Department of Computer Science
University of Leicester

Master of Science Dissertation

Deep Multi-Stream Convolutional Neural Networks in Human Re-Identification

Yoav Alon

CO7501 Individual Project
Mentor/First Grader: Dr. Huiyu Zhou (Reader/Associate Professor)
Second Grader: Dr. Emma Tonkin (Associate Tutor)
Date of Submission: 03. September 2018

Abstract

The Task of human re-identification is one of the many challenges of computer vision where deep learning has proofed to be outperforming all classical methods of computer vision like Haar cascades and manual feature extraction. The following dissertation has the objective to approach the optimization of multi-Stream convolutional neural networks in respect of various competitive metrics like accuracy per epoch. Siamese, Triplet- and Quadruplet loss as forms of multi-stream networks use the shared weights of the same network and with the appropriate loss function are able to train the network in a the most effective way. The issue of Triplet mining search to find propose Triplets which are hard, meaning that the network is able to learn many features in short time.

Contents

Chapter 1

Introduction and Motivation

1.1 The task of Human Re-Identification

Human re-identification in video surveillance aims to match a person in different non-overlapping camera views. Even after revolutionary improvements due to recent deep learning approaches which try to train discriminative features of individuals, the problem of human is still considered as not entirely solved due to challenges like changes of appearance of an individual as a result of movement and the associated change of body pose, complex camera angles and varying light conditions.

The task also includes a number of assumptions such as a non changing cloth of each individual. In recent years, due to increasing computing abilities and larger datasets, Convolutional Neural Networks proofed to outperform all previous approaches of computer vision to solve this task.

The task of human re-id is the need of a view-, light- and pose-insensitive feature extraction/representation in order to re-identify any individual.

1.2 Practical Applications

1.3 Research Objective, Requirements and Evaluation

The research objective is to create a change in architecture and method in order to improve the performance of training for deep neural network in human re-id in respect to the most common metrics applied.

In most papers a vast number of assumptions is made in order to compare the most common metrics of different models. The improvement of accuracy rates per epoch is one of the most important metrics in order to compare various architectures and loss functions.

1.4 Contribution

Starting out from the existing strategies of hard triplet mining and online mining used in Triplet-Loss and Quadrupel Loss functions we created a completely new Loss function based on the center of gravity of a cluster and the average/maximum distance to embeddings in the cluster, which has to be smaller the the distance of the center of gravity to the nearest center of gravity of another cluster. In this way the number of operations is much smaller then in online triplet loss, and the training is more effective and faster.

1.5 Results

We were able to proof the the proposed loss function has a better performance in nearly all metrics.

1.6 Challenges

Challenges of the human re-identification can mainly deducted to the change of appearance of an individual due to factors such as movement and associated change of body pose, the camera angle and light conditions. The task also includes a number of assumptions such as a non changing cloth of each individual. Some of the challenges like a changing of cloth can be addressed by the work flow of the re-id software, as detecting the change of cloth of a person in the same position as event, or using areas invisible to cameras such as restrooms as a stack storing the individuals entering and exiting also taking the time into account and identify a (drastically) change of appearance.

Another challenge is the enabling a re-id solution in real-time. This can be done using GPU's and pre-trained weights for the network using Transfer learning, which has pre-trained the network to automatically extract features which are common to most persons.

1.7 Related Work

The latest deep learning approaches of the human re-id task clearly outperform other machine learning approaches. The most papers

