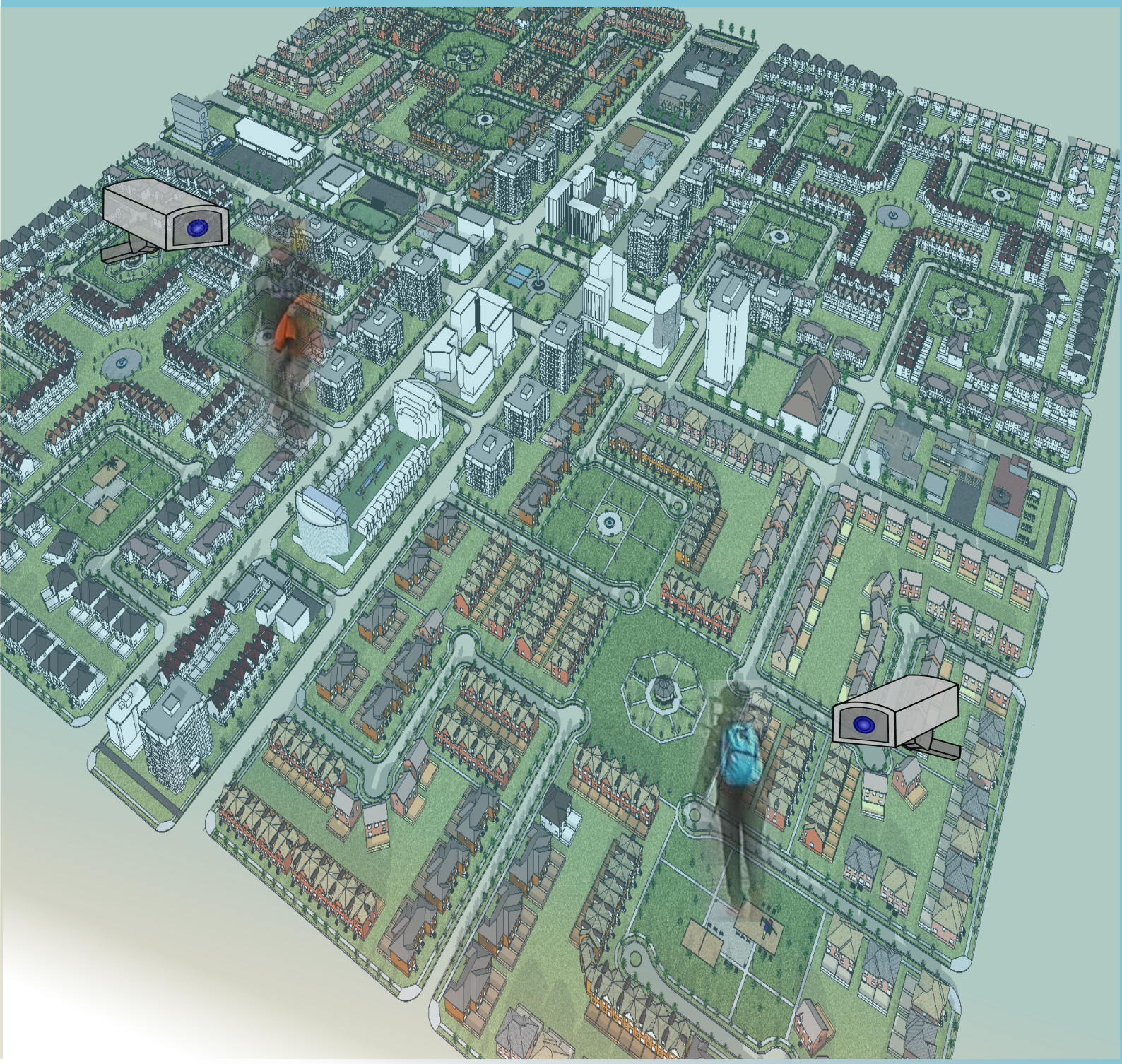
Enhancing Person Re-identification by Late Fusion

Aske R. Lejbølle



AALBORG UNIVERSITY
DENMARK

Master's Thesis
VGIS10
16gr1041
Aalborg University
June 2016

Copyright © Aalborg University 2016

Vision Graphics and Interactive Systems
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Enhancing Person Re-Identification by Late Fusion

Theme:
Master's Thesis: Vision, Graphics and Interactive Systems

Project Period:
Autumn 2015/Spring 2016 (Long Thesis)

Project Group:
15gr941/16gr1041

Participant(s):
Aske Rasch Lejbølle

Supervisor(s):
Kamal Nasrollahi

Copies: 3

Page Numbers: 101

Date of Completion:
June 2, 2016

Abstract:

Person re-identification is a problem of matching persons across several camera views. Due to challenges of view, scale and change in lighting conditions, this is a field still in research. Usually, proposed systems deals with creating a robust feature description from color and texture features and utilize a metric learning algorithm to improve performance. Due to superior results by late fusion in biometrics, this thesis proposes a system utilizing late fusion to fuse systems that well complement each other. This include high-level features from a CNN, a system extracting mid-level features using dictionary learning and a system extracting low-level features from local patches. In the fusing scheme, both rank aggregation and score-level fusing is applied. The proposed system is evaluated on VIPeR, PRID450, CUHK01 and CUHK03, extensively used in evaluation. Compared to previous state-of-the-art systems, the proposed system improves the rank-1 accuracy by 15.04% and 10.94% on CUHK01 and PRID450S, respectively. For CUHK03, the system performs similar to a state-of-the-art CNN using multiple training images. To reflect real-life situations, multi-shot and cross-dataset tests are conducted. In the multi-shot test, the proposed system increases the accuracy by 3.2% compared to the individual systems. The system achieves state-of-the-art results in the cross-dataset test, with an increased rank-1 accuracy of 5.85% compared to previous proposals. Late fusing only increases the total processing time by a maximum of 2.1% making it suitable for this task.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Contents

Preface

This Master's Thesis is the product of research in the topic of Person Re-identification, carried out in the period September 2015 to June 2016 at Aalborg University. It is written as a final part of the master's programme Vision, Graphics and Interactive Systems. The thesis covers the areas of soft biometrics, neural networks and other machine learning techniques and is targeted people with interest within Computer Vision and Machine Learning.

The first chapter covers the statement of the problems given the underlying challenges and the proposed problem statement to solve this. The second chapter provides an analysis for handling this problem and how the proposed system is evaluated. The third chapter describes in detail the individual components of the system and how they are combined. This furthermore ends up in evaluations on extensively used datasets. Fourth chapter covers the acceptance test of the proposed system by evaluating on real-life situations. Finally, the fifth chapter includes a discussion of potential changes in order to further improve the system which eventually ends up in an overall conclusion of the work. To get the best understanding of the techniques and results throughout the report, it is recommended to print in colors.

I would like to thank Kamal Nasrollahi for supervising throughout, bringing the underlying idea for the thesis.

Aske R. Lejbølle

Aalborg University, June 2, 2016

1. Introduction

Person re-identification (re-id) is a field in which much research is still done. It is a challenge of comparing images of persons extracted from video sequences captured from different camera views. The output is then either binary, whether two persons match or not, or a list of persons ranked by similarity. The purpose of doing person re-id can either be in a forensic matter where criminals need to be identified or if a certain person, or a group of persons, needs to be tracked across cameras in order to determine a path, for example in order to estimate the behavioural flow in an airport.

Challenges

Having a large network of cameras, video of the same person is captured in different settings with different views. This introduces challenges which have to be taken into account when doing re-id. The first challenges are about view and scale of a person which may change from camera to camera. In the first camera view, the person might be seen from the side walking close to the camera, while another only shows the back of the person far away. An example of a person captured at different scales can be seen in Figure 1.1 and Figure 1.2. This will cause texture on the clothing of the person to be more difficult to capture or even disappear.

Figure 1.1: Example of low scale image of a woman. The image is cropped to only contain the woman (Li et al., 2014).

Figure 1.2: The same woman in a different view shot at a larger scale (Li et al., 2014).

Another challenge regards the lighting conditions which changes depending on different variables. First to take into account is the cameras being used which may perceive color differently depending on the settings. Furthermore, a highly affective variable is the daylight which affects the perceived color of the clothing. Lastly, the placement of the camera is important, as placement in more obscure areas will cause lower color intensities. An example of a person captured at two different views and with different lighting conditions is shown in Figure 1.3 and Figure 1.4. Different lighting conditions might also lead to cluttering where parts of the person will have similar color as the background or other objects in the scene.

Figure 1.3: Example of back view shot of a man at good lighting conditions (Li et al., 2014).

Figure 1.4: The same man captured from a different view in bad lighting conditions (Li et al., 2014).

Lastly, a major challenge is occlusion in which the person is partly or entirely hidden behind another person or other objects in the scene. This will make it almost impossible to retrieve information about the particular person depending on the degree of occlusion. This will also be dependent on the placement of the camera according to the path on which people are walking. An example of an almost total occlusion is shown in Figure 1.5.

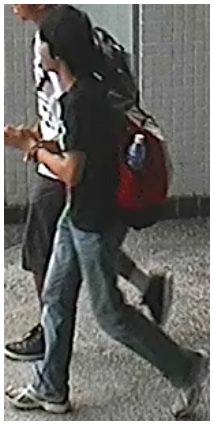


Figure 1.5: Example of a person almost occluded behind another person as they are both seen from the side (Li et al., 2014).

Pipeline

The task of re-id can be depicted as a pipeline as shown in Figure 1.6 in which each of the steps are important for the overall performance and accuracy.

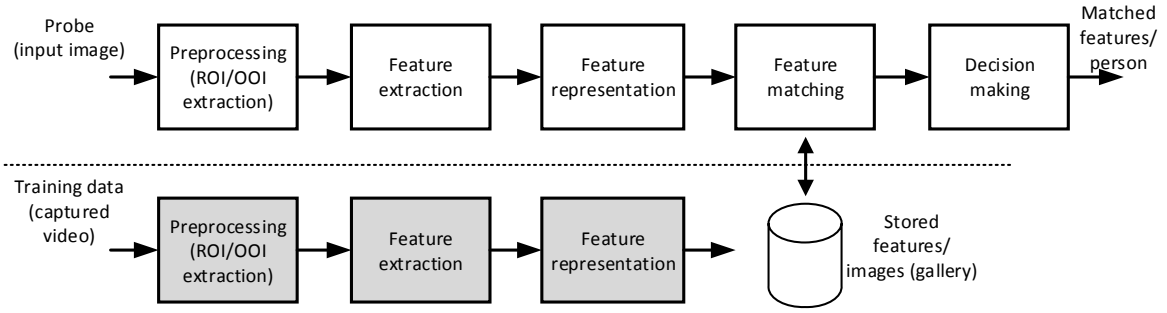


Figure 1.6: Pipeline of the re-id process which shows both the process of storing data from video (below dotted line) and the actual re-id (above dotted line).

The pipeline can be divided into two parts that run independently. The part above the dotted line is the main program in which a user can select and prompt the system with an image of a person, also known as a *probe*, whom matches shall be found from the database. The part below the dotted line is an online system which, depending on the requirement, may or may not, constantly process incoming video either on the fly or as video is stored. In this pipeline it is assumed, a tracker has automatically located the persons within the scene.

In both cases, preprocessing is carried out by extracting the region of interest (ROI) around the person or even only the object of interest (OOI) i.e. the person. Furthermore, different color spaces, contrast enhancing or noise removal can be applied to enhance the features extracted later.

Next, features such as edges, lines, colors, or other both global and more local are *extracted*. Due to the different challenges, certain features cannot be utilized alone. Scale and view challenges cause texture features to not work properly as stand-alone. Likewise, illumination challenges cause color features to not work in all cases and color and texture features can therefore be combined to complement each other. Furthermore, the *representation* of the features can be changed to increase robustness, for example by creating a color histogram, representing the color intensities in bins or by creating a covariance descriptor which combines color and texture information.

The extracted features and images are either *stored* in a database, known as the *gallery*, or used for *matching* with features in the gallery using a distance metric such as Euclidean distance. At this step, a metric learning algorithm is often utilized to improve accuracy by using a training set to learn similarities and dissimilarities across camera views. Matching with the gallery, the probe is compared with each image/set of features from each person, also known as a *query*. Having matched features from an image pair, the trained classifier then *decides* whether this pair are from the same person of not based on a probability or on a ranked list of similarity measures between the probe and the gallery.

Normally, a single probe is matched with a single query, assuming each person has at least one image, or feature description, in the database, also known as *single-shot* matching. In a real world scenario, it can not be entirely assured that the person searched for is contained in the database. Assuming this is not the case, several images of the same person will typically be contained in the database extracted from video sequences. Therefore, either one, several or a combination of images of the same person are often compared to several images of each person in the database, also known *multi-shot* matching.

There is a lot of potential in letting a computer doing person re-id. As we live in a world in which terrorism becomes an increasing thread, lately by the bombings in Paris (par, 2015), lots of time and resources are spent tracking down the persons responsible by looking through hours of surveillance videos, resources which could be used elsewhere if the task is automated. But because of the challenges which only become greater when looking at camera networks in big cities, for example Paris, accuracy is not yet high enough to entirely let a computer do the job. One initial solution could be to make the task semi-automatic i.e. having an operator supervise the re-id process and pick right matches from wrong afterwards. In this manner, the process becomes iterative with more information about the person after each iteration.

Of course, automating the process should lead to higher performance compared to letting an operator doing the job. Normally it takes an operator $\frac{1}{8}$ of the length of a video sequence to search through and identify (Systems, 2015). Given that features are extracted and stored from newly recorded video on the fly, an automated system should be able to run the top part of the re-id pipeline faster than the stated time for an operator.

1.1 Recent work

Lots of different systems have been proposed for person re-id through the last years (Gong et al., 2014). This includes both supervised and unsupervised systems with the supervised having higher accuracies. Lots of different features types and feature descriptors have been proposed including color histograms, shape using histogram of oriented gradients (HOG) (Dalal and Triggs, 2005) or texture using Gabor filters (Fogel and Sagi, 1989), local binary patterns (LBP) (Ojala et al., 1994), scale invariant feature transform (SIFT) (Liao et al., 2010) or region covariances (Tuzel et al., 2006). Furthermore, local patch features have been proposed to get more discriminative and salient features.

Some of the previously state-of-the-art systems primarily utilize color descriptors as in (Koestinger et al., 2012) where HSV and Lab histograms from overlapping blocks are used together with LBP features to compare images using a Keep It Simple and Straightforward MEtric learning (KISSME) algorithm based on Mahalanobis distance. In (Farenzena et al., 2010), color histograms are also being utilized, this time together with descriptors that capture information about highly recurrent patches in the body and stable color regions. Finally, (Zhao et al., 2013) utilizes color histograms in LAB color space together with SIFT features for local patches to compute a salience probability map of each image of each person. The current state-of-the-art systems also utilize color histograms or color names as in (de Carvalho Prates and Schwartz, 2015) in which color histograms in five different color spaces together with color names are used with KISSME as metric learning. An extended version of KISSME is used in (Liao et al., 2015) where color histogram maximization and texture features are extracted from local patches and used to learn a cross-view projection matrix which is used together with KISSME.

In recent years, neural networks have begun to also be used in person re-id with some of them showing state-of-the-art results. First in 2012 (Yi et al., 2014), a convolution neural network (CNN) was used match images using a siamese architecture together with a cosine connection function and a binomial deviance cost function making it able to learn to differences across camera views based on two inputs. A couple years later, another CNN was proposed (Li et al., 2014), also partly utilizing the siamese structure. Though, this time, patches from the output of two sub-CNN's are matched across views through multiplication. This time, a softmax function is used to train the network by calculating the probability of an image pair being similar.

More recently, in 2015, a CNN was presented which showed the largest accuracies on large datasets (Ahmed et al., 2015). This CNN is an improvement of the one presented the year before and therefore

has many of the same properties. Difference here lies in the way of matching patches which is done using differencing rather than multiplication. Furthermore, an architecture has been proposed combining CNN features with hand-crafted color and texture features. (Wu et al., 2016b) proposes a network which fine-tunes a pre-trained CNN based on the different feature types and use the resulting new feature type with a cross-view Kernel Marginal Fisher Analysis (KMFA) as metric learning to evaluate a test set.

1.2 Problem Statement

Even though, lots of system have been proposed for person re-id, non of them have shown the performance and accuracy it takes to be used in practice. Every year, new systems are proposed, some of them increasing the accuracy by a few percentages. But as more and more cameras are being used to monitor civilization in different environments, more resources are being used to handle the increasing amount of data. Therefore, other ways to improve the accuracy need to thought of in order to have a well working automated system.

In order to make such an improved system, a technique which combines the strengths of different features, either on an early or late state. This is more generally known as *fusion*. Only few presented systems have made use of fusion techniques, most recently in 2015, where the outputs of comparing images using color features including histograms in HSV colorspace and color names (CN) are combined with the outputs of comparing images using LBP and HOG features (Zheng et al., 2015b). (de Carvalho Prates and Schwartz, 2015) also makes use of fusion by aggregating results from different feature types. Common for both systems is that the fusion done on score-level but by independent features.

Instead, the fusion can also be done on matching level, by combining the decisions made by different systems. As CNN's have shown state-of-the-art results in recent years, it is evident to fuse one such with other systems utilizing hand-crafted features. This leads to the following problem statement.

> ***Is it possible to increase accuracy of a person re-identification system by fusing a state-of-the-art CNN with systems utilizing hand-crafted features?***

2. Technical Analysis

This chapter includes a description of techniques to fuse different modalities and how the proposed fusing scheme is evaluated. Further a description of the evaluated datasets are included followed by the evaluation protocols.

2.1 Features

To increase the accuracy of the fused outputs, proper systems that complement each other well have to be chosen. Even though, different color spaces are robust to certain challenges, for examples by converting to a different color space having the intensity mostly represented in a single component, combining systems that utilize same types of features in some of the same color spaces will not improve the fused accuracy by much. Even though, the metric learning algorithm differs, the outputs are likely to be much similar, leaving the fusion without any effect. Furthermore, fusing systems that extract features on the same level of abstraction will contain the same level of detail and hence assume to produce similar outputs. Therefore, systems working with globally extracted features should be fused with systems working with more locally extracted features, preferably utilizing different types of features.

Feature extraction can be categorized to three levels:

- **Low-level features** – extracted on per pixel level. For color, typically histograms are made based on the RGB values of each pixel. To extract texture, descriptors such as Local Binary Patterns (LBP) or Scale-Invariant Feature Transform (SIFT) are often utilized to make a robust representation.
- **Mid-level features** – extracted using a learned mapping from low-level features to a more high-level representation. This include learning new feature representation based on given low-level features.
- **High-level features** – as with mid-level features, high-level features are also extracted from a learned mapping from low-level features. These features represents entire objects. Examples of such features can be extracted from late layers of CNN's.

To have modalities that complement each other well, systems utilizing features on each of the three levels will be included.

2.2 Early and late fusion

In order to enhance the accuracy of a certain application, different types of information can be fused. These types of fusion are divided in three categories[1].

[1] An extensive review of multi-modality fusion in biometrics can be read in (Ross et al., 2006)

- **Data-level fusion** deals with combining data from different sensors (cameras in this case).
- **Early fusion** also more commonly known as *feature-level fusion* deals with combining extracted feature vectors to a single vector in order to decrease noise from the data.
- **Late fusion** deals with combining the output decisions or scores from different classifiers. This would correspond to having different experts evaluating the same input.

Often, data-level fusion is dealt with in the case of multi-shot matching when several images of the same person are available. In this case, different systems can be applied to fuse the images, for example by taking the mean of the pixel values at same spatial location or take the max value across all images.

Early fusion is the most utilized fusion techniques. Typically, different feature types based on different color spaces or texture are extracted and concatenated to a single feature vector. But because of often high correlation between some of the features, dimensionality reduction, such as Principle Component Analysis (PCA) is applied. Another way of combining features was proposed by (Yang et al., 2003). Instead of combining the features in serial, a parallel feature fusion scheme was introduced. From an evaluation on handwritten characters and images of faces, the new parallel technique showed to be better.

Even though, feature-fusion is the most utilized among the three different fusion techniques when it comes to person re-id, late fusion has previously shown to be the best and most applied when it comes to biometric multi-modality fusion (Ross et al., 2006). In addition, a certain level of noise may still be included in the feature descriptors which has to be dealt with when fusing these, making fusion at this level potentially more complex. Due to this, late fusion is employed to fuse outputs from different systems.

Fusing at a late stage comes with a cost, as independent classifiers are used for each system while the two former only make use of a single. Because of this, more memory is needed in order to store features from each individual system while more computational power is also required. This also increases the processing time as fusion on this stage is typically more complex than simply concatenating features and the increased processing time should, hence, be taken into account as well.

2.2.1 Late fusion

Late fusion can be applied in different manners depending on the output from the classifiers and three different ways are often discussed:

- **Rank-level fusion** also called *rank aggregation* is used when ranked lists of matches are output.
- **Score-level fusion** is used when the outputs are matching scores between the probe and the gallery.
- **Decision-level fusion** is used when when the outputs are certain labels assigned given the probe.

Rank-level fusion

In person re-id, a metric learning algorithm is often used to calculate the distance between a probe and the gallery and the output is typically represented as a ranked list in ascending order according to the distance. As the output of each classifier represents rankings for each person, they do not need to be mapped to a common metric making fusion directly applicable.

Different fusion schemes have been proposed, from more simple solutions using *Borda Count* in which the sum of the ranks are simply computed, as shown in Equation 2.1, to more complex solutions using statistics.

$$r_{new,p} = \sum_{i=1}^{K} r_{i,p}, \tag{2.1}$$

where $r_{i,p}$ is the ranking for person p for the i^{th} system using K different systems.

In 2003, (Stuart et al., 2003) presented a rank aggregation technique which makes use of the statistical placement of the ranked persons. First, each rank is converted to a numerical rank matrix with values in the interval [0,1]. This is done by dividing the rank of each id with the number of elements in the list causing lower values to be better. Having normalized the ranks, the new rank is then calculated by looking at the joint cumulative distribution of the rank over all systems following Equation 2.2 which is an optimized way of calculating the aggregated rank, proposed by (Aerts et al., 2006).

$$\begin{aligned} r_{new,p} &= K! \cdot V_{K+1} \\ V_{K+1} &= \sum_{k=1}^{K} \sum_{i=1}^{k} (-1)^{i-1} \frac{V_{k-i}}{i!} r^{i}_{K-k+1,p}, \end{aligned} \tag{2.2}$$

where r^i denotes the i^th system, K is the total number of systems and $V_0 = 1$.

This way of aggregating ranks was tested against other aggregation algorithm, including Max, Mean, and Robust Ranking Aggregation (RRA) (Kolde et al., 2012) and the results showed the Stuart algorithm to be best (de Carvalho Prates and Robson Schwartz, 2015). This algorithm is therefore utilized in the fusing scheme.

Score-level fusion

To do score-level fusion, the output from each classifier needs to be a similarity score between the probe and gallery. Next, the scores need to be normalized in order to have a common metric to remove any bias by having scores containing high values. Furthermore, the importance of a good score either being high or low should be taken into account.

In 1998, (Kittler et al., 1998) presented a framework based on Bayesian theory which stated different ways to combine classifiers. Given the Bayesian decision rule and theorem in Equation 2.3.

$$\begin{aligned} P(\omega_j|x_1,...,x_R) &= \max_k P(\omega_k|x_1,...,x_R) \\ P(\omega_k|x_1,...,x_R) &= \frac{p(x_1,...,x_R|\omega_k)P(\omega_k)}{p(x_1,...,x_R)}, \end{aligned} \tag{2.3}$$

where ω_j is the assigned class based on measurements $x_1,..,x_R$ where R is the number of classifiers and x_i is the measurement from the i'th classifier. From this, different rules of combining classifiers can be created. These rules are summarized in Table 2.1.

Combination Rule	Equation
Product	$P(\omega_j)\prod_{i=1}^{R} p(x_i\|\omega_j) = \max_{k=1}^{m}[P(\omega_k)\prod_{i=1}^{R} p(x_i\|\omega_k)]$
Sum	$(1-R)P(\omega_j) + \sum_{i=1}^{R} p(\omega_j\|x_i) = \max_{k=1}^{m}[(1-R)P(\omega_k) + \sum_{i=1}^{R} p(\omega_k\|x_i)]$
Median	$med_{i=1}^{R} P(\omega_j\|x_i) = \max_{k=1}^{m}[med_{i=1}^{R} P(\omega_k\|x_i)]$
Max	$\max_{i=1}^{R} P(\omega_j\|x_i) = \max_{k=1}^{m}[\max_{i=1}^{R} P(\omega_k\|x_i)]$
Min	$\min_{i=1}^{R} P(\omega_j\|x_i) = \max_{k=1}^{m}[\min_{i=1}^{R} P(\omega_k\|x_i)]$
Majority vote	$\sum_{i=1}^{R} \Delta_{ji} = \max_{k=1}^{m}[\sum_{i=1}^{R} \Delta_{ki}]$, where $\Delta_{ki} = 1$ if class k is chosen, otherwise 0

Table 2.1: Rules of combining classifiers based on Bayesian theory.

As an extension to the Bayesian decision rules, weights can be assigned depending on the performance of each classifier. This was utilized by (Zheng et al., 2015b), who proposed calculating a similarity score for each extracted feature by taking the dot product between two feature vectors as given by Equation 2.4.

$$s_{p,q}^{(i)} = F_p^{(i)} \bullet F_q^{(i)T}, \tag{2.4}$$

where $s_{p,q}^{(i)}$ is the similarity score between feature vector i of the probe image $F_p^{(i)}$ and the transposed feature vector i of the query image $F_q^{(i)T}$. Having calculated similarity scores for all features, the product rule is utilized to calculate the fused similarity score following Equation 2.5.

$$sim(p,q) = \prod_{i=1}^{K} (s_{p,q}^{(i)})^{w_q^{(i)}}, \qquad \sum_{i=1}^{K} w_q^{(i)} = 1 \tag{2.5}$$

where $w_q^{(i)}$ is the weight of feature $F^{(i)}$ for the query image.

In previous work, score-level fusion using the product rule has shown to be superior (Kittler et al., 1998) and is therefore utilized in as a secondary fusing technique in the fusing scheme. Furthermore, weights for each score are added following Equation 2.5, in order to enhance performance as the best performing system might differ depending on the probe.

Decision-level fusion

At decision-level fusion, the output from each classifier is either a class label or a binary label indicating match or not, depending on the application. Some of the most simple techniques in binary classification include the "AND" or "OR" rules that output a matched label if either all agree or if just one of the classifier outputs agree.

Other approaches make use of majority voting by selecting the class at which most classifiers agree. This can be done with or without assigned weights depending on assumption of equality between classifiers. Finally, the Bayesian decision rules stated in Table 2.1 can also be utilized to fuse decisions rather than scores.

As systems in person re-id almost always output a ranked list or a list of similarity scores, decision-level fusion is not utilized. Furthermore, previous evaluation show this type of fusion to be worse than score-level fusion and early fusion (Lip and Ramli, 2012).

Having performed late fusion, tests are conducted by techniques described in the following.

2.3 Evaluation Protocols

In person re-id, different techniques can be used to evaluate a proposed system depending on how the problem is addressed. There are two types evaluation protocols which currently dominates:

- Cumulative Match Characteristic (CMC).
- Mean Average Precision (mAP).

2.3.1 Cumulative Match Characteristic (CMC)

As in image retrieval, person re-id can be addressed as a ranking problem. Utilizing the sorted ranks, different accuracies can be calculated depending on the rank. For example, the rank-1 accuracy is calculated by the number of probes which have their corresponding query in the gallery ranked as most similar. Likewise, the rank-5 accuracy is calculated by taking the five most similar queries into account. Calculating different ranking accuracies, a CMC curve can be drawn as shown in Figure 2.1. When comparing different re-id systems, the rank -1, -5, -10 and -20 accuracies are typically used.

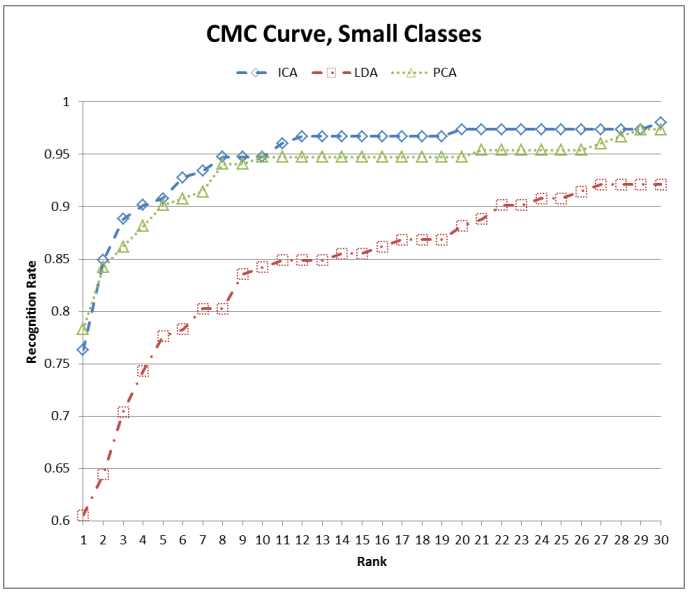


Figure 2.1: Example of a CMC curve drawn from different calculated accuracies (Hodges).

As CMC accuracies are calculated by only looking at the best rank of the queries from same person, the number of actual queries for each person is not taken into account. It therefore only makes sense to use if only one query for each person is contained in the gallery. In the case with more queries per person, as shown in Figure 2.2, the rank-1 accuracy is the same, even though, one of the queries is ranked 5. A different protocol therefore has to utilized.

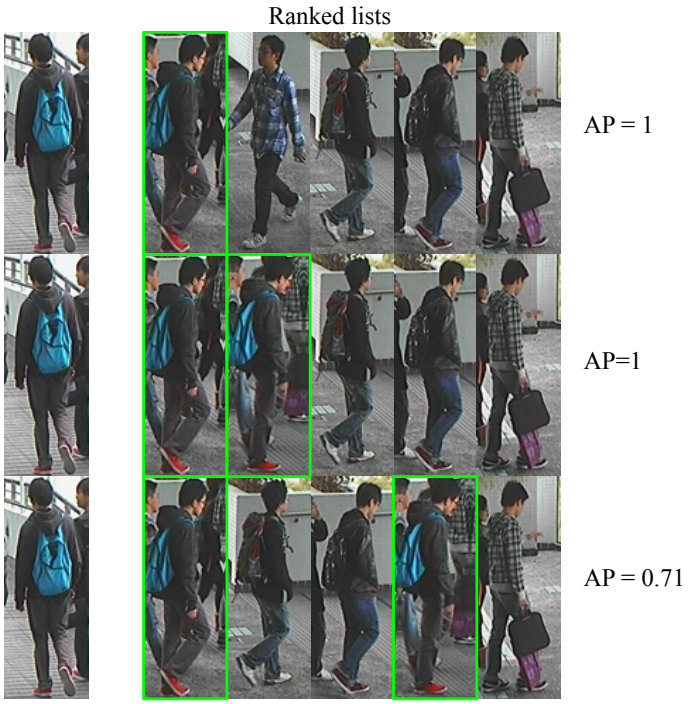


Figure 2.2: Example of the ranking accuracy not being suitable in multi-query case. The left image is the probe and the right images are gallery queries. The green boxes indicate correct match. In the top, there is one query and the ranking accuracy can be utilized. In case of the middle and bottom row, the rank-1 accuracy are similar, even though, the second query is ranked worse in the bottom row. It is there more suitable to look at the average precision (Li et al., 2012).

2.3.2 Mean Average Precision (mAP)

In order to also take the number of queries into account, the mean Average Precision (mAP) can be calculated. This is done by looking at the precision and recall for every probe. Precision can be defined as *"the number of correct matches compared to the number of matches looked at"* while recall can be defined as *"the number of correct matches in the matches looked at compared to all true matches"*. The combination of precision and recall is often used when evaluating image retrieval systems as a metric for the robustness and reliability. Another way of defining precision and recall is by introducing a confusion matrix as shown in Figure 2.3, in which terms based on actual and predicted matches are defined.

	Actual match	Actual mismatch
Predicted match	True Positive (TP)	False Positive (FP)
Predicted mismatch	False Negative (FN)	True Negative (TN)

Figure 2.3: Confusion matrix showing combinations having a true and predicted match.

Utilizing this matrix, precision and recall are calculated by Equation 2.6.

$$\begin{aligned} \text{Precision (P)} &= \frac{TP}{TP+FP} \\ \text{Recall (R)} &= \frac{TP}{TP+FN} \end{aligned} \tag{2.6}$$

By calculating the precision and recall from the ranked list iteratively, including the next rank at every

iteration, a Precision-Recall (PR) curve can be made as shown in Figure 2.4. Having a PR curve, the Average Precision (AP) can be calculated as the area under the curve following Equation 2.7. In order to interpolate between precisions, both the current and previous precision is taken into account.

$$AP = \int_0^1 P(R)dR = \sum_{n=1}^{N} (R^n - R^{n-1}) \frac{P^{n-1} + P^n}{2}, \tag{2.7}$$

where R^0 and P^0 are initialized as 0 and 1, respectively and N is the number of ranked elements. An example of the AP is shown in Figure 2.2 where the second row show an AP of 1 since the two queries are ranked first and second. In the last row, the second query is ranked fourth changing the AP to 0.71. Calculating the AP for each probe, the mAP can be calculated over all probes and used as a metric in the evaluation.

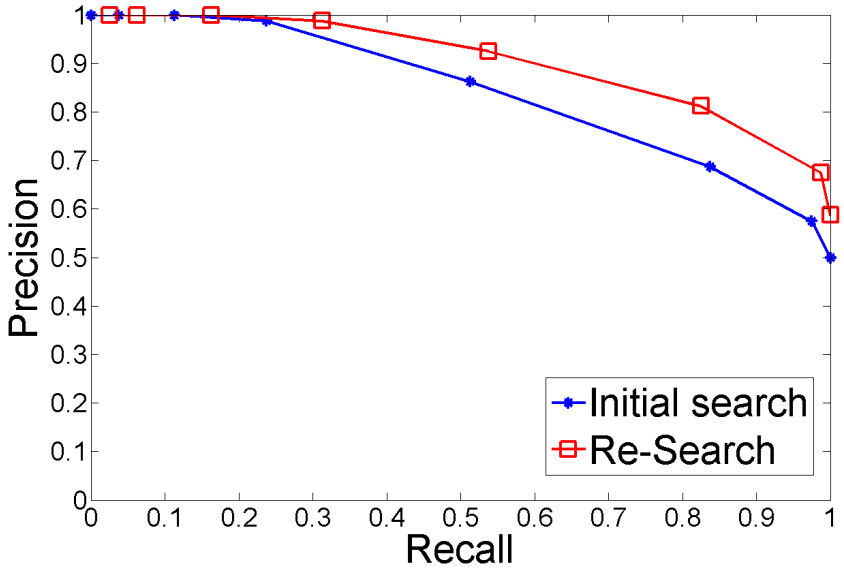


Figure 2.4: Example of a Precision-Recall curve, originally used for image matching (Kang et al., 2009).

As new datasets used for evaluating re-id systems include several queries per person, recent studies are starting to use mAP for evaluation. The larger datasets are created using several cameras, capturing images from different views in order to make evaluation more realistic.

