred):
    margin = 1
    similarloss = (1 - y_true) * 0.5 * B.square(y_pred)
    dissimilarloss = y_true * 0.5 * B.square(B.maximum(margin - ↙
      y_pred, 0))
    dist_loss = B.mean(similarloss + dissimilarloss)
    return dist_loss
```

The contrastive loss is based on the implementation from the examples but with the modification of multiplying with 0.5 before the two square operations. This is how the original contrastive loss is defined in [37].

The idea of choosing a siamese structure is that it makes a large variance between classes and a small variation within a class. Furthermore the way the siamese structure is implemented it also have to classify each of the input images. In Code 5.3 it can be seen how the siamese structure is applied using the network structure called `cnn`.

Code 5.3 Instantiating the siamese structure.

```
input_left = Input(shape=input_dim)
input_right = Input(shape=input_dim)
vect_left = cnn(input_left)
vect_right = cnn(input_right)
distance = Lambda(euclidean_distance,
          output_shape=eucl_dist_output_shape,
          name="distance")([vect_left,vect_right])
class_pred = Dense(personcount, init="glorot_normal")
class_left = ↙
    Activation("softmax",name="class_left")(class_pred(vect_left))
class_right = ↙
    Activation("softmax",name="class_right")(class_pred(vect_right))
```

The two input shapes are declared first, they are then used to produce the two branches of the network. The feature vectors from the two branches are then used to find how similar the input images are using euclidean distance, they are further used as input to the two classification layers.

These parts are then used to create the total model, and the model used for testing. The `testmodel` only consist of the network structure from one of the branches, such that it is possible to process only one image at the time. To make the testing more efficient. The instantiation of the network can be seen in Code 5.4.

Code 5.4 Building the models.

```
modelinput = [input_left,input_right]
modeloutput = [distance, class_left, class_right]

model = Model(input=modelinput, output=modeloutput)
testmodel = Model(input=input_left, output=vect_left)
loss = {"distance": contrastive_loss,
    "class_left": "categorical_crossentropy",
    "class_right": "categorical_crossentropy"}
model.compile(loss=loss, optimizer=optimizer)
```

The siamese examples are split into two classes, one containing a single person example called `PersonExample` and one called `SiameseExample` which contain two instances of the person example. The class called `PersonExample` can be seen in Code 5.5. The two most important values for training is the path and vectorid properties. The path contains the path to the image file for the example and the vectorid contains the position in the vector used to do classification. This is different from the personid which is the id from the dataset which is not continues as it for example contain id 3 and 5 but not id 4.

Code 5.5 Person example class.

```
class PersonExample:
```

```
2     def __init__(self, path: str, personid: int, vectorid: int, ↙
        trackid=-1, cameraid=-1, properties={}) -> None:
3         self.path = path
4         self.personid = personid
5         self.vectorid = vectorid
6         self.trackid = trackid
7         self.cameraid = cameraid
8         self.properties = properties
```

Creating siamese examples used for single images are done by creating matching and non matching examples for every tracklet. The matching examples are created by matching a random images from all tracklets, to a random image from all tracklets of the same person but from another camera view. The same procedure is used for producing non matching pairs, with the difference that the tracklets are matched to tracklets belonging to other persons. This means there is a lot more non matching examples then matching, to even out this difference a balancer is used. The balancer makes sure that the networks sees as many matching as non matching examples.

The `properties` variable for a person example describes how the image should be preprocessed for this specific example. This makes it possible to add data augmentation such as flipping, to some images while having the same example without flipping, making it is possible to use offline augmentation.

The siamese example can be seen in Code 5.6.

Code 5.6 Siamese example class

```
1 class SiameseExample:
2     def __init__(self, leftperson: List[PersonExample], ↙
        rightperson: List[PersonExample]) -> None:
3         self.leftperson = leftperson
4         self.rightperson = rightperson
```

The batchdata function from the siamese example maker can be seen in Code 5.7. This function do not load the data but creates the data structure which multiple threads load the examples into. The returned data structure is used directly as input to NN in Keras which requires data structure consisting of two elements packed inside of a tuple. The first element is all inputs as a list and the second element is all outputs as a list.

Code 5.7 batchdata function from siamese example maker.

```
1         images2 = np.zeros( (batchlength,) + self._shape, ↙
            dtype=np.float32 )
2         indexs = np.zeros( (batchlength) )
3         leftId = np.zeros( (batchlength, self._personCount), ↙
            dtype=np.float32 )
4         rightId = np.zeros( (batchlength, self._personCount), ↙
            dtype=np.float32 )
5         return ([images, images2],[indexs, leftId, rightId])
6
7     def loadexamples(self, siameseexamples: List[SiameseExample], ↙
        datastructure, batchindex: int):
```

The `loadexamples` function, shown in Code 5.8, is called by multiple threads where each thread loads a subset of the required examples for the batch. The examples that should be loaded by the specific thread is contained in the `siameseexamples` parameter and the `batchindex` indicates a offset in the data structure were the examples should be loaded into. The data structure is also given as a parameter called `datastructure`. The examples are loaded by unpacking the data structure into its individual data structures and then filling it with the data from the examples. The `same` variable indicates how large the euclidean distance should be between the two examples. This is 0 if the two examples are matching and 1 if they are not matching. This value will be the input to the `y_pred` parameter in the contrastive loss function shown in Code 5.2.

Code 5.8 loadexamples function from siamese example maker.

```
1          leftimage, rightimage = inputdata
2          same, leftid, rightid = output
3
4          for idx, example in enumerate(siameseexamples):
5              if example.leftperson[0].personid == ↙
                 example.rightperson[0].personid:
6                  same[batchindex + idx] = 0.0
7              else:
8                  same[batchindex + idx] = 1.0
9
10             leftimg = ↙
                 self._imageloader.load(example.leftperson[0].path)
11             rightimg = ↙
                 self._imageloader.load(example.rightperson[0].path)
12
13             leftprocessed = self._preprocessor.apply(leftimg, ↙
                 example.leftperson[0].properties)
14             rightprocessed = self._preprocessor.apply(rightimg, ↙
                 example.rightperson[0].properties)
15
16             leftimage[batchindex + idx, :, :, :] = leftprocessed
17             rightimage[batchindex + idx, :, :, :] = rightprocessed
18
19             leftid[batchindex + idx, ↙
                 example.leftperson[0].vectorid] = 1.0
20             rightid[batchindex + idx, ↙
                 example.rightperson[0].vectorid] = 1.0
```

The article reduces the learning rate after each epoch. This is implemented using the Keras callback functionality. This makes it possible to add a class whose functions are invoked after a certain time during training, such as the end of each epoch, end of batch and so forth. The implementation can be seen in Code 5.9. `K` is the Keras backend which is used for extracting the learning rate and setting it again. The `reducerate` factor describes how much of the learning rate is kept for each epoch, as it is multiplied with the previous learning rate to produce the new.

Code 5.9 Learning rate reducer.

```
1 class LRReducer(Callback):
2     def __init__(self, reducerate):
3         super(LRReducer, self).__init__()
4         self.reducerate = reducerate
5 
6     def on_epoch_end(self, epoch, logs = None):
7         lr = K.get_value(self.model.optimizer.lr)
8         K.set_value(self.model.optimizer.lr,(lr*self.reducerate))
```

The article subtracts a mean image from the input image. The mean image is created based on the Market-1501 dataset. This mean image can be seen in Figure 5.1.

Figure 5.1: Mean of all images in the market dataset.

A preprocessing module for subtracting the mean image was implemented for the framework, the module can be seen in Code 5.10.

Code 5.10 Mean image substract preprocessing module.

```
1 class MeanImgNorm(AbstractPreModule):
2     def __init__(self, imgpath):
3         self.img = cv2.imread(imgpath)
4 
5     def apply(self, image, properties):
6         return image-self.img
```

The input to the siamese NN consist of two examples which are made of either matching or non matching examples. There are fewer similar examples than dissimilar examples, therefore the article use data augmentation to make a more equal amount of dissimilar and similar examples. It specifically uses flip and translation for data augmentation. Translation is applied 4 times and is translated 0.05 or -0.05, of the width and height respectively, in both directions.

The implementation uses the new example creator to make the siamese examples and create the augmentation. The augmentation is applied by using the `properties` variable in the person example to set if flipping should be applied or how the image should be translated. The article does not describe how to create the siamese examples from the list of persons. Creating the siamese examples can be done in a lot of different ways. The method chosen creates all the possible combination of similar pairs for each person. Augmentation is used for each matching pair in order to create 5 times more examples. The dissimilar examples are created by choosing a random image of each person, and matching it with a random image from all other persons. This is repeated such there are as many dissimilar pairs as similar pairs.

The test of the siamese NN is done by using only a single instance of the two duplicate NN structures. The result of the NN is then a single feature vector. The test was made using the Market-1501 dataset which is already split into a training, test(gallery) and query set(probe). The gallery is a matrix consisting of the feature vectors produced by the NN and the probe set is iterated and each feature vector is compared against the gallery features.

The comparison of the gallery feature vectors and the probe feature vector was made by `EuclideanTest` class which can be seen in Code 5.11. The class is instantiated by giving 3 parameters called `galleryvectors`, `galleryindexs` and `cameraindexs`. The `galleryvectors` are the feature vectors for the gallery. `galleryindexs` are the person indexs of the persons in the gallery features. The last parameter `cameraindexs` are the camera indexs for the gallery features. The `rank` function returns the index of the first correctly matching gallery feature vector in the list of candidates sorted by the euclidean distance.

Code 5.11 Class handling calculating the euclidean distance and the rank.

```
1 class EuclideanTest:
2     def __init__(self, galleryvectors, galleryindexs, cameraindexs):
3         self._galleryvectors = galleryvectors
4         self._galleryindexs = galleryindexs
5         self._cameraindexs = cameraindexs
6 
7     def rank(self, probevector, index: int, cameraindex, debug=False):
8         diffvector = self._galleryvectors-probevector
9         eucdist = np.linalg.norm(diffvector, axis=1)
10        disabled = np.logical_and(self._galleryindexs == index, ↙
            self._cameraindexs == cameraindex)
11        eucdist[disabled] = 999999999.#
12        sortedindex = np.argsort(eucdist)
13        sameperson = self._galleryindexs[sortedindex] == index
14        index = np.nonzero(sameperson)[0][0]
15        rank = int(index+1)#CMC is not zero indexed
16        if debug:
17            return rank, self._galleryindexs[sortedindex][0]
18        else:
19            return rank
```

The network was then trained and tested. It was however only possible to get a rank-1 accuracy of approximately 37.1 %, rather than the 75 % from the article. The lacking accuracy might be due to a bug in the implementation or differences in the code due to missing implementation details. A solution with a good accuracy and available source code would therefore be desired.

5.2 ResNet

A solution with good results and open source code was found [38]. This was used to create a solution for still images. It is based on a pre trained version of ResNet-50 where the last layer is changed to classify the persons in the training set. The second last layer outputs a 2048 dimensional vector which is used during testing to compare persons using the euclidean distance. The code is provided by the makers of the Market-1501 dataset and is made using Caffe and Matlab.

ResNet is a deep neural network designed for the ImageNet challenge, the network is designed to handle input images of size 224×224 pixels. The most special thing about the residual network structure is the shortcuts across one block, where one block can be two to three convolutional layers. At the end of the block, the result of the convolutional layers are combined with the values from the shortcut using summing [39].

The structure of ResNet-50 with another classification layer can be seen in Table 5.2, where there is a shortcut for each block consisting of 3 convolutional layers.

Structure	Filters	stride	Number of blocks
7×7	64	2	
3×3	max pool	2	
$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \\ 1 \times 1 \end{bmatrix}$	$\begin{bmatrix} 64 \\ 64 \\ 256 \end{bmatrix}$		3
$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \\ 1 \times 1 \end{bmatrix}$	$\begin{bmatrix} 128 \\ 128 \\ 512 \end{bmatrix}$		4
$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \\ 1 \times 1 \end{bmatrix}$	$\begin{bmatrix} 256 \\ 256 \\ 1024 \end{bmatrix}$		6
$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \\ 1 \times 1 \end{bmatrix}$	$\begin{bmatrix} 512 \\ 512 \\ 2048 \end{bmatrix}$		3
Fully connected (751 classes)			

Table 5.2: The network structure used for performing re-identification on still images.

An illustration of how a shortcut works for one block can be seen in Figure 5.2.

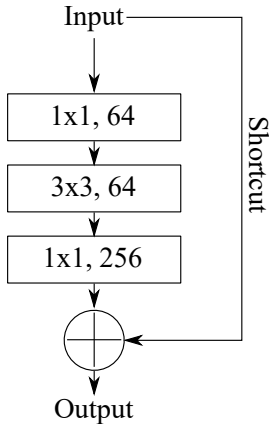


Figure 5.2: Illustration of one block from the ResNet-50 network with shortcut.

The existing solution based on Caffe became an issue since it used a lot of memory. The problem

existed in the way Caffe utilizes the memory on the GPU, where it assigns memory for the network to be trained (all the weights, gradients, images and so on). When Caffe then have to test the network, it simply tries to allocate the memory required for testing the network, on top of what is already allocated. Therefore it was chosen to use Keras with Tensorflow as backend, as it was possible to use both CPU and GPU at the same time. Meaning the network could be trained on one GPU and tested on another GPU or the CPU. This did however mean that the network had to be recreated in Keras.

This was done by using another existing implementation of ResNet-50 in Keras [40] and modifying it to match the existing solution from Caffe.

One of the things that needed to be added to the model was the variable from Caffe called "weight_decay". This parameter is used to set the regularization of the layers in the network, which combines the mean square error (mse) and mean square weights (msw) by a variable (λ in this report). The combination is calculated as in (5.1) and is used when performing back propagation, instead of the mse.

$$msereg = \lambda * msw + (1 - \lambda) * mse \tag{5.1}$$

Where *msereg* is the regularized mse [41]. The λ parameter is simply set as one variable in the solver file in Caffe. Whereas in Keras it have to be set for every layer, an example can be seen in Code 5.12 line 1, where it is defined that the regularization is L2 and the λ parameter is set to 0.0005.

Code 5.12 Weight decay implementation.