Chapter 2

Technical Requirements

2.1 Common Human Re-Id Datasets

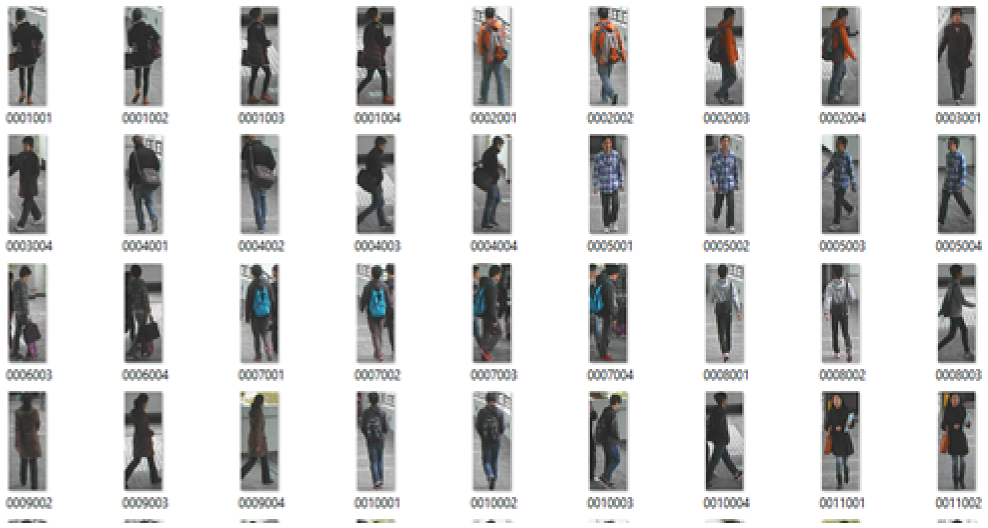
An important factor in the task of human re-id are the datasets use in (pre-)training of the network. As beeing a task of supervised learning each dataset is required to be labeled, which is a certain challenge. Even the biggest public available datasets are not enough get similar results which have been achieved in for example in face recognition, where the datasets are multiple times as big.

When analyzing the commune phrase 'public available dataset' we can easily imagine that governments or large companies may use non public datasets of with a hughe number of samples.

In the older datasets most detection, and setting of the bounding boxes in order to extract the image of a single person were done manually. In advanced sets we also have the ability to use data which is the result of automated detection, which proofs to produce samples more similar to real live data.

For initial experiments we used short movies and a detection algorithm which detects persons in a frame and and algorithm to follow up on a person. This having the advantage of using movement features of a person such as optical flow or optical flow farneback.

By using a dataset like the following **CUHK01** we ignore a large number of real live features obtainable from the movement of a person like pose, walking, etc. But in order to compare the results of different architectures in Human Re-ID we have to use standartized datasets.



For the initial experiments we used the dataset **CUHK01** ([Li] 2012) containing 971 identities with for images for each person tooken by two different camera angles (two images each), resulting in 3884 images overall.

The larger dataset **CUHK03** ([Li] 2014) contains 13164 images from 1360 identities.

In the state of the art dataset **Market-1501** (2015 [LZT]) containing 32668 images of 1501 identities we may choose between images extracted from an automated person detection algorithm and manually boxed images. Especially for production this is a important factor.

For purpose of comparing we also use the **VIPeR** (Gray, Brennan, and Tao 2007) dataset

2.2 Required Software

We used the standard state of the art **Tensorflow 1.x** in order to create the neural networks and flow necessary in order to train. For different purposed we use the tensorflow based **Keras** in order to experiment with simple architectures, both based on Python 3.6. **Numpy**, **Pandas** are of use for data preparation and **sklearn** assists us with a few methods for deep learning preprocessing like normalization used in Triplet Loss. **Matplolib** a python based library is used to vizualise our results.

2.3 Required Hardware

For almost all Tensorflow training session running on a normal CPU is time consuming. In our experiments the same Tensorflow training which took about an hour on a simple CPU could be completed within a few seconds using a GPU.

Todays AWS (Amazon Web Services) offers a cheap alternative to for research and production to use GPU calculation power.

For the most calculations done in this dissertation we used the Google Cloud based Colab GPU. The advantage in a short calculating time is the ability to try various hyper-parameters of a network training in a short period of time and find implications for further research (As done in the chapter regarding data augmentation).

Chapter 3

Theory of Convolutional Neural Networks

3.1 Convolutional Neural Networks

Convolutional Neural Networks are a subclass of deep neural networks typically used in computer vision, speech recognition and many other applications. Due to their deep feature learning architecture they are perfect for image classification like in human re-id. Similar to regular neural network they are made up of a architecture various layers such as convolutional layers. We will have a

3.1.1 Convolutional Layer

Two dimensional Convolution applies a filter K on an image I in form of a matrix multiplication. In a convolutional neural network there is a certain numbers of filters which will be applied on the image but the weights, the elements of the filter are not given, but have to be calculated.