2.4 Datasets

To test the constructed system, four commonly used datasets are included in order to compare the new proposed re-id system with previously state-of-the-art systems. These datasets all consist of images of persons taken from different camera views, hence, introducing the challenges described in chapter 1.

- **ViPER**(Gray and Tao, 2008) - this is the most commonly used dataset. It consists of 1264 images of 632 different persons with two images from two different camera views per person.
- **PRID450S**(Roth et al., 2014) - this dataset is a relatively new dataset (2014) which is build on PRID 2011 by (Hirzer et al., 2011). It consists of 900 images of different 450 persons captured in two different camera views, yielding in two images per person.

- **CUHK01**(Li et al., 2012) - this dataset is also relatively new (2012) compared to ViPER (2007). It consists of 3884 images of 971 different persons with four images in two different camera views per person, yielding at total of four images per person.
- **CUHK03**(Li et al., 2014) - this is one of the most recent datasets for person re-id (2014). It consists of 13,164 images of 1360 different person with and average of 4.8 images from two different camera views per person. A total of three camera pairs were used to capture images.

Examples of images from each dataset are shown in Figure 2.5. Generally, images in VIPeR and PRID450S are brighter making color features more important while texture is be more dominant when it comes to matching images in CUHK01 and CUHK03.

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

Figure 2.5: Examples of image from (a) VIPeR, (b) PRID450S, (c) CUHK01 and (d) CUHK03.

Testing protocols

In order to be able to compare the proposed system with previous proposals, similar testing protocols for the datasets are followed. For VIPeR and PRID450S, persons are randomly and equally divided into training and testing sets, leaving 316 persons in each group for VIPeR and 225 for PRID450S. For CUHK01, 485 persons are used for training while the remaining 486 are used for testing. More training data is available for CUHK03 at which 1160 persons are used for training and 100 are used for testing. The remaining 100 persons are used for validation in situations where parameters can be optimized, such as CNN.

For the three former datasets, 10 random splits are made and used in the tests. For CUHK03, 20 random splits are made following the protocol given by (Li et al., 2014). In this case, the test persons are given at each split while the validation set, if needed, must be randomly chosen from the remaining persons as they are not already provided.

2.5 Recap of Technical Analysis

In order to fuse systems that complement each other well, features at different levels have to be exploited. Features can be divided into three levels; low-level features that include color and texture features extracted on per pixel level and used to create a representation such as a histogram; mid-level creating a more sparse representation, extracted by learned projection based on low-level features; high-level features used to describe entire objects and also learned from low-level features. Furthermore, it is a good idea to extract features both locally and globally, in order to combine the level of detail and have both a global representation of each person, but also more discriminative information.

Fusing can be done on different levels in the matching process. While data-level and early fusion deals with combining either raw data different feature representations, late fusion combines the outputs of different classifiers. As late fusion has shown to be superior in fusion of different biometrics, a fusion scheme on this level is proposed. Further, this simplifies the fusion as potential noise does not has to be taken into account as when fusing on feature or data level.

When evaluating person re-id, the ranked output is often used to create a CMC curve and the rank-1 accuracy is compared calculated as an indicator of the ability to find the exact same person as the most similar. But as each person will often have several queries, datasets are being constructed which are evaluated by calculating the mAP, taking all queries from each probe into account.

Four datasets are often used to evaluate person re-id systems, including VIPeR, PRID450S, CUHK01 and CUHK03. While the two former are minor with 632 and 450 persons, respectively, the two latter are larger with 971 and 1360 persons, respectively. The two minor datasets contain only one image from each person in two views, while the two larger contain multiple images, hence, making them more realistic. To make results comparable to previous systems, similar test protocols are followed, splitting the first three datasets into equally sized training and testing sets while the latter is split into 1160 training persons and 100 test persons.

3. Proposed System

This chapter introduces the different systems used for fusion, including a state-of-the-art CNN and two state-of-the-art systems using hand engineered features. First, an overview of the proposed system is given followed by descriptions of the individual systems, the fusion scheme an evaluation.

3.1 System Overview

An overview of the proposed system is shown in Figure 3.1. The system follows the upper part of the re-id pipeline depicted in Figure 1.6 for each system.

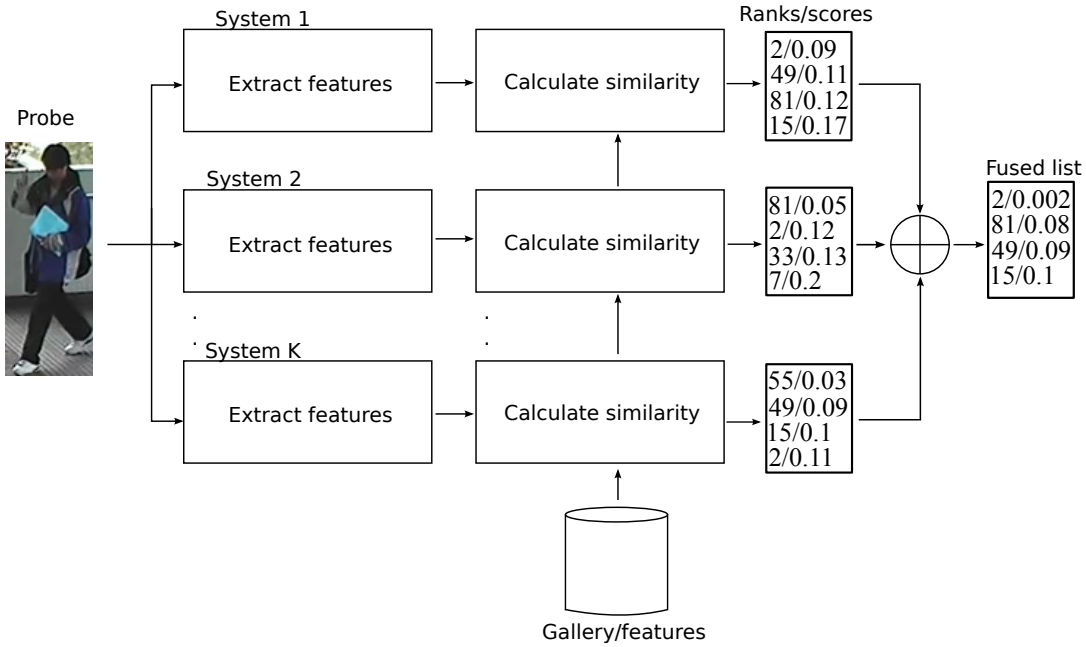


Figure 3.1: An overview of the proposed system in which an unspecified number of systems can be used. Each system will perform feature extraction and compute similarity with the gallery based on a learned metric learning algorithm. The outputs are ranked lists of persons and corresponding scores which are fused (indicated by $\oplus$) to a final ranked list.

A probe is passed to each of the systems which then extracts the specific features. These features are then passed to an already learned metric algorithm which is used to calculate similarities/distances to queries in the gallery. The output for each system will be a ranked list of persons with corresponding scores which are then fused to a final ranked list following the fusing scheme.

3.2 Convolution Neural Networks

Even though, the first CNN's date all the way back to the end of the 1980's (LeCun et al., 1989), it is not until recently, large enough sets of data combined with computational efficiency has made it possible to make properly use of CNN's. The first CNN's were only used for digit recognition using small images of digits as training data, but nowadays, CNN's are being used in a variety of different image processing tasks spanning from object recognition (Krizhevsky et al., 2012) to age and gender classification (Levi and Hassner, 2015) and even to determine, whether or not your selfie is good (Karpathy, 2015). The reason for the increase in popularity lies in accuracies which, as also seen in person re-id, outperforms previously state-of-the-art systems in respective tasks. This is due to the complexity of the networks which make them able to learn to extract very expressive high-level feature representations used for describing entire objects. For this reason, a CNN is used as a modality to extract high-level features. Furthermore, companies like NVIDIA have made it possible to utilize the power of parallel processing on the GPU, making performance much better. Because of the complexity, training a CNN also requires lots of data, in order not to overfit. Therefore, CNN's work best on large datasets while having similar or even worse performance compared to other systems when dealing with small datasets.

A CNN is a deep neural network consisting of different types of layers, each contributing with a change of representation to increase overall accuracy. The most commonly used layers include:

- **Convolution layer** is the most important layer and is used to capture information from an input by convolving it with different filters. Usually, several convolution layers are used and while the first layers learns to capture local features like edges, later layer learn to capture more high-level representations such as parts of doors, leaves etc.
- **Pooling layer** is used to make the CNN more robust to small changes in translation. Usually, this is done with a MAX pooling operation, forward passing the largest value in a neighborhood.
- **Fully Connected layer** is used to combine information from the previous layer to more high-level features which describes entire objects such as a house, a tree etc.
- **Activation layer** is typically placed after each convolution and fully connected layer to transform the input to a fixed interval depending on the utilized activation function.
- **Decision layer** is used to determine the output class dependent on the input to the network.
- **Loss layer** is used to calculate the loss used for back propagation.

In the following, each of these layer types will be further described.

3.2.1 Convolution Layer

The convolution layer maps the input of low-level representations to more high-level representations by convolving with a predefined set of filters as shown in Figure 3.2.

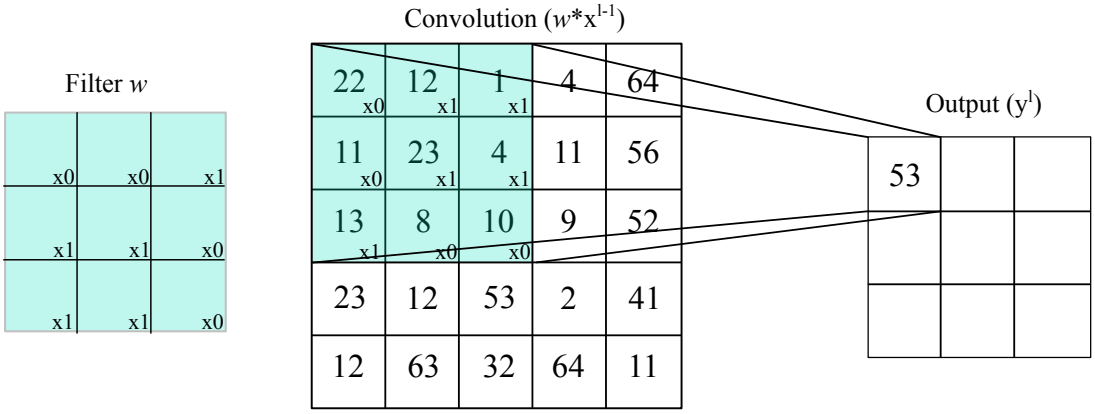


Figure 3.2: Example of 2-d convolution between filter W and feature map x^l. The values in the right bottom corner indicates the filter coefficients. Note that the filter is first vertically, then horizontally flipped.

The size of the filters are given by $N \times N \times C$ for each convolution layer where N denotes the height and width and C the depth. The size of the filters is typically between 3×3 to 15×15 depending on the application and placement of the layer. For the first convolution layer, the depth is 3 if the inputs are color images and 1 if grayscale images are used. Furthermore, the number of filters decides the number of output *feature maps* where each feature map is given as a filter convoluted with the input as shown by Equation 3.1. In addition, a bias term is usually added with each filter shown by b^l in the equation.

$$y^l_{i,j} = w^l * x^{l-1} = \sum_{n=-R}^{R} \sum_{m=-R}^{R} w^l_{m,n} \cdot x^{l-1}_{i-m,j-n} + b^l, \tag{3.1}$$

where $y^l_{i,j}$ is the neuron at location (i,j) at layer l, $w^l_{m,n}$ is a filter with radius R at layer l and $x^{l-1}_{i,j}$ is the neuron in the previous layer $l-1$ at (i,j). The size of the output feature maps depends on the size of the filters and stride at which they are run over the images and is given as $\frac{H^{l-1}-N}{S} + 1 \times \frac{W^{l-1}-N}{S} + 1$, where S denotes the stride while H and W are height and width of the feature maps in the previous layer.

When setting up a CNN, the weights of the filters and bias are initialized with random numbers. The bias is initialized with a constant, typically 0, while the weight initialization may differ depending on the architecture. The most common techniques used for weight initialization are:

- **Gaussian** – draws values from a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ where μ and σ are predefined.
- **Xavier** – draws values from the interval $[-\sqrt{\frac{3}{neurons^{l-1}}}, \sqrt{\frac{3}{neurons^{l-1}}}]$ where $neurons^{l-1}$ is the number of neurons in the previous layer.
- **MSRA** – draws values from a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ where σ is given as $\sqrt{\frac{2}{neurons^{l-1}}}$.

The number of learnable parameters depends on the number and size of the filters and is given as $K \cdot (N \cdot N \cdot C) + K$ where K is the number of filters and associated bias.

3.2.2 Pooling Layer

The pooling layer downsamples the feature maps by connecting a group of neurons in layer $l-1$ to layer l to make a representation which is invariant to small translational changes. It works by running a kernel of specified operation through each feature map with a predefined stride and outputs values based on the operation. Often, MAX kernels are utilized to output the maximum values within a neighborhood as shown in Figure 3.3.

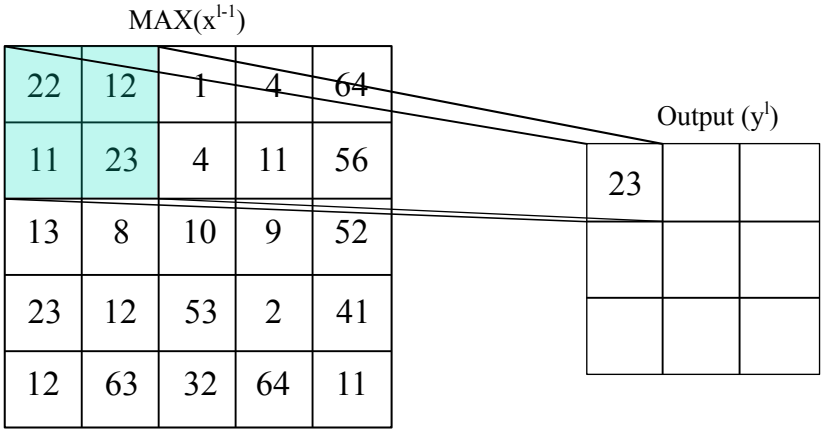


Figure 3.3: Example of 2-d MAX-pooling using a 2×2 MAX-kernel.

More generally, the outputs using MAX-pooling are calculated following Equation 3.2.

$$y_{i,j}^{l} = \max_{m,n \in [0,N-1]} x_{i+m,j+n}^{l-1} \tag{3.2}$$

where N is the size of the pooling kernel. Typically, a 2×2 kernel is used with a stride of 2. The size of the output feature maps depends on the kernel size and stride and is given similar to the convolution layer. The pooling kernels do not contain any weights or bias, hence, no learnable parameters are included in these layers.

3.2.3 Fully Connected Layer

The fully connected layer maps the dense output from the previous layer to a more sparse and high-level representation. This is done by connecting each input to each output as shown in Figure 3.4, hence, the name. The output is therefore calculated as the linear combination of the inputs and associated weights following Equation 3.3.

$$y_j^l = \sum_{i=1}^{N} w_{ji}^l \cdot x_i^{l-1} + b_j^l \tag{3.3}$$

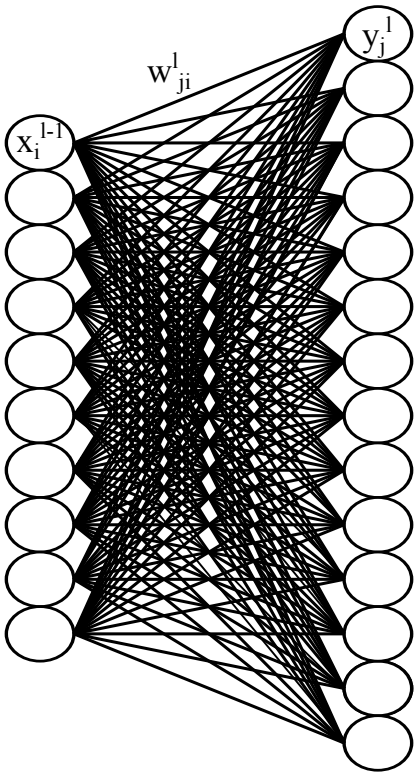


Figure 3.4: Example of a fully connected layer connecting every input x_j^{l-1} to every output y_j^l.

For each output, a weight is associated with each input along with a bias and the number of parameters is therefore given as $neurons_{in} \cdot neurons_{out} + neurons_{out}$. The size of the output furthermore depends on the number neurons in the specified layer.

3.2.4 Activation Layer

The activation layer increases the non-linearity of the network by mapping the input to an output using a non-linear function. This affects the rate at which the weights are updated which will be further explained in subsection 3.2.7. Different types of functions can be used for this purpose with Rectifier Linear Units (ReLU) or a sigmoid function, such as tanh or logistic, as the most common ones, each with a different interval as shown in Figure 3.5.

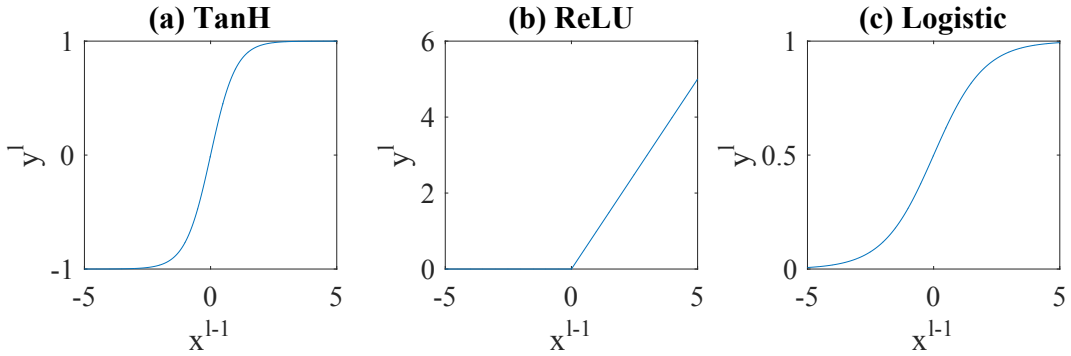


Figure 3.5: Example of three different activation functions and their input-output relation, including the tanh (a), ReLU (b) and logistic (c).

While the ReLU function is simply defined by $f(x) = max(0, x)$, the two sigmoid functions are slightly more complex as they are defined by $f(x) = \frac{1}{1+\exp^{-x}}$ and $f(x) = \frac{\exp^{x} - \exp^{-x}}{\exp^{x} + \exp^{-x}}$ for logistic and tanh, respectively.

Due to its simplicity and, though, still nonlinear character, the ReLU activation function has shown to be both faster when it comes to training and more efficient than the sigmoid functions (Glorot et al., 2011).

3.2.5 Decision Layer

The decision layer maps the high-level representation of the network input to a label indicating a certain class depending on the application as shown in Figure 3.6.

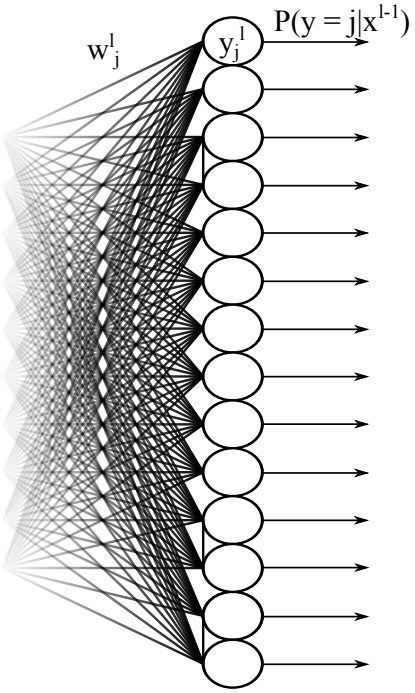


Figure 3.6: Example of a decision layer in which values in each output is mapped to a probability.

In most cases, the softmax function is used to calculate probabilities for each class following Equation 3.4.

$$P(y = j|x) = \frac{\exp^{x^T w_j}}{\sum_{k=1}^{K} \exp^{x^T w_k}}, \tag{3.4}$$

where $P(y = j|x)$ is the probability of class given data x, w_j is the weights for class j and K is the total number of classes. In principle, this corresponds to dividing each output from the previous layer with the sum of all outputs. The class will thus be given by the highest probability value. In the training phase, the decision layer is often combined with the loss layer.

3.2.6 Loss Layer

The loss layer calculates the error of the current forward propagation from a defined loss function based on the predicted and true class labels. When training a CNN, mini-batches are being utilized rather than

running a single images through the network in each iteration. This is done to better generalize on the training data. The goal is to minimize a loss function defined in Equation 3.5.

$$L(W) = \frac{1}{|D|}\sum_{i=1}^{|D|} f_w(x_i) + \lambda r(W), \tag{3.5}$$

where

$L(W)$ is the average loss

$|D|$ is the entire training dataset

$f_w(x_i)$ is the loss function given data x_i

$r(W)$ is a regularization term with weight λ

Since $|D|$ in practice can be very large, an approximation is made, calculating the loss for each mini-batch with size N following Equation 3.6.

$$L(W) \approx \frac{1}{N}\sum_{i=1}^{N} f_w(x_i) + \lambda r(W) \tag{3.6}$$

Having calculated the outputs in the decision layer, an example of a loss function is the multinomial logistic function. In this function, the loss is calculated as the negative log likelihood of the probabilities as given in Equation 3.7.

$$L(W) = -\frac{1}{N}\sum_{i=1}^{N} log(\hat{p}_i, l_i), \tag{3.7}$$

where $\hat{p}$ is the probability value at true label l.

3.2.7 Backpropagation and Parameters

To update the weights in the network, backward propagation is utilized by calculating the error of each layer from the output loss. Recall the outputs from the fully connected and convolution layer:

$x_i^l = \sum_j w_{ij} y_j^{l-1} + b_i^l$ (fully connected) $\vee$ $x_{ij}^l = \sum_{n=-R}^{R}\sum_{m=-R}^{R} w_{m,n}^l \cdot y_{i-m,j-n}^{l-1} + b^l$ (convolution).

$y_i^l = \sigma(x_i^l) \vee y_{ij}^l = \sigma(x_{ij}^l)$ where σ is the activation function.

The chain rule can then be utilized to back propagate the error following Equation 3.8.

$$\begin{aligned} \frac{\partial L}{\partial x^l} &= \frac{\partial L}{y^l}\frac{\partial y^l}{\partial x^l} \\ \frac{\partial L}{\partial y^{l-1}} &= \sum \frac{\partial L}{\partial x^l}\frac{\partial x^l}{y^{l-1}} = \sum \frac{\partial L}{y^l}\frac{\partial y^l}{\partial x^l}\frac{\partial x^l}{y^{l-1}} \end{aligned} \tag{3.8}$$

Further, the weight and bias gradients are calculated as defined in Equation 3.9.

$$\begin{aligned} \frac{\partial L}{\partial W^l} &= \frac{\partial L}{\partial x^l}\frac{\partial x^l}{\partial W^l} \\ \frac{\partial L}{\partial b^l} &= \frac{\partial L}{\partial x^l}\frac{\partial x^l}{\partial b^l} \end{aligned} \tag{3.9}$$

When dealing with the error in more specific cases, the type of layer defines how the error is calculated based on the weights and bias. The error is thus used to update the weights and bias in the given layer. In the following, error calculation and weight update for the three most common layers; convolution, pooling and fully connected layer, will be described.

For the fully connected layer, which is the first layer type in the back propagation, the back propagated error is defined from the derivative of the output given in Equation 3.8, following Equation 3.10.

$$\begin{aligned} \frac{\partial L}{\partial x_i^l} &= \frac{\partial L}{\partial y_i^l}\sigma'(x_i^l) \\ \frac{\partial L}{\partial y_j^{l-1}} &= \frac{\partial L}{y_i^l}\frac{\partial y_i^l}{\partial x_i^l}\frac{\partial x_i^l}{y_j^{l-1}} = \sum_i \frac{\partial L}{\partial y_i^l} w_{ij}^l \sigma'(x_i^l) \end{aligned} \tag{3.10}$$

Further, the weight error gradient depends on the calculated error along with the gradient of the activation of the previous output as given in Equation 3.11. Finally, the bias is solely based on the calculated error as also stated in the equation.

$$\begin{aligned} \frac{\partial L}{\partial w_{ij}^l} &= \frac{\partial L}{\partial x_i^l}\frac{\partial x_i^l}{\partial w} = \frac{\partial L}{\partial x_i^l} y_j^{l-1} = \frac{\partial L}{\partial x_i^l}\sigma'(x_j^{l-1}) \\ \frac{\partial L}{\partial b_i^l} &= \frac{\partial L}{\partial x_i^l} \end{aligned} \tag{3.11}$$

For the pooling layer, the case is simple if MAX pooling is utilized. In this case, only the neuron which contained the maximum value in a certain neighborhood receives the error as small changes in the input would only affect that particular neuron. More generally, this can be written following Equation 3.12.

$$\frac{\partial L}{\partial \max\limits_{m,n\in[0,N-1]} y_{i+m,j+n}^{l-1}} = \frac{\partial L}{\partial x_i^l} \tag{3.12}$$

Since the pooling layer does not contain learnable filters or bias, no gradients are derived.

Finally, the error for the convolution layers also depends on the filter and bias. Here, the error is calculated similarly to the fully connected layer, though, the back propagated error can now be stated as a correlation following Equation 3.13.

$$\begin{aligned} \frac{\partial L}{\partial x_{ij}^l} &= \frac{\partial L}{\partial y_{ij}^l}\sigma'(x_{ij}^l) \\ \frac{\partial L}{y_{ij}^{l-1}} &= \sum_{-R}^{R}\sum_{-R}^{R} \frac{\partial L}{\partial x_{i+m,j+n}^l}\frac{\partial x_{i+m,j+n}^l}{\partial y_{ij}^{l-1}} = \sum_{-R}^{R}\sum_{-R}^{R} \frac{\partial L}{\partial x_{i+m,j+n}^l} w_{m,n}^l \end{aligned} \tag{3.13}$$

The error weight gradient is once again calculated from the error and the and previous output following Equation 3.14. Though, in this case, the sum must be taken for all neurons associated with the weight.

$$\begin{aligned} \frac{\partial L}{\partial w_{m,n}^l} &= \sum_{i=1}^{H-R}\sum_{j=1}^{W-R} \frac{\partial L}{\partial x_{ij}^l}\frac{\partial x_{ij}^l}{\partial w_{m,n}} = \sum_{i=1}^{H-R}\sum_{j=1}^{W-R} \frac{\partial L}{\partial x_{ij}^l} y_{i-m,j-n}^{l-1} \\ \frac{\partial L}{\partial b_i^l} &= \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{\partial L}{x_{ij}^l} \end{aligned} \tag{3.14}$$

When updating the weights in the backward propagation, an optimization algorithm is used, often Stochastic gradient descent (SGD). This is done by calculating a weight update parameter given the negative loss gradient and the previous weight update as given in Equation 3.15.

$$V_{t+1} = \mu V_t - \alpha \nabla L(W_t), \tag{3.15}$$

where μ is the momentum and α the learning rate which are commonly known as the *hyperparameters* of the learning. V_{t+1} is the new weight update which is then used to update weights as shown in Equation 3.16.

$$W_{t+1} = W_t + V_{t+1} \tag{3.16}$$

The last hyperparameter is the weight decay, λ, which together with the regularization term, r(W), helps to avoid overfitting. The regularization term is often given by the L2 regularization given as $\frac{1}{2}w^2$. Including this term in the loss function and taking the gradient, the weight update parameter is changed to Equation 3.17.

$$V_{t+1} = \mu V_t - \alpha \nabla L(W_t) - \lambda W_t \tag{3.17}$$

Looking at the weight updates in both the fully connected and convolution layer, it can be seen that the gradient of the activation affects the weight update depending on the activation function utilized. For ReLU, the gradient will always be 1 for values larger than 0 causing the rate of the weight update to be dependent on the error. On the other hand, if dealing with sigmoid functions, the gradient varies within a certain interval as also seen in Figure 3.5 with values closer to 0 causing a larger gradient and, hence, a faster change in the filter weights.

As also mentioned in section 1.1, different CNN's have been proposed for person re-id, some of them which have achieved state-of-the-art results. The most recent and best working CNN's include the siamese networks proposed by (Ahmed et al., 2015) and the combined hand-crafted and CNN based network presented by (Wu et al., 2016b). While the siamese CNN works well on large datasets achieving high accuracies due to the nature of CNN features, accuracies on smaller datasets are lower. This problem is not seen at the second CNN as it also includes color features in the learning phase while utilizing a feature learning algorithm for reducing the feature dimension, a method which has shown to work well in other proposed systems.

As the two achieve decent results, re-implementations of those are made, tested and CNN achieving best results is further used in the fusing scheme.

3.2.8 Caffe

All re-implementations are done using the *Caffe* framework developed by the Berkeley Vision and Learning Center in California along with several open-source contributors (Jia et al., 2014).

Caffe is an open-source framework which offers simple GPU compatibility and already has lots of different types of layers implemented, including convolution, MAX pooling and fully connected layers[1].

For making CNN models, Caffe makes use of protocol buffers, developed by Google, and each new network is therefore written in a *prototxt* file along with a separate prototxt file working as a *solver* in which all settings are set. Each layer is defined by a name, type, inputs, outputs and corresponding parameters. An example of a defined convolution layer is shown in Figure 3.7.

[1]A layer catalogue can be found at http://caffe.berkeleyvision.org/tutorial/layers.html (Jia et al., 2014).

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

Figure 3.7: Example of a defined convolution layer in the network prototxt file. *Bottom* is input while *Top* is output. *param* are globally set parameters defining how much the weight update is multiplied with the learning rate (lr_mult) and weight decay (decay_mult). *convolution_param* are convolution specific parameters confer subsection 3.2.1.

```
net: "examples/FFN/train_val_fnn.prototxt"
base_lr: 0.00001
lr_policy: "step"
gamma: 0.1
stepsize: 20000
display: 50
max_iter: 100000
momentum: 0.9
weight_decay: 0.0005
snapshot: 1000
snapshot_prefix: "examples/FFN/ffn"
solver_mode: GPU
```

Figure 3.8: Example of settings in the solver file. *base_lr* is the base learning rate, *lr_policy* is the learning policy with following parameters and *snapshot* defines the interval at which intermediate results are saved.

The settings in the solver file include the different hyperparameters, maximum number of iterations, test interval, solver mode (GPU or CPU) etc. Figure 3.8 show an example of a solver file. For each re-implementation, a network solver and network model file is created.

For handling mini-batches during training and testing, *blob's* are utilized. These are 4D arrays, where the size is given by N×C×H×W and

N is the number of image pairs i.e. the size of a batch.

C is the number of image channels i.e. three for a color image.

H and **W** is the height and width of each image.

When defining a network, top and bottom blobs are specified. The bottom blobs are input to a layer while top blobs are output. Furthermore, blob memory is row-major, meaning that the physical location of a value at index (n,c,h,w) is given by $((n \cdot C + c) \cdot H + h) \cdot W + w$. This is important when accessing and storing data.

Besides training and testing neural networks, Caffe offers other tools to perform different tests. The *time* tool can be used to benchmark the time it takes for a forward and backward pass for a certain network. This can be done for a specified number of iterations using either the CPU or GPU. Furthermore, tools for fine-tuning, calculating an image mean or creating databases are available. Databases are created using a text file in which each line indicate the path to an image with a corresponding label as shown in Figure 3.9.

```
1 /home/aske/Datasets/Market-1501/all/0653_c6s2_045218_00.jpg 652
2 /home/aske/Datasets/Market-1501/all/1124_c5s2_157774_00.jpg 1123
3 /home/aske/Datasets/Market-1501/all/0392_c2s1_090221_01.jpg 391
4 /home/aske/Datasets/Market-1501/all/0500_c4s2_060473_00.jpg 499
5 /home/aske/Datasets/Market-1501/all/0937_c6s2_115018_00.jpg 936
6 /home/aske/Datasets/Market-1501/all/0264_c3s1_061292_00.jpg 263
7 /home/aske/Datasets/Market-1501/all/0263_c6s1_056151_00.jpg 262
8 /home/aske/Datasets/Market-1501/all/0629_c6s2_029568_00.jpg 628
9 /home/aske/Datasets/Market-1501/all/0200_c1s1_039976_00.jpg 199
10 /home/aske/Datasets/Market-1501/all/1479_c3s3_080744_06.jpg 1478
11 /home/aske/Datasets/Market-1501/all/0257_c3s1_064892_00.jpg 256
```

Figure 3.9: Example of a text file used to create a database. Each line is a path to an image with a corresponding label.

The conversion makes a serialized representation of each image along with the label and stores it to a Light Memory-Mapped Database (LMDB) which is a type of database that works well with large amounts of data due to its structure. This tool is used for creating databases when training and testing the re-implemented CNN's. Details about database creation can be found in Appendix B.

Last, Caffe offers both Matlab an Python interfaces, making it possible to monitor training and tests. Here, it is also possible to save intermediate results by saving a *caffemodel* file containing weight values and *solverstate* file containing the current settings, iteration number etc.

3.2.9 Joint Siamese CNN

Presented at the IEEE conference on Computer Vision and Pattern Recognition (CVPR) in 2015, (Ahmed et al., 2015) was the first CNN to really set the stage for using CNN in person re-id, achieving state-of-the-art results on CUHK01 and CUHK03 with rank-1 accuracies of 65% and 54.74%, respectively. For VIPeR, achieving a high accuracy is more difficult and the rank-1 accuracy in this case, thus, reached 34.81%. The overall architecture of the CNN is shown in Figure 3.10.

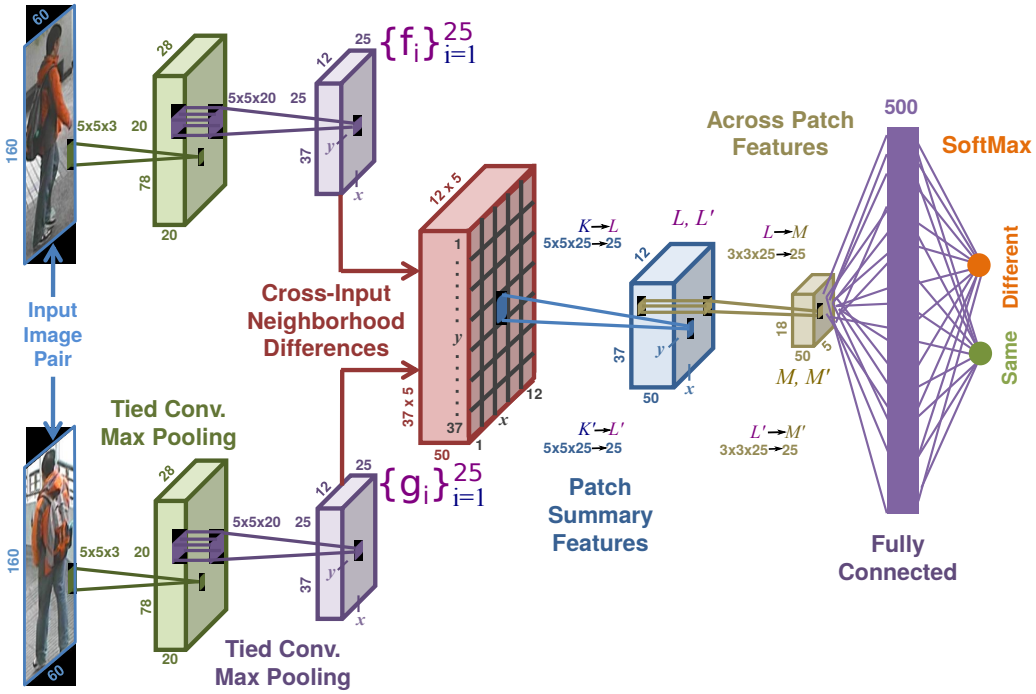


Figure 3.10: State-of-the-art CNN in person re-identification (Ahmed et al., 2015). The CNN follows a siamese structure until the fully connected layers that maps the features to a sparse representation.