```
1 x = Convolution2D(nb_filter1, 1, 1, name=conv_name_base + '2a', ↙
    W_regularizer=l2(0.0005))(input_tensor)
2 x = BatchNormalization(axis=bn_axis, name=bn_name_base + '2a')(x)
3 x = Activation('relu')(x)
```

There is furthermore added a dropout layer before the classification layer. The classification layer is also changed from the 1000 classes of the Image net challenge, to 751 classes representing the different people in the dataset. It is shown in Code 5.13 how the dropout layer and the fully connected layer is added.

Code 5.13 Dropout layer added.

```
1 x = Dropout(0.5)(x)
2 x = Flatten()(x)
3 output = Dense(751, activation='softmax', name='fc751', ↙
    W_regularizer=l2(0.0005), init=myinit)(x)
```

The last thing added to the model is the definition of which optimizer to use. Where the same settings as in the original Caffe model is chosen. With standard gradient descent optimization, a learning rate of 0.001 and a momentum of 0.9. This is shown in Code 5.14.

Code 5.14 Momentum.

```
1 sgd = SGD(lr = learningrate, momentum = 0.9)
```

One further detail for the optimization was that the learning rate had to be divided by 10 after 20000 batches, this was implemented using the callback function from Keras where it is called after each batch. The function called after each batch can be seen in Code 5.15. Where the variable `iterrun` is incremented for each batch.

Code 5.15 Learning rate reduction.

```
1 def on_batch_end(self, batch, logs=None):
2   if self._iterrun <= self._maxstepsize:
3     if self._iterrun % self._stepsize == 0 and not self._iterrun == 0:
4       lr = K.get_value(self.model.optimizer.lr)
5       K.set_value(self.model.optimizer.lr,(lr*self._reducerate))
6
7     self._iterrun += 1
```

The last thing that was implemented in the attempt to recreate the good results, was the data augmentation. The data augmentation used is only online flipping of the images, meaning that an image is only shown ones per epoch, where it can either be flipped or not. This is implemented by randomly selecting whether the image should be flipped or not for each epoch. The implementation of this can be seen in Code 5.16. Where the property "`flip`" is set to "`random`" when the network is trained on the Market-1501 dataset, and when it is trained on single images.

Code 5.16 Flipping images.

```
1 class Flip(AbstractPreModule):
2     def apply(self, image, properties):
3         if "flip" in properties:
4             if isinstance(properties["flip"], bool) or ↙
                (properties["flip"] == "random" and ↙
                random.randint(0,1)==0):
5                 return np.fliplr(image)
6             else:
7                 return image
8         else:
9             return image
```

The class seen in Code 5.16 is also used for flipping images when the network is trained for sequences. This is done by adding the property "`flip`", as a boolean, to a sequence of images that should be flipped. When the sequence should not be flipped the property is simply not added to the images.

For training these networks the training data is required, and to handle the data the framework described in Section 4.2 *Framework for NN Testing* is used. The `PersonExample` class from the siamese NN is reused, this class can be seen in Code 5.5 on page 24. It contains all the information about each example such as the id of the person, what camera the image is from and the path to the image. When all person examples are created they can be extracted. Where a class is made for doing this, it is called `SinglePersonMaker`. The class have two functions not considering the instantiation. The first function called `batchdata` (seen in Code 5.17) is used to produce the data containers for one batch. The data containers are required as the data is loaded by multiple threads at the same time.

Code 5.17 Creating data containers.

```
def batchdata(self, batchlength: int):
  images = np.zeros( (batchlength,) + self._shape, dtype=np.float32 )
  if self._makeidvector:
    indexs = np.zeros( (batchlength, self._personCount))
  else:
    indexs = np.zeros( (batchlength, ), dtype=np.int32 )

  if self._includecamera:
    camids = np.zeros( (batchlength, ), dtype=np.int32 )
    return (images, indexs, camids)
  else:
    return (images, indexs)
```

When the data containers are created they can be filled with data using the function from the `SinglePersonMaker` class called `loadexamples`. This function loads a given number of images determined by the length of the `examples` list. This list contains the person examples that should be loaded into the data container. Furthermore the function determines where in the list the data should be placed using the variable `batchindex`. It can be seen how this is implemented in Code 5.18.

Code 5.18 Loading image data.

```
def loadexamples(self, examples: List[PersonExample], ↙
    datastructure, batchindex: int):
  if self._includecamera:
    images, indexs, camids = datastructure
  else:
    images, indexs = datastructure

  for idx, example in enumerate(examples):
    img =  self._imageloader.load(example.path)

    processed = self._preprocessor.apply(img, example.properties)
    images[batchindex + idx, :, :, :] = processed

    if self._makeidvector:
      idvec = np.zeros(self._personCount)
      idvec[example.vectorid] = 1.0
      indexs[batchindex + idx, :] = idvec
    else:
      indexs[batchindex + idx] = example.personid
    if self._includecamera:
      assert example.cameraid != -1, "Camera id not initialized"
      camids[batchindex + idx] = example.cameraid
```

The solution was tested using the Matlab script from the existing re-identification solution which is also used as baseline code for the Market-1501 dataset. The existing Matlab script loads some Matlab

matrix files with the vectors, person ids and camera id for both gallery and probe examples. The Matlab script was changed to load data exported from Keras as HDF5 files.

The existing script performs normalization before making the euclidean distance comparison. The normalization can be seen in Code 5.19 where `galFea` and `probFea` are the gallery and probe feature that have been loaded. The normalization is done by finding the absolute mean value for each element in the feature vector, and dividing each element in each feature vector with the respective mean value.

Code 5.19 Normalization.

```
1 sum_val = sqrt(sum(galFea.^2));
2 for n = 1:size(galFea, 1)
3     galFea(n, :) = galFea(n, :)./sum_val;
4 end
5
6 sum_val = sqrt(sum(probFea.^2));
7 for n = 1:size(probFea, 1)
8     probFea(n, :) = probFea(n, :)./sum_val;
9 end
```

The function for calculating the euclidean distance between the gallery and probe vectors can be seen in line 2 in Code 5.20. Where the function is a smart way to calculate the euclidean distance as matrix computation, as it exploits that the distance between two points can be re-written using the rule in (5.2):

$$(P_1 - P_2)^2 = P_1^2 + P_2^2 - (2 * P_1 * P_2) \tag{5.2}$$

Where P_1 and P_2 are two arbitrary values.

Using this re-writing for all elements in the feature vector, it is possible to re-write the euclidean distance (shown in (5.3)) of many feature vectors to the function in Code 5.20.

$$\sqrt{\sum (V_1 - V_2)^2} \tag{5.3}$$

Where V_1 and V_2 are to arbitrary vectors of the same size.

Code 5.20 Euclidean Distance.

```
1 my_pdist2 = @(A, B) sqrt( bsxfun(@plus, sum(A.^2, 2), sum(B.^2, ↙
    2)') - 2*(A*B'));
2
3 dist_eu = my_pdist2(galFea', probFea');
4 [CMC_eu, map_eu, ~, ~] = evaluation(dist_eu, label_gallery, ↙
    label_query, cam_gallery, cam_query);
5
6 fprintf(['The IDE (' netname ') + Euclidean performance:\n']);
7 fprintf(' Rank1, mAP\n');
8 fprintf('%5.2f%%, %5.2f%%\n\n', CMC_eu(1) * 100, map_eu(1)*100);
```

The Cumulative match characteristic (CMC) curve is extracted in `evaluate`, where it follows the rules for the dataset. Where the rules are that all images with an id equal to -1 and images of the same person from the same camera view in the gallery set should be marked as junk images, and should therefore not

affect the accuracy. Whereas images with id equal to 0 is used to distract. The accuracy for each rank is returned as a vector.

The rank in the CMC describes the sorted gallery images for a given probe image. Where rank-1 is the best matching and rank-2 is the second best matching and so on to rank-n, where n describes the total number of images in the gallery set. Where the accuracy at the different ranks describe how many of the probe images are matched to the correct gallery id, given this number of guesses. This means that for rank-5 if one of the 5 best matches contain a correct match for that probe image, it will return an accuracy of 100 %. The mean accuracy is then calculated for the entire probe set, by summing the accuracy for the different images and dividing by the number of probe images.

Furthermore the function `evaluate` also returns the mean average precision (mAP), which is calculated based on how many true matches there is in the gallery for one probe image, and at which ranks the matches are found. The average precision (AP) is calculated as in (5.4).

$$AP = \frac{\sum_{k=1}^{n} P(k) * corect(k)}{N_{match}} \tag{5.4}$$

Where n is the index of the last true match from the gallery. $P(k)$ describe the accuracy at the different ranks, and $correct(k)$ is 1 when rank-k is a true match and 0 otherwise. N_{match} is the total number of true matches in the gallery for a probe image. [42]

To get the mAP of the entire probe set, the mean is calculated as usual by summing the AP for all probe images and dividing by the number of probe images.

The result from the existing implementation and the recreated implementation, tested using the Matlab script, can be seen in table 5.3

The training time for the recreated solution without dropout and 20 epochs was approximately 9 hours. The solution with dropout is tested after 49 epochs with normalization. The augmentation type used is live flip.

Implementation	Rank-1	mAP
Existing without dropout	75.62%	50.68%
Our without dropout	74.44%	51.84%
Difference (Our-Existing)	-1.18%	1.16%
Existing with dropout	78.92%	55.03%
Our with dropout	76.07%	53.80%
Difference (Our-Existing)	-2.85%	-1.23%

Table 5.3: Market-1501 test of implementations.

The rank-1 accuracy is about 1.2-2.85% less for the recreated solution then the existing solution. The mAP is also within that range with the exception of the solution without dropout which reaches about 1.2% more.

The difference suggest that there might be some small details that are missing. The accuracy is however high enough, to deem that the solution is useful for testing the accuracy when using only the lower body.

The solution worked well so it was decided to test it on other datasets. It is interesting to use the current solution as part of the video solution so video datasets are used for testing. The video datasets are converted into still image datasets by randomly selecting an image from the video sequence.

The MARS dataset is quite big and reflects the same type of scenario and is therefore tested. The smaller dataset called PRID2011 is also tested. The results of the tests can be seen in Table 5.4.

Dataset	Not Normalized	Normalized
MARS	69.15 %	70.78 %
PRID2011(B->A)	25.1 %	29.8 %

Table 5.4: Testing single images using video datasets.

The test with MARS is trained for 17 epoch and uses a dropout rate of 0.5.

The test using PRID2011 is trained using half of the persons that appear in both camera views together with all the persons only appearing in camera view B. When testing, the persons only located in camera view A are used to distract.

5.3 Lower Body Only

The test of the lower body is done by using the observation from [43] that the lower body usually consists of the 55 % of the bottom of the bounding box. This is an estimate, it is however chosen to use 50 % instead to reduce the risk of getting information from the upper body. This is thought necessary as it looks like there is information from the upper body when using 55 %. Examples of cropped images with 55 % vs 50 % can be seen in Figure 5.3.

Figure 5.3: PRID2011 cropped to focus on lower body, top row is cropped to keep 55 % of height, bottom row is 50 %.

The cropping is implemented using a new preprocessor which can be seen in Code 5.21. This module

is inserted as the first module such that resizing and etc. is done after cropping.

Code 5.21 Code for cropping to get lower body only.