3.1.2 Max Pool Layer

The purpose of the max pool layer is to reduce the dimensions of an image. Other methods like average pooling are possible, but max pool proofed good results for the task of convolutional neural network. Normally a 'block' consists of a number of convolutional layers with a max pool layer in the end. And the complete network consists of a few of these blocks with final layers.

3.1.3 Dense Layer

here text about dense layers

3.1.4 Activation function

Activation means, which neurons to fire. The activation function used is ReLu function f(x) = max(0,x). It is not a layer, but applied element wise after each layer. This introduces nonlinearity which makes generalization possible.

Training of the above neural networks means to use backpropagation in order to find the weights and biases for example of the convolutional filters of each layer which optimize the final cost function which in convolutional neural networks is based on cross entropy since the nature of the network is a classifying one.

3.2 Data Augmentation

3.3 Transfer learning

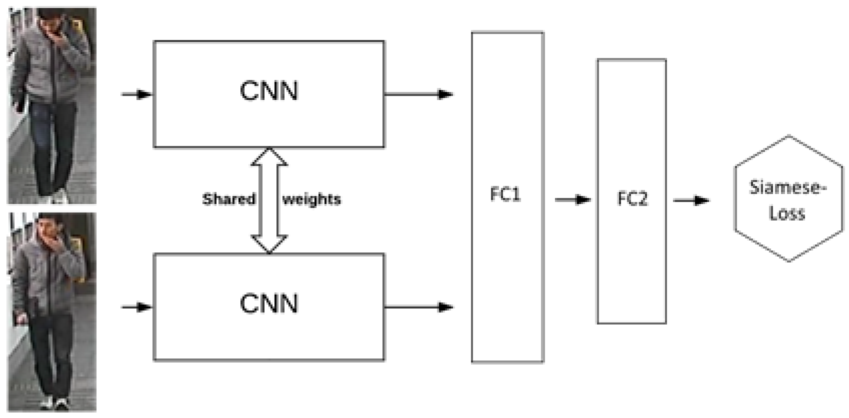
3.4 Backpropagation and Optimization Algorithm

Chapter 4

Multi-Stream Convolutional Neural Networks

4.1 Siamese Networks

Siamese Networks can extract a metric for similarity of images. In this way a Convolutional Neural Network can be trained by minimizing the Siamese-Loss with gradient descent for identical identities. Both of the Neural Networks are identical in structure of the model and share their weights. In this way two streams are send to the Siamese Loss.



The first image is passed through the CNN parallel two the second and the outputs are processed by fully connected layers and in the end a loss function can compare the outcome and extract a metric for similarity. This metric can be minimized. And on the other hand the Siamese loss of two different identities should be higher than the loss of two identical identities.

If the first person is x_i and the second is x_j and the encoding resulting from a pass through the convolutional neural network is $f(x_i)$ or short f_j then the Siamese loss can be defined as:

L(i,j) $= \|f_i - f_j\|_2^2$

4.2 Applications

Applications.

4.3 Future Work

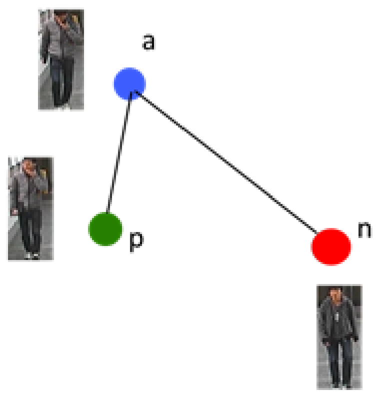
Future Work.