The input consists of an image pair, both of size 160×60×3, with a corresponding label, 1 or 0, depending on whether the images are of the same person or not. The two images are individually passed through two convolution layers, both followed by a MAX pooling layer. The two sub-networks share the same filter parameters i.e. they are *tied*. The outputs from the last MAX pooling layers are two 12×37×25 matrices i.e. 25 feature maps of size 12×37, called F_i and G_i where $i = 1:25$.

The two matrices are compared using a **Cross-Input Neighborhood Differences** layer, which pixel-wise up-scales and subtracts two matrices from one another by creating 5×5 patches for every pixel in each feature map. The idea of using this layer is to make the network more robust to positional differences between the image pair. Having a feature map pair, F_i and G_i, where i denotes the feature map number, the resulting patches for that feature map are calculated using Equation 3.18.

$$K_i(x,y) = F_i(x,y) \cdot J_5 - N_5(G_i(x,y)) \tag{3.18}$$

where

$F_i(x,y)$ is the pixel value in feature map i at coordinate (x,y).

J_5 is a 5×5 matrix of ones.

$N_5(G_i(x,y))$ is the 5×5 neighborhood around pixel coordinate (x,y) in feature map i.

This results in 25 difference maps of size 12×37×5×5. Another 25 difference maps, $K'_i(x,y)$, are created by switching the feature maps in Equation 3.18. The two outputs (first 25 difference maps and 25 last difference maps) are then passed through each their convolution layer with a filter stride of 5, called **patch summary** layer. This again reduces the size of each difference map to 12×37. Another convolution layer followed by a MAX-pooling layer is added to each sub-network resulting in two outputs of size 25×5×18. This layer is also called **across patch** layer as it learns filters based on the cross-subtracted patches. Therefore, the two sub-networks do not share filter parameters.

The outputs from the across patch layer are followed by a fully connected layer containing 500 neurons which results in a 500 size feature vector as output. This vector is passed to another fully connected layer containing two neurons. Finally, a softmaxwithloss layer, which is a combined softmax and multinomial loss layer, is placed at the output. This layer maps the output values of the previous layer to probability values of the image pair being from the same person or not and calculates the loss based on these probabilities.

The following layer types are necessary for re-implementing the CNN:

- **Data** layer to import data from a folder.
- **Convolution** layers for the first two layers, the patch summary and across patch layers.
- **MAX Pooling** layers used after the first two convolution layers and the across patch layer.
- **ReLU** layers for use as activation function.
- **Cross-input neighborhood difference** layer for calculating the neighborhood differences.
- **Concatenation** layer for concatenating the two outputs from the across patch layer before passing to the fully connected layer.
- **Inner product** layer for implementing the fully connected layers.
- **Softmaxwithloss** layer to calculate the probabilities and following loss.

Luckily, all but the cross-input neighborhood difference (cross-input) layer are already implemented in Caffe which eases the task. Though, due to lack of details certain assumptions have to be made in the re-implementation. This include:

- **Weight initialization** – as the algorithm used for initializing weights is not specified, the *Xavier* algorithm is used as this is common for most CNN's.
- **Bias** – it is not specified whether or bias is included for all layers. Typically this is the case and it will therefore also be included here.
- **Back propagation** – it is not specified how the errors are back propagated in the cross-input neighborhood layer and a different definition might therefore be used.
- **Parameters** – the cross-input layer is mentioned to contain learnable parameters, though, without specifying the structure of the weights or initialization. The entire network is further specified to contain 2,308,147 parameters, a number which is reached by including bias for all layers and weights for all but the cross-input layer. The re-implementation of the layer will therefore not include learnable parameters.

To implement a new layer, *Reshape()*, *Forward_cpu()* and *Backward_cpu()* functions have to be written to take care of memory allocation and the forward and backward propagation, respectively.

The reshape function allocates memory to the output depending on the size of the top blob(s) and verifies sizes of top and bottom blob(s). For this layer, the sizes of the two bottom blobs are verified to be similar. For the forward propagation, the algorithm should follow Equation 3.18. The algorithm is shown in Pseudocode 1, hence output two top blobs of similar size. To access data from the bottom blobs, a *data_at()* function is being used which returns the value stored at index (n,c,h,w). Furthermore, an *offset()*

function is used to store data at a specific index calculated by the function. The algorithm is summarized in Pseudocode 1.

input: $bottom[0], bottom[1]$ - Two N×25×12×37 blobs with input data
output: $top[0], top[1]$ - Two N×25×12×37×5×5 blobs with output data

$J_5 \leftarrow$ 5×5 matrix with ones ;
for *Each image, n, in input batch* **do**
  for *Each feature map, c, in image* **do**
    for *Each row, h, in feature map* **do**
      for *Each column, w, in feature map* **do**
        $F_c(w,h) \leftarrow [bottom[0] \rightarrow data\_at(n,c,h,w)]$;
        $G_c(w,h) \leftarrow [bottom[1] \rightarrow data\_at(n,c,h,w)]$;
        $N_5(G_c(w,h)) \leftarrow$ Get 5×5 neighborhood around pixel at index *(w,h)* ;
        $N_5(F_c(w,h)) \leftarrow$ Get 5×5 neighborhood around pixel at index *(w,h)* ;
        ;// Calculate patch
        $K_c(w,h) \leftarrow F_c(w,h) \cdot J_5 - N_5(G_c(w,h))$;
        $K'_c(w,h) \leftarrow G_c(w,h) \cdot J_5 - N_5(F_c(w,h))$;
        ;// Save to top blobs at correct index
        $[top[0] \rightarrow offset(n,c,5 \cdot h, 5 \cdot w)] \leftarrow K_c(w,h)$;
        $[top[1] \rightarrow offset(n,c,5 \cdot h, 5 \cdot w)] \leftarrow K'_c(w,h)$;
      end
    end
  end
end
return $top[0], top[1]$;

Pseudocode 1: Algorithm for calculating the neighborhood difference between two bottom blobs.

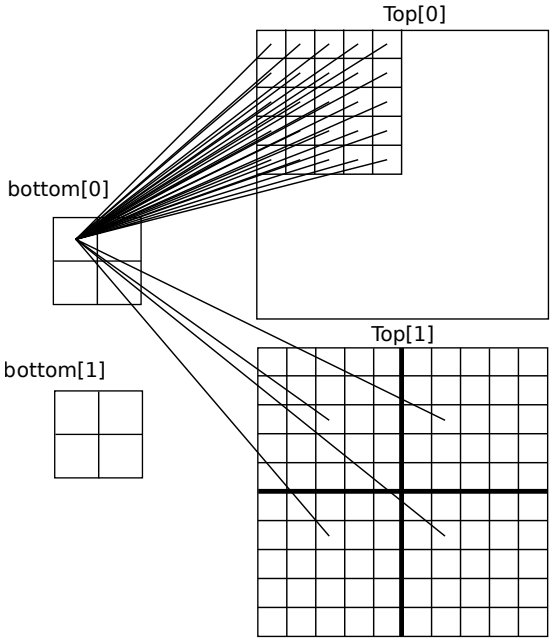


Figure 3.11: The connection between neurons in the input and output of the Cross-input neighborhood difference layer.

As the back propagation for the layer is not specified, assumptions are made. As neurons in feature

map F_i is connected to neurons in both K_i and K'_i as shown in Figure 3.11, the error gradient at each index in bottom[0] has to be calculated from errors in both top[0] and top[1] and vice versa for G_i. As the chain rule is applied when back propagating the error through the network, the error gradient from K_i with respect to a neuron in F_i can be written following Equation 3.19.

$$\begin{aligned}\frac{\partial E}{\partial F_i(x,y)} &= \frac{\partial E}{\partial K_i(x,y)} \cdot \frac{\partial}{\partial F_i(x,y)}(F_i(x,y) - N_5(G_c(x,y))) \\ \frac{\partial E}{\partial F_i(x,y)} &= \frac{\partial E}{\partial K_i(x,y)} \cdot 1\end{aligned} \tag{3.19}$$

That is, all errors from the connected neurons in K_i are summed. Likewise, the error gradients from K'_i with respect to a neuron in F_i can be written following Equation 3.20.

$$\begin{aligned}\frac{\partial E}{\partial F_i(x,y)} &= \frac{\partial E}{\partial K'_i(x,y)} \cdot \frac{\partial}{\partial F_i(x,y)}(N_5(G_c(x,y)) - F_i(x,y)) \\ \frac{\partial E}{\partial F_i(x,y)} &= \frac{\partial E}{\partial K'_i(x,y)} \cdot -1\end{aligned} \tag{3.20}$$

Therefore, the negative of the errors from connected neurons in K'_i are also added to the sum. In order to not back propagate a too large error which might make the model overfit, the average of the summed errors is calculated. The same applies to the error gradients at G_i where signs at 1 are simply switched. In the implementation, a function *diff_at()* is used like *data_at()* to access error data at a certain index. The algorithm is shown in Pseudocode 2.

input: $top[0], top[0]$ - Two N×25×12×37×5×5 blobs containing gradients
output: $bottom[0], bottom[1]$ - Two N×25×12×37 blobs containing gradients

for *Each image, n, in output batch* **do**
  for *Each feature map, c, in image* **do**
    for *Each row, h, in feature map* **do**
      for *Each column, w, in feature map* **do**
        $count = 0$;
        $E_5(K_c(w,h)) \leftarrow$ Retrieve errors from patch $K_c(w,h)$;
        $E'_5(K'_c(w,h)) \leftarrow$ Retrieve errors from patch $K'_c(w,h)$;
        ; // Add up errors from patches
        $[bottom[0] \rightarrow offset(n,c,h,w)] \leftarrow$
        $sum(E_5(K_c(w,h))) + (-1) \cdot sum(assoc(E'_5(K'_c(w,h))))$;
        $[bottom[1] \rightarrow offset(n,c,h,w)] \leftarrow$
        $sum(E'_5(K'_c(x,y))) + (-1) \cdot sum(assoc(E_5(K_c(w,h))))$;
        $count \leftarrow$ #neurons in $E_5(K_c(x,y))$+ #associated neurons in $E'_5(K'_c(x,y))$;
        ; // Divide error with number of associated neurons
        $[bottom[0] \rightarrow offset(n,c,h,w)] \leftarrow [bottom[0] \rightarrow offset(n,c,h,w)]/count$;
        $[bottom[1] \rightarrow offset(n,c,h,w)] \leftarrow [bottom[1] \rightarrow offset(n,c,h,w)]/count$;
      end
    end
  end
end
return $bottom[0], bottom[1]$;

Pseudocode 2: Algorithm for back propagating error gradients in the cross-input layer. $assoc()$ expresses the associated neurons in the particular patch.

To make the algorithm run on the GPU, *forward_gpu()* and *backward_gpu()* functions are written as well, following the same algorithms. A test of the re-implemented layer is carried out before the accuracy test in subsection 3.2.13.

3.2.10 Feature Fusion Network

Presented at the IEEE Winter Conference on Applications of Computer Vision (WACV) in March 2016, (Wu et al., 2016b) combines conventional features when training a CNN to make a new Feature Fusion Network (FFN) that takes advantage of both low-level hand engineered and high-level CNN features. Results on VIPeR, CUHK01 and PRID450s have shown rank-1 accuracies of 41.69%, 47.53% and 58.02%, respectively and is therefore also considered one of the state-of-the-art systems.

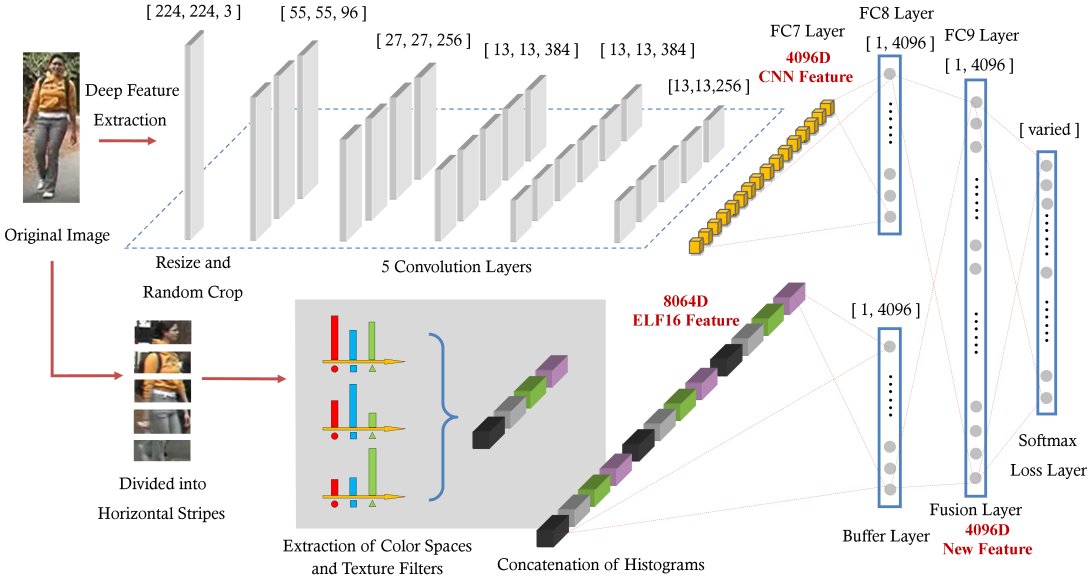


Figure 3.12: Overview of FFN (Wu et al., 2016b). Features using a CNN and conventional color and texture features are extracted, before they are combined in a fully connected layer leading to a softmax function determining the label depending on the input.

The FFN, showed in Figure 3.12, consists of two parts; one containing a common CNN and another containing hand-crafted low-level color features, including both color and texture.

3.2.11 Feature extraction

For the CNN, the architecture follows the AlexNet CNN presented in 2012 (Krizhevsky et al., 2012). The network was originally used for object recognition and consists of five convolution layers, all but the third followed by a MAX pooling and Local Response Normalization (LRN) layer. The LRN layers work by normalizing each output from the previous layer by dividing by the sum of outputs at the same location for adjacent feature maps following Equation 3.21. This has shown to help generalization of the network.

$$y^{l}_{i,j,c} = y^{l-1}_{i,j,c} \left(\frac{1}{\kappa + \alpha \sum_{c=\max(0,k-n/2)}^{\min(K-1,k+n/2)} (y^{l-1}_{i,j,c})^2} \right)^{\beta}, \tag{3.21}$$

where κ, α and β are hyperparameters, n is the number of adjacent feature maps taken into account and K is the number of feature maps. The hyper-parameters are set to 2, 10^{-4} and 0.75, respectively while n is set to 5. A single image is resized and cropped to 224×224×3 and used as input and the output is a 4096-dimensional CNN feature vector from the last pooling layer.

For the other part, the image is horizontally divided into 18 equally sized parts and histograms are extracted from color spaces RGB, HSV, Lab, YCbCr and YIQ along with texture features from 16 different gabor filters. Details for color conversion from RGB can be found in Appendix A. For HSV, YIQ and Lab, the intensity components L, V and Y are left out to make it more robust to lighting changes. For the texture histograms, the image is first converted to grayscale before convoluted with a Gabor filter defined by multiplying a sinusoidal wave with a Gaussian function following Equation 3.22. Examples of different gabor filters are shown in Figure 3.13.

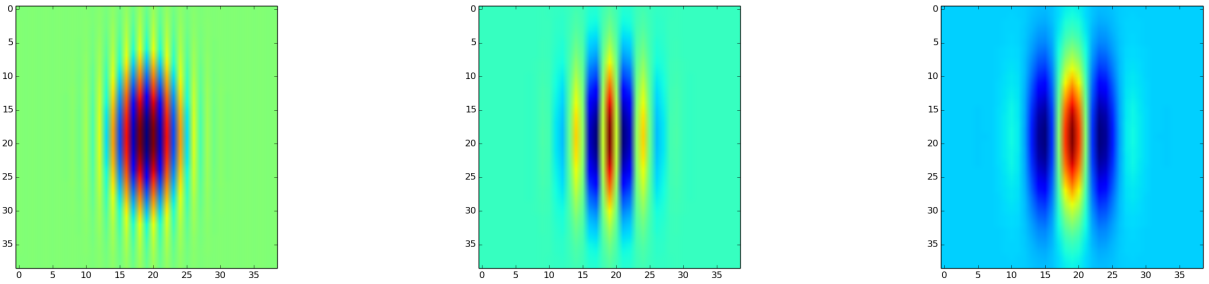


Figure 3.13: Gabor filters with three different wavelengths (left to right), 2, 5 and 10. The remaining parameters: $\sigma = 5, \theta = 45, \psi = 0.5, \phi = 0$

$$g(x', y'; \lambda, \theta, \phi, \sigma, \gamma) = \frac{\pi}{2\sigma^2} \exp(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}) \cdot \cos(\frac{2\pi}{\lambda} + \phi), \tag{3.22}$$

where $x' = x\cos(\theta) + y\sin(\theta)$, $y' = -x\sin(\theta) + y\cos(\theta)$ and

λ represents the wavelength of the cosine and therefore the smoothness of the value changes in the filter.

θ represents the rotation of the filter.

ϕ represents the filter offset i.e. how much the filter values are shifted.

σ represents the standard deviation of the Gaussian function and affects the size of the filters and values.

γ represents the aspect ratio between the values in the horizontal and vertical direction of the filter and thus its ellipticity.

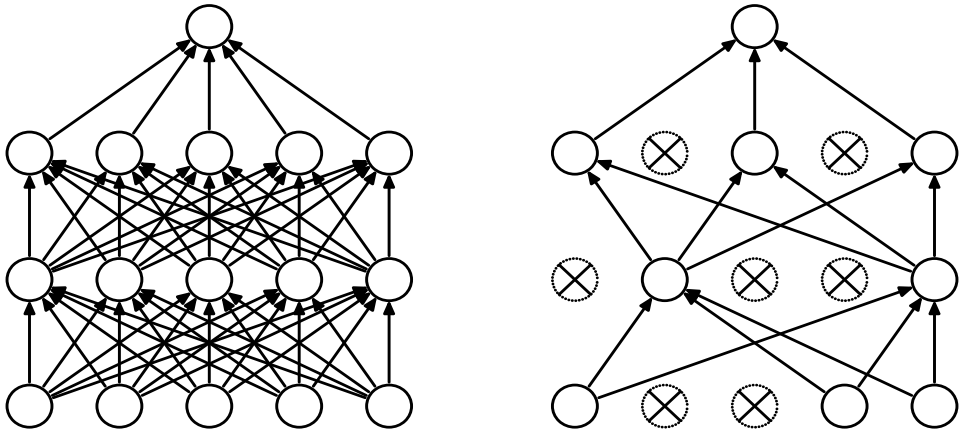
Table 3.1 show the filter coefficients used in this context.

	γ, θ, λ, σ, ϕ
Filter1	0.3, 0, 4, 2, 0
Filter2	0.3, 0, 8, 2, 0
Filter3	0.4, 0, 4, 1, 0
Filter4	0.4, 0, 8, 1, 0
Filter5	0.3, 90, 4, 2, 0
Filter6	0.3, 90, 8, 2, 0
Filter7	0.4, 90, 4, 1, 0
Filter8	0.4, 90, 8, 1, 0
Filter9	0.3, 45, 4, 2, 0
Filter10	0.3, 45, 8, 2, 0
Filter11	0.4, 45, 4, 1, 0
Filter12	0.4, 45, 8, 1, 0
Filter13	0.3, 135, 4, 2, 0
Filter14	0.3, 135, 8, 2, 0
Filter15	0.4, 135, 4, 1, 0
Filter16	0.4, 135, 8, 1, 0

Table 3.1: Filter coefficients used for different Gabor filters used to extract texture features.

Each color and texture histogram contains 16 bins and is L_1-normalized before all histograms are concatenated to a 8064-dimensional feature vector, originally named ELF16.

Before fusing features, both feature types are followed by a fully connected layer, called **buffer layers** that each outputs a 4096-dimensional feature vector. The outputs of the buffer layers are then concatenated and fed to a third fully connected layer, called **fusion layer**, combining information from each of the feature types. For the fully connected layers, dropout is utilized to combat overfitting when training the network. This is done by randomly temporarily leaving out neurons in a layer with a certain probability, as shown in Figure 3.14, in this case 0.5. Furthermore, ReLU activation functions are utilized throughout the network.



(a) Standard Neural Net (b) After applying dropout.

Figure 3.14: Example of dropout implemented in a neural network resulting in each neuron having a certain change of being left out in the current iteration (Srivastava et al., 2014).

As for the siamese CNN described in subsection 3.2.9, a softmaxwithloss layer is used to predict a label and calculate the loss based on the comparison to the true label.

To re-implement the network, the following layer types are needed:

- **Data** layer to import data from a folder.
- **Convolution** layers for five convolution layers.
- **MAX Pooling** layers used after the first and last two convolution layers.
- **Local Response Normalization** layers placed after the first and last two convolution layers.
- **Dropout** layers after the two first fully connected layers.
- **ReLU** layers for use as activation function.
- **Concatenation** layer for concatenating the two outputs from the CNN and hand-crafted features.
- **Inner product** layer for implementing the fully connected layers.
- **Softmaxwithloss** layer to calculate the probabilities and following loss.

For the FFN, an AlexNet model provided by Caffe is modified as shown in the illustration of the architecture placed in Appendix C. Two data layers are declared, one with an image and label as input and a second with the corresponding ELF16 feature. The image is forward propagated through the convolution, pooling and normalization layers to the first fully connected CNN buffer layer. The ELF16 feature is passed to the fully connected ELF16 buffer layer. A concatenation layer is added, in order to create a single vector from the two buffer layers. The concatenated feature is passed to the fully connected fusion layer which outputs to the softmaxwithloss layer. After training the FFN, the model is used to extract features from the fusion layer (FFN features). By combining hand-crafted features with CNN features during training, the new FFN features includes both the sparse representation of the texture from CNN and the distinctiveness of the colors captured in different color spaces, making the overall representation more robust.

Depending on the layer, the representation of the features differ, an example of this is shown in Figure 3.15, in which the output of each convolution layer is shown along with the final FFN, feature represented as a histogram. It can be seen that the output of the first layers reminds of the output from convolving with edge filters such as a sobel, while the local texture information is much more dense in the later layers. At the final fully connected layer, the texture information is thus represented as a sparse feature vector.

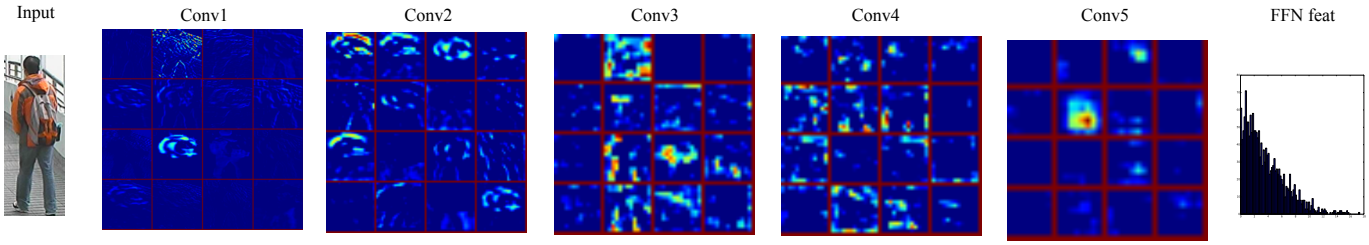


Figure 3.15: Example of the first 16 outputs of each convolution layer and the final FFN layer given an input. As the image progresses through the network, more dense representations of the local texture are present before a sparse representation is extracted at the FFN layer.

Further, Figure 3.16 shows the output features represented as histograms of all positive values for the same person in two camera views compared to a different person. Here, It can be seen that the distributions of the same person in the two different views are much more similar compared to a different id. First thing to notice is the values which lies within a larger range for the other person. Furthermore, the

second person has a larger frequency between 4-8 compared to the first person caused by the difference in texture.

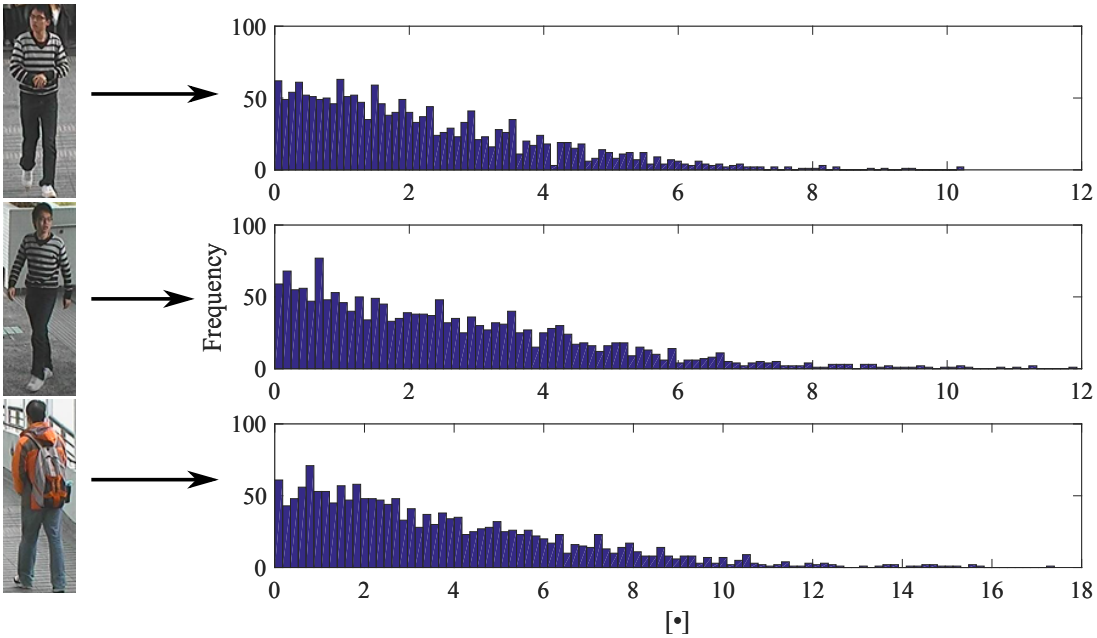


Figure 3.16: Example of features from three persons in two different view. The first two images is the same person in view A and B while the last image is a second person in view B. The features are represented in histograms using 100 bins.

3.2.12 Metric learning

As metric learning, Mirror Kernel Marginal Fisher Analysis (Mirror-KMFA), presented by (Chen et al., 2015b) is then utilized since it has shown good performance compared to other metric learning algorithms.

The main idea of Mirror-KMFA is to transform features from persons in different views by a projection. This is done from the assumption that features from the same persons will have different distributions depending on the view in which they are extracted. This is done by creating a mirror representation of the feature distributions as shown in Figure 3.17.

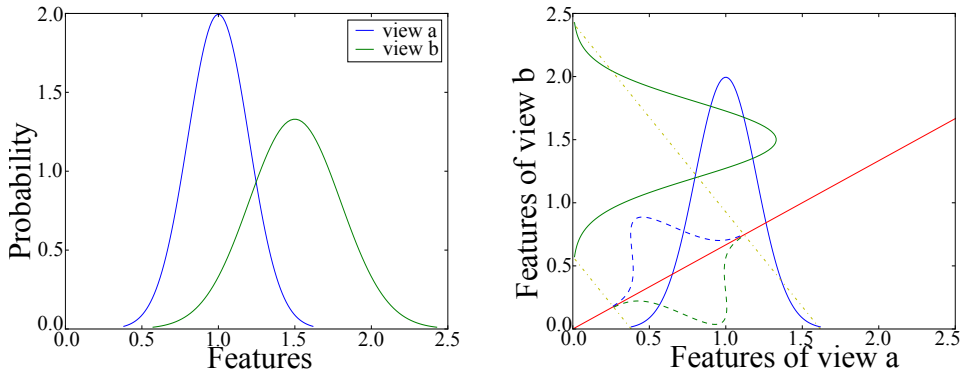


Figure 3.17: Change of feature distributions in two views. Left image shows the distributions of features in a single dimension in two different views which are different. Right image show the mapped distributions (dashed green and blue curves) based on the learned subspace (solid red line) (Chen et al., 2015b).

First, a kernel trick is applied that maps the features to a kernel space. The idea behind this is to make the the original non-linearly separable features linearly separable by applying the learning algorithm in

kernel space. This is done by using a specific kernel function to map an input pair to an output using a predefined function. In this case, the Chi-square Funtion (χ^2) kernel is used following Equation 3.23.

$$k(x, y) = \sum_{i=1}^{n} \frac{2x_i y_i}{x_i + y_i}, \tag{3.23}$$

where z and x are two feature vectors and k is the kernel function. Having feature matrices X_1 and X_2, two resulting matrices X_1^k and X_2^k are created.

Next, the features in the kernel space are augmented by multiplying with the transformation matrices [R,M] and [M,R] in order to deal with the misalignment of the feature distributions before they are projected to a subspace. The two matrices are calculated following Equation 3.24.

$$R = \frac{1+r}{L} I \qquad M = \frac{1-r}{L} I, \tag{3.24}$$

where I is an identity matrix, r is a parameter determining the difference between [R,M] and [M,R] and L is a normalization term chosen to be L_2-normalization. The parameter r is calculated by looking at the subspaces in each view calculated from a PCA. By applying a Singular Value Decomposition (SVD) to a linear combination of the two subspaces, the corresponding principle angles between the subspaces, can then utilized following Equation 3.25.

$$r = \frac{1}{D} \sum_{d=1}^{D} (1 - \cos^2(\theta_d)), \tag{3.25}$$

where θ_d is the principle angle and D is the covariance matrix after applying SVD. The augmented features will thus be $X_{1,aug}^k = [R, M]' \cdot X_1^k$ and $X_{2,aug}^k = [M, R]' \cdot X_2^k$.

Finally, the mirror representation is created following Equation 3.26.

$$\begin{aligned} X_1^M &= \Lambda^{-\frac{1}{2}} \cdot P^T \cdot X_{1,aug}^k \\ X_2^M &= \Lambda^{-\frac{1}{2}} \cdot P^T \cdot X_{2,aug}^k, \end{aligned} \tag{3.26}$$

where Λ and P are matrices containing all eigenvalues and corresponding eigenvectors, respectively, from an eigenvalue decomposition of a matrix $C = [K, -\beta \cdot K; -\beta \cdot K, K]$, where $K = [X_1^k, X_2^k]$ and β is a regularization term, also calculated by looking at the PCA subspace following Equation 3.27. The calculation of the mirror representation is summarized in Pseudocode 3.

$$\beta = \frac{1}{D} \sum_{d=1}^{D} \cos(\theta_d) \tag{3.27}$$

input: X_1, X_2 - Two d×n matrices with features from each person
output: X_1^M, X_2^M - Two mirror transformed matrices of size 2·n×n

$Z \leftarrow [X_1, X_2]$;
; // Calculate kernel matrices using Equation 3.23
$X_1^k \leftarrow k(Z, X_1)$;
$X_2^k \leftarrow k(Z, X_2)$;
; // Calculate transformation matrices using Equation 3.24
$R \leftarrow \frac{1+r}{L} I$;
$M \leftarrow \frac{1-r}{L} I$;
; // Calculate augmented features
$X_{1,aug}^k \leftarrow [R, M]' \cdot X_1^k$;
$X_{2,aug}^k \leftarrow [M, R]' \cdot X_2^k$;
; // Calculate mirror transformed features using Equation 3.26
$K \leftarrow [X_1^k, X_2^k]$;
$C \leftarrow [K, -\beta K; -\beta K, K]$;
$[\Lambda, P] \leftarrow$ eigenvalues and eigenvectors of C ;
$X_1^M \leftarrow \Lambda^{-\frac{1}{2}} P^T X_{1,aug}^k$;
$X_2^M \leftarrow \Lambda^{-\frac{1}{2}} P^T X_{2,aug}^k$
return X_1^M, X_2^M ;

Pseudocode 3: Algorithm for calculating the kernalized mirror representation for an input.

Having the mirror representation of the features, MFA is used to calculate a projection matrix that maps kernel features to a subspace before calculating the distance between feature vectors for the test set. The projection matrix is calculated by first form similarity matrices representing within-class and between-class relationships. Having calculated the distance between each image pair in X_1^M and X_2^M using Equation 3.28, the within-class similarity matrix W is formed by looking at the k-nearest neighbors within the same class for each data point following Equation 3.29. The distance function defined in Equation 3.28 is specific for the kernel space.

$$\mathcal{D}_{x_{1,i},x_{2,j}} = x_{1,i} \cdot x_{1,i} + x_{2,j} \cdot x_{2,j} + 2 \cdot x_{1,i} \cdot x_{2,j} \tag{3.28}$$

$$W_{i,j} = \begin{cases} 1, & \text{if } x_{2,j}^p \in k_{nn}(x_{1,i}^q) \text{ or } x_{1,i}^p \in k_{nn}(x_{2,j}^q) | (p = q) \\ 0, & \text{otherwise} \end{cases} \tag{3.29}$$

where $k_{nn}(x_{1,i}^p)$ and $k_{nn}(x_{2,j}^q)$ are the sets of nearest neighbors in the class p and q for sample $x_{1,i}$ and $x_{2,j}$, respectively. Similarly, the between-class matrix W^P is formed by looking at the k-nearest neighbors in between the classes for each data point following Equation 3.30.

$$W_{i,j}^P = \begin{cases} 1, & \text{if } x_{2,j}^p \in k_{nn}(x_{1,i}^q) \text{ or } x_{1,i}^p \in k_{nn}(x_{2,j}^q) | (p \neq q) \\ 0, & \text{otherwise} \end{cases} \tag{3.30}$$

Having formed the similarity matrices, within-class and between-class scatter matrices are calculated following Equation 3.31.

$$\begin{aligned} S^w &= X_1 X_2^M \cdot (D - W) \cdot X_1 X_2^{M(T)} \\ S^b &= X_1 X_2^M \cdot (D^P - W^P) \cdot X_1 X_2^{M(T)}, \end{aligned} \tag{3.31}$$

where D and D^P are diagonal matrices calculated as $D_{i,i} = \sum_j W_{i,j}$ and $D^P_{i,i} = \sum_j W^P_{i,j}$, respectively.