```
1 class CropHeight(AbstractPreModule):
2     def __init__(self, ratio):
3         self.ratio = ratio
4
5     def apply(self, image, properties):
6         return image[int(image.shape[0]*self.ratio):,:]
```

The implementation always rounds down the amount of pixels, found by multiplying the height with the ratio, to the nearest integer. This mean the crop is not pixel perfect this is however not necessary considering that the ratio is only an estimate and vary from person to person. Notice that the ratio is the ratio of the height that should be cropped of the top. The ratio should be between 0 and 1, where 0 is no cropping and 1 is the entire image cropped away. An example could be the lower 55 % should be kept, then the ratio should be 0.45.

Test Results

In this section tests are made to show the influence of only using the lower body for re-identification with single images. The accuracy of the lower body will be compared to that of the upper body and the full body. As these test are considering single images, the video dataset will report average of multiple tests as the images are selected randomly from each sequence.

The first test was made on the Market-1501 dataset with the previously used settings with the addition of the `CropHeight` preprocessor module. The augmentation used was online flipping which have been used for the best performing implemented solution. Results are shown in Table 5.5 and Table 5.6.

Epoch	With Normalization	Without Normalization	Training accuracy
10	36.52 %	36.76 %	95.95 %
20	36.73 %	36.31 %	99.89 %
40	37.26 %	37.35 %	99.98 %
50	37.35 %	37.11 %	100 %

Table 5.5: Market-1501 rank-1 accuracy using the lower body only.

Rank	With Normalization	Without Normalization
5	59.06 %	59.53 %
10	67.64 %	68.05 %

Table 5.6: Market-1501 epoch 50 rank-5 and rank-10 accuracies.

A second test was made with static flip and translation instead of online flip, on the Market-1501 dataset. Results are shown in Table 5.7.

Epoch	With Normalization	Without Normalization	Training accuracy
10	29.69 %	30.64 %	94.70 %
20	31.21 %	32.60 %	99.34 %
40	32.07 %	32.96 %	99.91 %
50	32.07 %	33.22 %	99.92 %

Table 5.7: Market-1501 rank-1 accuracy using offline flipping

The rank-5 and rank-10 of epoch 50 gave 54.54 % and 64.22 % without normalization and 53.68 % and 63.69 % with normalization.

Training on the MARS dataset with single images also used the `CropHeight` preprocessing module. Furthermore all the settings are the same as the best results on the Market-1501 dataset and the MARS dataset. Results are shown in Table 5.8.

Epoch	With Normalization	Without Normalization	Training accuracy
6	29.63 %	29.2 %	99.45 %
12	29.65 %	29.1 %	99.98 %
50	29.87 %	29.81 %	100 %

Table 5.8: MARS rank-1 accuracy using the lower body only, average of 5 test.

Training on PRID2011 followed the same procedure as above. The average of 10 results can be seen in Table 5.9.

Epoch	With Normalization	Without Normalization	Training accuracy
10	10.9 %	9.4 %	84.6 %
30	10.2 %	8.7 %	97.8 %
50	12.6 %	9.8 %	98.5 %

Table 5.9: PRID2011 rank-1 accuracy using the lower body only, average of 10 test.

The rank-5 and rank-10 of epoch 50 gave 21.1 % and 27.6 % without normalization and 23.7 % and 31.9 % with normalization.

A summery of the results can be found in Table 5.10, where the second line indicates whether normalization in the Matlab test script is used or not.

Dataset	Lower body only		Full body		Lower body only % of Full body	
Normalization	No	Yes	No	Yes	No	Yes
Market-1501	37.11 %	37.35 %	74.58 %	76.07 %	49.76 %	49.10 %
MARS	29.81 %	29.87 %	69.15 %	70.78 %	43.11 %	42.20 %
PRID2011	12.60 %	9.80 %	25.10 %	29.80 %	50.20 %	32.89 %

Table 5.10: Comparison of lower body only vs full body rank-1 accuracy.

These results shows that the accuracy drops approximately with 50 % of the full body accuracy when only focusing on the lower body, when using single images.

Another test was made using only the top 50 % of the image in order to see what the ratio is in terms of accuracy between the lower and upper part of the body. A preprocessing module similar to `CropHeight`

were used but it would crop the bottom of the image instead. The tests used online flip and the result for the Market-1501 dataset can be seen in Table 5.11

Epoch	With Normalization	Without Normalization	Training accuracy
10	41.06 %	42.40 %	97.15 %
20	42.28 %	42.52 %	99.92 %
40	42.28 %	42.55 %	99.98 %
50	42.13 %	42.87 %	99.99 %

Table 5.11: Market-1501 rank-1 accuracy using only top 50 % of the images.

The rank-5 and rank-10 of epoch 50 gave 61.46 % and 69.51 % without normalization and 62.08 % and 70.22 % with normalization.

The test was repeated using the PRID2011 dataset with only the upper body. The results can be seen in Table 5.12, which is the average of 10 tests.

Epoch	With Normalization	Without Normalization	Training accuracy
10	7.4 %	6.7 %	98.53 %
30	8.7 %	7.0 %	99.05 %
50	9.0 %	7.1 %	99.91 %

Table 5.12: PRID2011 rank-1 accuracy using the upper body, average of 10 test.

5.4 Summary

It is found that the best performing solution implemented, for single images, is the ResNet-50 structure trained for classification. Where the data augmentation resulting in the highest rank-1 accuracy is online flipping of images. It was further found that the implementation of the Gated siamese structures were missing some details which were not present in the article.

The accuracy reached using the ResNet-50 structure was deemed high enough for continuing to designing a system for sequences.

Test showed that when using only the lower body, the accuracy of the system drops to around 50 % of the full body accuracy. The tests further shows that using the upper body for re-identification result in close to the same accuracy as using the lower body but with a slightly higher accuracy.

6 Video

This section will describe the design of a NN that work for datasets containing video. The NN for video will mainly be trained on the MARS dataset[44]. The dataset is separated into two parts, one for test and one for training. The amount of persons is split approximately half-half, which means the NN will be trained using 625 persons. The other half of the dataset is used for testing.

Dataset	Results
MARS	81.21 % [4]
iLIDS-VID	58.0 % [17]
PRID-2011	77.3 % [44]

Table 6.1: State of the art performance for video dataset (rank-1 accuracy).

The network designed for the Market-1501 dataset, is used to evaluate the MARS dataset. This is done in order to have a baseline for performance on the MARS dataset. This results in a 70.78 % rank-1 accuracy and a mAP of 54.35 % using normalization.

6.1 Recurrent Neural Network

The initial idea is to add a RNN after the ResNet-50 structure in order to utilize temporal information. It is therefore necessary to make a new example class, example maker and example creator which can accommodate sequences.

The new example class is called `SeqPersonExample` which contains a list of paths to images. It furthermore have a function called `get_random_path`. The `get_random_path` function is used to get a new random subsequence of paths for each epoch. It is also possible to set a parameter called `staticpath` to a sequence length which causes `get_random_path` to return the same random subsequence for all epochs.

The example maker called `SeqPersonMaker` uses the `get_random_path` and loads all images and applies the preprocessing step to each image. It also implements export of cameras and real person IDs. Flipping the images randomly between each epoch is implemented in the maker since all images in one sequence have to be either flipped or not.

The example creator called `allPersonsSeq` uses the same principal as `allPersons` used for single images but without live flip as explained above. It also adds a `staticpath` parameter which makes it able to pick random subsequences that are used for all the epochs.

The next step is to use the entire sequences, therefore the ResNet-50 structure is used as input to a recurrent layer. Two types of recurrent layers are tested, which are LSTM and GRU.

This require that the ResNet model should be able to handle sequences. The Keras API does not include specific convolutional layers or so forth for sequences, but does instead have a layer wrapper called `TimeDistribution`. `TimeDistribution` is applied on all the convolutional, batch normalization and fully connected layers in the ResNet-50 structure, an example of this can be seen in Code 6.1.

Code 6.1 Time distributed.

```
shortcut = TimeDistributed(Convolution2D(nb_filter3,
                                         1, 1,
                                         subsample=strides,
                                         name=conv_name_base + ↙
                                           '1',
                                         W_regularizer=l2(0.0005))
                           )(input_tensor)
shortcut = TimeDistributed(BatchNormalization(axis=bn_axis,
                                              name=bn_name_base+'1')
                           )(shortcut)

x = merge([x, shortcut], mode='sum')
x = Activation('relu')(x)
```

There is no need for wrapping the activation function in the timedistribution layer, according to the examples provided by Keras. The merge layer used for the residual connections can not be wrapped in the timedistribution layer but do however run without it. The network is trained with different settings but only results in approximately 40 % rank-1 accuracy at most. There are some concerns regarding if the merge layer will create problems or not doing back propagation, as the merge layer is not wrapped in the time distribution.

The subject is therefore researched further. It is found that Timedistribution can be used for the entire ResNet Model. This works by creating a model and using the time distribution layer around it, as shown in Code 6.2.

Code 6.2 Time distributed.

```
img_input = Input((sequencelength, 224, 224, 3))
single_image = Input((224, 224, 3))
resmodel = ResNet50(learningrate,include_top=False, ↙
  input_tensor=single_image)
x = TimeDistributed(resmodel)(img_input)
```

This do however create another error, as sharing weights are not implemented correctly for batch normalization. This means that the batch normalization have to be set to mode 2, which means that the batch normalization uses per-batch statistics instead of moving average of testing.[45]

The results does however get worse than before, and deliver approximately 10% rank-1 accuracy. This might suggest that the network is to complex compared to the number of examples or that batch normalization using mode 2 works poorly.

The size of the recurrent layers internal state have been 2048 which was quite high considering that the amount of persons used for training was 625. Therefore it is tested what a reduction in the size of the recurrent layers will do.

This do not give the desired results, therefore it is decided to check the batch normalization layers. Batch normalization are among other things used to avoid vanishing or exploding gradients. The weights of the first and last convolutional layer are therefore checked which are shown in figure 6.1 and 6.2.

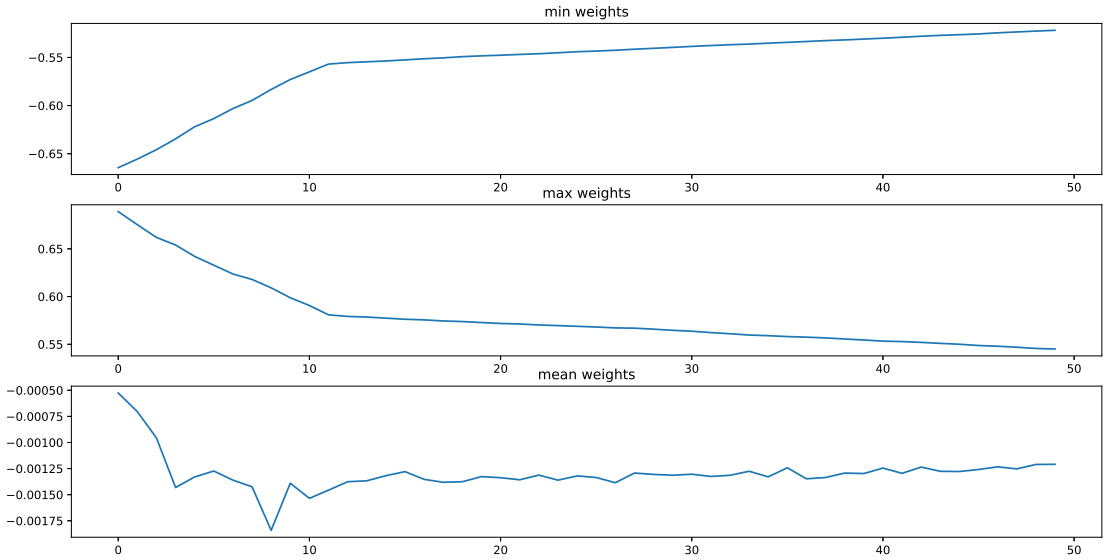


Figure 6.1: First convolution layer weights.

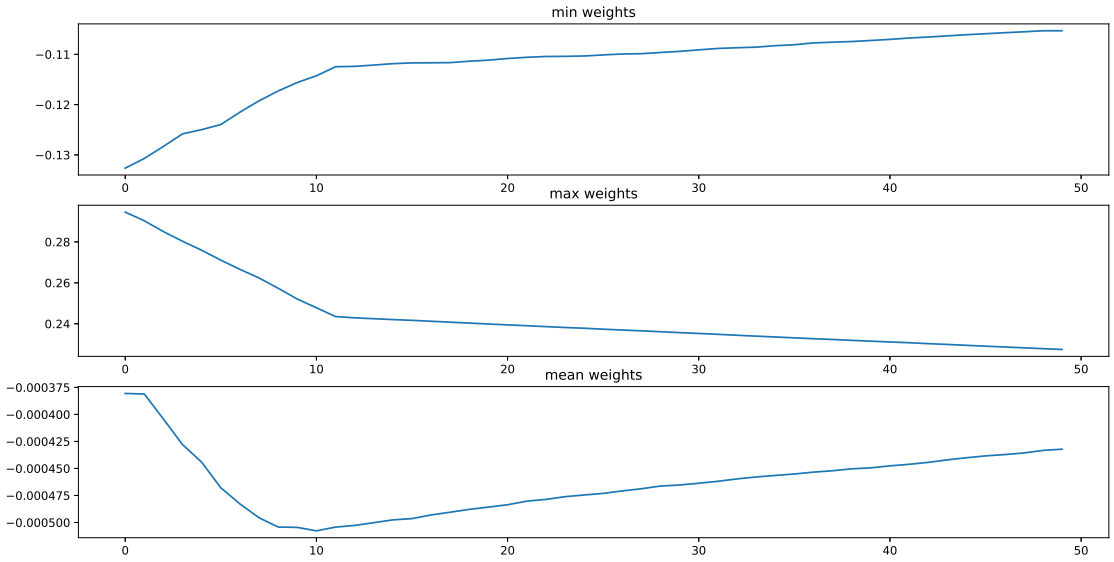


Figure 6.2: Last convolution layer weights.

It shows that the weights are updated through out the entire network. The reduction in learning rate after 20000 updates or 11 epoch in this case is also visible. This reduction might be too early considering the loss and accuracy shown in table 6.2.

Epoch	Accuracy (1=100%)	Loss
1	0.0379609550143	9.44175852181
2	0.0703784055554	8.903405037
3	0.115570018838	8.21983156626
4	0.177271634929	7.47836398699
5	0.245360332954	6.81826533401
6	0.316823338045	6.2710305169
7	0.395396490134	5.78473956618
8	0.454687887071	5.39653001124
9	0.51711256875	5.04668801039
10	0.57073995016	4.74083074871
11	0.618582803593	4.4891064411
12	0.662689817601	4.24248980169
13	0.734634863834	3.96759517603
14	0.765967714083	3.85957116511
15	0.778862387341	3.8252458115

Table 6.2: Training accuracy and loss using GRU with 512 size for 15 epoch.

Another test is made as the training accuracy did not reach close to 100 %. In this test the learning rate reduction is made after 40000 updates instead of 20000. This results in a rank-1 accuracy of 14.75 % after 40 epoch 15.66 % after 67 epochs.

The next test for the GRU uses a learning rate of 0.005 instead of 0.001 and a hidden state size of 128. The result is a rank-1 accuracy of 23.89 % after 40 epochs and 24.9 % after 100 epochs.

Another test is made to see how only adjusting the learning rate will affect the performance. The learning rate is set to 0.01 instead of 0.005. This results in a rank-1 accuracy of 24.19 % after 100 epochs.

This is worse and a new test is therefore made. Where the old learningrate of 0.005 is used with a hidden state size of 64 for the GRU. This results in a rank-1 accuracy of 22.27 % after 40 epochs and 21.52 % after 100 epochs.

A newer version of Keras is also tested but is not able to train when using both batch normalization and time distribution.

It is possible that the lack of a moving average for batch normalization causes the big problems. The first time distributed version of ResNet gave much better results despite not being time distributed across all layers. It is therefore possible that back propagation works fine with the merge layer. A test is therefore made with a 128 GRU size with the layer individually time distributed. This does however result in a rank-1 accuracy of 3.33% after 40 epochs.

Another test is made with an even smaller GRU size of 64 which do not get past 34% classification accuracy on the training data. This indicates that the last tested network is not complex enough to separate the persons shown.

6.2 Using Static CNN

As there were many challenges using time distribution to train the network, it was chosen to train the ResNet-50 structure on single images from the MARS dataset. Then use the pre trained weights for the CNN structure to produce the input to the GRU or LSTM and then only train the RNN, this approach is inspired by [6].

This was done using the possibility of telling Keras that one layer should not be updated in the training phase, it is shown in Code 6.3 how each layer in the ResNet structure is set to not be trained.

Code 6.3 Excluding layers from training.

```
1 resmodel = ResNet50(learningrate, include_top=False, ↙
    input_tensor=single_image)
2 for layer in resmodel.layers:
3   layer.trainable=False
4 x = TimeDistributed(resmodel)(img_input)
```

Training the CNN structure and the LSTM separately as a classification problem did not result in a high rank-1 accuracy, only around 5 % and without much varying depending on the LSTM cell size.

The low accuracy when training the parts separate could indicate, that the problem of not reaching a high re-identification accuracy, might be the way of making the CNN structure able to handle sequences by using time distribution from Keras. Or that the RNN finds features that are to specific for each person from the training set.

6.3 Siamese Structure

Trying to reach an accuracy close to state of the art with a RNN after the ResNet-50 structure, did not give promising results using the previously described methods. Where some of the attempts were trying to reduce the GRU or LSTM cell size, or making the ResNet-50 structure static (the weights are not updated). As these attempts did not reach the desired results it was chosen to experiment with a siamese structure. As it is a common assumption that siamese structure is well suited for re-identification as it learns to separate between images of different people and group images of the same person, the siamese structure used in this report is shown in Figure 6.3.

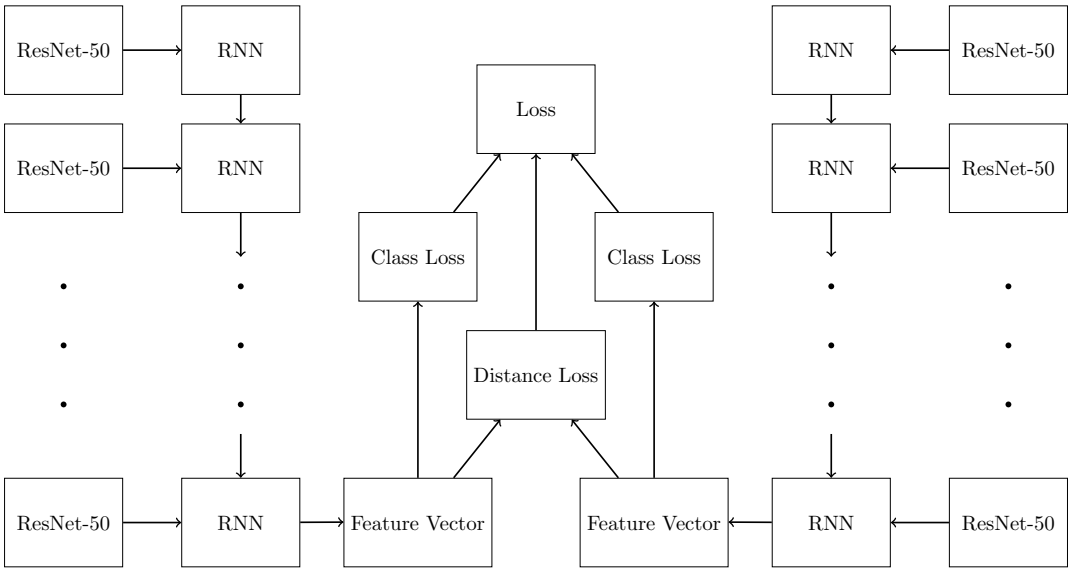


Figure 6.3: Illustration of the siamese structure used.

This did not give the desired results either, and only reached around 10 % rank-1 accuracy at most.

This low accuracy further indicate that the problem lies with either the RNN or the way of using time distribution for wrapping the network, or a combination of both.

6.4 Test of Time Distribution

Non of the attempts above have given good results when using a RNN, therefore it was decided to test what caused the low accuracy.

The first test removed the recurrent layer and used average pooling instead, in order to see the influence of the recurrent layers. The individual time distributed version of the ResNet using sequences was used with a sequence length of 5. Epoch 7 resulted in a rank-1 test accuracy of 34.04 %.

Another test was made to check if the batch normalization was the cause of the issue. This was done using the ResNet structure on single images from the MARS dataset with batch normalization set to mode 2. As mode 2 normalization is recommended when using time distributed layers [45]. This resulted in approximately 9 % rank-1 accuracy after 17 epochs. This is a lot lower accuracy compared to the baseline result for the MARS dataset (approximately 70 %).

Some further differences are however found between the current and the baseline implementation. The weights used for the baseline implementation are pre trained on the Market-1501 dataset. The current implementation furthermore uses offline flipping and translation. The current implementation with batch normalization set to mode 2 is therefore recreated using correct weights and augmentation. This do however result in a rank-1 accuracy of approximately 5 %. This indicate that the issue with accuracy might be caused by the batch normalization in mode 2.

Therefore another test is made identical to the previous test, but with batch normalization set to mode 0. This results in a rank-1 accuracy at approximately 69 %. This clearly shows that batch normalization in mode 2 can not be used for this project.

It is possible to make the individually time distributed version run with batch normalization in mode 0. Some other tests were therefore made using RNN, where the correct weight and augmentation is used together with batch normalization in mode 0. The first test uses a GRU size of 128, this results in a rank-1 accuracy a approximately 12 %. Therefore a test using a higher memory size of 512 using a LSTM is made. This results in approximately 55 % rank-1 accuracy after one epoch. The accuracy do however drop to 31 % after 4 epochs.

Further test are made with different GRU sizes and the correct weights, batch normalization and data augmentation. The results from the tests can be seen in Table 6.3.

Epoch	GRU of size 256	GRU of size 1024
1	44.17 %	55.66 %
2	41.11 %	50.92 %
3	38.79 %	47.4 %
4	35.85 %	44.35 %

Table 6.3: Rank-1 accuracy on the MARS dataset with different GRU sizes.

The results shows that the re-identification accuracy of the system declines as the system is trained for more epochs, this might be caused by the system beginning to overfit after only a few batches.

6.5 Static CNN Updated

After the problems with the normalisation and the data augmentation is fixed, the network with a RNN is trained all together. This however results in decreasing rank-1 accuracy on the test data for every epoch, as described in Section 6.4 *Test of Time Distribution*. As this does not give the desired result it is chosen to test with a static CNN again. Where the correct batch normalisation and correct data augmentation is used. Otherwise the test is identical to the one described in Section 6.2 *Using Static CNN*. The results of the static CNN with a recurrent unit at the end can be seen in Table 6.4.

	GRU size 128	GRU size 1024	GRU size 2048
Epoch 1	59.33 %	66.69 %	67.19 %
Epoch 2	57.29 %	66.33 %	66.57 %
Epoch 4	57.75 %	65.22 %	67.23 %
Epoch 8	59.35 %	65.83 %	66.82 %

Table 6.4: Rank-1 test accuracy for different Memory size using a static CNN.

As the result from using a pure static CNN seems to stay close to the starting accuracy throughout the training it is decided to let the network update the weights after a given number of epochs. This approach is inspired by [6]. Activating the layers for training is done as shown in Code 6.4, which is placed inside the training loop where epochNb is incremented after each epoch. This resulted in similar results to those in Table 6.4.

Code 6.4 Training CNN after epoch 3.