Having the two scatter matrices, the goal is to find proper projection matrix M which can be done by solving the generalized eigenvalue problem $S^w m_i = \lambda_i S^b m_i$ for each $i \in 1, .., n$ resulting in $M = \{m_1, m_2, ...m_n\}$. This projection matrix is used for projecting the test data to a lower dimensional space before calculating the pairwise distances following Equation 3.28. The training is summarized in Pseudocode 4.

input: $X_1^M, X_2^M, X_1^k, X_2^k$ - Mirror transformed features 2n×n and kernel matrices
output: M - MFA projection matrix

$X \leftarrow [X_1^M, X_2^M]$;
; // Calculate distances in kernel space using Equation 3.28
$N \leftarrow \mathcal{D}_{X_1^k, X_2^k}$;
; // Construct similarity matrices using Equation 3.29 and Equation 3.30
for *each row in* N **do**
    if $x^p_{2,j} \in k_{nn}(x^q_{1,i})$ *or* $x^p_{1,i} \in k_{nn}(x^q_{2,j}) | p = q$ **then**
        $W_{i,j} = 1$
    end
    if $x^p_{2,j} \in k_{nn}(x^q_{1,i})$ *or* $x^p_{1,i} \in k_{nn}(x^q_{2,j}) | p \neq q$ **then**
        $W^P_{i,j} = 1$
    end
end
; // Calculate scatter matrices using Equation 3.31
$S^w \leftarrow X_1 X_2^M \cdot (D - W) \cdot X_1 X_2^{M(T)}$;
$S^b \leftarrow X_1 X_2^M \cdot (D^P - W^P) \cdot X_1 X_2^{M(T)}$;
; // Calculate projection matrix
$M \leftarrow \text{eig}(S^w m = \lambda S^b m)$;
return M ;

Pseudocode 4: Algorithm to calculate the projection matrix M.

3.2.13 Tests

To validate the correctness of the re-implemented CNN's, a test of the accuracy is carried for both systems and compared to the original results. For FFN, code for training the Mirror-KMFA is provided by (Chen et al., 2015b).

Before testing the accuracy, the re-implemented cross-input layer described in subsection 3.2.9 is tested in order make sure the outputs are correct.

Layer test

When creating a new layer in Caffe, both the output of the forward propagation and the error gradient in the backward propagation need to be tested. Furthermore, the dimension of the output blobs are tested.

This is done by creating a script in which dummy blobs are created containing either random or hard-coded data. Letting the CNN run the forward pass, Caffe test macros can then be used to test the equivalence between an expected and actual value. In these tests, the *EXPECT_EQ* macro is used to verify that two values are similar.

The first test is the dimension of the top blobs. For this test, two dummy bottom blobs of size 2×3×3×3 are made. This should then result in two blobs of size 2×3×15×15.

The next test is the output from a predefined input after a forward pass. This is done by hard coding two bottom blobs, run the forward pass and check for similarity in the output. To simplify the test by not causing too large top blobs, two 2×2×2×2 bottom blobs are hard coded with the following values:

$$\begin{aligned} bottom[0] &= \begin{matrix} 2 & 3 \\ 4 & 5 \end{matrix} \\ bottom[1] &= \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \end{aligned} \tag{3.32}$$

This should result in two 2×2×10×10 top blobs with values as shown in Figure 3.18.

Top[0]

2	2	2	2	2	3	3	3	3	3
2	2	2	2	2	3	3	3	3	3
2	2	1	1	2	3	2	2	3	3
2	2	1	1	2	3	2	2	3	3
2	2	2	2	2	3	3	3	3	3
4	4	4	4	4	5	5	5	5	5
4	4	3	3	4	5	4	4	5	5
4	4	3	3	4	5	4	4	5	5
4	4	4	4	4	5	5	5	5	5
4	4	4	4	4	5	5	5	5	5

Top[1]

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	-1	-2	1	1	-1	-2	1	1
1	1	-3	-4	1	1	-3	-4	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	-1	-2	1	1	-1	-2	1	1
1	1	-3	-4	1	1	-3	-4	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Figure 3.18: Resulting top blobs from bottom blobs defined in Equation 3.32.

The last test verifies the numerical correctness of the backward propagation. This is done by making the output of the forward pass imitate the error gradients and then run a backward pass. The resulting error gradients should then be equal to those stated in Equation 3.33.

$$\begin{aligned} bottom[0] &= \begin{matrix} 1.72414 & 2.72414 \\ 3.72414 & 4.72414 \end{matrix} \\ bottom[1] &= \begin{matrix} 0.03448 & 0.03448 \\ 0.03448 & 0.03448 \end{matrix} \end{aligned} \tag{3.33}$$

When testing in Caffe, all tests are made four times; by representing the values as both double and float using both the CPU and GPU. The results are shown in Figure 3.19.

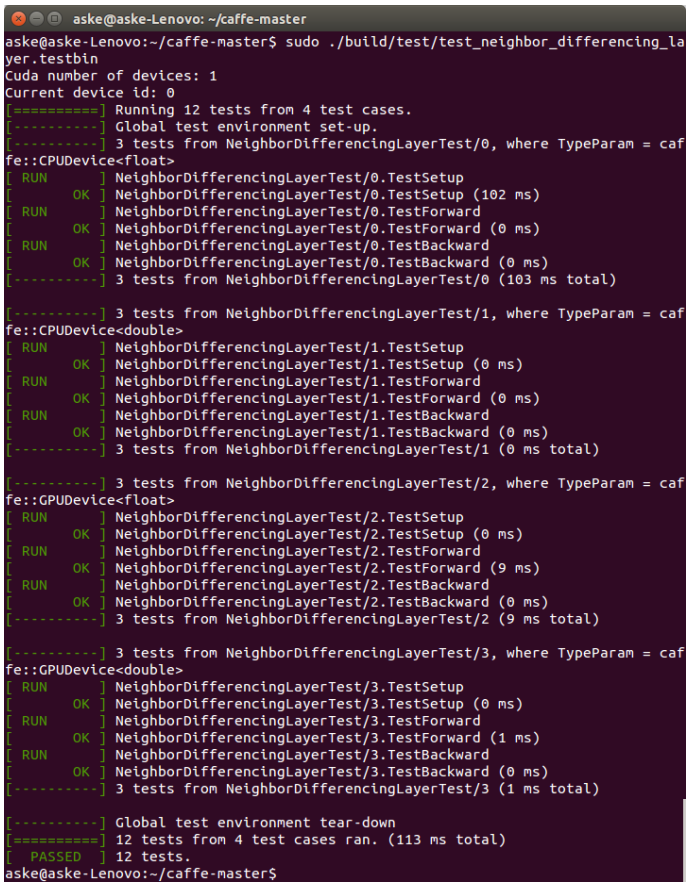


Figure 3.19: Results from the pretests shows all tests pass, including output shape (TestSetup), forward pass (TestForward) and backward pass (TestBackward).

As shown in Figure 3.19, all tests pass which means, the layer behaves as expected.

Accuracy tests

Before testing the CNN's they are trained and the loss is monitored in order to know when it converges. Depending on the CNN, different settings and databases are used for training. In both cases, training and test is carried out using a GeForce GT 760GTX GPU with 4GB of available memory.

First, the siamese CNN is trained and tested. Training is carried out by the use of SGD described in section 3.2 with a mini-batch size of 150 and base learning rate of 0.01. The learning rate is then decreased using an inverse policy where the learning rate in each iteration is calculated as $\alpha^i = \alpha^0(1+\gamma\cdot i)^{-p}$, where α^0 is the base learning rate and γ and p are parameters affecting the speed at which the rate learning decrease and are set to 10^{-4} and 0.75, respectively. The two remaining hyper-parameters, momentum and weight decay, are set to $\mu = 0.9$ and $\lambda = 5\cdot 10^{-4}$.

The first training is done on the CUHK01 dataset where the data is divided into 871 persons used for training and 100 persons used for testing. As the number of positive image pairs in the training set is only 3484, while the number of negative image pairs is almost four million, data augmentation is carried out. For each image, five images of size 150×50 are subsampled from the four corners and center. This increases the number of positive image pairs to 87,100 which is still much smaller than the number of negative. Therefore, negative image pairs are randomly sampled to make twice the number of positive. The monitored training loss is shown in Figure 3.20.

The first result (run 1) shows that the loss becomes infinite (indicated by a loss of 87.33) after just a few

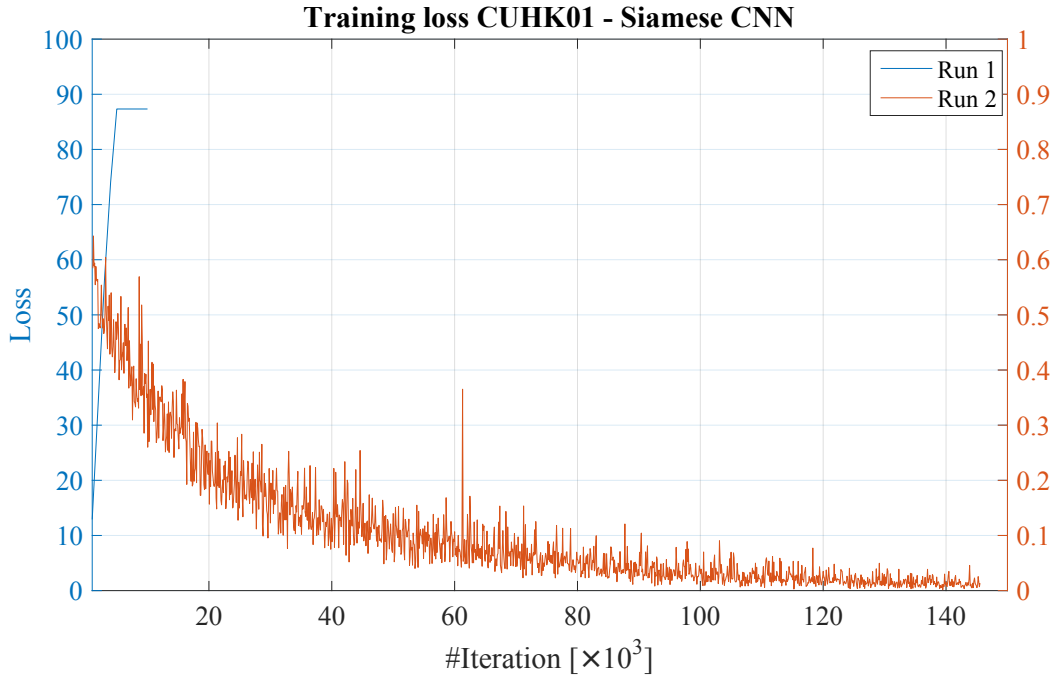


Figure 3.20: Training loss on CUHK01 in which the first run fails my reaching constant high loss (87.33) and the second run quickly starts to converge.

iterations meaning that the network diverges. This is most likely due to the cross-input layer which does not handle back propagation as originally and therefore not correct. This can cause errors to be larger than supposed which quickly increases the value of the filter weights to a high level. Solutions for this can either be to scale the input data or reduce the base learning rate or finally increase the weigh decay.

An attempt is therefore made by decreasing the base learning rate to 0.001 and rerun training. The second attempt show better results as the loss quickly starts to converge as shown in Figure 3.20 (run 2). The loss starts to converge right away and seem to have converged at iteration 140,000 at a loss of 0.01.

In the tests, there are 99 negative image pairs and one positive for each person. Having run all image pairs through the network, the rank accuracies are calculated using only the probabilities of an image pair being similar. The accuracy is then based on the number of persons at which the highest probability is the true match. The results can be seen in Table 3.2.

Rank	Accuracy [%]
1	30
5	62
10	76
20	93

Table 3.2: Ranking accuracies after training on CUHK01 (p=100) for 140,000 iterations.

Compared to the original, the rank-1 accuracy is only half the stated (65%). This confirms that the cross-input layer is not implemented as originally. Even though, results are at a more respectable level, important missing details are needed in order to achieve similar accuracies as original.

To train the FFN, the Market-1501 dataset is used as it is the largest person re-id dataset available. It contains 38,195 images of 1501 persons with images of each person capture in up to six cameras from which some of them have overlapping view. Rather than training from the bottom, a pretrained AlexNet model provided by Caffe is fine-tuned (Jia et al., 2014). Training is carried out using SGD with a mini-batch

size of 25 and a base learning rate of 0.00001. The learning rate is then decreased by a step policy given by $\alpha^i = 0.1 \cdot \alpha^{i-1}$ for every 20,000 iteration. Momentum and weight decay is set similarly to the siamese CNN. Training runs for about 40,000 iterations (Run 1) before convergence as shown in Figure 3.21.

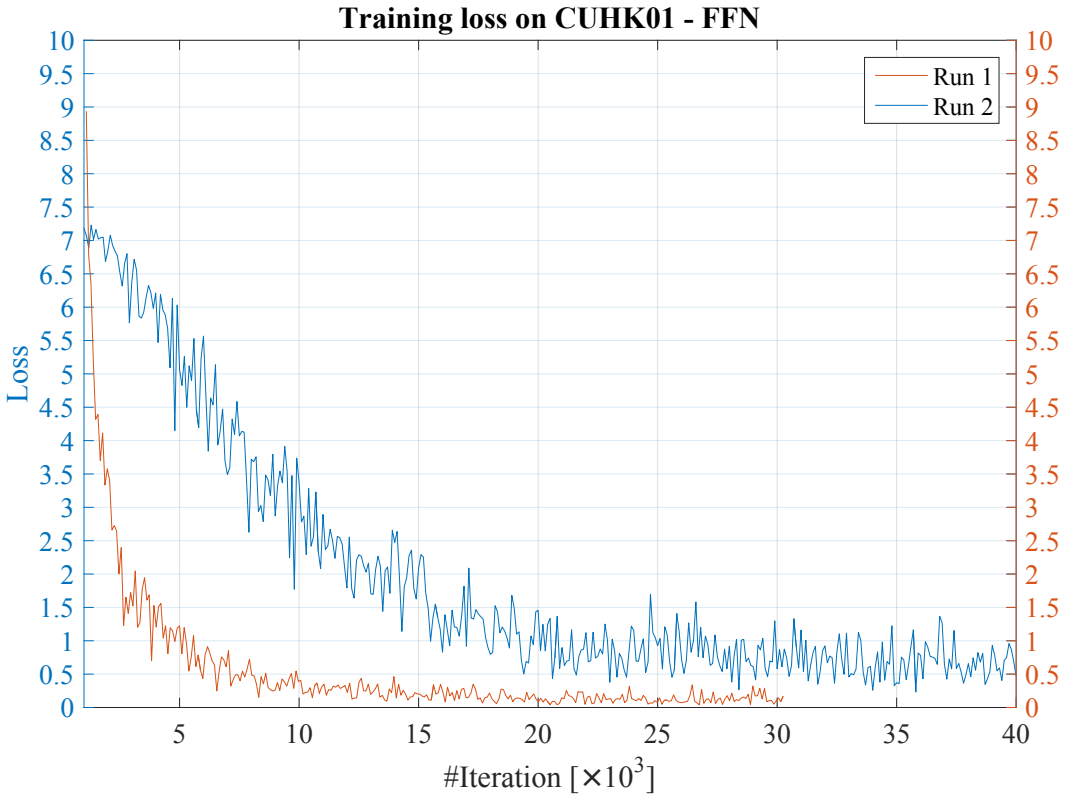


Figure 3.21: Training loss on CUHK01 in which the first run uses all images from Market-1501 and the second run only uses misclassified images for further fine-tuning.

After first convergence, wrongly labelled images are identified and further fine-tuning is carried out using only these images with a learning rate starting at 10^{-6}. For persons at which only up to two images are misclassified, the misclassified images are excluded which results in 2600 images used for further fine-tuning. The final loss converges around iteration 25,000 at 0.1 (Run 2) as shown in Figure 3.21.

The FFN is then used to extract features from the datasets used for testing and the features from the training set is used to train the Mirror-KMFA. As features are also available from the original implementation, the results are compared with those. Following the original paper, tests are conducted on CUHK01 and VIPeR following the protocols defined in section 2.4. The results are shown in Table 3.3 and Table 3.4 for VIPeR and CUHK01, respectively.

System/Rank	r = 1	r = 5	r = 10	r = 20
FFN_{orig}	37.66	66.61	77.85	87.31
FFN_{own}	34.49	62.28	73.45	84.05

Table 3.3: Results on VIPeR (p=316) using original and re-implemented features.

System/Rank	r = 1	r = 5	r = 10	r = 20
FFN_{orig}	35.27	58.25	68.33	77.53
FFN_{own}	32.28	56.95	66.73	76.03

Table 3.4: Results on CUHK01 (p=486) using original and re-implemented features.

As shown, results are slightly below the original. Reasons for this may be how the persons are divided or how the new layers in the neural network part of the FFN are defined in terms of weight initialization, weight decay and learning rate which are not originally stated. Further, it might be noted that the results in both situations are lower than stated in the beginning of this section. As the metric learning algorithm

is used directly out the box using the same settings as stated by the author, wrongly or missing details are the cause of the lower accuracy. Both a linear and Radial Based Chi-square Function kernel (RBF-χ^2) have been tested instead but without any increase in accuracy, thus, missing details in the paper are needed in order to figure out the reason for this slight decrease.

As the results compared to original only vary by a small margin for FFN compared to the siamese CNN, while also being better, it will be used further in the fusion scheme. This is furthermore argued by the fact that the siamese CNN was tested on only 100 persons while 486 were used for FFN and using similar division between test and training data for the former most likely reduce the accuracy. For the tests, the features extracted using the re-implementation will be used. This system will go by **FFN** in the fusing scheme.

3.3 Cross-View Dictionary Learning

For mid-level features, the representation of low-level features is changed be more sparse using a feature learning algorithm. While neural networks can be seen as one such algorithm, dictionary learning is another way of making the data more sparse. By making the data more sparse, a more discriminative representation of features can be made and is therefore beneficial for this task.

A benchmark of different person re-id systems has been made by the Smart Surveillance Interest Group (SSIG) at Federal University of Minas Gerais, Brazil (Schwartz and Carlos, 2016). On this benchmark, the top ranked CMC results are given on different datasets, including VIPeR, PRID450S and CUHK01. A few papers utilizing mid-level features are present on these lists, including *Learning Mid-level Filters for Person Re-identification* (MLF) proposed by (Zhao et al., 2014) at CVPR in 2014. The system is shown in Figure 3.22 and utilizes Support Vector Machines (SVM) to learn a set of mid-level features using *efficient* patches i.e. local patches that are neither over- or under-represented among all images in the training set. The algorithm tries to cluster similar patches by matching patches from an image in view A with all patches at a constrained spatial location at images in view B. Even though, this system is robust to misalignment and scaling, training easily takes up lots of memory and processing time when dealing with larger datasets.

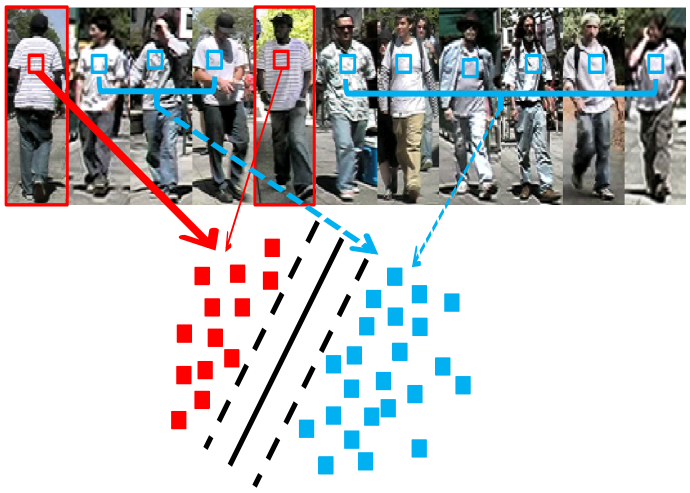


Figure 3.22: Principle of MFA in which SVM is utilized to distinguish correctly matched patches from same person across views (red boxes) from incorrectly matched patches from different persons (blue boxes).

Another proposal was presented at the International Joint Conference on Artificial Intelligence (IJCAI) in July 2015, by (Li et al., 2015), called Cross-View Projective Dictionary Learning for Person Re-

identification (CPDL). The system, shown in Figure 3.23, utilizes dictionary learning in order to create a more sparse feature representation which, unlike the former presented system, is solved as an optimization problem and is hence faster to solve. Furthermore, dictionaries on both patch level and image level are learned, in order to take advantage of two levels of abstraction as done in this fusion scheme. Meanwhile, it still makes use of discriminative features extracted from local patches like MLF

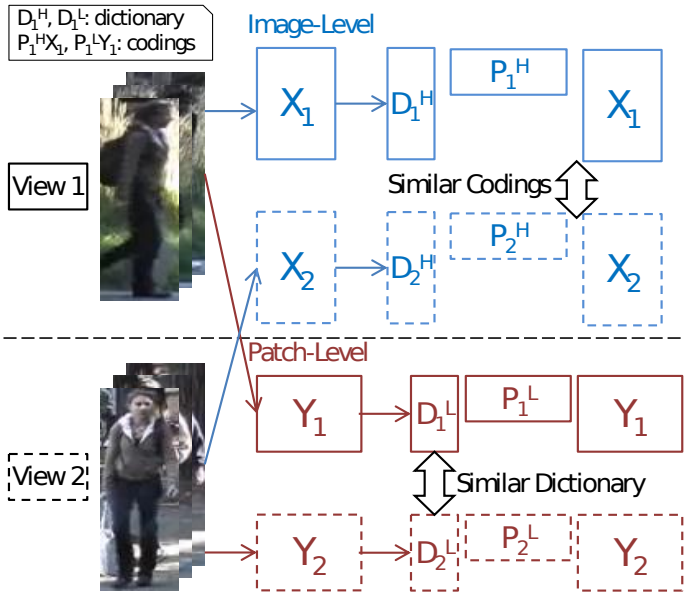


Figure 3.23: Dictionary learning based on cross-view images (Li et al., 2015). On both image-level and patch-level, dictionaries and projection matrices are learned. On image-level, projection matrices are shared across views, while dictionaries are shared on patch-level.

As the system has shown decent results on both VIPeR and CUHK01 with rank-1 accuracies of 33.99% and 59.47%, respectively, it will be utilized in this fusing scheme. Furthermore, the features at patch-level are sparse representations of local areas, unlike FFN which is based on more global representations and it is therefore suitable to fuse with FFN.

3.3.1 Feature extraction

Before dictionary learning, low-level features are extracted. The images are first converted to the Lab color space, details are found in Appendix A, and a 10×10 window is run through the image with a step size of 5. For each patch, 32-dimensional color histograms along with 128-dimensional SIFT features are extracted in each channel. Furthermore, color histograms are extracted from down sampled images with scaling factors of 0.5 and 0.75, resulting in a final 672-dimensional feature vector for each patch. All histograms are L_2-normalized before dictionary learning. By including SIFT descriptors as a different way to represent the local textures, the system becomes more suitable when fusing with the FFN.

Usually, SIFT descriptors are calculated from detected key points i.e. points in the image that stand out as local maxima, found in the entire image but in this case, a descriptor is calculated for each patch leaving out key point detection. For a patch, filtering is performed using the first order derivatives of a Gaussian filter, known from Canny edge detection, following Equation 3.34. Each output will, thus, contain the gradients in the horizontal and vertical direction, respectively.

$$\begin{aligned} I_x = I * G_x \qquad I_y = I * G_y \\ G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{(x^2+y^2)}{2\sigma^2}} \end{aligned} \tag{3.34}$$

where G_x and G_y are the gradient in the x and y direction, respectively, for the Gaussian filter.

Next, the gradient magnitudes and orientation are calculated following Equation 3.35.

$$I_M = \sqrt{I_x^2 + I_y^2} \qquad I_\theta = \tan^{-1}\left(\frac{I_y}{I_x}\right) \tag{3.35}$$

The patch is then divided into 4×4 cells and the orientations of the gradients are quantized into 8 bins with each bin representing a range of angles.

3.3.2 Metric learning

Having extracted features, dictionaries are learned on both patch level and image level. Generally, dictionary learning is about learning a dictionary matrix $D \in \mathbb{R}^{d\times m}$ along with sparse coefficient matrix $Z \in \mathbb{R}^{m\times n}$ such that the data matrix $X \in \mathbb{R}^{d\times n}$ can be represented as $X = DZ$. This is solved by the objection function defined in Equation 3.36.

$$\begin{aligned} \min_{D,Z} ||X - DZ||_F^2 + \lambda Z \\ \text{s.t. } ||d_i \leq 1||_2, \end{aligned} \tag{3.36}$$

where λ is a regularization term and d_i is the i'th column vector of the dictionary matrix where the vector is constrained in order to avoid large values causing the Z to have very low values.

But since solving Z is an NP-hard problem, the problem is reformulated to $X = DPX$, where $P \in \mathbb{R}^{m\times d}$ is a projection matrix with m≪d. Having a dictionary and projection matrix for each camera view, the objection function thus changes to Equation 3.37.

$$\begin{aligned} \min_{D_1,D_2,P_1,P_2} ||X_1 - D_1P_1X_1||_F^2 + ||X_2 - D_2P_2X_2||_F^2 + \lambda f(D_1, D_2, P_1, P_2) \\ \text{s.t. } ||d_{1,i} \leq 1||_2, \quad ||d_{2,i} \leq 1||_2 \end{aligned} \tag{3.37}$$

where $f(D_1, D_2, P_1, P_2)$ is a regularization function that affects the similarity between dictionary or projection matrices in the two views. To distinguish between dictionaries and projection matrices at patch and image level, the superscripts L and H are used to represent patch level and image level matrices, respectively.

At patch level, features from patches at the same spatial location in two different views do most likely not share same subspace due to misalignment, though, they can be assumed to share the same dictionary making the regularization function $||D_1^L - D_2^2||_F^2$. By adding a variable matrix A, each of the first two terms in Equation 3.37 can be rewritten to Equation 3.38.

$$\begin{aligned} \min_{D_1^L,P_1^L,A_1^L} ||X_1 - D_1^LA_1^L||_F^2 + \beta||P_1^LX_1 - A_1^L||_F^2 + \lambda_1||D_1^L - D_2^L||_F^2 \\ \text{s.t. } ||d_{1,i}^L \leq 1||_2, \quad ||d_{2,i}^L \leq 1||_2, \end{aligned} \tag{3.38}$$

where β is a balance parameter. The solution to the matrices A_1^L and P_1^L can then be formulated as in Equation 3.39.

$$\begin{aligned} A_1^L &= (D_1^{L(T)}D_1^L + \beta I)^{-1}(D_1^{L(T)}X_1 + \beta P_1^LX_1) \\ P_1^L &= A_1^LX_1(X_1X_1^T + \gamma I)^{-1}, \end{aligned} \tag{3.39}$$

where γ is a regularization parameter. The dictionary is then updated in an iterative process until convergence following Equation 3.40. Convergence is defined as a desired difference between the current

and previous dictionary which is set to maximum 10^{-9}.

$$\begin{aligned} D_1^{L,i} &= (\rho(S^i - T^i) + [X_1, X_2] \cdot [A_1^L, A_2^L]^T)^{-1}(\rho I + [A_1^L, A_2^L] \cdot [A_1^L, A_2^L]^T \\ S^i &= \min(1, \sqrt{\sum_n (D_{1,n}^{L,i} + T^{i-1})^2})^{-1}(D_1^{L,i} + T^{i-1}) \\ T^i &= T^{i-1} + D_1^{L,i} - S^i, \end{aligned} \tag{3.40}$$

where S^0 and T^0 are initialized to D^0 and 0, respectively with D^0 being initialized with random numbers drawn from a Gaussian distribution. In S^i, $D_{1,n}^{L,i}$ is the n'th row in the dictionary matrix D_1^L at iteration i. Having updated the dictionary once, the process from calculating A_1^L and P_1^L is run again for N number of iterations, in this case 30. Likewise, the matrices D_2^L, P_2^L and A_2^L can be found using similar solutions.

At image level, features from each patch are concatenated to a single feature vector describing the entire image. Therefore, similar dictionaries for the same person in two different views does not make sense, since this would cause to general dictionaries. On ther other hand, features in two different views can be assumed to share the same subspace i.e. the projection matrix should be similar. For this reason, the regularization term $||P_1^H X_1 - P_2^H X_2||_F^2$ is used.

Since the feature vector for each image can be up to 332,000-dimensional, depending on the image size and hence number of patches, PCA is first applied to reduce the dimension to 190. Like at patch level, each of the first two terms in Equation 3.37 is changed to Equation 3.41.

$$\begin{aligned} \min_{D_1^H, P_1^H, A_1^H} ||X_1 - D_1^H A_1^H||_F^2 + \alpha||P_1^H X_1 - A_1^H||_F^2 + \lambda_2||A_1^H - A_2^H||_F^2 \\ \text{s.t. } ||d_{1,i}^H \leq 1||_2, \quad ||d_{2,i}^H \leq 1||_2, \end{aligned} \tag{3.41}$$

Because of difference in similarity between dictionary matrices, A_1^H is calculated differently and the solutions therefore follows Equation 3.42.

$$\begin{aligned} A_1^H &= (D_1^{H(T)} D_1^H + (\alpha + \lambda_2)I)^{-1}(D_1^{H(T)} X_1 + \lambda_2 A_2^H + \alpha P_1^H X_1) \\ P_1^H &= A_1^H X_1 (X_1 X_1^T + \gamma I)^{-1}, \end{aligned} \tag{3.42}$$

Similar solutions are given for A_2^H and P_2^H. The dictionary is update following the procedure from Equation 3.40. The training at patch-level is summarized in Pseudocode 5. At image-level, the solutions

for A and P can simply be replaced by the equations in Equation 3.42 and leaving out line 12.

input: X_1, X_2 - feature matrices from view A (X_1) and B (X_2)
output: D_1^L, D_2^L - dictionaries at patch level

; // Initialize matrices

1 $P_1^L \leftarrow$ random numbers from $\frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$;
2 $D_1^L \leftarrow$ random numbers from $\frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$;
3 $S^0 \leftarrow D_1^L$;
4 $T^0 \leftarrow 0$;
5 **for** *each iteration n* **do**
; // Calculate variable and projection matrix using Equation 3.39
6 $A_1^L \leftarrow (D_1^{L(T)} D_1^L + \beta I)^{-1} (D_1^{L(T)} X_1 + \beta P_1^L X_1)$;
7 $P_1^L \leftarrow A_1^L X_1 (X_1 X_1^T + \gamma I)^{-1}$;
8 **while** $D_1^{L,i} - D_1^{L,i-1} > 10^9$ *or* $i < 100$ **do**
9 Update $D_1^{L,i}$, S^i and T^i by Equation 3.40 ;
10 **end**
11 **end**
12 $D_2^L \leftarrow D_1^L$;
13 **return** D_1^L, D_2^L ;

Pseudocode 5: Algorithm for iteratively dictionary learning.

When training dictionaries on both patch level and image level, the parameters λ_1 and λ_2 are set to 2 and 1, respectively based on an analysis of the accuracy with changing values. Furthermore β and α are empirically set to 1 and 2, respectively. Tests on the VIPeR dataset were originally conducted with different dictionary sizes, resulting in the most optimal size being 50, meaning each feature vector is projected to a 50×1 vector. At both patch and image level this mapping is done by solving the optimization problem $\min_{X_D} \|X - DX_D\|_2^2$ s.t. $\|X_D\|_0 \leq T$ where X is the original feature, D is the dictionary, X_D is the projected feature and T is the size of the dictionary. An example of a projection of patches at same spatial location from two different persons in two different views is shown in Figure 3.24. Here it can be seen that the sparse representation of the two patches from the same person are more similar compared to the patch from the second person, especially when looking at the first 15 and last 10 values, due to the difference in appearance.

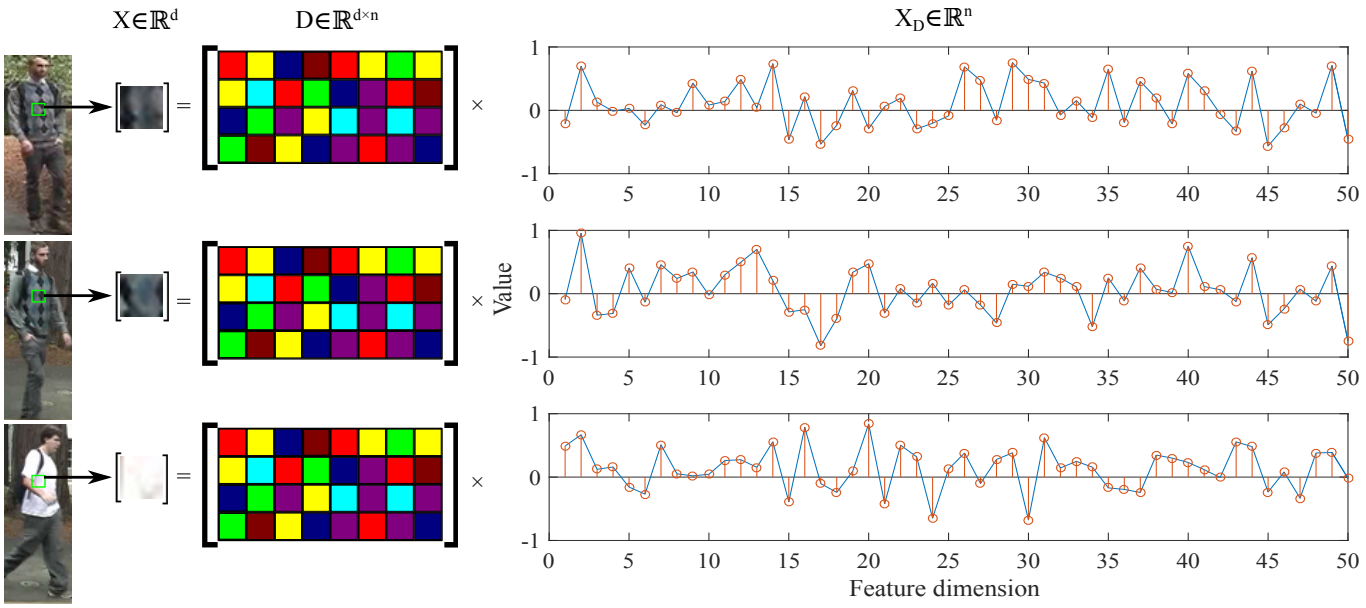


Figure 3.24: An example of patches X extracted in different views at same spatial location and their corresponding sparse representation X_D given the dictionary D. Notice that the dictionary is similar in both views.

Because of the challenge of misalignment between camera views, a patch in one view cannot solely be compared to the patch at same spatial location in a different view. Therefore, the patch is compared with all patches at the same horizontal level as shown in Figure 3.25. This is done by calculating the Euclidean distance between each patch pair and base the similarity score for the patch on the nearest patch distance as shown in Pseudocode 6. The score between a probe and a query is finally given as the sum of all patch distances in the image.

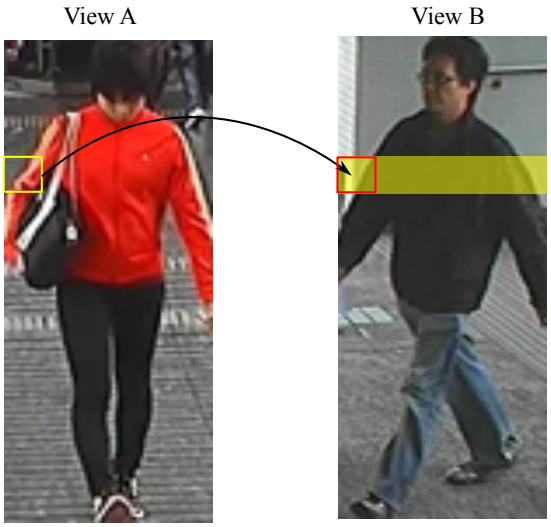


Figure 3.25: Matching patches between camera views. Patches in view A are matches with patches at the same horizontal level (yellow area) in view B and the shortest match is returned.

At image level, the similarity $Score_I(p,q)$ between a probe and a query is simply calculated as the dot product between the two corresponding feature vectors. Having normalized both patch level and image level scores, the two are fused following Equation 3.43.

$$Score(p,q) = Score_P(p,q) + \lambda Score_I(p,q), \tag{3.43}$$

where λ is a pre-defined weight parameter set to be between [0,1].

input: $X_p^L, X_p^H, X_q^L, X_q^H$ - sparse patch and image representations of probe and query
output: *Score* - matching score between probe and query

; // Calculate patch score
for *each patch i at horizontal line j* **do**
  $X_{q,j} \leftarrow$ acquire all patches at horizontal level j ;
  $\mathcal{D}(X_{p,ij}^L, X_{q,i}^L) \leftarrow \min(\mathcal{D}(X_{p,i}^L, X_{q,j}^L)$;
end
$Score_P(p,q) \leftarrow sum(\mathcal{D}(X_{p,ij}^L, X_{q,i}^L)$;
; // Calculate image score
$Score_I(p,q) \leftarrow X_p^{H(T)} X_q^H$;
; // Calculate total score
$Score(p,q) \leftarrow Score_P(p,q) + \lambda Score_I(p,q)$;
return $Score(p,q)$;

Pseudocode 6: Algorithm for matching scores at patch- and image-level and fuse the results.

For dictionary learning, code is provided by (Li et al., 2015) and will be named **DICT** in the fusing scheme.

3.4 LOcal Maximal Occurrence Representation and Metric Learning (LOMO)

For low-level features, discrimination can be increased by employing techniques that extract features locally. But by making use of locally extracted features, time complexity increases and features which are fast to extract should therefore be used. Different systems utilize patches, including *Person Re-identification by Salience Matching* by (Zhao et al., 2013) who proposed creating a salience probability map for each person as shown in Figure 3.26. This is done by taking the distances from each patch at a person in view A to all patches at a constrained area from each person in view B into account. Patches with low distances are thus defined as being more salient. Even though, this creates a more discriminative representation for each person, the system does not take the change of feature distribution between views into account and the system will thus be challenged if differences between views are too large. Furthermore, the system is comprehensive since it matches several patches both when creating the probability maps and when calculating matching scores.

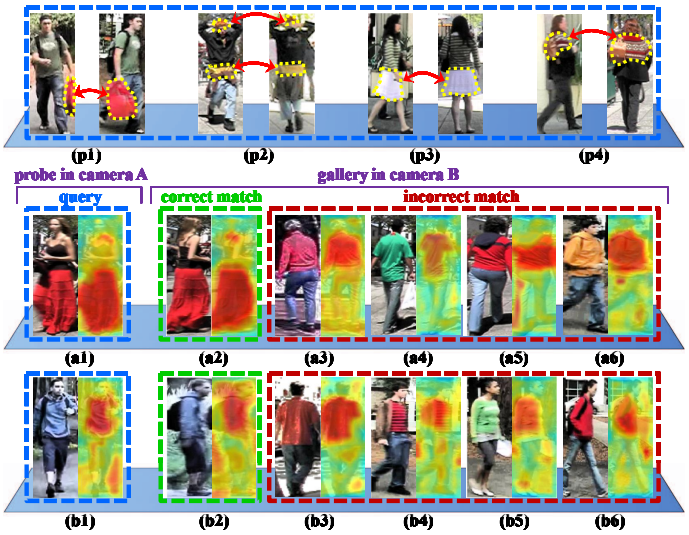


Figure 3.26: Principle of matching by salience probability maps (Zhao et al., 2013). The top row shows different salient areas at persons in two views while the two bottom rows show the salience probability maps for a probe in view A and a number of queries in view B and the corresponding matches.

A more simple matching is performed by (Zheng et al., 2015b) who proposed extracting different low-level features and improve the results by fusing scores calculated for the individual features, using the product ruled described in section 2.2. Initially, this system is performed unsupervised and accuracy is therefore not as high as proposed systems that utilized supervised learning. Such system was presented by (Liao et al., 2015) at CPVR in June 2015 year, named *Person Re-identification by LOcal Maximal Occurrence Representation and Metric Learning* (LOMO). The system shown in Figure 3.27, utilizes a feature representation made from locally extracted low-level features and Mahalanobis distance to match. The system is, thus, fast both when it comes to feature extraction and matching. Furthermore, it has shown decent results on both small and large datasets with a rank-1 accuracy on the VIPeR of 40%, 52.20% on the CUHK03 and 63.20% on CUHK01 using multiple-shot settings. The system makes use of both local color features extracted in HSV color space and texture features based on local patterns. As the low-level features in this system is different from those in both FFN and DICT combined with the level of discrimination, it is well suitable to include in this fusion scheme.

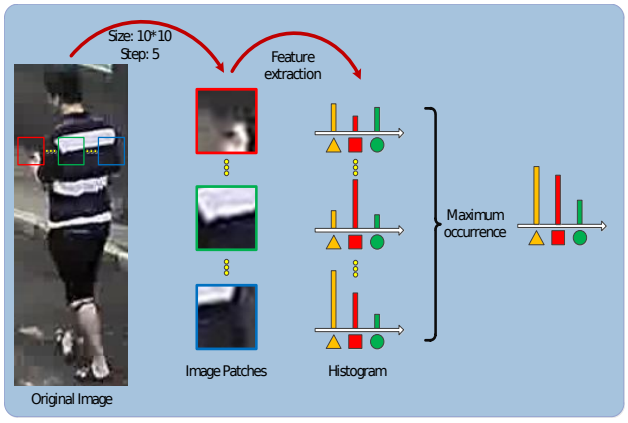


Figure 3.27: Overview of the LOMO feature extraction system (Liao et al., 2015). Local patches of size 10×10 are extracted and different types of histogram features, including HSV and SILTP are extracted. For patches in same horizontal position, largest bin values are taken for each bin.

3.4.1 Preprocessing

Before extracting features, each image is preprocessed by applying the Retinex algorithm (Jobson et al., 1997) which aims at enhancing color information, especially in shadowed regions, making the colors more similar to the colors perceived by humans. This is done by convolute the image with a Gaussian filter and subtract it from the original image, following Equation 3.44.

$$R_i(x,y) = \log(I_i(x,y)) - \log(F(x,y) * I_i(x,y)), \tag{3.44}$$

where R_i is the single-scale retinex output for channel i and I_i is the intensity value and F is a Gaussian filter calculated as $F(x,y) = K \exp^{-\frac{x^2+y^2}{\sigma^2}}$ where σ is the standard deviation of the Gaussian filters which determines the scale and K is chosen such that $\int\int F(x,y)dxdy = 1$. A smaller σ provides a better dynamic range compression while a larger value cause better color constancy under different lighting conditions.

By combining single-scale retinex outputs at different scales, both advantages are combined and a better representation is made. This is done by simple summation following Equation 3.45

$$R_{MSR,i} = \sum_{n=1}^{N} \omega_n R_{i,n}, \tag{3.45}$$

where ω_n is the weight at scale n, N is the total number of scales and $R_{MSR,i}$ is the multi-scale retinex. In this context, scale values of 5 and 20 are utilized. Figure 3.28 show an example of in image before and after applying the Retinex algorithm.

(a) (b)

Figure 3.28: Example of Retinex. (a) shows an image before applying Retniex, while (b) shows the image after applying the Retinex. It can be seen that the overall color representation is enhanced.

3.4.2 Feature extraction

Having preprocessed the images, features are extracted at locally from patches of size 10×10 pixels with a step size of 5 pixels. For each patch, a joint HSV histogram is extracted with 8 bins for each channel along with Scale Invariant Local Ternary Pattern (SILTP) features (Liao et al., 2010) which, compared to more commonly known LBP features, are more robust to image noise and at the same time achieves invariance to intensity scale changes. This is achieved by adding a scaling parameter τ that indicates the comparing

range, having a smaller value more sensitive to noise but better discrimination. From the usual LBP, the image is encoded as shown in Figure 3.29 and given in Equation 3.46.

$$s(I_c, I_n) = \begin{cases} 1, & \text{if } I_n - I_c \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.46}$$

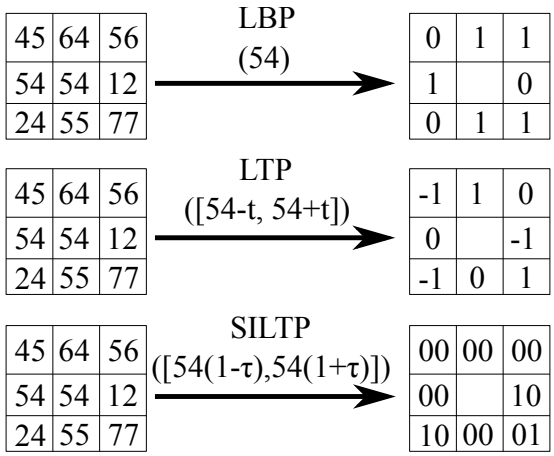


Figure 3.29: Example of different encoding using LBP, LTP and SILTP. For LTP, $t = 5$ and for SILTP, $\tau = 0.2$.

where I_c is the intensity of the center pixel and I_n the intensity of the compared neighbor pixel. To make the LBP robust to noise, Local Ternary Patter (LTP) was proposed which added a threshold, t, when comparing the center pixel to the neighbors and the encoding hence changed to that of Equation 3.47 as shown in Figure 3.29.

$$s_t(I_c, I_n) = \begin{cases} 1, & \text{if } |I_n - I_c| + I_c \geq I_c + t \\ 0, & \text{if } |I_n - I_c| < t \\ -1, & \text{if } |I_n - I_c| + I_c \leq I_c - t \end{cases} \tag{3.47}$$

Multiplying the intensities with a predefined value instead, furthermore add robustness to scaled intensities. The encoding thus changes to Equation 3.48 as shown in Figure 3.29.

$$s_\tau(I_c, I_n) = \begin{cases} 01, & \text{if } I_n > (1+\tau) I_c \\ 10, & \text{if } I_n < (1-\tau) I_c \\ 00, & \text{otherwise} \\ 11, & \text{undefined} \end{cases} \tag{3.48}$$

Having the encoded information, each pixel can be represented by a decimal value calculated from the encoded representation following Equation 3.49.

$$S_\tau(x_c, y_c) = \sum_{n=1}^{N-1} 3^{n-1} (s_{\tau,01}(I_c, I_n) + s_{\tau,10}(I_c, I_n)) \cdot 2, \tag{3.49}$$

where N is the number of neighboring points taken into account which can be either 4 or 8 and $\{s_{\tau,01}, s_{\tau,10}\}$ are the two cases of binary encoding defining either values larger or lower than the threshold. From the new SILTP images, histograms are created at each image scale.

Having extracted features, histograms of all patches at the same horizontal level are compared and maximized as shown in Figure 3.27, in order to make up for view point changes between camera views.

Further, the image is down sampled two times by applying a 2×2 average pooling kernel and similar features are extracted. Finally, log transformation is applied to the features to suppress large bin values. Figure 3.30 shows an example of the color feature representations of two different persons in two different views. In this case, only the largest bin values are taken into account to emphasize on the more discriminative differences. It can be seen that the blue colors from person 1 are captured in both views while less color is captured at the second person.

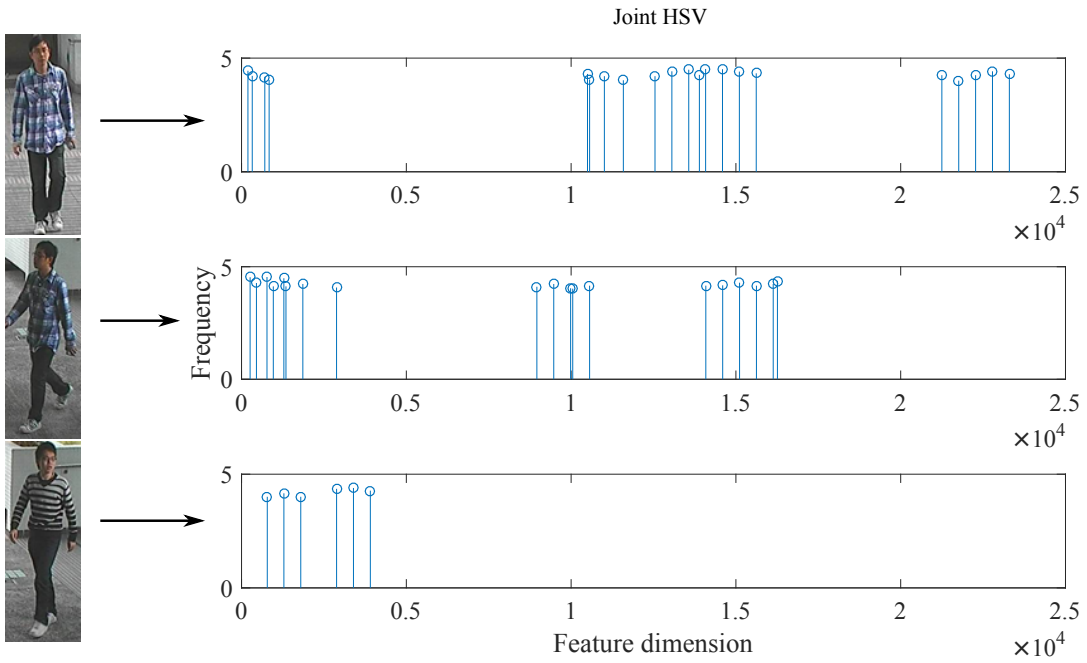


Figure 3.30: Example of HSV color histograms extracted from two different person in two different views, including only the largest bin values. More color features are included for the first person due to his blue chequered shirt.

Meanwhile, Figure 3.31 show the texture features from the very same persons. Comparing the texture in these images, there are to some extend similarities with horizontal lines causing the features to be more similar. Using the combination of the color and texture features thus makes the algorithm better at distinguishing between these persons.

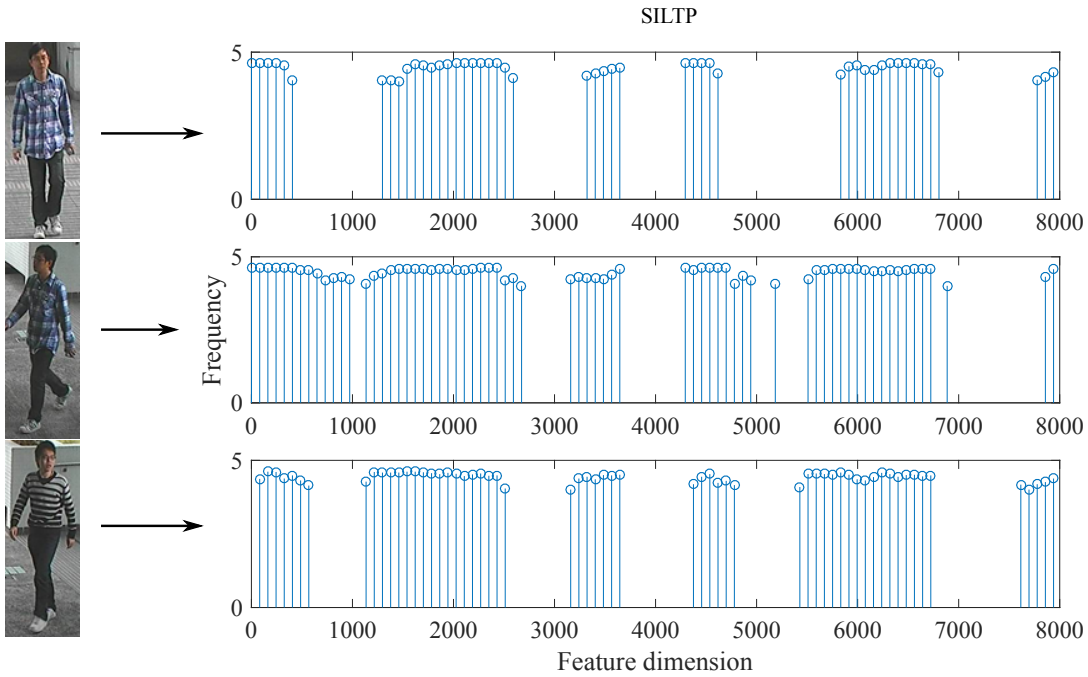


Figure 3.31: Example of SILTP features from two different persons in two different views. Similarities are seen as both persons are wearing a clothe with horizontal texture.

3.4.3 Metric learning

As similarity measure, an extended version of KISSME is applied, named Cross-view Quadratic Discriminant Analysis (XQDA). Because of the similarity to KISSME, it is only slightly slower (≈0.5 seconds) when training. KISSME (Koestinger et al., 2012) is used to calculate similarity between images. The algorithm utilizes Mahalanobis distance given in Equation 3.50.

$$d_M^2(x_i, x_j) = (x_i - x_j)^T M (x_i - x_j), \tag{3.50}$$

where the matrix, M, is estimated by taking the difference between similar and dissimilar image pairs into account as defined by Equation 3.51.

$$\begin{aligned} M &= \Sigma_{y_{ij}=1}^{-1} - \Sigma_{y_{ij}=0}^{-1} \\ \Sigma_{y_{ij}=1} &= \sum_{y_{ij}=1} (x_i - x_j)(x_i - x_j)^T \qquad \Sigma_{y_{ij}=0} = \sum_{y_{ij}=0} (x_i - x_j)(x_i - x_j)^T, \end{aligned} \tag{3.51}$$

where y_{ij} indicates similar (1) or dissimilar (0) image pairs from the training data.

The covariance matrix, M, is often estimated after applying PCA to reduce the feature dimension as it is not desired to store high-dimensional features further provides a better estimate. But as the PCA does not consider metric learning, this case is not suitable. Therefore, XQDA considers a projection matrix, W, to map features from different views to a common subspace before metric learning. This matrix is estimated by looking at the variance of the within-class and between-class differences. Including a basis, w, the variances can be given as $\sigma_I(w) = w^T \Sigma_I w$ and $\sigma_E(w) = w^T \Sigma_E w$. As it is desired to maximize $\frac{\sigma_E(w)}{\sigma_I(w)}$, an objection function following Equation 3.52 can be defined.

$$\max_w w^T \Sigma_E w, \quad \text{s.t. } w^T \Sigma_I w = 1 \tag{3.52}$$

This can be solved by the generalized eigenvalue decomposition similar to Linear Discriminant Analysis (LDA). The solution to this is simply the N largest eigenvalues of $\Sigma_I^{-1}\Sigma_E$ where N is the number of dimensions kept dimensions in the learned subspace.

Having the projection matrix, W, the distance function hence changes to Equation 3.53.

$$\begin{aligned} d_M^2(x_i, x_j) &= (x_i - x_j)^T W M W^T (x_i - x_j) \\ M &= \Sigma'^{-1}_{y_{ij}=1} - \Sigma'^{-1}_{y_{ij}=0} \end{aligned} \tag{3.53}$$

where $\Sigma'_{y_{ij}=1} = W^T \Sigma_{y_{ij}=1} W$ and $\Sigma'_{y_{ij}=0} = W^T \Sigma_{y_{ij}=0} W$. The learning process is summarized in Pseudocode 7. In the fusing scheme, this system will be named LOMO.

input: X_1, X_2 - feature matrices for training set from view A and B
output: W, M - projection and covariance matrix

; // Calculate similarity and dissimilarity matrices following Equation 3.51
for *each image pair* i, j *in* X_1 *and* X_2 **do**
    $\Sigma_{y_{ij}=1} \leftarrow \sum_{y_{ij}=1} (x_i - x_j)(x_i - x_j)^T$;
    $\Sigma_{y_{ij}=0} \leftarrow \sum_{y_{ij}=0} (x_i - x_j)(x_i - x_j)^T$;
end
; // Find W by solving objection function given in Equation 3.52
$W \leftarrow \text{eig}(\Sigma_I \Sigma_E^{-1})$;
; // Calculate M following Equation 3.53
$\Sigma'_{y_{ij}=1} \leftarrow W^T \Sigma_{y_{ij}=1} W$;
$\Sigma'_{y_{ij}=0} \leftarrow W^T \Sigma_{y_{ij}=0} W$;
$M \leftarrow \Sigma'^{-1}_{y_{ij}=1} - \Sigma'^{-1}_{y_{ij}=0}$;
return W, M ;

Pseudocode 7: Algorithm for calculating projection matrix W and covariance matrix M.

3.5 Fusion Scheme

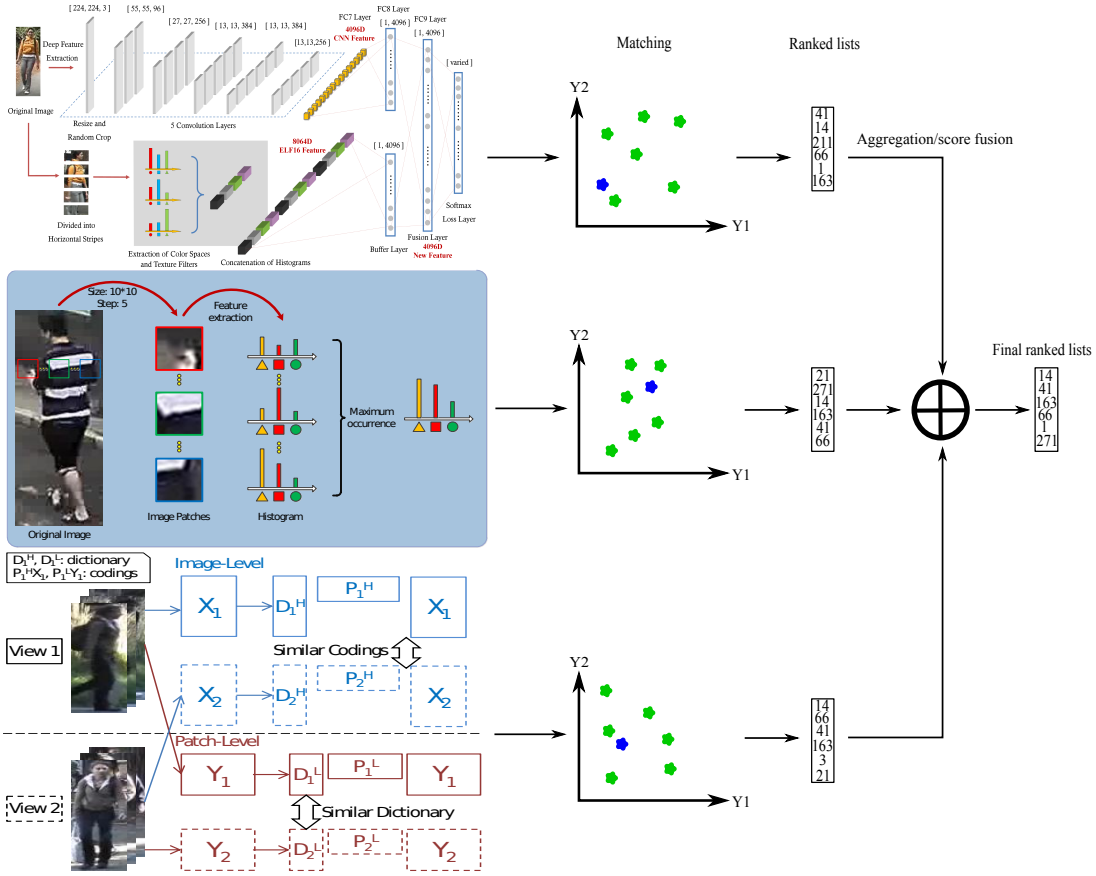


Figure 3.32: ...

Figure 3.32 show the fusing scheme. When fusing results from the three presented systems, rank aggregation using Stuart algorithm and score-level fusion using the product rule are applied in two separate tests.

For rank aggregation, the procedure follows Pseudocode 8 i.e. for each probe, a ranked list is returned indicating the most similar queries. These lists are concatenated to a single rank matrix $R \in \mathbb{R}^{3\times q}$ with q indicating the number of matched queries. The matrix is then passed to the rank aggregation function

which calculated the new aggregated ranks using Equation 2.2.

input: $r_{ffn}, r_{dict}, r_{lomo}$ - ranked lists from each system
output: r_{new} - aggregated ranks

$K \leftarrow$ #systems
for *each probe p in lists* **do**
    ;// Combine ranks for probe
    $r_p \leftarrow r_{ffn,p}, r_{dict,p}, r_{lomo,p}$;
    ;// Normalize ranks
    $r_p \leftarrow \frac{r_p}{\#p \text{ in lists}}$;
    ;// Calculate new rank using Equation 2.2
    $r_{new,p} \leftarrow K! \cdot V_{K+1}$;
end
return r_{new} ;

Pseudocode 8: Algorithm to perform rank aggregation.

The score-level fusion includes more intermediate steps, including score normalization and weight assignment. As the values in the calculated distance matrices might differ depending on the system, the scores need to be normalized to a common metric to remove any bias. Here, min-max normalization is used following Equation 3.54 as it has shown to be superior compared to other normalization algorithms such as decimal scaling or z-score normalization.

$$\hat{d}_M^2(x_i, x_j) = \frac{d_M^2(x_i - x_j) - \min d_M^2(\mathbf{x})}{\max d_M^2(\mathbf{x}) - \min d_M^2(\mathbf{x})} \tag{3.54}$$

Afterwards, test score curves are defined for each person using the normalized scores. Furthermore, reference curves are defined from the scores calculated on the training data, in order to calculate weights associated with each test score. For each test score curve, the Euclidean distance is calculated to all reference curves and the K-nearest reference curves are then averaged and subtracted the test curve. Testing different numbers of K, the best results were achieved by setting $K = 15$.

In (Zheng et al., 2015b), the weight for each feature is determined by the area under the subtracted score curve (AUC) using Equation 3.55.

$$w_q^{(i)} = \frac{\frac{1}{A_i}}{\sum_{k=1}^{K} \frac{1}{A_k}}, \tag{3.55}$$

where A_i is the area under score curve for the i'th feature and K is the number of features. In that context, higher scores are better resulting in larger weights by lower AUC.

As distance functions in this case are used to calculate similarity, smaller values indicate better results. Because of this, the reference curve and weight assignment have to be dealt with differently. The average reference curve is still calculated same way as before, an example of a test curve and such reference curve is shown in Figure 3.33 (a).

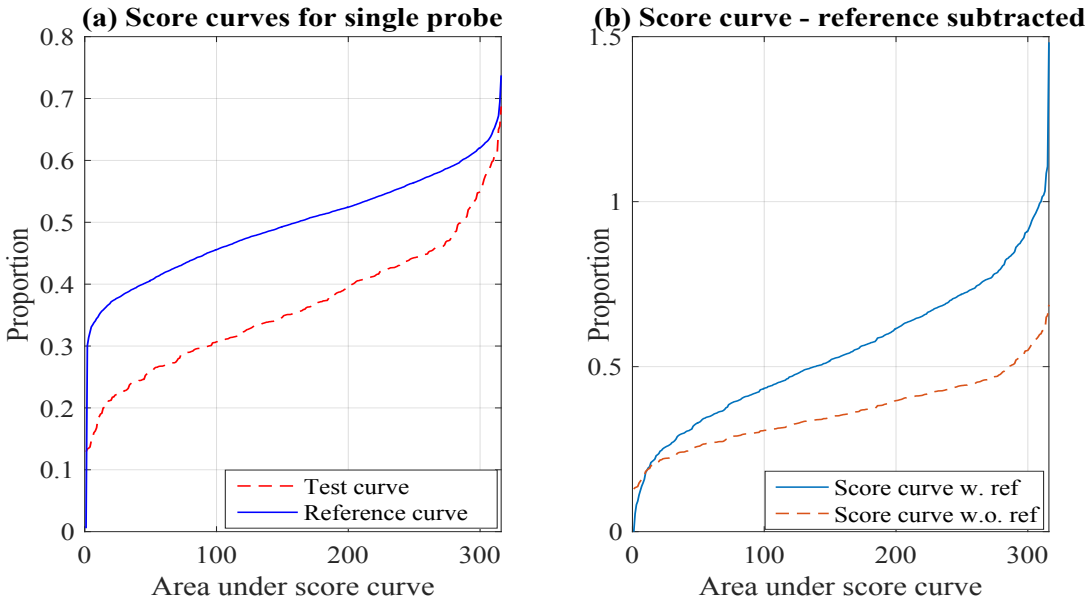


Figure 3.33: Example of a test curve and the corresponding average reference curve for a single probe.

As it is desired to keep the scores for the most similar low and high for the rest, the AUC should be large. Therefore, the reference curve is horizontally mirrored before subtracted the test curve. This way, the AUC is enlarged while keeping the similarity between the probe and the most similar queries low as also shown in Figure 3.33 (b).

As it is desired to have a large AUC, the weight assignment is changed to be calculated by Equation 3.56.

$$w_q^{(i)} = \frac{A_i}{\sum_{k=1}^{K} A_k} \tag{3.56}$$

Having assigned weights, the product rule given by Equation 2.5 is used to fuse the results. An

overview of the score-level fusion is shown in Pseudocode 9.

input: $dist_{ffn}, dist_{dict}, dist_{lomo}$ - score test matrices from each system
$dist_{ffn,ref}, dist_{dict,ref}, dist_{lomo,ref}$ - reference scores matrices
output: $dist_{fuse}$ - score-fused matrix

; // Normalize scores using Equation 3.54
for *each feature type k* **do**
  $dist_k \leftarrow \frac{dist_k - \min(dist_k)}{\max(dist_k) - \min(dist_k)}$
end
for *each probe p in dist* **do**
  for *each feature type k* **do**
    ; // Calculate distance between test score and reference scores
    $sco_{dist,k} \leftarrow \mathcal{D}(dist_{p,k}, distk, ref)$;
    ; // Find a subtract mean of K most similar reference curve
    $sco_{min,k} \leftarrow dist_{p,k} - \frac{1}{K}\sum_{k} \min(sco_{dist,k})$;
    ; // Find weight as AUC using Equation 3.56
    $w_p^k \leftarrow \frac{AUC(sco_{min,k})}{\sum_{k}^{K} AUC(sco_{min,k})}$;
  end
  ; // Fuse scores using Equation 2.5
  $dist_{fuse,p} \leftarrow dist_{p,ffn}^{w_p^{ffn}} \cdot dist_{p,dict}^{w_p^{dict}} \cdot dist_{p,lomo}^{w_p^{lomo}}$;
end
return $dist_{fuse}$;

Pseudocode 9: Algorithm to perform score-level fusion.

3.6 Fusion Tests

Two types of tests are conducted in order to analyse the performance of the fusion scheme:

- **Accuracy test** – conducted in order to see improvement, not only compared to the individual systems, but also other proposed state-of-the-art systems.
- **Processing time test** – conducted in order to analyse whether it makes sense to combine different systems leading to higher computational time compared to the increased accuracy.

First, tests of the accuracy are conducted on the datasets presented in section 2.4 using the defined protocols. Next, the processing time is tested and compared to the achieved accuracies in order to conclude the benefit of the fusion scheme. All results are provided in Appendix C.

3.6.1 Accuracy test

Tests are conducted using different combination of the fusing scheme, pair-wise and all three at once, in order to see how well the systems complement each other. In each test, subscripts are used to distinguish

between the rank aggregated result, shown by the subscript *agg*, and score-fused result, shown by the subscript *sco*. Further, **All** refers to the result by fusing all three systems.

The first tests are conducted using ranking aggregation and the resulting CMC curves are shown in Figure 3.34. In reality, it is not desired to have a large list of potential candidates returned, therefore, rankings are shown for rank -1, -5, -10 and -20. Results from both rank aggregation and score-level fusion are summarized in Table 3.5-3.8.

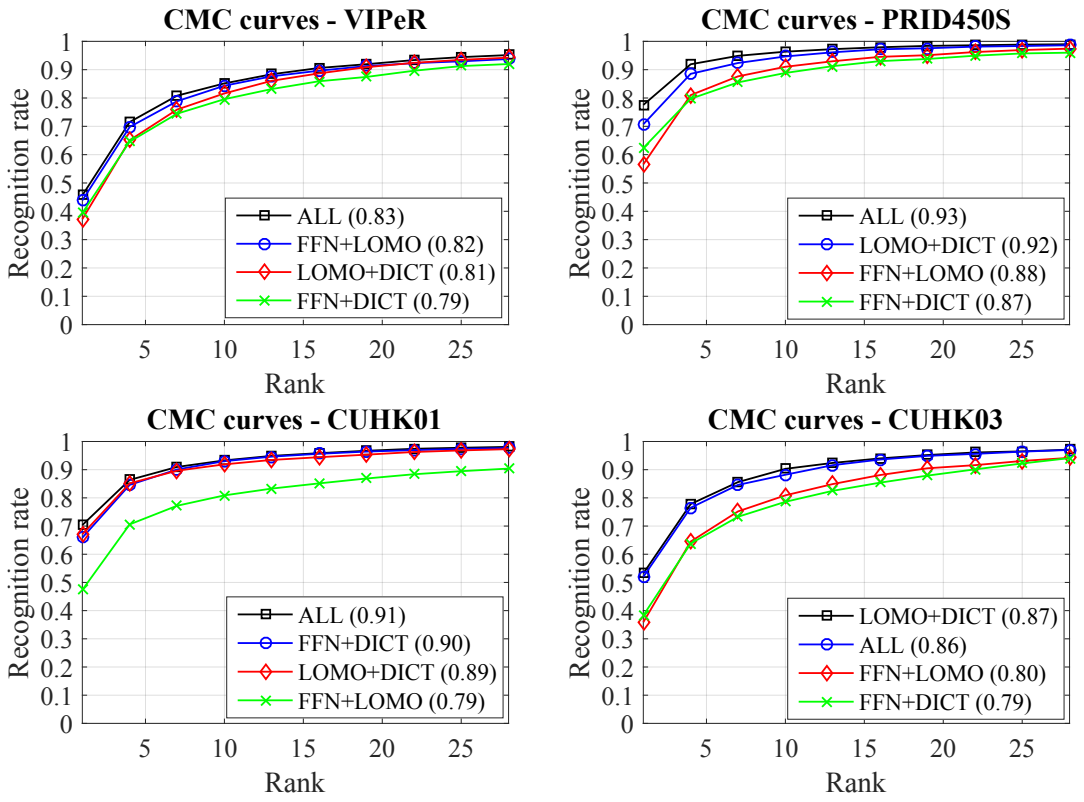


Figure 3.34: Results on four different datasets fusing by applying rank aggregation.

For all datasets except CUHK03, fusing all three systems provides the best results. For CUHK03, the results from FFN are much worse than DICT and LOMO which is the reason the fused results being worse than only fusing LOMO and DICT. For VIPeR, the fused result is 1.87% better than fusing only FFN and LOMO. For PRID450S, the rank-1 accuracy is increased by 6.85% while the accuracy is increased 3.29% at CUHK01. Compared to the best individual results, the fusing increases the rank-1 accuracies with 7.91%, 17.83%, 16.91% and 9.05% for VIPeR, PRID450S, CUHK01 and CUHK03, respectively. It therefore seems beneficial to late fuse different systems to increase the overall accuracy.