```
1 if epochNb == 3:
2   for layer in self.model.layers:
3     layer.trainable = True
```

6.6 Single Images

It is decided to test how well the ResNet structure will work without a RNN at the end, as there are problems with the performance when trying to use a recurrent layer.

The first idea of how to use the trained network is to just choose one random image from each tracklet and then test these single images using the market baseline code. Where the feature vectors have to be given to the Matlab script together with the camera ID and the person ID. The second way the trained ResNet structure could be used, is to average pool all feature vectors produced from one tracklet, resulting in one feature vector for each tracklet. This is illustrated in Figure 6.4. Where `Input 0` to `Input N` represent all images from one tracklet.

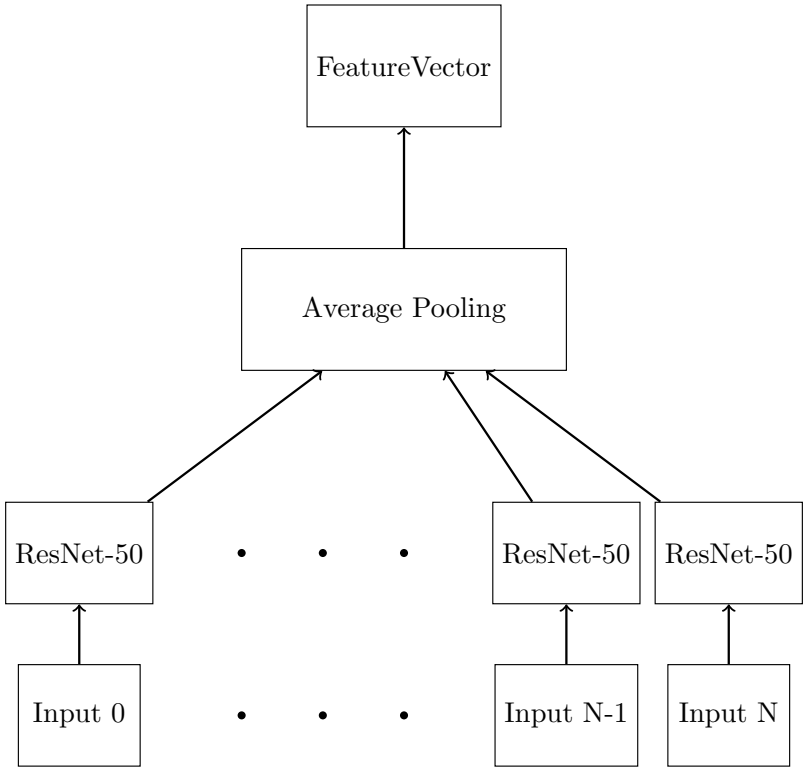


Figure 6.4: Illustration of how average pooling is applied.

The second way of using the ResNet structure is extended such it also included a vector that is pooled using `min` and `max`. Furthermore it is tested how well the three pooling methods worked separately.

The implementation of these pooling layers can be seen in Code 6.5, where the batch size is set to 1 such the network only processes one sequence at the time. The pooling is only applied when the network is tested, to give an average of how the features look through the entire tracklet.

Code 6.5 Average Pooling.

```
1 if not averagefeatures:
2   vectors = model.predict(images)
3 else:
4   batchimages = images
5   vectors = model.predict(batchimages)
6   vectors = np.average(vectors, axis=0)
```

As this method have proven to deliver good results on the MARS dataset, it is chosen to test the network on other video datasets for re-identification. The network is trained on the MARS dataset and tested on another dataset, the results of these tests can be seen in Table 6.5.

When performing the test of the system, both the pooling and single image solution is used. The single image solution can give some varying results as it chooses a random single image from each tracklet each time the test is performed. Therefore an average of 10 test is presented as the result for the single image solution.

Method/Dataset	MARS		PRID2011 A->B		PRID2011 B->A		ILIDS-VID	
Normalization	No	Yes	No	Yes	No	Yes	No	Yes
Single images	69.15%	70.78%	?	?	8.8 %	8.3 %	4.8 %	5.8 %
Average pooling	77.93 %	80.25 %	12 %	11.5 %	12.5 %	11 %	9.67 %	9.33 %
Max pooling	74.34 %	77.47 %	?	?	?	?	?	?
Min pooling	64.39 %	70.56 %	?	?	?	?	?	?
Combi pooling	75.66 %	77.63 %	?	?	?	?	?	?

Table 6.5: Results when trained only on MARS dataset.

Table 6.5 shows that the highest rank-1 accuracy reached on the MARS dataset using this method is ≈80 %. This accuracy is close to the current state of the art rank-1 accuracy which to the best of our knowledge is ≈ 81 % on the MARS dataset. It can be seen from the test that the normalization, from the Market-1501 dataset evaluation, does not always have a positive influence.

6.7 Optical Flow

Optical flow features might be useful for re-identification using the lower body, as it might provide extra information, such as gait features, which can be used to distinguish people.

The optical flow method is tested with a combination of two NN merged together, with one using optical flow images and another using normal rgb images. The NN using optical flow images comes from [24] which uses convolutional layers for processing optical flow images from 25 frames of a sequence. The method for single images is the ResNet-50 structure used so far. The two network structures are combined by concatenating the feature vectors just before the classification layer.

The structure of the optical NN can be seen in Table 6.6.

Type	Output Dim	FS	Stride
Input	$* \times * \times 5$	-	-
Convolutional	$* \times * \times 96$	7×7	1
Max Pool	$* \times * \times 96$	2×2	2
Convolutional	$* \times * \times 192$	5×5	2
Max Pool	$* \times * \times 192$	2×2	2
Convolutional	$* \times * \times 512$	3×3	1
Max Pool	$* \times * \times 512$	2×2	2
Convolutional	$* \times * \times 512$	2×2	1
Fully Connected	512	-	-
Dropout	512	-	-
Fully Connected	256	-	-
Dropout	256	-	-

Table 6.6: Structure of gait NN

The MARS dataset is used for the first test as it is one of the biggest datasets. It unfortunately have sequences down to only 5 frames, which mean that it is not possible to use 25 frames for producing the optical flow features for all sequences. The first test therefore only uses 5 frames for computing the optical flow features.

The implementation of the optical flow network uses the `SeqPersonExample` class and adds a special example maker class called `OpticalMaker`. The part of the loadexamples function used to deal with the single image and optical flow is shown in Code 6.6.

Two preprocessors are used in the `OpticalMaker` class; one for the single image and one for the images used for optical flow. The loading and conversion of images to optical flow images can been seen in line 5-11 of Code 6.6. An additional preprocessing step is included in the example maker, which is converting the images into grey scale. As it is necessary for computing the optical flow images. The method using grey scale images for calculating the optical flow is the same as the gait article. The parameters for the Farneback optical flow had to be found through experimentation, since they were not include in the article. The parameters for the Farneback optical flow were:

- Pyramid scale: 0.5.
- Pyramid levels: 3.
- Window size: 10.
- Number of iterations: 3.
- Neighbourhood size in pixels: 5.
- Standard deviation of the Gaussian filter: 1.2.

Where pyramid scale is a scalar that describes how much the image size should be reduced for each pyramid level. With 0.5 meaning it should be half the size and 1 resulting in an image at the same size. Window size describes the kernel size of the Gaussian filter.

Code 6.6 Part of loadexamples function for OpticalMaker.

```
1              paths = person.get_random_paths(self._sequencelength)
2              images = []
3              for path in paths:
4                  unproimg = self._imageloader.load(path)
5                  prepro = self._opticalpreprocessor.apply(unproimg, ↙
                     properties)
6                  images.append(cv2.cvtColor(prepro, ↙
                     cv2.COLOR_BGR2GRAY))
7
8              for i in range(0,len(images)-1):
9                  motion[batchindex + idx,:,:,(i*2):(i*2)+2] = ↙
                     cv2.calcOpticalFlowFarneback(images[i], ↙
                     images[i+1],None, 0.5, 3, 10, 3, 5, 1.2, 0)
10
11             rgbimg = self._imageloader.load(paths[0])
12             image[batchindex + idx, :, :, :] = ↙
                 self._preprocessor.apply(rgbimg, properties)
13
14
15 class SeqVarPersonMaker:
```

The first test uses a width of 64 and a height of 128 which is half the size of the images from the MARS dataset. This results in a rank-1 accuracy of 7.47 % after 12 epochs and 8.08 % after 25 epochs, in both cases the training accuracy are above 99.9 %.

The next test tries to use a size more similar to the original size. The original size was a width of 60 and a height of 60. The test however uses 80 for height instead and 60 for width in order to keep more data. The average window size, for the Farneback optical flow was set to 15 instead of 10. This do not help, and resulted in a rank-1 accuracy of 7.07 % after 5 epochs and 8.03 % after 25 both with a training accuracy of 99.9 %.

A possible problem can be that the merge was made right before the classification layer. This can be problematic as the classification layer is removed and the merge vector became the output of the network when testing. To solve this a additional fully connected layer with a size of 2048 is added between the merge layer and classification layer. This did not help and the rank-1 accuracy was around the same as before, with a rank-1 accuracy of 8 % after 12 epochs.

The pre trained weights from the Market-1501 dataset have not been loaded correctly. This is fixed and results in a rank-1 accuracy of approximately 8 % again.

The bounding boxes for the MARS dataset are found using the Deformable Parts Model, which means the bounding boxes can vary significantly from frame to frame, this can be seen in Figure 6.5.

(a) Frame 9 **(b)** Frame 10 **(c)** Frame 11 **(d)** Frame 12

Figure 6.5: Person 5, track 2 in the MARS dataset, bounding box wrong size

Another problem created by the automatic handling is occlusion where an example can be seen in Figure 6.6.

(a) Frame 7 **(b)** Frame 8 **(c)** Frame 9 **(d)** Frame 10

Figure 6.6: Person 81, track 7 in the MARS dataset, person occluded by a umbrella.

Due to the automatic bounding boxes in the MARS dataset it is chosen to use the PRID2011 dataset instead. The bounding boxes from the PRID2011 dataset are made by hand, and therefore does not have the same issues with jumps in bounding box sizes. It is therefore decided to train on the PRID2011 dataset with the same settings as above. The dataset is split into a training and testing set by randomly selecting half of the 200 people that are in both of the two camera views. The extra people that are only in camera A are also used for training. The extra people from camera B are used as distractors in the gallery set when testing. The results can be seen in Table 6.7.

Epoch	With Normalization	Without Normalization	Training accuracy
7	35.1 %	30.7 %	96.9 %
25	36.1 %	31.7 %	99.4 %

Table 6.7: Average of 10 results for Optical flow NN.

The results are far from state of the art but it is interesting to see if they are better than only using the ResNet-50 structure.

A baseline is made by training on the PRID2011 dataset using only the ResNet-50 structure for single images. The results can be seen in Table 6.8, which shows that the addition of optical flow increase the accuracy. Single image results are the average of 10 test.

Dataset	Single images		Average pooling	
Normalization	No	Yes	No	Yes
PRID2011 A->B	25.1 %	29.8 %	20 %	22 %

Table 6.8: Results when trained on the PRID2011 dataset after 25 epochs.

The better results on PRID2011 means that another test is made with the ResNet-50 and GRU with a size of 512 which is tested on PRID2011. The results using average of 10 test are 4.9 % without

normalization and 6.8 % with normalization rank-1 accuracy, after 16 epochs. The results after 25 epochs are 4.2 % without normalization and 5.5 % rank-1 accuracy.

A second test is made with a sequence length of 25. The PRID2011 dataset do have tracks that are shorter then 25. These are simply included with the missing frames set to zero. The originally intent was that the zero frames should be discarded, unfortunately there are problems, using the masking function together with the time distribution, in Keras that made this impossible. The training accuracy reached 100 % after 28 epochs, but the testing accuracy were quite bad as shown in Table 6.9.

Epoch	With Normalization	Without Normalization	Training accuracy
5	1.1 %	1.2 %	4.9 %
10	0 %	0 %	39 %
16	0.3 %	0.2 %	82.2 %

Table 6.9: Average of 10 results for NN.

This shows that the irregular bounding boxes is not the cause of the bad performance with the GRU.

6.8 Recurrent Unit

The implemented NNs attempting to utilize temporal information works poorly, therefore an existing solution is used instead. The solution chosen is the Siamese NN with a recurrent unit at the end[17]. The rank-1 accuracy reported in the article is close to state of the art performance on the PRID2011 and ILIDS-VID datasets, furthermore the source code is available. The source code is written in the scripting language Lua and depends on the NN framework called Torch.

The source code includes the code required for training and testing. Testing is done with persons that are present in two cameras and is not in the training set. The images are augmented 3 times, first flipping the images, second cropping the images and last a combination of both. The two cameras are split into probe and gallery. Features are extracted from the images for both cameras four times using the NN, one time with the original image and three times with augmentation. The euclidean distance is calculated between the two view each time and all the distances are summed together. The summed distances between all persons in the two cameras are used to find the best matches. This is done by finding the smallest distance from a person in one camera view a person in the other camera view. The matched persons are then used for calculating the CMC accuracy.

Their code differs from the previous implementations by having a recurrent unit instead of a GRU or LSTM. The CNN only consist of 3 convolutional layers. This is simpler than the other CNNs used so far, such as the ResNet-50. They also use online flip and translation as data augmentation, rather than the offline translation augmentation used in the previously described implementations. Another difference is the use of tanh as activation function instead of ReLU.

The batch size for training the NN is 1, where the training consists of switching between a similar and dissimilar examples of a persons. The number of batches per epoch is twice the number of persons in the training set, which can be seen in line 5 in Code 6.7.

Code 6.7 Loop for training Siamese NN with a recurrent unit.

```
1     for eph = 1,opt.nEpochs do
2     ...
3         local order = torch.randperm(nTrainPersons)
4
5         for i = 1,(nTrainPersons*2) do
6       ...
7             if i % 2 == 0 then
8                 ...
9                 startA,startB,seq_length = ↙
                    datasetUtils.getPosSample(personImages, ↙
                    trainInds,order[i/2], opt.sampleSeqLength)
10                netInputA = ↙
                    personImages[trainInds[order[i/2]]][camA][{{startA,startA ↙
                    + seq_length - 1},{},{}}]:squeeze()
11                netInputB = ↙
                    personImages[trainInds[order[i/2]]][camB][{{startB,startB ↙
                    + seq_length - 1},{},{}}]:squeeze()
12                netTarget = {1,(order[i/2]),(order[i/2])}
13
14            else
15                ...
16                seqA,seqB,camA,camB,startA,startB,seq_length = ↙
                    datasetUtils.getNegSample(personImages, ↙
                    trainInds, opt.sampleSeqLength)
17                netInputA = ↙
                    personImages[trainInds[seqA]][camA][{{startA,startA ↙
                    + seq_length - 1},{},{},{}}]:squeeze()
18                netInputB = ↙
                    personImages[trainInds[seqB]][camB][{{startB,startB ↙
                    + seq_length - 1},{},{},{}}]:squeeze()
19                netTarget = {-1,seqA,seqB}
20            end
```

The similar examples are created by using the images from two cameras for a person. The person is chosen based on a randomly shuffled list with all the persons. The dissimilar person examples are instead completely randomly chosen. This is done shuffling a list containing the indexes of the list with training examples. Two different persons are found by taking the first and second of the list with shuffled indexes of training examples. This can be seen in the two next lines. This can be seen in line 3 in Code 6.8.

Code 6.8 Function to pick dissimilar examples of persons.

```
1 function dataset_utils.getNegSample(dataset,trainInds,sampleSeqLen)
2
3     local permAllPersons = torch.randperm(trainInds:size(1))
4     local personA = permAllPersons[1]
5     local personB = permAllPersons[2]
6
7     local camA = torch.floor(torch.rand(1)[1] * 2) + 1
8     local camB = torch.floor(torch.rand(1)[1] * 2) + 1
```

A baseline is created by running the code 10 times and checking that it matches the results reported in the paper. The code is run with some specific seeds in order to be able to repeat the test again with different parameters or only the lower body. The seeds used are 1 to 10. A test of PRID2011 with the average of the 10 results at different maximum sequences length can be seen in Table 6.10.

Sequence length	Rank1	Rank5	Rank10
1	44.1 %	73.9 %	85.6 %
2	49.1 %	78.1 %	88.3 %
4	52.9 %	82.1 %	89.7 %
8	55.9 %	84.4 %	91.0 %
16	61.4 %	87.0 %	92.2 %
32	64.9 %	90.0 %	93.8 %
64	68.7 %	91.2 %	95.4 %
128	70.5 %	91.8 %	96.1 %

Table 6.10: Accuracies for different sequence length on PRID2011. Average of 10 results.

The accuracy comes close to the accuracy reported in the article which can be seen in Table 6.11.

Method	Rank1	Rank5	Rank10
Article	70 %	90 %	95 %
Recreated	70.5 %	91.8 %	96.1 %
Difference(Recreated-Article)	+0.5 %	+1.8 %	+1.1 %

Table 6.11: Accuracies from article and reproduced results(our) for PRID2011.

The test is also repeated again with iLIDS-VID using the same procedure. The results can be seen in Table 6.12.

Sequence length	Rank1	Rank5	Rank10
1	10.4 %	26.7 %	35.8 %
2	16.7 %	38.8 %	51.1 %
4	21.6 %	47.2 %	60.3 %
8	25.2 %	55.7 %	68.8 %
16	36.7 %	66.8 %	78.0 %
32	45.3 %	73.6 %	84.2 %
64	51.5 %	79.4 %	89.4 %
128	53.2 %	81.2 %	90.4 %

Table 6.12: Accuracies for different sequence length on ILIDS-VID. Average of 10 results.

The accuracy difference is bigger with ILIDS-VID which can be seen in Table 6.13.

Method	Rank1	Rank5	Rank10
Article	58 %	84 %	91 %
Recreated	53.2 %	81.2 %	90.4 %
Difference(Recreated-Article)	−4.8 %	−1.8 %	−0.6 %

Table 6.13: Accuracies from article and reproduced results for ILIDS-VID.

This is a quite big difference so a second test is made with some different seeds which are 11 to 20. It is also possible that the original paper have tested with a higher max sequence length. The maximum sequence length for the ILIDS-VID dataset is 192 so this is tested as well. The results of this can be seen in Table 6.14.

Sequence length	Rank1	Rank5	Rank10
1	9.3 %	26.7 %	36.9 %
2	15.5 %	36.4 %	49.3 %
4	20.5 %	44.4 %	58.8 %
8	26.5 %	55.4 %	69.6 %
16	37.8 %	67.0 %	79.0 %
32	45.3 %	73.9 %	83.9 %
64	50.9 %	78.9 %	89.4 %
128	53.9 %	80.9 %	90.3 %
192	53.7 %	81.0 %	90.1 %

Table 6.14: Accuracies for different sequence length on ILIDS-VID seed 11-20. Average of 10 results.

The new result shows little improvement at a sequence length of 128, going from 53.2 % to 53.7 %. The longer maximum sequence length of 192 delivered a worse rank-1 accuracy than a sequence length of 128. The maximum rank-1 accuracy for a single test was 62 % and the minimum was 44 % so there is a quite big difference between the individual test results. The two test have a average accuracy close to each other as shown in Table 6.15. The difference in accuracy from the article is very unlikely to come from the tests having a bad combination of seeds. The difference from the articles results either come from a very lucky combination of seeds or the code provided is different from the articles original test.

Tests	Rank1	Rank5	Rank10
Test1	53.2 %	81.2 %	90.4 %
Test2	53.7 %	81.0 %	90.3 %
Difference(Test1-Test2)	−0.5 %	0.2 %	0.1 %

Table 6.15: Difference between the two test made for ILIDS-VID at maximum sequence length of 128.

It is interesting to see if it is possible to get a better accuracy by using a more advanced RNN layers such as a GRU or LSTM. A test is therefore made with a GRU in order to see if it is possible to get better results than using the recurrent unit. The results are shown in Table 6.16.

Sequence length	Rank1	Rank5	Rank10
1	36.0 %	68.6 %	81.2 %
2	37.3 %	68.4 %	82.1 %
4	39.0 %	71.5 %	83.0 %
8	42.1 %	75.7 %	84.9 %
16	46.5 %	77.7 %	86.9 %
32	49.8 %	80.6 %	87.9 %
64	50.6 %	82.9 %	88.3 %
128	53.9 %	82.4 %	88.9 %

Table 6.16: Accuracies for different sequence length on PRID2011 with GRU without dropout. Average of 10 results.

The accuracy for a GRU is a lot worse than using a recurrent unit. One possible explanation is the lack of dropout between the hidden states. The `p` parameter is used to add dropout. The description of the parameter from the documentation can be seen below.

> *"For $p > 0$, it becomes Bayesian GRUs [Moon et al., 2015; Gal, 2015]. In this case, please do not dropout on input as BGRUs handle the input with its own dropouts. First, try 0.25 for p as Gal (2016) suggested, presumably, because of summations of two parts in GRUs connections."*[46]

The documentation is followed and the dropout layer between the CNN and the GRU is removed. The `p` parameter is also set to 0.25. A test is made with the same procedure as before and the results can be seen in Table 6.17.

Sequence length	Rank1	Rank5	Rank10
1	34.3 %	65.6 %	77.6 %
2	36.3 %	66.5 %	79.4 %
4	36.8 %	69.1 %	79.9 %
8	37.2 %	68.7 %	79.9 %
16	38.7 %	71.8 %	83.0 %
32	42.4 %	75.1 %	84.4 %
64	45.5 %	75.9 %	85.2 %
128	47.1 %	77.3 %	85.7 %

Table 6.17: Accuracies for different sequence length on PRID2011 with GRU with dropout 0.25. Average of 10 results.

The addition of the dropout between the time steps seem to result in an even worse accuracy. The LSTM is not tested due to bad results for the GRU, as the LSTM is more complex then the GRU. The worse results for the GRU is consistent with the earlier findings for the ResNet-50 structure which suggest that the GRU is unsuited for re-identification when combined with a CNN.

Another area that might be improved is the CNN structure. It is currently very minimal with the 3 convolutional layers and max pooling layers. The ResNet structure is therefore tried with 20 layers. The implementation is made by adapting an existing implementation from [47]. Batch normalization do however not work for a batch size of 1. Batch normalization is therefore removed to see the affect on the NN.

The rank-1 accuracy after 500 epochs reached 48 % on the training data and a rank-1 accuracy on the testing data of 14 % using a sequence length of 16. The amount of epochs was therefore increased to 1000 instead of 500 epochs. Using 1000 epochs results in the rank-1 accuracy reaching 96 % and a rank-5 accuracy of 99 % on the training data. The results of different sequences tested once is shown in Table 6.18.

Sequence length	Rank1	Rank5	Rank10
1	10 %	26 %	39 %
2	10 %	28 %	46 %
4	12 %	34 %	46 %
8	13 %	37 %	49 %
16	14 %	40 %	55 %
32	17 %	46 %	62 %
64	19 %	51 %	64 %
128	23 %	53 %	64 %

Table 6.18: Accuracies for different sequence length on ILIDS-VID after 1000 epoch(tested once).

The test accuracy is quite low, which could be caused by the lack of batch normalization. This causes the test accuracy to increase a lot slower then the training accuracy shown in Table 6.19, where tests are made after 200 epocs.

Epoch	Test Rank1	Training Rank1
200	9 %	17 %
400	13 %	39 %
600	11 %	70 %
800	13 %	87 %
1000	14 %	96 %

Table 6.19: Accuracies for test and training after each 200 epoch with a sequence length of 16 for ILIDS-VID.

6.9 Lower Body Only

This section describes the test when using the lower body only, on sequences of images.

On the PRID2011 dataset two different methods are tested, the first being a NN consisting of the ResNet-50 structure combined with a GRU. The other tested method is the optical flow method which combines gait features. The results of the optical flow method with 5 frames can be seen in Table 6.20.

Epoch	With Normalization	Without Normalization
10	24.4 %	22.2 %
30	25.3 %	23 %
50	25.9 %	22.6 %

Table 6.20: Rank-1 accuracy on PRID2011 using lower body only, with optical flow method.

A test is made with the existing solution from [17] by cropping the images from the top by 50 % like the method used for the other tests but with the open source code for the siamese recurrent NN. The average of 10 results on the PRID2011 dataset can be seen in Table 6.21.