Comparing the pairwise results, there is a pattern that the best and worst individually performing system cause the worst pairwise combination. For both VIPeR and CUHK01, LOMO is the best performing while FFN is the worst. Looking at the pairwise combinations in these datasets, FFN+LOMO is the worst of the three. This shows that different low-level features are an important factor when doing fusion in order to have good complementation that increases the overall performance.

The second tests are made using score-level fusion and the resulting CMC curves are shown in Figure 3.35.

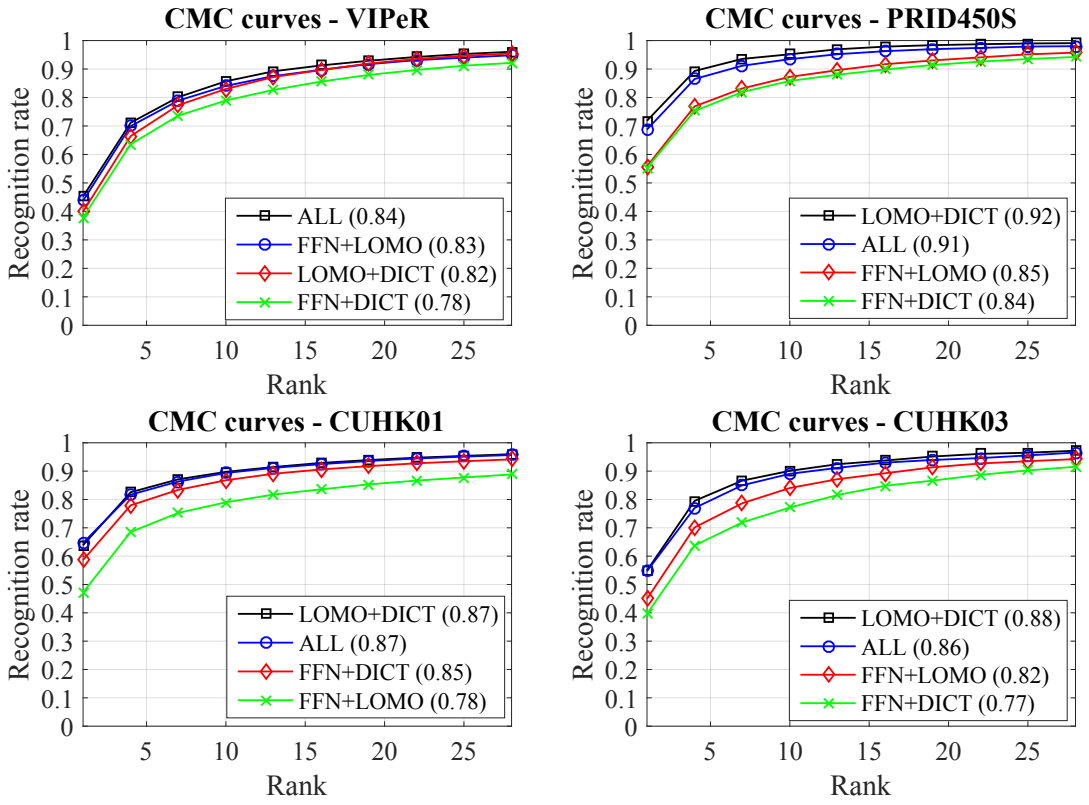


Figure 3.35: Results on four different datasets by applying score-level fusion.

Unlike ranking aggregation, score-level fusion does not seem to perform better when combining three systems than just two. In case of VIPeR, the fused result is 1.45% better than fusing only LOMO and FFN. For CUHK01, the result is slightly increase by 0.81% while being slightly worse at CUHK03, 1.03% below fusing LOMO and DICT. Compared to the individual results, the fused are still improved by 7.52%, 9.03%, 11.23% and 11.51% for VIPeR, PRID450S, CUHK01 and CUHK03, respectively.

The reason for getting worse results when fusing all systems, might lie in the calculated distances which more easily affects the ranked positions if too different. This will be further analysed in subsection 3.6.2.

Comparing aggregation and score-level fusion, the results also show ranking aggregation to be the best way to apply late fusion having better results at both VIPeR, PRID450S and CUHK01. A reason for this might be that the fusion solely relies on the ranks and not underlying distance which may cause problems in score-level fusion. On the other hand, score-level fusion has the advantage of being more robust to single bad performing systems as seen with CUHK03 at which the score-fused result is better than the rank aggregated. Furthermore, as the training set is much larger compared to the other datasets, there is a higher probability of finding a similar reference curve at systems that performs well, leaving higher weights.

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	**45.63**	**75.06**	85.16	92.47
All_{sco}	45.24	75.02	**85.70**	**93.47**
FFN+$LOMO_{agg}$	43.86	73.89	84.34	91.84
FFN+$LOMO_{sco}$	43.73	73.81	84.08	92.17
FFN+$DICT_{agg}$	39.40	68.04	79.59	88.23
FFN+$DICT_{sco}$	37.85	67.56	78.96	88.50
LOMO+$DICT_{agg}$	37.22	69.84	81.61	91.46
LOMO+$DICT_{sco}$	39.94	71.06	83.01	92.48
FFN	30.70	57.72	69.15	78.96
LOMO	37.72	67.59	80.06	90.51
DICT	28.23	55.41	70.54	82.97

Table 3.5: Results on VIPeR (p=316) for both rank aggregation(*agg*) and score-level fusion(*sco*). The best results are in bolt.

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	**77.56**	**93.47**	**96.09**	98.44
All_{sco}	68.76	88.49	93.47	97.11
FFN+$LOMO_{agg}$	56.58	84.27	91.02	95.29
FFN+$LOMO_{sco}$	55.82	79.60	87.24	93.36
FFN+$DICT_{agg}$	62.36	82.04	88.84	94.09
FFN+$DICT_{sco}$	55.29	78.13	85.82	91.84
LOMO+$DICT_{agg}$	70.71	90.00	94.67	97.69
LOMO+$DICT_{sco}$	71.84	91.24	95.18	**98.56**
FFN	37.11	61.42	73.38	82.84
LOMO	59.73	82.58	89.69	94.80
DICT	38.00	54.80	62.22	73.11

Table 3.6: Results on PRID450S (p=225) for both rank aggregation(*agg*) and score-level fusion(*sco*). The best results are in bolt.

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	**70.35**	**88.46**	**93.29**	**96.95**
All_{sco}	64.67	83.81	89.35	93.90
FFN+$LOMO_{agg}$	47.35	73.19	80.88	87.28
FFN+$LOMO_{sco}$	47.17	71.39	78.99	85.65
FFN+$DICT_{agg}$	65.86	86.85	93.02	96.58
FFN+$DICT_{sco}$	58.60	80.11	86.78	92.08
LOMO+$DICT_{agg}$	67.06	87.16	91.89	96.62
LOMO+$DICT_{sco}$	63.86	84.42	89.74	94.26
FFN	32.28	56.95	66.73	76.03
LOMO	41.77	66.19	74.86	83.07
DICT	53.44	78.85	86.95	93.11

Table 3.7: Results on CUHK01 (p=486) for both rank aggregation(*agg*) and score-level fusion(*sco*). The best results are in bolt.

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	52.25	80.40	88.95	94.90
All_{sco}	54.72	81.25	89.40	94.60
FFN+$LOMO_{agg}$	35.75	69.00	81.20	91.10
FFN+$LOMO_{sco}$	44.75	74.20	84.08	91.88
FFN+$DICT_{agg}$	37.90	67.05	79.00	89.70
FFN+$DICT_{sco}$	39.68	67.33	78.28	87.80
LOMO+$DICT_{agg}$	53.00	81.35	90.15	**96.20**
LOMO+$DICT_{sco}$	**55.75**	**83.27**	**90.47**	95.87
FFN	20.30	42.20	53.90	68.70
LOMO	43.20	75.60	87.10	93.30
DICT	33.70	62.60	72.90	85.90

Table 3.8: Results on CUHK03 (p=100) for both rank aggregation(*agg*) and score-level fusion(*sco*). The best results are in bolt.

Comparison to state-of-the-art

In order to see the improvement, the obtained results are compared to previous state-of-the-art results. This includes some the systems presented in section 1.1, namely the original FFN feature fused with LOMO features, the original Mirror-KMFA, MLF, Salient Color Name based Color Descriptor (SCNCD) which utilizes color names to describe different parts of the human body trained with KISSME and Ensemble Color Model (ECM) which makes use of color names along with regular color features, also utilizing KISSME as metric learning algorithm. Furthermore, the state-of-the-art CNN also re-implemented is compared.

The compared result are summarized in Table 3.9-3.12.

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	45.63	75.06	85.16	92.47
All_{sco}	45.24	75.02	85.70	93.47
FFN+LOMO (Wu et al., 2016b)	**51.06**	**81.01**	**91.39**	**96.90**
Mirror-KMFA (Chen et al., 2015b)	42.97	75.82	87.28	94.84
SCNCD (Yang et al., 2014)	37.80	68.50	81.20	90.40
Deep Re-id (Ahmed et al., 2015)	34.81	63.72	76.24	81.90
ECM (Liu et al., 2015)	38.90	67.80	78.40	88.90
MLF+LADF (Zhao et al., 2014)	43.39	73.04	87.28	93.70

Table 3.9: Comparison between the fused systems and other state-of-the-art systems on the VIPeR dataset (p=316).

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	**77.56**	**93.47**	**96.09**	**98.44**
All_{sco}	68.76	88.49	93.47	97.11
FFN+LOMO (Wu et al., 2016b)	66.62	86.84	92.84	96.89
Mirror-KMFA (Chen et al., 2015b)	55.42	63.72	87.72	93.87
ECM (Liu et al., 2015)	41.90	66.30	76.90	84.90
SCNCD (Yang et al., 2014)	41.60	68.90	79.40	87.80

Table 3.10: Comparison between the fused systems and other state-of-the-art systems on the PRID450S dataset (p=225).

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	**70.35**	**88.46**	**93.29**	**96.95**
All_{sco}	64.67	83.81	89.35	93.90
FFN+LOMO (Wu et al., 2016b)	55.51	78.40	83.68	92.59
Deep Re-id (Ahmed et al., 2015)	47.53	72.10	80.53	88.49
Mirror-KMFA (Chen et al., 2015b)	40.40	64.63	75.34	84.08
MLF (Zhao et al., 2014)	34.30	55.12	64.91	74.53

Table 3.11: Comparison between the fused systems and other state-of-the-art systems on the CUHK01 dataset (p=486).

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	52.25	80.40	88.95	94.90
All_{sco}	54.72	81.25	89.40	94.60
LOMO+XQDA (Liao et al., 2015)	52.20	82.23	**92.14**	96.25
Deep Re-id (Ahmed et al., 2015)	**54.74**	**86.42**	91.50	**97.31**
FPNN (Li et al., 2014)	20.65	51.32	68.74	83.06

Table 3.12: Comparison between the fused systems and other state-of-the-art systems on the CUHK03 dataset (p=100).

In case of CUHK01 and PRID450S, both the score-level and rank aggregated results are better than previous proposed systems. For CUHK01, the best result is improved by 15.04% while the best result is improved by 10.94% on PRID450S. For VIPeR, feature fusion of FFN and LOMO provides better results than the achieved. This should, though, be held in relation to the fact that FFN in this fusing scheme achieved results slightly below the original despite using the same metric learning algorithm. With the knowledge of the missing details, the overall accuracy will most likely be improved from a better performance by FFN.

Lastly, the fused results on CUHK03 provides similar results to the neural network proposed by (Ahmed et al., 2015). This CNN, though, had the advantage of training on multiple images of each person, hence, begin able to better adapt to new data.

3.6.2 Analysis of fusing

In order to get a better understanding of how the results of the three systems affect the fusion, results from each system are compared with the fused in four different cases:

1. The true match is ranked within the top-5 for all individual systems.
2. The true matched is ranked uneven, though, within top-10 for all systems.
3. The true match is ranked high for the first two of the systems, while ranked badly by the last.
4. The true match is ranked high by the first system, while ranked badly by the last two.

In all cases, examples from CUHK03 are being used (Li et al., 2014).

Figure 3.36: Example of case 1 in which true match is ranked high by all three systems. Green boxes indicate true match and yellow numbers in images are corresponding normalized distances.

Figure 3.36 shows the first case in which LOMO ranks the true match as the most similar, while it is ranked third by both FFN and DICT. At both rank aggregation and score-level fusion this cause the true match to be most similar. Generally, the distances for DICT are higher than the other two which may affect the results either positively or negatively. In this case, even though, the true match is ranked 1, the difference between the most and second most similar becomes smaller after fusing (0.0012) than looking at LOMO alone (0.0743). In order to verify the correctness of the classification, this difference is desired to be as large as possible.

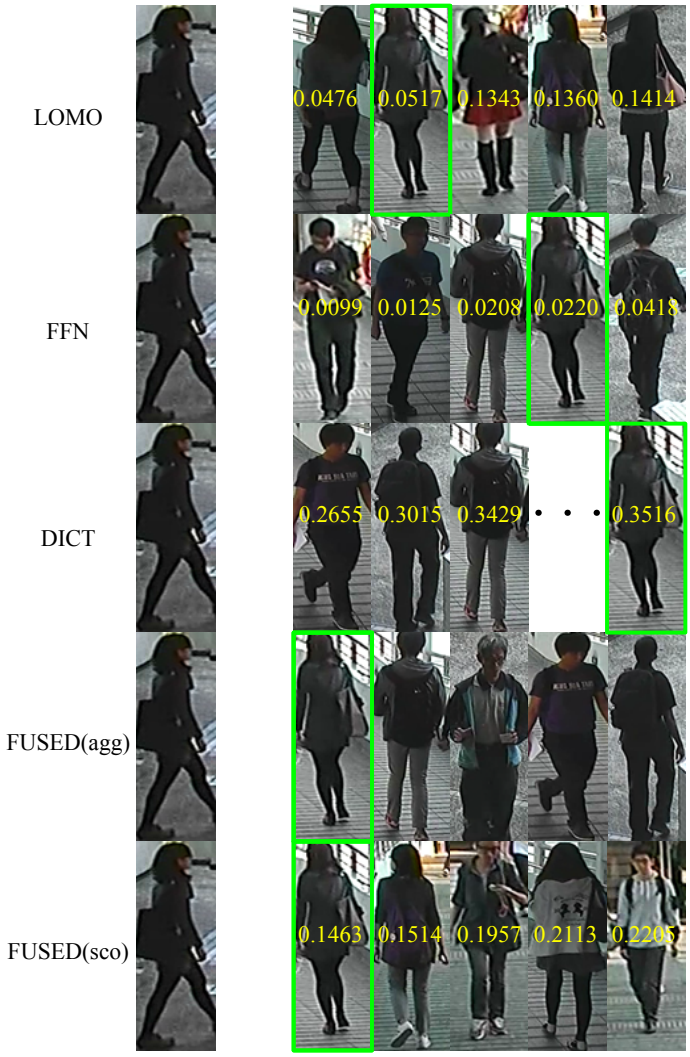


Figure 3.37: Example of case 2 in which true match is differently for three systems, still within top-10. For DICT, the true match is ranked outside top-5 indicated by the dots. Green boxes indicate true match and yellow numbers in images are corresponding normalized distances.

An example of the second case is shown in Figure 3.37. Here, the true matched is ranked 2,4 and 7 by LOMO, FFN and DICT, respectively. But as also seen, each system ranks the remaining persons differently, making the true match ranked first when fusing the results. This show that fusing is beneficial, even though, results may vary, at least when still within a certain range.

Figure 3.38: Example of case 3 in which true match is ranked top for two systems, but only 21 by the last, indicated by the dots. Green boxes indicate true match and yellow numbers in images are corresponding normalized distances.

The third case is shown in Figure 3.38. Here, the true match is ranked 1 by both LOMO and DICT, but only 21 by FFN. Due to this, the aggregated result only rank the true match 2. This shows one of the disadvantages at aggregation when outliers are present. Meanwhile, score-level fusion is more robust to such situation, as the distances are taken into account and the two best systems are thus weighted higher than the last. Compared to the other cases, a backpack is shown in one camera view while not in the other which is a challenge when dealing with a high-level representation which is the case for FFN.

An example of the last case is shown in Figure 3.39 in which only LOMO ranks the true match as 1, while FFN ranks it as 23 and DICT as 11. This cause the true match to not even be in top-5 when aggregating the results and show the importance of having at least two of the systems ranking the true match within top-10. The score-fused result is better as the true match is ranked 5 which is most likely due to the distance which is low at, not only LOMO, but also FFN despite the ranking. This cause LOMO to be the dominating factor and leaves the true match at the score-fused result to have just a slightly higher distance than this system alone.

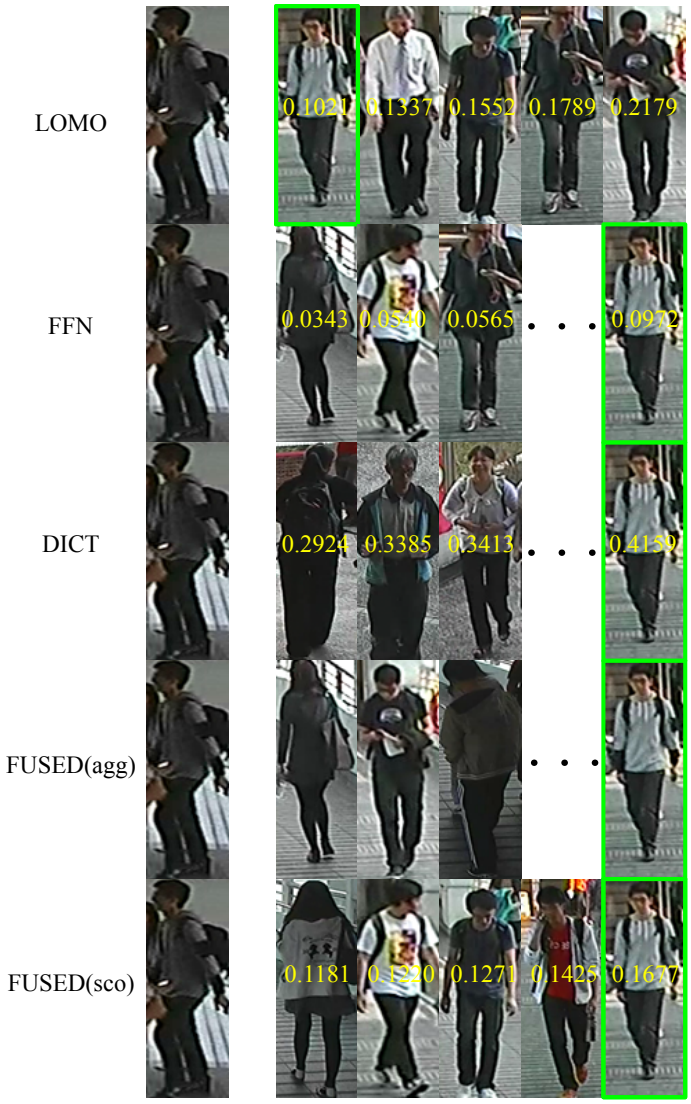


Figure 3.39: Example of case 4 in which true match is ranked top for one system, but only 23 and 11 by FFN and DICT, respectively, indicated by the dots. Green boxes indicate true match and yellow numbers in images are corresponding normalized distances.

As seen in all cases, the different systems might rank the true match differently, but are also ranking the false matches differently which is an important factor as it leaves a higher probability for the true match to be ranked high. In order to have a true match ranked as the most similar, all three systems must rank the match within top-10 when it comes to rank aggregation, while one system may rank it low if the other two ranks it high when it comes to score-level fusion.

3.6.3 Processing time

In order to analyse whether it makes sense to make use of late fusion of different systems, tests of the processing time for the metric learning algorithms, matching and fusion techniques are conducted and compared to the increase in accuracy.

The tests are made by average the timings of 10 test iterations on the VIPeR dataset using an Intel i7-4700MQ CPU @ 2.4GHz and 16GB of RAM, and the results are shown in Table 3.13.

As shown, training and matching using cross-view dictionary takes up large part of the total process-

Mirror-KMFA(train)	Mirror-KMFA(match)	XQDA(train)	XQDA(match)	Dict.(train)	Dict.(match)
30.33	10.30	1.79	0.18	49.71	292.38

Table 3.13: Average timings for training and matching in seconds over 10 iterations on VIPeR dataset.

Rank aggregation	Score-level
6.26	0.12

Table 3.14: Average timings for late fusion in seconds over 10 iterations on VIPeR dataset.

ing time. If only looking at matching and score-level fusion, dictionary matching takes up 96.5% of the 302.98 seconds it takes, in total. This is due to the patch matching in which 30 distance calculations are carried out when just comparing a single image pair with each 30×10 patches. Training is not as critical as it can be done offline, but for testing this is critical if an algorithm should run in real-time and a different matching algorithm should therefore be considered. The main part of matching at KMFA is due to the more complex χ^2 kernel applied. If matching time should be reduced, regular MFA can be utilized or a simple linear kernel with a potential change of drop in accuracy.

Furthermore, it can be seen that both ways of fusion are fast, though, with score-level fusion being up to 30 times faster making it the most desirable. Though, compared to the achieved accuracies, changes should be made as the overall achieved results show rank aggregation to be the better. This will be further discussed in section 5.1. Compared to the accuracies for to the total processing time for matching and fusion, it does seem beneficial as it only improves the processing time by 2.1% and 0.04% for rank aggregation and score-level fusion, respectively. Leaving out the dictionary matching, this corresponds to a processing time of 33.5 ms and 53 ms per person when matching and fusing for aggregation and score-level, respectively. This corresponds to $\approx$30fps and $\approx$18fps, making the system able to run in real-time if score-level fusion is applied. This would also require fast feature extraction techniques that does not add any particular increase in processing time.

4. Acceptance Test

This chapter includes an acceptance test reflecting real-life situations to get a more realistic view on the performance. This include a multi-shot and cross-dataset test.

In order to get a more realistic view on the performance of the proposed system, an acceptance test is conducted, based on two real world scenarios:

- A known dataset for the specific purpose is not available and it is undesired to make a new training set in every new context.
- Videos are available in different views making it possible to extract several images of each person.

From these two scenarios, two different tests are conducted. The first test deals with a cross-dataset scenario, in which training is performed on one dataset while test is conducted on another. The second test makes use of multiple images of each person which can be used to increase the robustness of the representation of the persons.

4.1 Cross-dataset test

In a a real world scenario, the gallery is potentially large and the id's of the persons included are unknown. Therefore, videos from different camera views need to be examined, in order to find and label persons which are present in two or more camera views. This is a comprehensive and time consuming task which is necessary, in order to make use of a supervised system with a high accuracy.

To avoid this task, another option is to train on an already known dataset and then test on the unknown. This increases the challenge as environmental settings and camera views will be totally different, things that, even though still different, had certain similarity within each dataset. An example of this is shown in Figure 4.1.

Figure 4.1: Example of differences between dataset. Left and middle image is from of the same person in the VIPeR dataset and the right image is from CUHK01 dataset. Even though, there are differences in view and lighting within the VIPeR images, similarities are found in the environment and vertical view.

To have a well working system, it should be able to generalize, thus, still have a decent accuracy independent of which dataset is used for training. Tests are therefore also conducted with cross-dataset settings. Only a few proposed systems have conducted cross-dataset tests with the combination of CUHK02 (Li et al., 2012) and VIPeR being used. CUHK02 is an extension on the also tested dataset, CUHK01 and consists of an additional 845 person, bringing the total number of persons to 1816. In this situation, training is performed on CUHK02, while testing is done on VIPeR using the same number of persons as in within-dataset test. The results are available in the attached appendix and is summarized in Figure 4.2 and Table 4.1. The results and scores are further available in Appendix C.

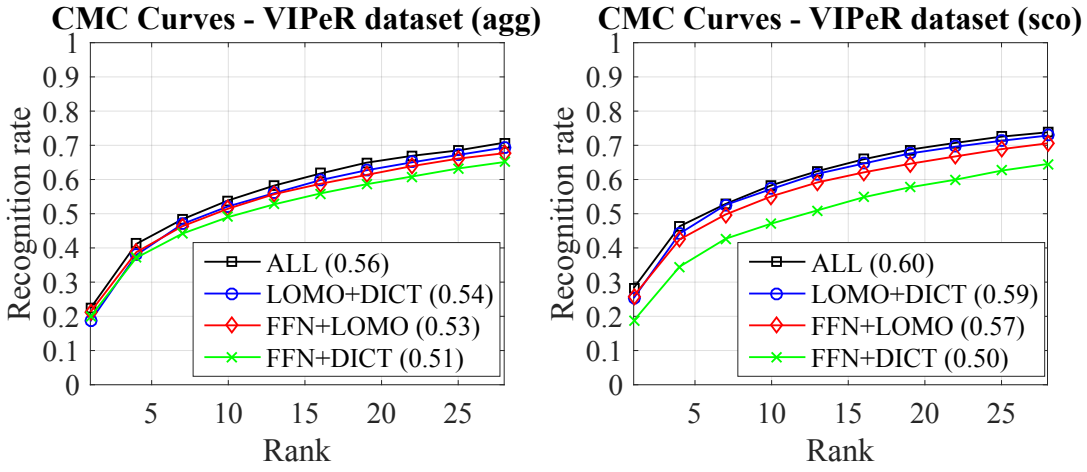


Figure 4.2: CMC curves for cross-dataset tests on VIPeR (p=316) by both rank aggregation (agg) and score-level fusion (sco).

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	22.31	43.89	53.89	65.66
All_{sco}	**28.26**	**49.10**	**58.34**	**69.35**
FFN+$LOMO_{agg}$	21.23	41.58	51.49	61.96
FFN+$LOMO_{sco}$	25.92	45.25	55.11	65.36
FFN+$DICT_{agg}$	19.87	40.19	49.02	59.49
FFN+$DICT_{sco}$	18.72	37.50	47.14	58.39
LOMO+$DICT_{agg}$	18.54	41.27	52.15	63.48
LOMO+$DICT_{sco}$	25.32	47.45	57.25	68.23
FFN	14.11	31.90	40.06	48.07
LOMO	17.31	38.48	50.82	62.18
DICT	14.49	31.14	42.03	55.13

Table 4.1: Results when training on CUHK02 and testing on VIPeR (p=316). The best results are in bold.

As for the within-dataset test for CUHK03, score-level fusion dominates the results being 5.95% better than rank aggregation when combining all three. Comparing the individual results, LOMO show to be best while FFN and DICT show similar results. This is also reflected in the pairwise combinations with the combination of FFN and DICT being the worst of the three.

Table 4.2 shows a comparison with two CNN's systems trained on CUHK02. The results show an increased rank-1 accuracy of 5.85% compared to the second best. With this increase, it should also be noted that the two CNN's were trained on multiple images of each person in the training set while only a single image was used in this context.

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	22.31	43.89	53.89	65.66
All_{sco}	**28.26**	**49.10**	**58.34**	69.35
DeepRank(Chen et al., 2015a)	22.41	–	56.39	**72.72**
DML(Yi et al., 2014)	16.17	–	45.82	57.56

Table 4.2: Comparing to other state-of-the-art results when training on CUHK02 and testing on VIPeR (p=316). "–" indicates not available results. Best results are in bold.

4.2 Multi-shot test

In reality, images might be sampled at a certain frame rate from a video stream and, hence, several images from each person in each view are available. As described in section 2.4, a few datasets provide several images of each person in different views, including CUHK03 with an average of 4.8 images per view, per person. Therefore, this dataset is used in the multi-shot test.

Similar protocol as for the single-shot test is followed, though, instead of only comparing each probe to a single query image, the probe is compared to all queries in the gallery following a single-multi matching protocol. For the training set and test probes, features from all images at each person are combined to a single feature vector. This was also done by (Ustinova et al., 2015) in which average pooling of features showed the best results, therefore, this technique is also utilized in this context. In the case of the dictionary learning, only patches from a single image from each person is used to learn the patch dictionary. This is done to avoid dictionaries learned from too generalized patches due to the misalignment problem which can cause the performance to drop. At image-level the average of each

representation is still calculated.

For this test, both CMC Curve and mAP are given. For the CMC curve, the most similar match from each person is taken into account. The resulting CMC curve is shown in Figure 4.3 and the ranked results along with the mAP for pairwise and all fused systems are shown in Table 4.3. All results are available in Appendix C.

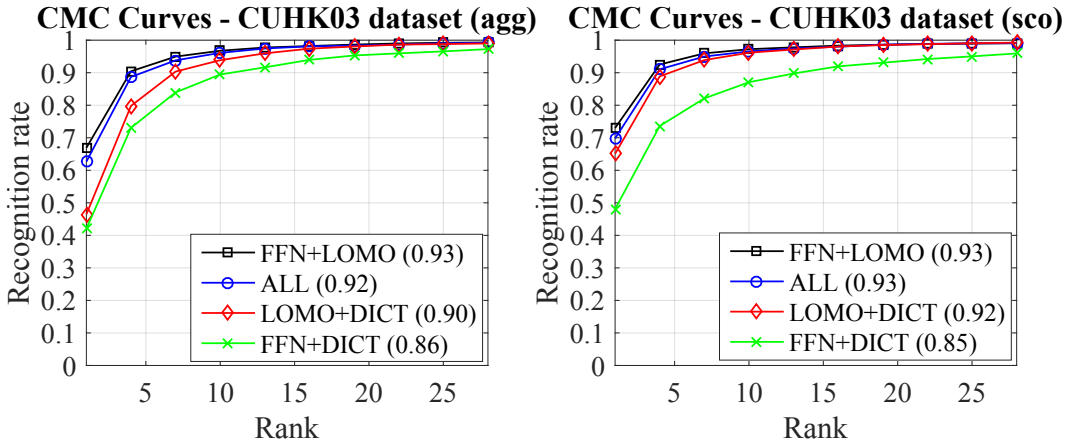


Figure 4.3: CMC curves for multi-shot tests on CUHK03 (p=100) by both rank aggregation (agg) and score-level fusion (sco).

System/Rank	r = 1	r = 5	r = 10	r = 20	mAP
All_{agg}	62.65	91.15	96.10	98.55	53.47
All_{sco}	69.80	93.05	96.55	**98.77**	59.82
$FFN+LOMO_{agg}$	66.90	92.20	96.75	98.85	57.30
$FFN+LOMO_{sco}$	**73.00**	**94.48**	97.23	98.75	**61.99**
$FFN+DICT_{agg}$	42.05	77.95	89.45	95.50	36.43
$FFN+DICT_{sco}$	48.07	76.93	87.07	93.55	40.96
$LOMO+DICT_{agg}$	46.45	85.10	93.85	98.30	42.43
$LOMO+DICT_{sco}$	65.07	91.53	96.15	98.58	54.72
FFN	49.35	78.85	87.35	93.65	39.32
LOMO	69.65	93.95	**97.35**	98.65	58.71
DICT	22.20	47.80	59.85	75.15	18.37

Table 4.3: Results from multi-shot testing on CUHK03 (p=100). The best results are in bold.

The results show that LOMO alone show high performance compared to the fused and pair-wise combinations. As it was also seen in the single-shot test, score-level fusion show better results, being slightly better than LOMO, though, still worse than FFN+LOMO. For this combination, the best single-shot result is improved by 17.25%. This shows the strength of combining several images of the probe. The better results by score-level fusion once again show that this technique performs better when more reference curves are included when assigning weights. Even though, FFN also show decent accuracy being almost 30% higher than in the single-shot case, the combination with LOMO does not improve performance when fusing by ranking aggregation. This was also seen in the case of single-shot on the PRID450S dataset, at which the accuracy in the combination of LOMO and FFN showed a lower accuracy than LOMO alone, but when including DICT, the performance rose to the highest in the test. The only difference here is the negative impact by DICT which show very poor performance despite multi-shot training and testing. A reason for this might be patch matching which does not perform well when the probe is compared to several queries giving higher probability of incorrect matched patches.

The best obtained results are compared to current state-of-the-art results on CUHK03 using multishot settings. Only two previous systems are known to have conducted multi-shot tests on CUHK03. The first system, *Person Re-Identification meets Image Search* (PRIM) by (Zheng et al., 2015a) utilizes a Bag-Of-Words (BoW) model trained on an independent dataset to extract more sparse features, as in the case of dictionary learning, along with color name features. Score-level fusion is then utilized to combine scores from different features. The second system *Multiregion Bilinear Convolution Neural Networks for Person Re-Identification* (MB-CNN) by (Ustinova et al., 2015) is a recent proposed CNN architecture utilizing part based siamese CNN's each making use of bilinear operations to achieve a more part-based sparse texture description which is more robust to large changes in pose. For both systems, average pooling is applied to fused multiple image features. The compared results are shown in Table 4.4.

System/Rank	r = 1	r = 5	r = 10	r = 20	mAP
All_{agg}	62.65	91.15	96.10	98.55	53.47
All_{sco}	69.80	93.05	96.55	**98.77**	59.82
FFN+$LOMO_{sco}$	73.00	**94.48**	**97.23**	98.75	**61.99**
PRIM (Zheng et al., 2015a)	22.95	42.50	53.00	–	20.33
MB-CNN (Ustinova et al., 2015)	**80.60**	92.90	95.50	98.00	–

Table 4.4: Results from multi-shot testing on CUHK03 (p=100) compared to other state-of-the-art systems. Rank 5, 10 and 20 for PRIM and MB-CNN are estimated from the CMC curve since exact results are not given. Furthermore, the mAP for MB-CNN is not given. The best results are in bold.

Compared to PRIM, the results are clearly better which was expected as the system was trained on an independent dataset. Though, compared to MB-CNN, the combination of FFN and LOMO is worse at rank-1 which show that part-based CNN's might be more suitable in person re-id as they are more robust to pose changes which is one of the major challenges. Even though, the bilinear CNN achieves good results on this particular multi-shot dataset, problems might occur at minor multi-shot datasets due to the nature of training CNN and in those cases, the systems used in this fusing scheme might prove better. It should, though, also be noted that the results from rank-5 and up are similar to those obtained in the fusing scheme.

4.2.1 Analysis of multi-shot

In order to get a better understanding of the different situations in which the performance of each system is either good or bad, five cases are analysed:

1. All individual systems provide a low AP, resulting in a low AP at fusion.
2. All systems show a high AP resulting in a high AP at fusion.
3. FFN show a higher AP than LOMO and DICT.
4. LOMO show a higher AP FFN and DICT.
5. DICT show a higher AP than FFN and LOMO.