Sequence length	Rank1	Rank5	Rank10
1	21.0 %	46.0 %	62.3 %
2	27.4 %	54.0 %	69.8 %
4	33.2 %	63.4 %	75.8 %
8	39.6 %	69.3 %	81.1 %
16	47.5 %	75.5 %	85.2 %
32	49.8 %	78.8 %	86.8 %
64	53.0 %	81.4 %	89.3 %
128	57.3 %	84.0 %	90.0 %

Table 6.21: Rank-1 accuracy for different sequence length. Average of 10 results.

A comparison of the loss in accuracy of lower body only compared to full body at different sequence lengths can be seen in Table 6.22.

Sequence length	Lower body only	Full body	Lower body only % of Full body
1	21.0 %	44.1 %	47.6 %
2	27.4 %	49.1 %	55.8 %
4	33.2 %	52.9 %	62.8 %
8	39.6 %	55.9 %	70.8 %
16	47.5 %	61.4 %	77.4 %
32	49.8 %	64.9 %	76.7 %
64	53.0 %	68.7 %	77.1 %
128	57.3 %	70.5 %	81.3 %

Table 6.22: Lower body only vs full body on PRID2011.

A test with only the upper body is made on the PRID2011 dataset which can be seen in Table 6.23.

Sequence length	Lower body only	Full body	Lower body only % of Full body
1	24.8 %	51.4 %	64.5 %
2	28.6 %	58.4 %	70.0 %
4	31.8 %	60.8 %	73.1 %
8	33.5 %	64.4 %	78.2 %
16	38.4 %	70.8 %	82.9 %
32	43.1 %	74.3 %	84.8 %
64	45.2 %	78.4 %	87.6 %
128	47.6 %	80.4 %	89.4 %

Table 6.23: Upper body only on PRID2011.

The accuracies when using only the lower body on the ILIDS-VID dataset is shown in Table 6.24.

Sequence length	Rank1	Rank5	Rank10
1	4.5 %	14.7 %	22.6 %
2	7.5 %	20.6 %	31.9 %
4	8.4 %	23.3 %	33.7 %
8	13.9 %	36.3 %	50.5 %
16	23.5 %	50.2 %	65.1 %
32	29.9 %	58.8 %	72.8 %
64	36.4 %	68.6 %	80.8 %
128	39.0 %	71.0 %	83.1 %

Table 6.24: Accuracies for different sequence length on ILIDS-VID. Average of 10 results.

The accuracy of lower body only compared to full body at different sequence lengths on ILIDS-VID is shown in Table 6.25.

Sequence length	Lower body only	Full body	Lower body only % of full body
1	4.5 %	10.4 %	43.3 %
2	7.5 %	16.7 %	44.9 %
4	8.4 %	21.6 %	38.9 %
8	13.9 %	25.2 %	55.2 %
16	23.5 %	36.7 %	64.0 %
32	29.9 %	45.3 %	66.0 %
64	36.4 %	51.5 %	70.7 %
128	39.0 %	53.2 %	73.3 %

Table 6.25: Comparison of accuracies for different sequence length on ILIDS-VID. Average of 10 results.

An upper body only version is also tested on the ILIDS-VID dataset. The results can be seen in Table 6.26.

Sequence length	Rank1	Rank5	Rank10
1	5.3 %	16.1 %	25.1 %
2	10.7 %	33.4 %	46.3 %
4	15.8 %	43.2 %	57.7 %
8	20.8 %	49.6 %	64.5 %
16	28.6 %	58.7 %	72.7 %
32	36.8 %	67.2 %	80.2 %
64	42.2 %	75.5 %	85.0 %
128	45.5 %	76.7 %	86.6 %

Table 6.26: Upper body only on ILIDS-VID. Average of 10 results.

The ResNet-50 network using average pooling on the MARS dataset delivers the best results. Therefore a test using this method is also performed when using only the lower body, which can be seen in Table 6.27.

Epoch	Rank1	Rank5	Rank10
17	65.35 %	100 %	100 %
30	67.83 %	82.42 %	86.77 %
50	68.69 %	100 %	100 %

Table 6.27: Lower body only on MARS using ResNet-50 with average pooling.

A comparison of the accuracies when using the full body and lower body only can be seen in Table 6.28. The SI+5OF is the single images combined with 5 optical flow frames. The SRNN method is the original code for the siamese recurrent NN from [17].

Method	Dataset	Lower body only		Full body		Lower body only % of Full body	
Normalization		No	Yes	No	Yes	No	Yes
SI+5OF	PRID2011	22.6 %	25.9 %	31.7 %	36.1 %	71.3 %	71.7 %
SRNN	PRID2011	57.3 %	-	70.5 %	-	81.3 %	-
SRNN	ILIDS-VID	39.0 %	-	53.2 %	-	73.3 %	-
ResNet-50 average	MARS	-	68.69	-	80.25	-	85.6 %

Table 6.28: Comparison of lower body only vs full body rank-1 accuracy for sequence methods.

6.10 Summary

Accuracy close to state of the art is reached using different methods for the different datasets. The best performing method on the MARS dataset is using the ResNet-50 structure where all feature vectors for each frame in the tracklet are average pooled. Where all attempts to use the ResNet-50 structure together with a LSTM and GRU fails, as they did not reach as high accuracy.

For both the PRID2011 and ILIDS-VID dataset, the recurrent implementation described in Section 6.8 *Recurrent Unit* delivers the best performance which is close to state of the art. This implementation uses optical flow images together with normal RGB images to extract the feature vectors, meaning this method utilizes the temporal information compared to the ResNet-50 solution. The method using optical flow furthermore uses a siamese structure, which does not give good results with the ResNet-50 solution. As there were problems implementing a siamese structure in Keras.

It is further found that online flipping of images as data augmentation gives the best results when using the ResNet-50 structure, batch normalization should further be set to mode 0 for the best results.

The tests of using only the lower body shows that the rank-1 accuracy only drops a maximum of 30 % compared to the full body accuracy, when utilizing the temporal information. This is a considerable lower loss of accuracy compared to using still images, where the loss of rank-1 accuracy compared to the full body accuracy is around 50 %.

7 Combining Images and Video Method

In this chapter it will be attempted to fuse the two top performing methods from the PRID2011 dataset and the MARS dataset.

7.1 Combination of CNN Structures

The first attempt to combine the two structures, is to use the two CNN structures and combine the output of these using a fully connected layer. This means that only one of the branches from the siamese structure, used on the PRID2011 dataset, is used in the fusion with the ResNet-50 structure. The fully connected layer that fuses the features from the two CNN structures contains 2048 units. A simplified version of the structure can be seen in Figure 7.1.

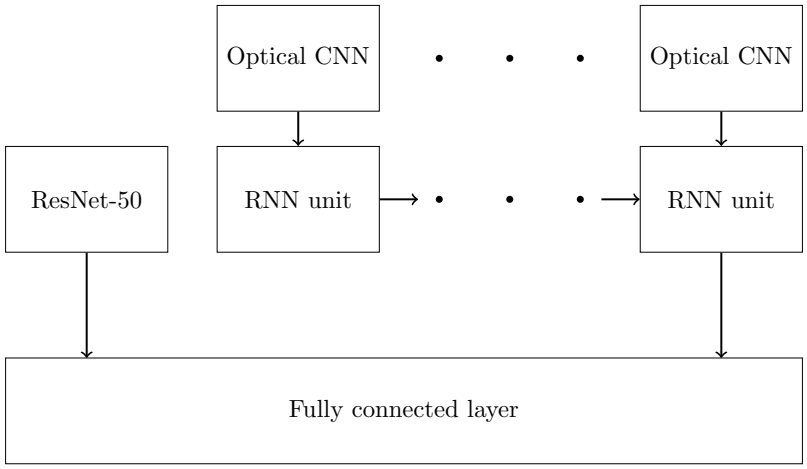


Figure 7.1: Illustration of the tested fused structure.

This structure is used as it is assumed that the ResNet-50 part will contribute with good texture and colour features, and the **Optical CNN** will contribute with temporal features. The **Optical CNN** blocks represent the CNN structure from the recurrent unit solution.

After the ResNet-50 structure a dropout layer is used with a dropout rate at 0.5, for the recurrent unit a dropout rate of 0.6 is used for both the input and the previous hidden state. Furthermore online flipping is used as data augmentation. The ResNet-50 structure only sees the first image of the sequence shown to the RNN which is trained using a sequence length of 16. The fused network is trained for a classification problem.

The results of the tested network can be seen in Table 7.1.

Epoch	With Normalization	Without Normalization	Training accuracy
8	8.82 %	7.49 %	98.35 %
15	9.32 %	7.62 %	98.39 %
25	10.63 %	8.92 %	99.24 %
36	11.84 %	9.6 %	99.73 %

Table 7.1: Results from the fused structure on the MARS dataset.

The low accuracy compared to the test accuracy could indicate that the network does not create features that are general enough for separating the test subjects. The test accuracy does however increase

faster then the training accuracy through time, indicating that the networks might start to generalize more. A solution might be to reduce the size of the fully connected layer merging the two structures, which could force the network to make a more general description of the persons. Another thing that might influence the result is that the last layer from the ResNet-50 structure uses a ReLU activation function, where a tanh activation function is used for the recurrent network. Furthermore the ResNet-50 structure delivers 2048 output values compared to the 128 from the recurrent unit.

Another solution that might better suited for re-identification is the siamese structure.

7.2 Siamese fusion

The best performing NN for PRID2011 and ILIDS-VID used a siamese structure instead of only classification. It is therefore interesting to see if it makes a difference in the fusion if a siamese structure is used for the recurrent network structure.

It is chosen to try to implement the structure in Keras, as the original source code for this network have some challenges. One challenge is for example that the original source code loads all the images into memory which is not possible when using the MARS dataset.

Implementing the recurrent network structure in Keras does however also presents some challenges, such as implementing the average pooling after the recurrent unit. A positive thing by choosing to implement the fusion in Keras is that the ResNet-50 structure is implemented and have pre trained weights. Furthermore the way of loading the MARS dataset is already implemented, the creation of siamese examples does however have to be changed to be more similar to the one implemented in the original source code.

The original source code creates the siamese examples by first picking a similar example from a randomly shuffled list and then pick two other persons at random. This was recreated by making a `Example iterator` that receives the `Persons` class from the dataset loader then produces its own examples. This skips the `exampleCreator` step and therefore uses less memory. Furthermore it is simpler to implement. The `batchIterator` uses a method similar to the siamese example creator from the original source code. It furthermore includes the cameraid and trackid in the list of person index. This is necessary as the MARS dataset have more than two cameras and several tracks. The method for finding a dissimilar pair is the same as from the original source code.

Two implementation are made to create the siamese network. The first model reused the optical NN from Section 7.1 *Combination of CNN Structures* and wraps the model. The model is called twice with two different inputs, to create the siamese structure. There are however problems getting a lower loss. Together with previous problems with this way of creating a siamese structure is the reason for creating a new implementation of the siamese structure. The new implementation made the siamese network by letting each layer take two inputs. Part of the implementation can be seen Code 7.1.

Code 7.1 Layer wise siamese implementation.

```
zero1 = ZeroPadding2D((4, 4))
conv1 = Convolution2D(16, 5, 5, subsample=(1, 1), name='conv1')
max1 = MaxPooling2D((2, 2), strides=(2, 2))
...
for input_img in (input_img1, input_img2):
  x = TimeDistributed(zero1)(input_img)
  x = TimeDistributed(conv1)(x)
  x = Activation('tanh')(x)
  x = TimeDistributed(max1)(x)
```

The loss of both implementations do however not get reduced after one epoch. An example of the loss from the new siamese implementation can be seen in Figure 7.2.

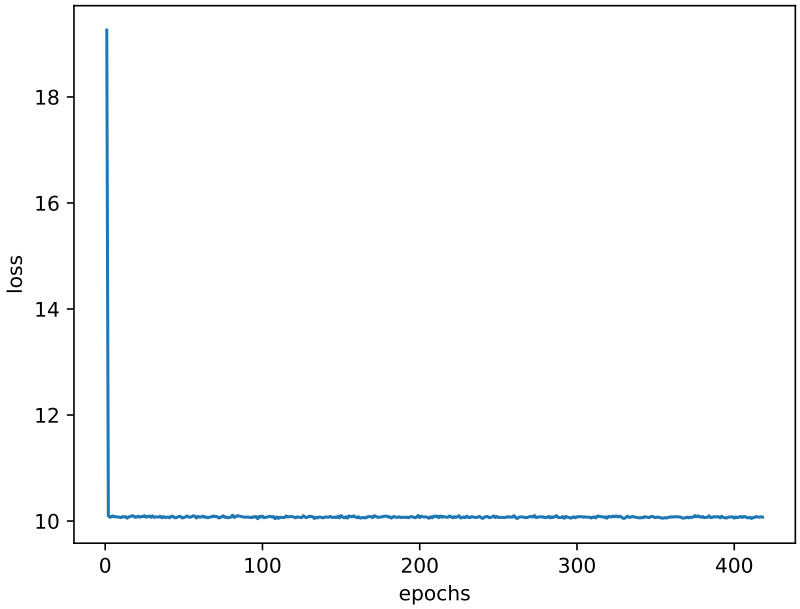


Figure 7.2: Loss for layer wise siamese implementation.

As there are many problems creating the siamese structure, it might be possible to implement the fusion into the original source code. This would however require some large changes in the data structure, as the dataset should be loaded from the hard drive on the fly. Furthermore the way of creating examples should also be changed to consider more then two camera views, as the MARS dataset consist of 6 different camera views.

8 Discussion

The single image results on the Market-1501 dataset do not reach state of the art performance, it only reached a rank-1 accuracy of 76.07 % compared to state of the art of 91.75 %. It should however be noted that the state of the art results have not been published at the time of writing[48] and is only accessible on arXiv.org[4]. The single image accuracy is however deemed high enough that a test using only the lower body is made. The lower body tests show that removing half of the image results in approximately half of the original accuracy.

Testing single images, with only the lower body, extracted from the video datasets MARS and PRID2011 shows an even larger drop in rank-1 accuracy, ranging from approximately 30 % to 50 % of the full body accuracy. The results does without normalization however range from approximately 43 % to 50 %, which shows that when only half of the image is used only half the accuracy or less is reached.

The upper body test on the Market-1501 dataset shows a slightly better performance than lower body. The upper body results reaches 55 % of the full body accuracy. This is however not consistent and tests on the PRID2011 dataset with the upper body only which results in a worse performance than lower body only.

The paper [13] reports the lower body to be the least discriminative, and the upper body to be a lot more discriminative compared to the lower body. It is therefore interesting to see that the upper body does not have a much higher accuracy than the lower body. One possible reason for this could be that the article splits the body into five parts instead of only lower and upper body. Another reason might be that the NNs used in this project are better at finding good features for the lower body.

The video based results for MARS got very close to state of the art results with 80.25 % compared to 81.21 % from [4]. The method uses ResNet-50 for single images and uses the video sequences by making a mean of the feature vectors produced for each frame in the tracklets. The lower body only results in an accuracy that is 85.6 % of the full body accuracy. This is quite a big improvement from the single images solutions.

The state of the art method uses quite similar NN since it also uses ResNet-50 but with a triplet structure instead but without the mean of the feature vectors. It could be interesting see what the accuracy of the state of art method will be using the mean of the feature vectors of each frame in the tracklet. It would furthermore be interesting to see if another more complex NN structure could improve the accuracy. This could be a larger version of ResNet or the GoogLeNet.

One possible source of inaccuracy for some tests might be the testing of the test dataset accuracy for different epochs. This might cause the accuracy to be specific the test dataset and not generalize. One solution could be to only test once after a specific epoch. Another possibility could be to use a validation set. The disadvantage of using a validation set is that fewer example of people can be used for training and testing. One possible to solution could be to use cross validation.

The solution does however not utilize the temporal information between frames. Attempts to utilize temporal information using a GRU or LSTM do however deliver worse results and are problematic to implement in Keras. One of the main problems using the ResNet-50 structure in Keras with RNNs is implementing batch normalization together with time distribution. These two combined create some problems where it is not possible to use per batch normalization, and the implementation of moving average normalization is not able to handle sharing its weights.

The siamese recurrent NN from [17] have the possibility of utilizing the temporal information by using optical flow and a recurrent unit. The code from the paper is used to recreate the original results for the PRID2011 dataset but something caused the ILIDS-VID results to lack 4.8 % rank-1 accuracy. Both are however reasonably within the range of state of the art which makes it interesting to see how the temporal

solution works for the lower body.

The percentage of the original accuracy when using only the lower body compared to the full body is 81.3 % for PRID2011 and 73.3 % for ILIDS-VID. The upper body only accuracy is again not consistently better or worse. The accuracy of the upper body only on PRID2011 is again worse then lower body only with a rank-1 accuracy of 47.6 % compared to 57.3 % for the lower body.

The siamese recurrent NN also uses the mean of features vectors produces by multiple frames in the tracklet. The higher accuracy of the lower body only compared to the full body might come from the mean of these feature vectors. This is however not necessarily the case when considering the test of the ResNet50 combined with the optical flow frames on PRID2011. It does not use the mean of feature vectors but still results in 71.7 % of the full body accuracy when using the lower body only.

Another attempt is made to improve the performance by using a GRU instead of the recurrent unit. This time using the original code from the siamese recurrent NN. The results using a GRU again delivers worse accuracy. This could indicate that a GRU is unsuited for the task of re-identification or that the datasets are to small for using such a complex unit compared to the simpler recurrent unit.

The fusion of the single images and sequences newer reached a accuracy that is worth testing with lower body only. Some of the methods for sequences are however based on the the single image solution, therefore it might not be that interesting to fuse these with the single image solution.

All the results shows that a lower body only solution should use temporal information in order to avoid losing half or more of the full body accuracy. The temporal information might simply be average of features from multiple frames but more inter-frame temporal features can also be used. The accuracies will in these cases be in the range of approximately 70 % to 85 % of the full body accuracy.

This drop in accuracy is not negligible but the solution makes it possible to track people that changes cloth like hats, jackets and so forth. Another possibility with the lower body only solution is the ability to enhance the privacy. It is possible to place cameras such that only the lower body is recorded which makes it more difficult to find the identity of a person but can still be used for re-identification.

There, to the best of our knowledge, is no research within the area of re-identification using only the lower body. The findings in this report can therefore build the base for further research within this area.

There is still some aspects that needs to be investigated before a lower body only or even re-identification can be used in automatic systems. More research is also need in order to get a higher accuracy.

It is however possible that the system can be usable for queue estimation since it does not require everybody to be re-identified. This will though require the system to know how certain a match is. This will require an investigation into the possibility of discarding false positive matches. It might for example be possible to sort away false positive matches using a maximum threshold for the euclidean distance.

Physical settings might also make it possible to make some constraints for the matching. It might be necessary for people to pass one camera before arriving at another, or the distance between two cameras might be to large for the person to travel that distance in a given time. These physical constraints might help increase the accuracy of the re-identification.

9 Conclusion

The state of the art results are not reached with the single images solution. The rank-1 accuracy for the single images is 76.07 % whereas the state of the art is 91.75 %. The accuracy for the lower body is 49.1 % of the full body accuracy. The solution is trained and tested on the video datasets MARS and PRID2011 using single images extracted from the sequences. The test of the lower body only results in a accuracy between 32.89 % to 50.2 % of the full body accuracy. The results for the upper body only is either worse or better than lower body only depending on the dataset.

The state of the art results are almost reached with the video based solution. The rank-1 accuracy for the MARS dataset is 80.25 % whereas the state of the art is 81.21 %. The lower body accuracy is 85.6 % of the full body. The datasets PRID2011 and ILIDS-VID are tested using a siamese recurrent NN. The lower body accuracy of these compared to the full body are within the range of 73.3 % to 81.3 %. A NN using only optical flow for temporal information gets a lower body only accuracy of 71.3 % compared to the full body. The results for the upper body only is again either worse or better than lower body only depending on the dataset.

The state of the art results are not reached with a fusion of the single images and video based solution. The fused network have a too low accuracy to be tested on lower body only.

The video based solution performing best on the MARS dataset is based on the single image solution, and is therefore not fused with the single image solution.

The results show that temporal information can be used to increase the accuracy of a lower body only solution.

Bibliography

[1] E. David. (2016, July) Disney patent would track park goers by their shoes. https://siliconangle.com/blog/2016/07/29/disney-patent-would-track-park-goers-by-their-shoes. SiliconAngle. Last seen May 2017.

[2] Queue measurement. https://blipsystems.com/queue-measurement-3/. Blip Systems. Last seen May 2017.

[3] L. Zheng, Y. Yang, and A. G. Hauptmann, "Person re-identification: Past, present and future," *CoRR*, vol. abs/1610.02984, 2016. [Online]. Available: http://arxiv.org/abs/1610.02984

[4] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," 2017.

[5] H. Liu, J. Feng, M. Qi, J. Jiang, and S. Yan, "End-to-end comparative attention networks for person re-identification," *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 14, 2016.

[6] M. Geng, Y. Wang, T. Xiang, and Y. Tian, "Deep transfer learning for person re-identification," 2016.

[7] R. R. Varior, M. Haloi, and G. Wang, "Gated siamese convolutional neural network architecture for human re-identification," *Proceedings of Computer Vision - ECCV 2016*, pp. 791–808, 2016.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[10] L. Wu, C. Shen, and A. van den Hengel, "Personnet: Person re-identification with deep convolutional neural networks," *CoRR*, vol. abs/1601.07255, 2016. [Online]. Available: http://arxiv.org/abs/1601.07255

[11] N. McLaughlin, J. M. del Rincon, and P. Miller, "Person re-identification using deep convnets with multi-task learning," *IEEE Transactions on Circuits and Systems for Video Technology*, 2016.

[12] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Deep metric learning for person re-identification," in *ICPR*, 2014.

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

[14] 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*, June 2014, pp. 152–159.

[15] D. Chen, Z. Yuan, B. Chen, and N. Zheng, "Similarity learning with spatial constraints for person re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1268–1277.

[16] L. Wu, C. Shen, and A. van den Hengel, "Deep recurrent convolutional networks for video-based person re-identification: An end-to-end approach," *CoRR*, vol. abs/1606.01609, 2016. [Online]. Available: http://arxiv.org/abs/1606.01609

[17] N. McLaughlin, J. Martinez del Rincon, and P. Miller, "Recurrent convolutional network for video-based person re-identification," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[18] T. Matsukawa and E. Suzuki, "Person re-identification using cnn features learned from combination of attributes," *CVPR2016*, pp. 1363–1372, 2016.

[19] R. R. Varior, B. Shuai, J. Lu, D. Xu, and G. Wang, *A Siamese Long Short-Term Memory Architecture for Human Re-identification.* Cham: Springer International Publishing, 2016, pp. 135–153. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46478-7_9

[20] J. van de Weijer, C. Schmid, J. Verbeek, and D. Larlus, "Learning color names for real-world applications," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[21] Y. Yan, B. Ni, Z. Song, C. Ma, Y. Yan, and X. Yang, *Person Re-identification via Recurrent Feature Aggregation.* Cham: Springer International Publishing, 2016, pp. 701–716. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46466-4_42

[22] B. Prosser, W.-S. Zheng, S. Gong, and T. Xiang, "Person re-identification by support vector ranking," in *Proceedings of the British Machine Vision Conference.* BMVA Press, 2010, pp. 21.1–21.11, doi:10.5244/C.24.21.

[23] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang, "Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification," *CoRR*, vol. abs/1508.04535, 2015. [Online]. Available: http://arxiv.org/abs/1508.04535

[24] F. M. Castro, M. J. Marin-Jimenez, N. Guil, and N. P. de la Blanca, "Automatic learning of gait signatures for people identification," 2016.

[25] Z. Liu, Z. Zhang, Q. Wu, and Y. Wang, *Enhancing Person Re-identification by Integrating Gait Biometric.* Cham: Springer International Publishing, 2015, pp. 35–45. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16628-5_3

[26] Fully connected layer. https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/fc_layer.html. Last seen April 2017.

[27] A. Deshpande. (2016, July) A beginner's guide to understanding convolutional neural networks. https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/. Last seen April 2017.

[28] (2015, August) Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Last seen April 2017.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[30] K. Yao, T. Cohn, K. Vylomova, K. Duh, and C. Dyer, "Depth-gated recurrent neural networks," *arXiv:1508.03790v2*, 2015.

[31] M. Mazur. A step by step backpropagation example. https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/. Last seen April 2017.

[32] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145 – 151, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608098001166

[33] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv:1502.03167v3*, 2015.

[34] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[35] Tensorflow. https://www.tensorflow.org/. Last seen May 2017.

[36] (2017, Jan) Keras examples. https://github.com/fchollet/keras/tree/1.2.1/examples. Last seen May 2017.

[37] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," *Proceedings of Computer Vision and Pattern Recognition 2006*, 2006.

[38] L. Zheng, "Code for ide baseline on market-1501," Mar 2017. [Online]. Available: https://github.com/zhunzhong07/IDE-baseline-Market-1501

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[40] Fchollet, "fchollet/deep-learning-models," Mar 2017. [Online]. Available: https://github.com/fchollet/deep-learning-models

[41] MathWorks. Improve neural network generalization and avoid overfitting. https://se.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html.

[42] W. Su, Y. Yuan, and M. Zhu, "A relationship between the average precision and the area under the roc curve," in *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, ser. ICTIR '15. New York, NY, USA: ACM, 2015, pp. 349–352. [Online]. Available: http://doi.acm.org/10.1145/2808194.2809481

[43] A. Møgelmose, C. Bahnsen, T. B. Moeslund, A. Clapes, and S. Escalera, "Tri-modal person re-identification with rgb, depth and thermal features," in *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2013, pp. 301–307.

[44] L. Zheng, Z. Bie, Y. Sun, J. Wang, C. Su, S. Wang, and Q. Tian, *MARS: A Video Benchmark for Large-Scale Person Re-Identification.* Cham: Springer International Publishing, 2016, pp. 868–884. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46466-4_52

[45] Keras. Error when using batchnormalization in timedistributed. https://github.com/fchollet/keras/issues/2827.

[46] (2017, January) Gru. https://github.com/Element-Research/rnn/blob/master/GRU.lua. Last seen May 2017.

[47] (2017, February) Resnet. https://github.com/facebook/fb.resnet.torch. Last seen May 2017.

[48] (2017, May) Triplet-based person re-identification. https://github.com/VisualComputingInstitute/triplet-reid. Last seen May 2017.