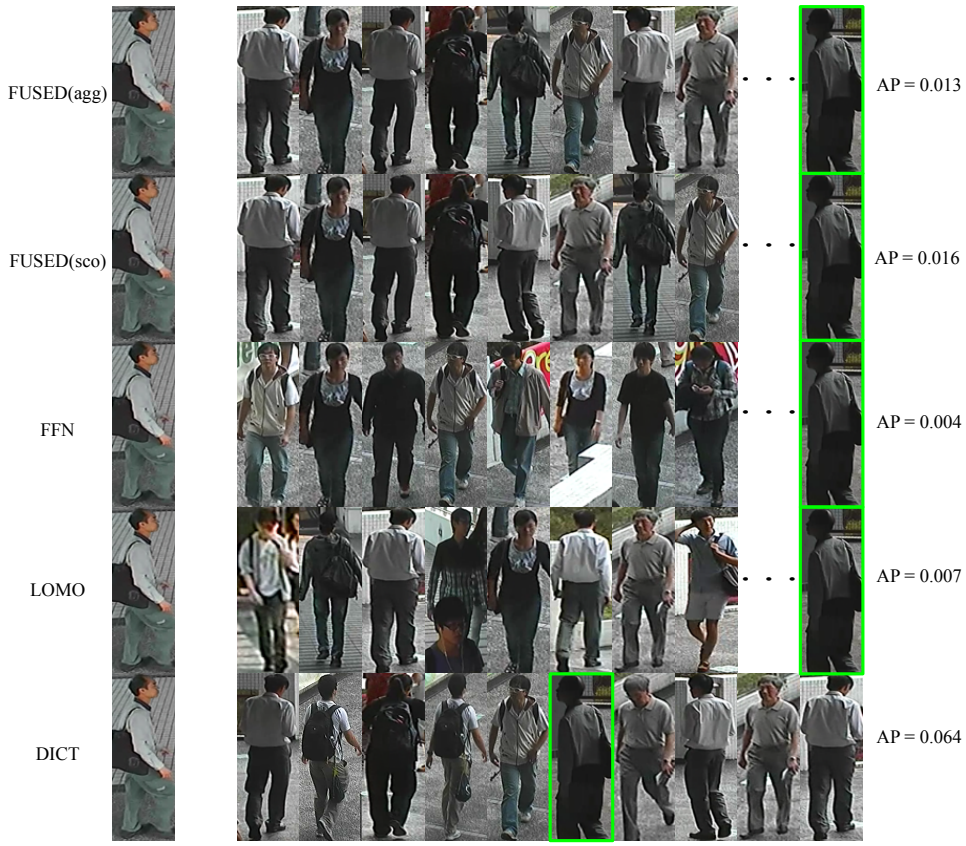


Figure 4.4: An example of the first case in which all systems have a low AP also result in a low AP when fused. The green boxes show the true matches. The most similar true query is ranked 38, 206 and 129 at rank aggregated, FFN and LOMO, respectively.

An example of the first case is shown in Figure 4.4 in which all three systems perform badly also resulting in a low AP when fusing the results. In this situation, the probe is very similar to the background, causing the feature representation in all cases to be noisy and hence making it difficult to classify correctly. Furthermore, the colors are very neutral which increases the challenge as also seen by the matches which are dominated by the same neutral gray color. The best result is given by DICT which has the most likely true match ranked 6 while the best true match is ranked 129 and 206 for LOMO and FFN, respectively. The reason for DICT providing better results might be due to the SIFT descriptor which might be more invariant to lighting changes compared to LOMO and FFN.

In the second case shown in Figure 4.5, all systems score a high AP causing the fused AP to also be high. Both the rank aggregated and score-level fused result ranks all true matches as most similar, resulting in an AP of 1. Similar result is seen for DICT, while LOMO and FFN score slighty below with AP's of 0.908 and 0.796, respectively. In all cases, other persons also carrying a backpack show to be similar and the rotated query of the true match might be the reason for being ranked lower by both FFN and LOMO. DICT does not suffer similar problem, as SIFT is more robust to minor changes. Furthermore, one of the true matches is occluded and parts of the feature representation is therefore ruined. DICT is more robust to this problem as patches are matched across the same horizontal line and only the lowest match is used, thus, eliminating the match between the occluded part and the probe.

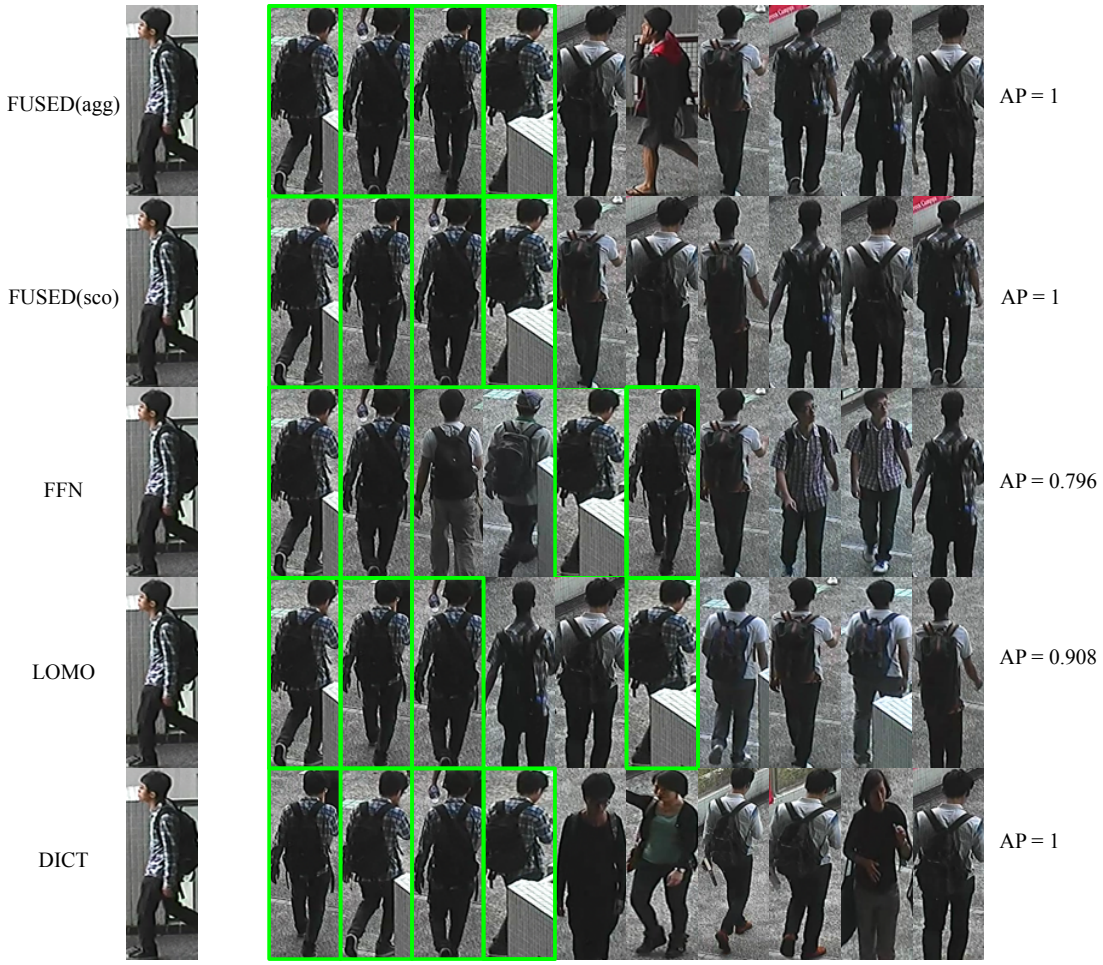


Figure 4.5: An example of the second case in which all systems have a high AP resulting in a high AP for the fused result. The green boxes show the true matches.

Figure 4.6 shows a case in which FFN show good performance with an AP of 0.878 and worse for LOMO and DICT with AP's of 0.209 and 0.030, respectively. This unfortunately cause the fused result to be lower than FFN alone with an AP's of 0.647 and 0.723 which is still decent. Looking at the figure, the lightings are very different between the probe and the true matches. This is one of the disadvantages for LOMO despite utilizing the HSV color space due to including the V channel. Usually, SIFT is robust to noise, though, in this case of extracting descriptors and specific local areas and not by key points, this invariance is reduced. As the probe is blurred compared to the queries of the same person, the SIFT descriptors are different causing the lower performance. Therefore, the most likely true match is only ranked 34 for this DICT. FFN is better at handling this issue as the intensity components are left out in the hand-craftet features which are used to adapt the FFN. Furthermore, as FFN represents the texture on a global scale combined with also being invariant to lighting changes, it better captures the same texture in different images. For the fused result, score-level fusion again seem to be more robust to the worse results by LOMO and DICT, having a higher mAP than rank aggregation.

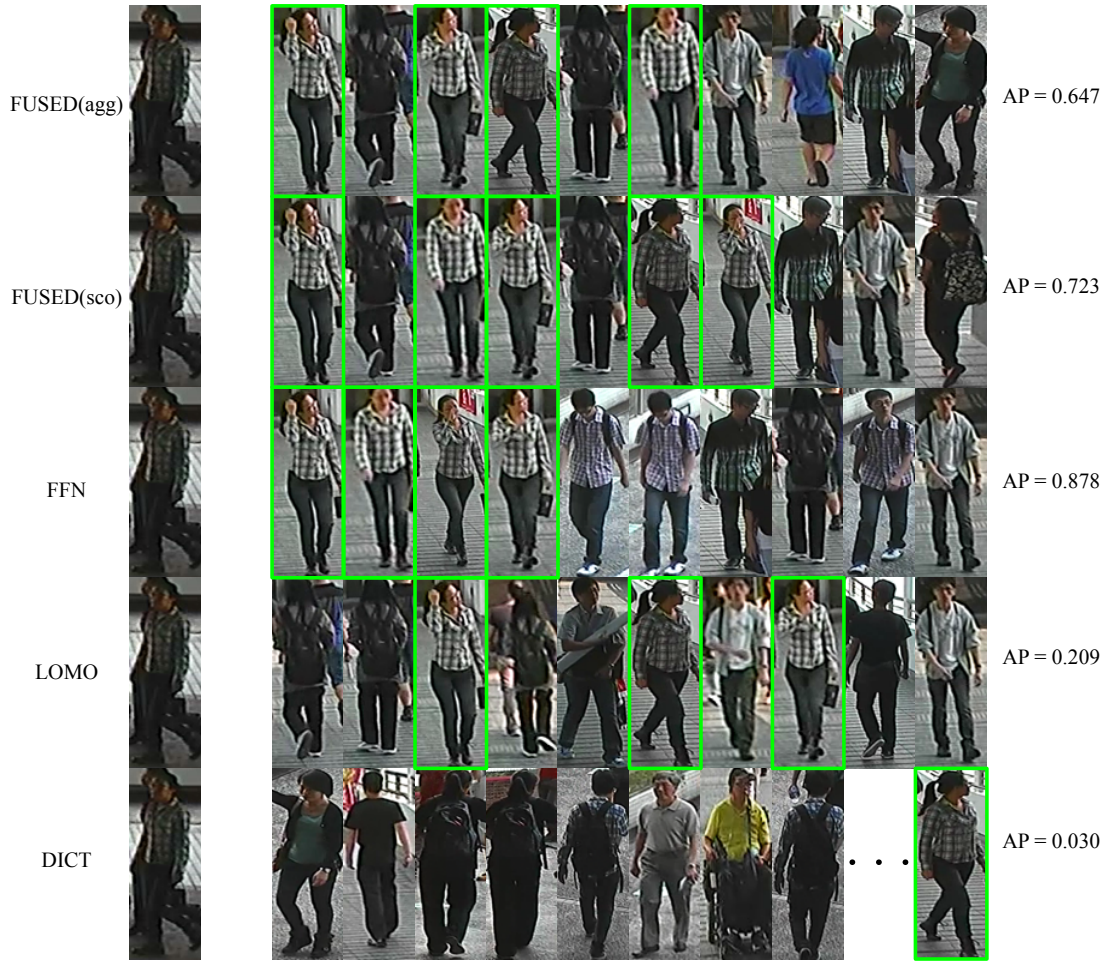


Figure 4.6: An example of the third case in which FFN has a high AP while the AP's for LOMO and DICT are lower. The green boxes show the true matches. For DICT, the most similar true query is ranked 34.

An example of the fourth case is shown in Figure 4.7, in which LOMO achieves almost perfect results with an AP of 0.963 while the AP's for FFN and DICT are lower at 0.187 and 0.019, respectively. For DICT, the most similar true match is only ranked 35 in this case. Looking at the probe, texture plays are large part while a part of the backpack is also colored. Due to the SILTP features in LOMO, the system performs well in distinguishing between different textures and therefore only the color affect the results which is seen by the girl in the red jacket being ranked better than one of the true matches. For FFN, due to the large change in view and because of the colored backpack, queries in which similar color is more dominant are ranked higher, seen by ranking a boy wearing a red shirt and a girl carrying a red backpack within top-10. This can be seen as a disadvantage of including hand-crafted features when training a CNN. For DICT, a disadvantage by matching patches at same horizontal level is shown, as persons wearing color similar to the top of the backpack carried by the probe are ranked higher. As also seen in the previous case, the score-level fusion provides decent results, even though, FFN and DICT do no not perform well. Though, compared to the last case, the results are decreased more, meaning that LOMO overall seem to be more dominating when fusing.

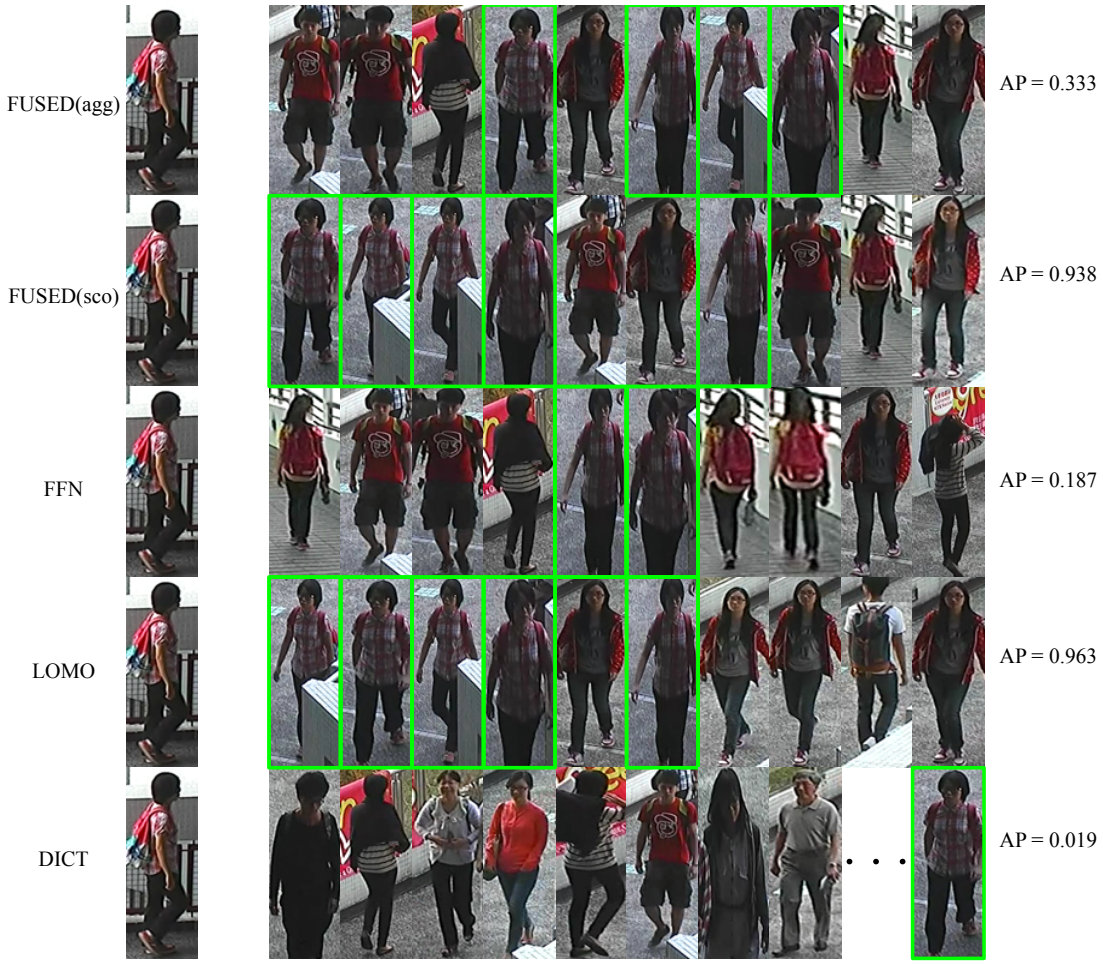


Figure 4.7: An example of the fourth case in which LOMO has a high AP while the AP's are low for FFN and DICT. The green boxes show the true matches. For DICT, the most similar true match is ranked 35.

An example of the last case is shown in Figure 4.8. In this case, DICT has a high AP of 0.828 while the AP of LOMO is worse, though, still at a decent 0.523 and only 0.073 for FFN. Again, texture is dominant and apart from the other examples, the worn shirt has a distinct change in appearance at the chest making the local patches at this area more discriminative. As the patch matching accumulates all shortest distances for all patches, the total distance becomes shortest, even though, some of the parts might be more similar to other images, for example having the part from the chest and up most similar to the second most likely query, but having the part from the chest and down more similar to the third most likely query. In this case, rank aggregation provide better results than score-level fusion unlike the other cases. The reason for this might be the impact of the weight assignment which assigns large weights to FFN if all distances are large and evenly distributed. This cause a large weight for FFN, despite a low AP resulting in a negative impact on the fused result. This weight assignment problem will be further discussed in section 5.1.

Figure 4.8: An example of the fifth case in which the AP of DICT is high while they are lower for LOMO and FFN. The green boxes show the true matches.

Overall, cases in which the images are too uniform in both color and texture seem the be very difficult if this is also the case for more persons in the gallery. On the other hand, the systems provide good results when both texture and color is apparent within the image with texture being more important. In order to improve performance in the cases of low AP, intensity components, such as the V component in HSV, can be removed to make the systems more robust to lighting changes. Furthermore, the patch matching in DICT can be further constrained to only match the local patches witin a local neighborhood, as a reasonable assumption can be made that the person is centred in the bounding box. In addition, a feauture which is more robust to scale changes, such as the SILTP could be considered to be included in FFN to capture the same local texture.

5. Evaluation

After having tested the proposed system with a more realistic view, different changes are discussed to potentially improve performance both with respect to accuracy, processing time and memory usage. The discussion will end up in a conclusion based on the work carried out in this report.

5.1 Discussion

Changes can be made to each part of the proposed system in order to enhance performance even further. In this section, the following topics are discussed:

- Lowering the dimension of the LOMO feature vector using different metric learning algorithms.
- Change the architecture of the re-implemented siamese CNN using only already available Caffe layers, in order to improve accuracy.
- Leaving out certain features to avoid coherent feature representations between fused systems.
- Other ways to assign weights when dealing with score-level fusion.

5.1.1 Metric learning algorithms

As one of the key elements in the person re-id pipeline, a robust metric learning algorithm has a large impact in increasing the performance and accuracy of a system. As matching using dictionary learning showed to be slow, systems that solely relies on fast matching algorithms can be considered. Furthermore, the dimension of the LOMO feature vector is 26,960 using images of just a size of 128×48. Representing each number as a *double*, a single feature vector takes up ≈0.21Mibs, a number which quickly rises to several GB when dealing with large databases. This is what is avoided by learning a subspace as in XQDA.

In order to see if other algorithms utilizing such subspace can enhance the performance, some of the most recent and best performing metric learning algorithms are compared. In total, five different metric learning algorithms are compared, including:

- XQDA (Liao et al., 2015).
- Mirror-KMFA (Chen et al., 2015b).
- Metric Learning Accelerated Proximal Gradient (MLAPG) (Liao and Li, 2015).
- Cross-View Discriminant Component Analysis (CVDCA) (Chen et al., 2016).

- Large Scale Similarity Learning (LSSL) (Yang et al., 2016).

As XQDA and Mirror-KMFA were used in the fusing scheme, these are included in comparison. These algorithms are described in section 3.4 and subsection 3.2.10. The remaining are shortly introduced in the following.

MALPG

Published at the IEEE International Conference on Computer Vision (ICCV) in December 2015, MALPG is a recent metric learning algorithm which, as many others, is based on the well known KISSME algorithm. The algorithm achieved a rank-1 accuracy of 40.73% on VIPeR using LOMO features.

The algorithm includes a constrain to the covariance matrix M such that the matrix becomes a positive definite matrix. This is done by introducing a convex loss functions following Equation 5.1.

$$f_M(x_i, x_j) = \log\left(1 + \exp^{y_{ij}(D_M^2(x_i,x_j)-\mu)}\right), \tag{5.1}$$

where $D_M^2(x_i, x_j)$ is the Mahalanobis distance between sample x_i and x_j, y_{ij} is 1 for a positive pair and 0 for negative and μ is a constant positive bias term. The overall loss function is then given as Equation 5.2.

$$F(M) = \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} f_M(x_i, x_j), \tag{5.2}$$

where $w = \frac{1}{N_{pos}}$ if $y_{ij} = 1$ and $\frac{1}{N_{neg}}$ otherwise while N_{pos} and N_{neg} are the number of similar and dissimilar image pairs, respectively. The algorithm, thus, more robust to the difference between the number of positive and negative image pairs which is often the case in person re-id. The objection function is thus given by Equation 5.3.

$$\min_M F(M) \qquad \text{s.t. } M \succeq 0 \tag{5.3}$$

The solution to Equation 5.3 is given by an SVD and the matrix can therefore be decomposed to $M = PP^T$. By removing columns with zero on the diagonal, the matrix P becomes a projection matrix. Therefore, the Euclidean distance can be used between image pairs instead of Mahalanobis.

CVDCA

In the IEEE Transaction on Circuits and Systems for Video Technology published in January 2016, CVDCA is an algorithm which also maps features to a common subspace using image discrepancies. Using this technique, the algorithm achieved a rank-1 on the VIPeR dataset of 39.72%.

The algorithm learns transformation matrices U^p, where $p = 1, 2, ..., N$ and N is the number of camera views, by both taking intra-view differences and cross-view differences into account. By also taking the intra-view differences into account, the algorithm is more robust when dealing more than just two different camera views, a challenge which is not considered in current databases for person re-id but is likely to occur in real life. Furthermore, the algorithm adapts according to very different positive image

pairs. This is done by solving the function defined in Equation 5.4.

$$\begin{aligned} \min_{U^1,U^2,...,U^N} f(U) \qquad & f(U) = f_{cross}(U) + \eta f_{intra}(U) \\ \min_{U^1,U^2,...,U^N} \sum_{p=1}^{N-1} \sum_{q=p+1}^{N} \sum_{i=1}^{n^p} \sum_{j=1}^{n^q} W_{ij}^{p,q} \|U^{pT} x_i^p - U^{qT} x_j^q\|_2^2 + \sum_{p=1}^{N} \sum_{i=1}^{n^p} \sum_{j=1}^{n^p} W_{ij}^{p,p} \|U^{pT} x_i^p - U^{pT} x_j^p\|_2^2 \\ + \sum_{p=1}^{N-1} \sum_{q=p+1}^{N} \|U^p - U^q\|_F^2 \\ \text{s.t. } U^{kT} M^k U^k = I, \quad k = 1, 2, ..., N, \end{aligned} \tag{5.4}$$

where n^p and n^q are the number of images in view p and q, respectively, $W_{ij}^{p,q}$ is a weight term and $M^k = X^k X^{kT} + \mu I$ is a constraint in which X^k denotes the features in the k'th view added to avoid non-zero projection matrices. Furthermore, the last term is a regularization term which takes largely different positive image pairs into account by avoiding largely different transformation matrices caused by this difference in appearance.

By reformulating the problem, the solution can in the end be solved as a generalized eigenvalue problem and from this solution, each transformation basis $u_1^p, u_2^p, ..., u_C^p$ for the projection matrix in the p'th view is calculated following Equation 5.5.

$$u_c^p = \frac{\delta_p(u_c)}{\sqrt{\delta_p(u_c)^T M \delta_p(u_c)}}, \tag{5.5}$$

where δ_p indicates the p'th subvector in the transformation matrix and M is a block matrix defined as $diag(M^1, M^2, ..., M^N)$.

To calculate distances between image pairs, the cosine similarity is calculated and subtracted 1, making the algorithm fast in the matching process.

LSSL

Published recently at the AAAI Conference on Artificial Intelligence in February 2016, LSSL is another algorithm which is inspired by the KISSME algorithm. By only taking similar image pairs into account, the algorithm is very fast, but still manages to keep a decent accuracy with a rank-1 accuracy on VIPeR of 47.8%.

Recalling that the two matrices Σ_E and Σ_I in KISSME are calculated by the difference between similar and dissimilar image pairs, respectively, LSSL only calculates Σ_I to also estimate Σ_E. This is done by looking at the difference and commonness of similar image pairs defined in Equation 5.6.

$$\begin{aligned} e_{ii} &= x_i - z_i \\ m_{ii} &= x_i + z_i, \end{aligned} \tag{5.6}$$

where e_{ii} represents the difference and m_{ii} the commonness between a positive image pair. As when calculating Σ_I, the MLE is used to calculate the two new covariance matrices Σ_{mI} and Σ_{eI} following Equation 5.7.

$$\Sigma_{mI} = \frac{1}{N} \sum_{i=n}^{N} m_{ii} m_{ii}^T \qquad \Sigma_{eI} = \frac{1}{N} \sum_{i=n}^{N} e_{ii} e_{ii}^T \tag{5.7}$$

By introducing a pair-constraint Gaussian assumption, it can be shown that corresponding covariance matrices for dissimilar image pairs are equal and they can, thus, be calculated following Equation 5.8.

$$\Sigma_{mE} = \Sigma_{eE} = \Sigma = \frac{1}{2}(\Sigma_{mI} + \Sigma_{eI}) = \frac{1}{2N}\sum_{i=1}^{N}(m_{ii}m_{ii}^T + e_{ii}e_{ii}^T) \tag{5.8}$$

Instead of solely make use of the difference covariance matrices when calculating the Mahalanobis distances, the term can be split up to a similarity measure combined with a dissimilarity measure as shown in Equation 5.9.

$$r(x,z) = m^T Am - \lambda e^T Be, \tag{5.9}$$

where λ is a parameter used to balance the difference in similarity and dissimilarity scores and A and B are defined as $\Sigma^{-1} - \Sigma_{mI}^{-1}$ and $\Sigma_{eI}^{-1} - \Sigma$, respectively. The final score measure is, thus, defined as a combination of a bilinear similarity and Mahalanobis similarity. In order to achieve a good estimation of the similarity and dissimilarity distributions, LSSL utilizes PCA to reduce the number of feature dimensions to 74.

Tests

The VIPeR dataset is used in the tests following the protocol defined in section 2.4. In each test, a different size feature vector is used, including feature sizes of $\{500, 250, 125, 75\}$. Furthermore, the processing time for training and matching is measured and compared. The results are found in Appendix C.

Figure 5.1 shows the rank-1 accuracy compared to the feature dimension for each of the algorithms. It can be seen that XQDA and MLAPG both keep a stable level with only a slight decrease in accuracy when reducing dimension. Mirror-KMFA and CVDCA drops slightly more, but still maintains a decent accuracy even at a feature dimension of 500. These algorithms show the ability to learn a subspace which is robust to the number of dimensions as the projection matrices also takes the metric learning into account. The impact of this is also seen at LSSL which uses PCA reduced features, hence, does not include metric learning in the subspace learning. For this reason, the accuracy starts to drop already at a dimension of 125.

The results are further summarized in Table 5.1. It can be seen that the overall best performance is by using a feature dimension of 125 using CVDCA which achieves a rank-1 accuracy of 43.13%. Using this number of dimensions is further shown to provide the best results for almost all algorithms with LSSL as an exception.

Algorithm/feature dimension	**75**	**125**	**250**	**500**
XQDA	37.25	38.23	38.04	38.86
KMFA	39.18	40.47	40.13	36.14
MLAPG	37.97	38.77	38.96	**39.15**
LSSL	32.97	30.60	10.54	7.18
CVDCA	**42.15**	**43.13**	**42.97**	35.44

Table 5.1: Rank-1 accuracies for the five different metric learning algorithms compared to the number of feature dimensions. The best result at each feature dimension is in bold.

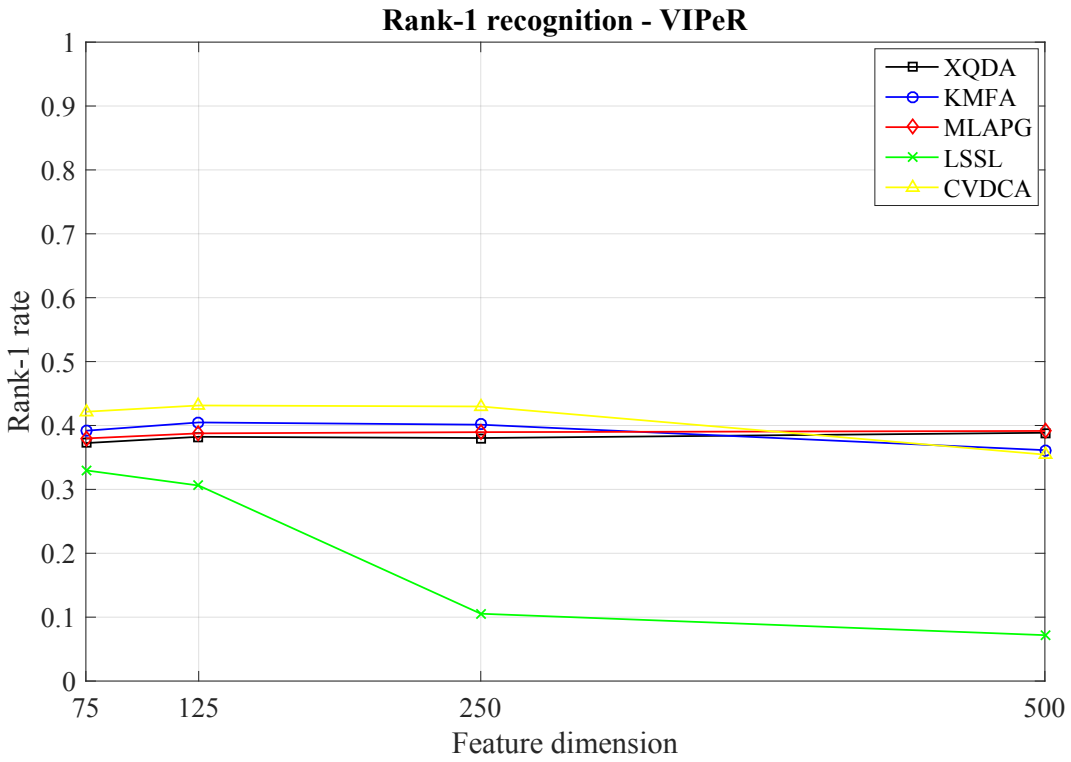


Figure 5.1: The rank-1 accuracy for XQDA, Mirror-KMFA, MLAPG, LSSL and CVDCA using different size of feature dimensions tested on VIPeR dataset (p=316).

To measure training and matching time, an average over all 10 iterations on the VIPeR dataset is taken and the results are summarized in Table 5.2. This is done with a feature dimension of 500.

	Train [s]	**Match [s]**
XQDA	1.86	0.21
KMFA	83.51	71.82
MLAPG	26.38	0.23
LSSL	0.12	0.03
CVDCA	27.71	1.22

Table 5.2: Comparison of training and matching time across the five tested metric learning algorithms by calculating average timings over 10 iterations on VIPeR dataset (p=316).

It can be seen that matching is fast in all cases, but Mirror-KMFA. This is due to the use of the more complex χ^2 kernel as described in subsection 3.6.3. This is furthermore the reason for the training time which is also significantly higher than the other algorithms. As both CVDCA and MLAPG utilizes PCA to initially reduce the number of dimensions, training time is higher than XQDA, though, without being critically high. For LSSL, training was performed on already reduced features and the training time will therefore in reality be slightly larger.

For future work, including some of these algorithms could be interesting, in order to see how different subspaces affects the results when using similar features, such as LOMO.

5.1.2 Coherent features

All three systems in the proposed fusing scheme makes use of different low-level features. LOMO makes use of joint HSV histograms and SILTP histograms while DICT makes use of Lab histograms and SIFT descriptors. Finally FFN combines RGB, HSV, YCrCb, Lab and XYZ color histograms along with gabor features. As CNN features used in FFN are also based on the texture of the image, it can be argued whether texture features in the other cases can be left it to avoid redundant information. For this reason, tests are conducted excluding the SILTP features in LOMO, making it solely rely on color features.

Tests are carried out on VIPeR and CUHK01 datasets due to the difference in color and texture appearence between the two. Similar protocols as followed in the previous test are also followed here and the resulting CMC curves are shown in Figure 5.2. Futhermore, the results are summarized in Table 5.3-5.4, in which the previous results are included in comparison. To distinguish between previous and new results, the superscript *col* is used for those in which SILTP is excluded. The results are found in Appendix C.

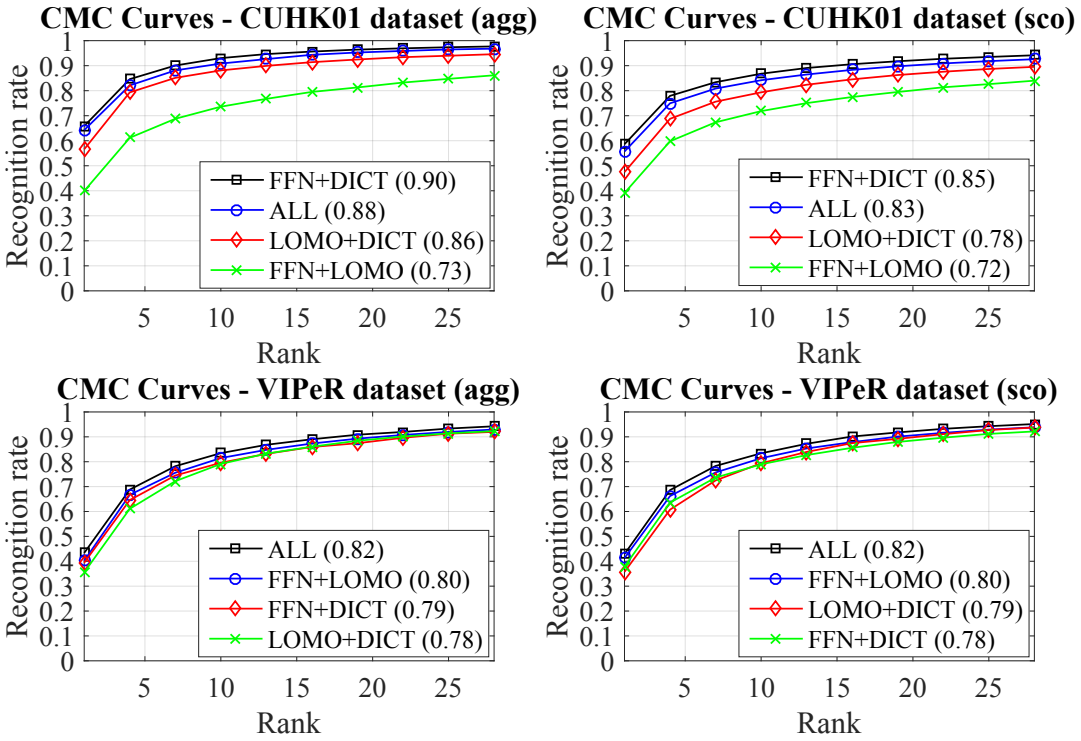


Figure 5.2: Tests on VIPeR (p=316) and CUHK01 (p=486) using both aggregation and score-level fusion and LOMO features without SILTP.

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	**45.63**	**75.06**	85.16	92.47
All_{sco}	45.24	75.02	**85.70**	**93.47**
All^{col}_{agg}	43.35	72.78	83.45	91.33
All^{col}_{sco}	42.91	72.29	83.39	92.10
LOMO	37.72	67.59	80.06	90.51
LOMO ^{col}	32.62	59.64	73.26	86.14

Table 5.3: Results on VIPeR (p=316) for both rank aggregation(*agg*) and score-level fusion(*sco*) by excluding SILTP in LOMO. The best results are in bolt.

System/Rank	r = 1	r = 5	r = 10	r = 20
All_{agg}	**70.35**	**88.46**	**93.29**	**96.95**
All_{sco}	64.67	83.81	89.35	93.90
All^{col}_{agg}	64.26	84.38	90.84	95.53
All^{col}_{sco}	55.69	77.30	84.19	90.21
LOMO	41.77	66.19	74.86	83.07
LOMO^{col}	26.77	48.54	59.16	69.81

Table 5.4: Results on CUHK01 (p=486) for both rank aggregation(*agg*) and score-level fusion(*sco*) excluding SILTP in LOMO. The best results are in bolt.

For CUHK01, the rank-1 accuracy for LOMO alone is dropped by 15% which was expected as the dataset relies much on texture information. This is one of the causes for the fused results also being lower with a drop in rank-1 accuracy of 6.09% and 8.98% for rank aggregation and score-level fusion, respectively. Leaving out SILTP, the combination of FFN and DICT now scores higher than all three systems combined, though, still being worse than the result with SILTP included. For VIPeR, the result is only slightly worse then the previous with a rank-1 accuracy drop of 5.1% at LOMO alone and 2.28% and 2.33% at ranking aggregation and score-level fusion, respectively. The drop in accuracy is more respectable in this case and the fused results still keep the highest accuracies compared to the pair-wise combinations.

Overall, excluding SILTP features in LOMO does not seem to have any positive impact when fusing with systems that also exploits texture. One general reason for this is the affect of fusing a system which alone has a low accuracy as seen with CUHK01. Another reason is the level of abstraction exploited by the different systems which causes different representations and can therefore complement each other.

5.1.3 Siamese CNN variations

The accuracy of the re-implemented siamese CNN descriped in subsection 3.2.9 was only half the originally achieved. In order to improve this accuracy, variations of the CNN are trained and tested. Very recently, a new CNN has been proposed for person re-id, building on the current state-of-the-art. The new PersonNet (Wu et al., 2016a) similarly makes use of a cross neighborhood difference layer. This network is, though, deeper than the original and have other differences, including two additional fully connected layers, different number of feature maps in each layer and finally using tanh activation functions rather than ReLU. This makes the basis of the variations proposed. Because of the missing information on the neighborhood difference layer, a different type of difference layer, already implemented in Caffe, is utilized. The following variations are proposed:

- Exchange **Neighborhood difference** layer with **Element wise** layer.
- Add to additional **Fully connected** layers, each with 4096 neurons.
- Use **Tanh** activation function instead of **ReLU**.

For all variations, training and testing is carried out on the CUHK01 dataset. The different layers are added in the order presented above.

Element wise layer

Having concluded, the problem lie in the cross-input layer, this layer is changed to a different type of difference layer to still let the network learn, based on the difference between positive and negative image pairs. In this case, Caffe has an already implemented **Element wise** layer, which can be used for element wise operations between two inputs. Following the original strategy, element wise subtraction is carried out as given by Equation 5.10.

$$K_i(x, y) = F_i(x, y) - G_i(x, y) \tag{5.10}$$

Two variations are tested; one which retains the siamese structure by having to element wise layers, one subtracting G_i from F_i and another vice versa, and one only doing the former subtraction.

Additional fully connected layers

In order to increase the non-linearity of the network, additional fully connected layers are added. Following the PersonNet, the size of these layers are 4096. To avoid overfitting, **dropout** layers are added with a probability of leaving out a neuron of 50% in a single training iteration. This furthermore enhances training as it can be seen as a combination of many different networks as different neurons are excluded in each training pass. An example of how the dropout layer works was shown in Figure 3.14.

Tanh activation function

Until now, ReLU activation functions have been utilized throughout the network. Though, for computing element wise differences, the output from the element wise layer may be zero across most of the output if differences between the two inputs are large, causing the network not to learn as the gradient at this point is 0 when using ReLU. Instead, an activation function that also maps to negative values can be used and therefore, the tanh activation function is utilized. This activation function maps the input to the interval [-1,1] as also described in section 3.2.

Train and tests

In all cases, training is carried out as described in subsection 3.2.13. The training loss for the different variations is seen in Figure 5.3 while the accuracies are summarized in Table 5.5. The original result (Orig) is included in comparison.

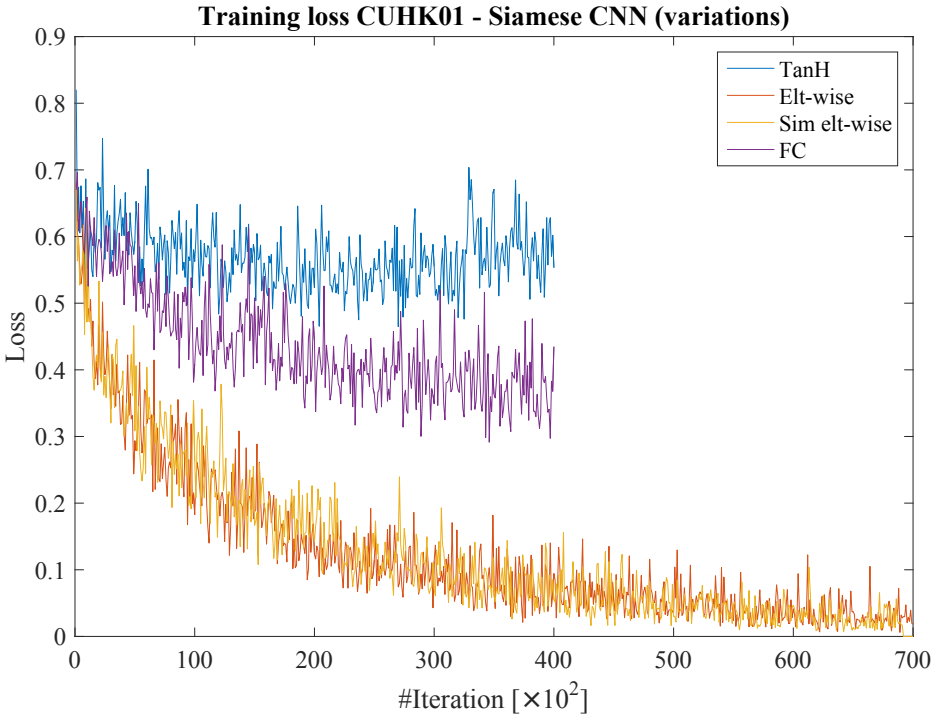


Figure 5.3: Training loss on CUHK01 for variations of Siamese CNN.

Variation/Rank	1	5	10	20
Orig	30.00	62.00	76.00	93.00
Elt-wise	26.00	65.00	76.00	91.00
Sim elt-wise	20.00	56.00	72.00	86.00
TanH	8.00	12.00	23.00	38.00
FC	22.00	56.00	68.00	85.00

Table 5.5: Rank accuracies for the four different variations of the re-implemented siamese CNN, including the original result.

As seen in the results, neither of the variations tends to improve the accuracy of the siamese CNN. It is, though, worth noticing that the solution of having two symmetric element-wise layers seem to be worse than just using a single, and it is therefore unnecessary to keep the siamese structure until the first fully connected layer. This can be argued by the lack of asymmetry which is present in the cross-input layer. Furthermore, the loss for adding additional fully connected layers or switch to TanH activation functions quickly converges at a relatively high loss which is also reflected in the results. One reason for this might be the size of the training data which is now too small when dealing with such deep network. This cause the network to not learn well, thus maintaining a high error.

5.1.4 Weight assignment

When assigning weights, the area under the distance curves are utilized. One problem might, though, arise negatively affecting the results. While it is desirable to have a curve following that of Figure 5.4 (a) at which the distance to the most similar is low but high for the rest, similar weights might be calculated if all distances are high and evenly distributed as shown in Figure 5.4 (b). In this case, the bad performing system would be assigned same weight as the well performing.

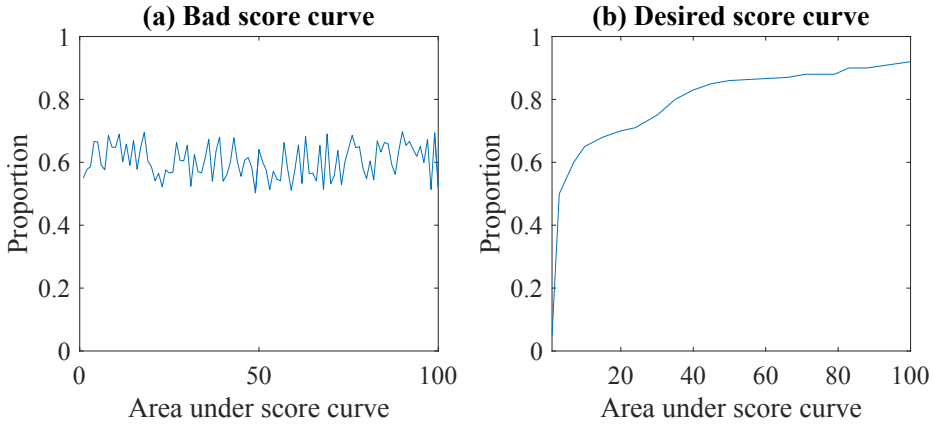


Figure 5.4: An example of (a) a bad score curve causing undesirable large weights and (b) a desired score curve cause desirable large weights.

Another weight assignment might therefore be considered. One solution to this is to only look at the distances to the most similar. By comparing the distances to the three most similar as defined in Equation 5.11, a ratio will be calculated determining the level of distinction for that particular person.

$$w = \frac{d_2 - d_1}{d_3 - d_1}, \tag{5.11}$$

where d_1, d_2 and d_3 are the distances to the most, second most and third most similar, respectively.

5.2 Conclusion

Person re-identification is a topic still in research due to difficult challenges such as change in pose, view, lighting and scale. Therefore, lots of effort is put into optimizing each step of the process, including creating a robust feature representation and develop a metric learning algorithm based on the cross-view changes. Lots of different systems have been proposed for this task, more recently including different CNN's, as they are dominant in several areas of image processing. Usually, different feature types which individually are invariant to certain cases, are fused for a more robust representation. Though, as stated in literature, late fusion of different biometrics often provides better results and should therefore also be considered in this case. Only a few proposed systems utilize late fusion, including rank aggregation and score-level fusion, though, using results from individual features. To further enhance performance, late fusion was instead applied to different systems which also utilize metric learning algorithms. To compare performance of the previously utilized late fusion techniques, both techniques were applied.

In order to enhance performance by late fusion of different systems, they should complement one another well. Therefore, features on different levels were combined, in order to combine the strengths of having low-level and sparse representations. Further, features extracted locally were combined with more globally extracted to have different representation of the features.

As CNN's have shown state-of-the-art results due to the ability to express local texture as a sparse feature vector, a re-implementation of one such was included. Two re-implementations were made; a siamese utilizing patch differences across dual input and a combined CNN and hand-crafted architecture called which learned the filters in a CNN based on low-level color and texture features. Due to lack of critical details in the siamese CNN, accuracy of the re-implementation was only half the stated and due to also higher accuracy by FFN, this network was utilized in the fusing scheme. The new FFN features were further used in combination with a Mirror-Kernel MFA metric learning algorithm to improve accuracy. Matching was then performed by a distance metric defined in kernel space.

To get capture more local features, though, still using sparse representations, a modified dictionary learning was utilized by the use of a projection matrix. From extracting SIFT and color features, dictionaries and projection matrices were learned for two different views on both patch-level but also a more global image-level having a single feature representation for each image. From having learned dictionaries, matching on patch was done by matching all patches on same horizontal level, define the patch distance as the shortest and accumulate all patch distances to a final distance. On image-level, the cosine similarity was calculated between two image representations. Finally, the scores of the two levels were fused by addition.

To also have a feature representation based solely on low-level features, LOMO was utilized due to decent results both in feature processing and matching. For a more discriminative representation, color and texture features were extracted from local patches and combined to a single representation. This included joint HSV and the scale and noise invariant SILTP feature. To make the features more robust to misalignments, histograms extracted at same horizontal level were maximized. Using the extracted features, a combined metric and subspace learning algorithm was utilized by calculating a projection matrix which was further used to calculate the covariance matrix in the Mahalanobis distance used to calculate similarities. Matching was, thus, performed after projecting features to a common subspace.

Having both ranks and scores as output for each system, rank aggregation using the Stuart algorithm was utilized by combining the ranks from a statistical point of view. Furthermore, the product rule was used due to its superiority in the field to fuse the output scores. In this case, weights were assigned to each system using the area under the score curve for each test curve.

Tests were conducted on four different datasets commonly used in the field, including two minor,

VIPeR and PRID450S, and two larger, CUHK01 and CUHK03, each representing the mentioned challenges. Having VIPeR, PRID450S and CUHK01 including 632, 450 and 971 persons, data was split evenly between training and test data and 10 test iterations were run. For CUHK03, including 1360 persons, 1160 were used for training while 100 were used for testing following a predefined protocol. 20 test iterations were then run. To visualize the results, CMC curves created showing the accuracies at each rank. After testing by fusing all three systems and pair-wise, it was clear that fusing all showed the overall best results when utilizing rank aggregation. This showed that rank aggregation worked better when having systems that all showed decent results. On the other hand, score-level fusion were better at handling situations at which one system would have bad performance. For the largest dataset, the combination of LOMO and dictionary learning showed better results due to bad performance by the FFN and the increased number of reference curves. Compared to state-of-the-art results the obtained accuracies were either higher or close to similar. For CUHK01 and PRID450S, accuracies were improved by 15.04% 10.94%, respectively, while the accuracy for VIPeR was 5.63% worse. Finally, the accuracy for CUHK03 was 0.02% worse than a state-of-the-art CNN which had used multiple images for each person in the training phase.

To get more realistic idea of the performance, a cross-dataset test was conducted by training on one dataset and test on an entirely different. For this test, the extended version of CUHK01, CUHK02, was used for training while tests were conducted on VIPeR. This showed the challenge of having data which is very different, a scenario which is likely in real life, by only achieving a rank-1 accuracy of 28.26% using score-level fusion. This was an improvement of 10.95% compared to the best individual system and showed the benefits of late fusion. Further, this was an improvement of 5.85% compared to previous state-of-the-art results. Further, a multi-shot test was conducted using several images from each person in the training phase and when matching across views. As matches were calculated for several queries from each person in a different view, an AP was calculated, indicating the ability to return similar images. A mAP was then calculated over all test probes. Utilizing multi-shot settings, the accuracy was clearly improved by 15.08% when fusing all three systems. Though, due to the matching at dictionary learning, the combination of only LOMO and FFN provided the highest accuracies, being 17.25% better than the single-shot case. Further, this combination showed the largest mAP being 61.99%. Compared to previous systems providing multi-shot results, the combination of FFN and LOMO showed the largest mAP, though, with a very recently proposed MB-CNN being better when looking at the rank-1 accuracy. This system showed an accuracy which was 7.6% higher than the proposed system.

By looking at cases in which the different systems performed differently, the biggest challenges were found at very uniform images both with respect to both color and texture. On the other hand, images with both clear texture and color information showed the best results despite of differences in view and lighting. Further, while LOMO worked well in images suffering from differences in scale and view, FFN seemed better in situations with changes in lighting and view. Finally, DICT showed decent results in situations with large variations across the image.

As the results show a potential in late fusing of different systems, each method could be improved by making the feature representations more robust to lighting changes as seen in the case of LOMO or changing the way of matching which was a problem with DICT when dealing with images including similar local patches across a horizontal view. Further, the score-level fusion showed problems in large evenly distributed scores, a problem which can be dealt with by assigning weights, only taking the lowest scores into account.

In order to analyse whether fusing of different systems makes sense, processing time was measured for the individual systems and the fusion techniques. From this test, the most beneficial fusing method was shown to be score-level fusion being almost 30 times faster than rank aggregation. The only critical point was the dictionary matching which took up 96.5% of the total matching and fusing time when doing score-level fusion, hence, a different way of matching should be considered.

A. Color Conversion

This appendix gives an overview of conversion from RGB to different color spaces utilized throughout the report, including:

- RGB to HSV
- RGB to Lab
- RGB to YCbCr
- RGB to YIQ

A.1 RGB to HSV

HSV consists of the three color components *Hue* (H), *Saturation* (S) and *Value* (V) in which the latter is mostly affected by lighting changes. The color space can be represented as a cylinder as shown in Figure A.1 where the hue is given by the angle of the circumference, saturation by the distance from the centre to the circumference and value by the hight at the cylinder.

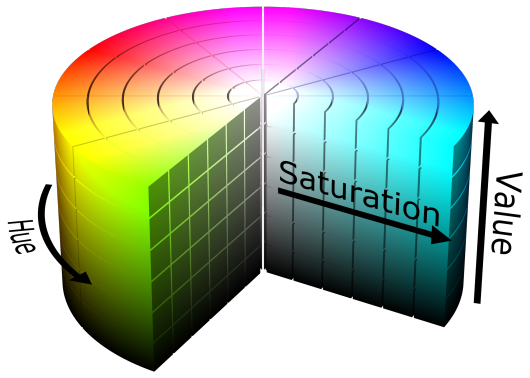


Figure A.1: HSV color space represented as a cylinder (SharkD, 2015).

The values of the color space are calculated following Equation A.1.

$$
\begin{aligned}
H &= \begin{cases} \frac{G-B}{V-min\{R,G,B\}} \cdot 60^\circ, & \text{if } V = R \text{ and } G \geq B \\ (\frac{B-R}{V-min\{R,G,B\}} + 2) \cdot 60^\circ, & \text{if } V = G \\ (\frac{R-G}{V-min\{R,G,B\}} + 4) \cdot 60^\circ, & \text{if } V = B \\ (\frac{R-B}{V-min\{R,G,B\}} + 5) \cdot 60^\circ, & \text{if } V = R \text{ and } G < B \end{cases} \\
H &\in [0, 359] \\
S &= \frac{V - min\{R,G,B\}}{V}, \; S \in [0,1] \\
V &= max\{R,G,B\}, \; V \in [0,255]
\end{aligned}
\tag{A.1}
$$

A.2 RGB to Lab

Lab consists of the color component *Lightness* (L) and the two non-linear components *a* and **b**, having L as the component mostly affected by lighting changes. The color space can be represented as a sphere, as shown in Figure A.2, in which L is the lightness coordinate, a is the red/green coordinate and b is the yellow/blue coordinate.

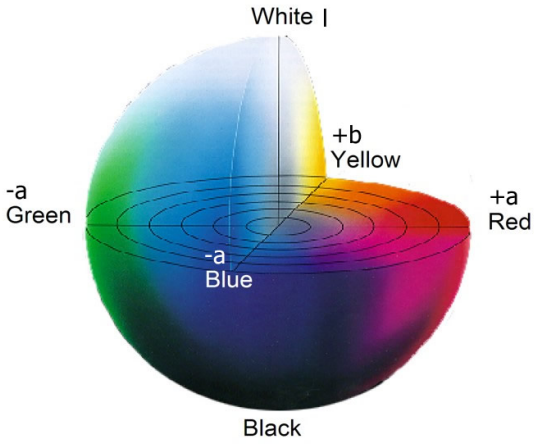


Figure A.2: Lab color space represented as a sphere (Solutions, 2014).

The values of each components are calculated by first converting to a different XYZ color space given in Equation A.2 (Lindbloom, 2003).

$$
\begin{aligned}
&X = \frac{0.49 \cdot R + 0.31 \cdot G + 0.20 \cdot B}{0.17697} \quad Y = \frac{0.17697 \cdot R + 0.81240 \cdot G + 0.01063 \cdot B}{0.17697} \quad Z = \frac{0.01 \cdot B + 0.99 \cdot B}{0.17697} \\
&L = 116 f_y - 16, \; L \in [0,100] \quad a = 500(f_x - f_y), \; a \in [-128,127] \quad b = 200(f_y - f_z), \; b \in [-128,127] \\
&f_x = \begin{cases} \sqrt[3]{x_r} & \text{if } x_r > \epsilon \\ \frac{\kappa x_r + 16}{116} & \text{otherwise} \end{cases} \quad f_y = \begin{cases} \sqrt[3]{y_r} & \text{if } y_r > \epsilon \\ \frac{\kappa y_r + 16}{116} & \text{otherwise} \end{cases} \quad f_z = \begin{cases} \sqrt[3]{z_r} & \text{if } z_r > \epsilon \\ \frac{\kappa z_r + 16}{116} & \text{otherwise} \end{cases},
\end{aligned}
\tag{A.2}
$$

where $x_r = \frac{X}{X_n}$, $y_r = \frac{Y}{Y_n}$ and $z_r = \frac{Z}{Z_n}$ while X_n, Y_n and Z_n are white point references with values of 95.047, 100 and 108.883, respectively. Furthermore, $\epsilon = 0.008856$ and $\kappa = 903.3$ following CIE standards (Lindbloom, 2003).

A.3 RGB to YCbCr

YCbCr consists of a intensity component Y and two chroma components Cb and Cr. The color space can be depicted as a rotated RGB cube as for shown in Figure A.3.

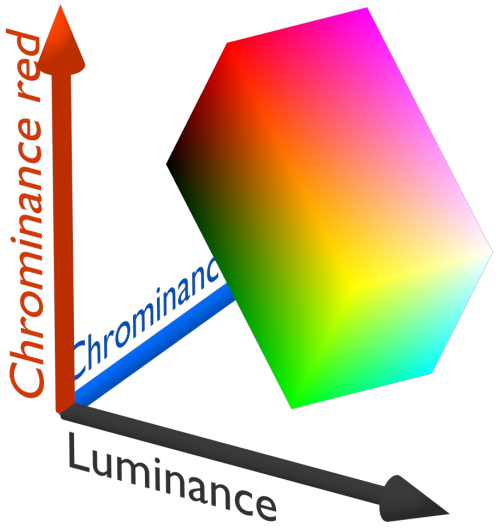


Figure A.3: YCbCr color space given as a rotated RGB cube (Peters, 2011).

The conversion from RGB follows Equation A.3.

$$\begin{aligned} Y &= 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B,\ Y \in [0, 255] \\ Cb &= -0.169 \cdot R + -0.331 \cdot G + -0.500 \cdot B + 128,\ I \in [0, 255] \\ Cr &= 0.211 \cdot R + -0.523 \cdot G + 0.312 \cdot B + 128,\ Q \in [0, 255] \end{aligned} \tag{A.3}$$

A.4 RGB to YIQ

YIQ is the color space used by NTSC and this thereby also known by that abbreviation and is similar to YCbCr in the sense that it also contains the luma intensity Y and two chroma components I and Q. Likewise, the color space can also be depicted as a rotated RGB cube.

The values of the components are calculated from RGB values following Equation A.4 with Y being similar to that in YCbCr.

$$\begin{aligned} Y &= 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B,\ Y \in [0, 255] \\ I &= 0.596 \cdot R + -0.274 \cdot G + -0.322 \cdot B,\ I \in [-151.98, 151.98] \\ Q &= 0.211 \cdot R + -0.523 \cdot G + 0.312 \cdot B,\ Q \in [-133.37, 133.37] \end{aligned} \tag{A.4}$$

B. LMDB Creation

When dealing with large sets of data, like in the case of training a neural network, it is beneficial to restore the images in a database for better I/O performance. Caffe supports both LMDB or the more compressed LevelDB databases when handling large datasets. Furthermore, a tool for converting a set of images to LMDB is provided.

As shown in Figure B.1, a list of images with a corresponding label must be given. The representation of each of these images is then changed to $C \times H \times W$ to fit with the blobs utilized in Caffe. Next, the image is written to a *Datum* struct which stores the information of the image dimensions, the image data itself and the corresponding label. Finally, the Datum is serialized to a string before written to the LMDB database. When writing to the database, the data can be chunked as batches for faster writes. In Caffe, a batch size of 1000 is used. The entire flow is summarized in Pseudocode 10.

```
1 /home/aske/Datasets/Market-1501/all/0653_c6s2_045218_00.jpg 652
2 /home/aske/Datasets/Market-1501/all/1124_c5s2_157774_00.jpg 1123
3 /home/aske/Datasets/Market-1501/all/0392_c2s1_090221_01.jpg 391
4 /home/aske/Datasets/Market-1501/all/0500_c4s2_060473_00.jpg 499
5 /home/aske/Datasets/Market-1501/all/0937_c6s2_115018_00.jpg 936
6 /home/aske/Datasets/Market-1501/all/0264_c3s1_061292_00.jpg 263
7 /home/aske/Datasets/Market-1501/all/0263_c6s1_056151_00.jpg 262
8 /home/aske/Datasets/Market-1501/all/0629_c6s2_029568_00.jpg 628
9 /home/aske/Datasets/Market-1501/all/0200_c1s1_039976_00.jpg 199
10 /home/aske/Datasets/Market-1501/all/1479_c3s3_080744_06.jpg 1478
11 /home/aske/Datasets/Market-1501/all/0257_c3s1_064892_00.jpg 256
```

Figure B.1: Example of a text file used to create a database. Each line is a path to an image with a corresponding label.

input: *ImList* - List of images with labels
output: DB - LMDB database

; // Define new database and batch
$DB \leftarrow$ open new LMDB ;
$WB \leftarrow$ new batch ;
for *each image img in ImList* **do**
    $count \leftarrow 1$;
    ; // Write image data to datum
    $img_{C,H,W} \leftarrow$ transpose($img_{H,W,C}$) ;
    $datum.channel \leftarrow$ #Channels in img ;
    $datum.height \leftarrow$ #Height in img ;
    $datum.width \leftarrow$ #Width in img ;
    $datum.data \leftarrow img$;
    $datum.label \leftarrow img_{label}$;
    ; // Write datum to batch
    $WB \leftarrow$ Serialize($datum$) ;
    if $count = batch\ size$ **then**
        ; // Write batch to DB and create new batch
        $DB \leftarrow WB$;
        $WB \leftarrow$ new batch ;
        $count = 1$
    else
        $count = count + 1$;
    end
end
return DB ;

Pseudocode 10: Algorithm for creating LMDB database.

C. Structure of attachment

A. Single-shot test results.

B. Multi-shot and cross-dataset test results.

C. Discussion test results.

D. Drawings of re-implemented CNN and FFN.

Bibliography

Paris attacks: What happened on the night. `http://www.bbc.com/news/world-europe-34818994`, December 2015.

Stein Aerts, Diether Lambrechts, Sunit Maity, Peter Van Loo, Bert Coessens, Frederik De Smet, Leon-Charles Tranchevent, Bart De Moor, Peter Marynen, Bassem Hassan, et al. Gene prioritization through genomic data fusion. *Nature biotechnology*, 24(5):537–544, 2006.

Ejaz Ahmed, Michael Jones, and Tim K Marks. An improved deep learning architecture for person re-identification. In *Proc. CVPR*, pages 3908–3916, 2015.

Shi-Zhe Chen, Chun-Chao Guo, and Jian-Huang Lai. Deep ranking for person re-identification via joint representation learning. *arXiv preprint arXiv:1505.06821*, 2015a.

Ying-Cong Chen, Wei-Shi Zheng, and Jianhuang Lai. Mirror representation for modeling view-specific transform in person re-identification. In *Proc. IJCAI*, pages 3402–3408, 2015b.

Ying-Cong Chen, Wei-Shi Zheng, Jian-Huang Lai, and Pong Yuen. An asymmetric distance model for cross-view feature mapping in person re-identification. *IEEE Transactions on Circuits and Systems*, 2016.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

Raphael Felipe de Carvalho Prates and William Robson Schwartz. Appearance-based person re-identification by intra-camera discriminative models and rank aggregation. In *Proc. ICB*, pages 65–72, 2015.

Raphael Felipe de Carvalho Prates and William Robson Schwartz. Cbra: Color-based ranking aggregation for person re-identification. In *Proc. ICIP*, pages 1975–1979, 2015.

Michela Farenzena, Loris Bazzani, Alessandro Perina, Vittorio Murino, and Marco Cristani. Person re-identification by symmetry-driven accumulation of local features. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2360–2367. IEEE, 2010.

Itzhak Fogel and Dov Sagi. Gabor filters as texture discriminator. *Biological cybernetics*, 61(2):103–113, 1989.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

Shaogang Gong, Marco Cristani, Shuicheng Yan, and Chen Change Loy. *Person re-identification*, volume 1. Springer, 2014.

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

Martin Hirzer, Csaba Beleznai, Peter M Roth, and Horst Bischof. Person re-identification by descriptive and discriminative classification. In *Image Analysis*, pages 91–102. Springer, 2011.

Adam Hodges. A cmc curve depicting the performance of each respetive facial recognition algorithm. `http://www.ajhodges.com/#slide6`.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

Daniel J Jobson, Zia-ur Rahman, and Glenn A Woodell. A multiscale retinex for bridging the gap between color images and the human observation of scenes. *Image Processing, IEEE Transactions on*, 6(7):965–976, 1997.

Hongwen Kang, Alexei A Efros, Martial Hebert, and Takeo Kanade. Image matching in large scale indoor environment. In *Proc. CVPR*, pages 33–40. IEEE, 2009.

Andrej Karpathy. What a deep neural network thinks about your #selfie. `http://karpathy.github.io/2015/10/25/selfie/`, October 2015.

Josef Kittler, Mohamad Hatef, Robert PW Duin, and Jiri Matas. On combining classifiers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226–239, 1998.

Martin Koestinger, Martin Hirzer, Paul Wohlhart, Peter M Roth, and Horst Bischof. Large scale metric learning from equivalence constraints. In *Proc. CVPR*, pages 2288–2295, 2012.

Raivo Kolde, Sven Laur, Priit Adler, and Jaak Vilo. Robust rank aggregation for gene list integration and meta-analysis. *Bioinformatics*, 28(4):573–580, 2012.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1097–1105, 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Gil Levi and Tal Hassner. Age and gender classification using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 34–42, 2015.

Sheng Li, Ming Shao, and Yun Fu. Cross-view projective dictionary learning for person re-identification. In *Proceedings of the 24th International Conference on Artificial Intelligence, AAAI Press*, pages 2155–2161, 2015.

Wei Li, Rui Zhao, and Xiaogang Wang. Human reidentification with transferred metric learning. In *Proc. ACCV*, pages 31–44, 2012.

Wei Li, Rui Zhao, Tong Xiao, and Xiaogang Wang. Deepreid: Deep filter pairing neural network for person re-identification. In *Proc. CVPR*, pages 152–159, 2014.

Shengcai Liao and Stan Z Li. Efficient psd constrained asymmetric metric learning for person re-identification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3685–3693, 2015.

Shengcai Liao, Guoying Zhao, Vili Kellokumpu, Matti Pietikäinen, and Stan Z Li. Modeling pixel process with scale invariant local patterns for background subtraction in complex scenes. In *Proc. CVPR*, pages 1301–1306, 2010.

Shengcai Liao, Yang Hu, Xiangyu Zhu, and Stan Z Li. Person re-identification by local maximal occurrence representation and metric learning. In *Proc. CVPR*, pages 2197–2206, 2015.

Bruce Justin Lindbloom. Useful color equations, April 2003. URL `http://www.brucelindbloom.com/index.html?Equations.html`.

Chia Chin Lip and Dzati Athiar Ramli. Comparative study on feature, score and decision level fusion schemes for robust multibiometric systems. In *Frontiers in Computer Education*, pages 941–948. Springer, 2012.

Xiaokai Liu, Hongyu Wang, Yi Wu, Jimei Yang, and Ming-Hsuan Yang. An ensemble color model for human re-identification. In *Proc. WACV*, pages 868–875, 2015.

Timo Ojala, Matti Pietikainen, and David Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 582–585. IEEE, 1994.

Christoph Peters. Ycbcrcolorspace perspective, Februrary 2011. URL `https://en.wikipedia.org/wiki/Talk%3AYCbCr`.

Arun A Ross, Karthik Nandakumar, and Anil Jain. *Handbook of multibiometrics*, volume 6. Springer Science & Business Media, 2006.

Peter M Roth, Martin Hirzer, Martin Köstinger, Csaba Beleznai, and Horst Bischof. Mahalanobis distance learning for person re-identification. In *Person Re-Identification*, pages 247–267. Springer, 2014.

William Robson Schwartz and Antonio Carlos. Person re-identification results, 2016. URL `http://www.ssig.dcc.ufmg.br/reid-results/`.

SharkD. Hsv color solid, December 2015. URL `http://doc.qt.digia.com/qq/qq26-adaptivecoloring.html`.

Coats Sewing Solutions. Colour by numbers, September 2014. URL `http://www.coatsindustrial.com/en/information-hub/apparel-expertise/colour-by-numbers`.

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

Joshua M Stuart, Eran Segal, Daphne Koller, and Stuart K Kim. A gene-coexpression network for global discovery of conserved genetic modules. *Science*, 302(5643):249–255, 2003.

Milestone Systems. The superior search and analysis video surveillance solution now integrated with milestone xprotect, 2015. URL `http://www.tcit-us.com/wp-content/uploads/2015/01/Milestone_TCIT-Search-Brochure_20150108.pdf`.

Oncel Tuzel, Fatih Porikli, and Peter Meer. Region covariance: A fast descriptor for detection and classification. In *Computer Vision–ECCV 2006*, pages 589–600. Springer, 2006.

Evgeniya Ustinova, Yaroslav Ganin, and Victor S. Lempitsky. Multiregion bilinear convolutional neural networks for person re-identification. *CoRR*, abs/1512.05300, 2015. URL `http://arxiv.org/abs/1512.05300`.

Lin Wu, Chunhua Shen, and Anton van den Hengel. Personnet: Person re-identification with deep convolutional neural networks. *CoRR*, abs/1601.07255, 2016a. URL `http://arxiv.org/abs/1601.07255`.

Shangxuan Wu, Ying-Cong Chen, and Wei-Shi Zheng. An enhanced deep feature representation for person re-identification. In *Proc. WACV*, 2016b.

Jian Yang, Jing-yu Yang, David Zhang, and Jian-feng Lu. Feature fusion: parallel strategy vs. serial strategy. *Pattern Recognition*, 36(6):1369–1381, 2003.

Yang Yang, Jimei Yang, Junjie Yan, Shengcai Liao, Dong Yi, and Stan Z Li. Salient color names for person re-identification. In *Proc. ECCV*, pages 536–551. Springer, 2014.

Yang Yang, Shengcai Liao, Zhen Lei, and Stan Z Li. Large scale similarity learning using similar pairs for person verification. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Deep metric learning for person re-identification. In *Proc. ICPR*, pages 34–39, 2014.

Rui Zhao, Wanli Ouyang, and Xiaogang Wang. Person re-identification by salience matching. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2528–2535. IEEE, 2013.

Rui Zhao, Wanli Ouyang, and Xiaogang Wang. Learning mid-level filters for person re-identification. In *Proc. CVPR*, pages 144–151, 2014.

Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jiahao Bu, and Qi Tian. Person re-identification meets image search. *CoRR*, abs/1502.02171, 2015a. URL `http://arxiv.org/abs/1502.02171`.

Liang Zheng, Shengjin Wang, Lu Tian, Fei He, Ziqiong Liu, and Qi Tian. Query-adaptive late fusion for image search and person re-identification. In *Proc. CVPR*, pages 1741–1750, 2015b.