Chapter 5

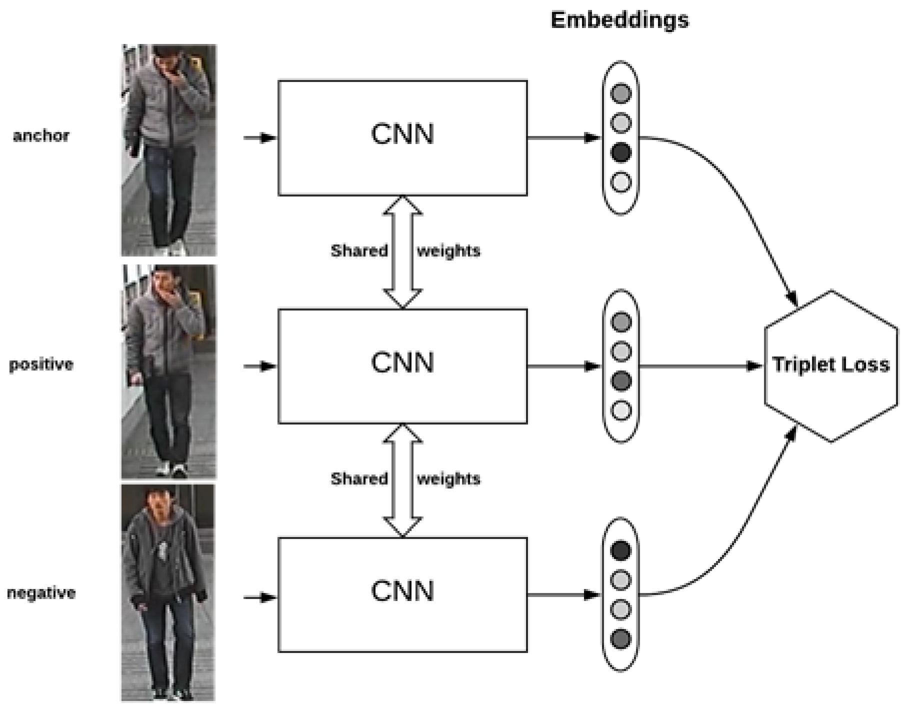
Triplet Loss in 3-stream Convolutional Neural Networks

5.1 Simple Triplet Loss

As we have concluded in respect to Siamese Convolutional Neural Networks, that the Siamese Loss as a metric for similarity should be higher when having to identical identities then when having different identities. This is exactly what we use in triplet loss. We create a Triplet containing two different images of the same identities and an image of a different person. The first image serves as an anchor a, the other image with the same identity serves as positive images p and the image of the different identity is n for negative. In this way we a batch of Triplets (a,p,n).



We now could apply a and p in our Siamese Neural Network and minimize Loss, or use a and n and maximize loss. Instead we send all tree images to our Convolutional Neural Network and extract a so called embedding. A representation of similarity in R^2.



The Euclidian distance of the embeddings in R^2 of the pair of anchor and positive $\|f_a - f_p\|_2^2$ must be smaller than the Euclidian distance of the pair of anchor and negative distance $\|f_a - f_n\|_2^2$

$$\|f_a - f_p\|_2^2 \leq \|f_a - f_n\|_2^2$$

For each image in the triplet a two dimensional position is extracted from the neural network, and the distance of two images represents their similarity.

We can restructure the equitation as:

$$\|f_a - f_p\|_2^2 - \|f_a - f_n\|_2^2 \leq 0$$

But this equitation would be easily fulfilled so we have to add a margin α

$$\|f_a - f_p\|_2^2 - \|f_a - f_n\|_2^2 + \alpha \leq 0$$

or in another representation following [AHL17a]

$$L_{tri}(\theta) = \sum_{a,p,n} [\alpha + D_{a,p} - D_{a,n}]_+$$

where as $D_{a,p}$ is a abreviation for the distance $\|f_a - f_p\|_2^2$ So the loss function can be considered as:

$$L(a,p,n) = max(\|f_a - f_p\|_2^2 - \|f_a - f_n\|_2^2 + \alpha, 0)$$

Usually the triplets can be chosen randomly, but the best results for a training are achieved when the similarity of anchor and positive $\|f_a - f_p\|_2^2$ is very similar to the similarity of anchor and negative $\|f_a - f_n\|_2^2$. If this is the case the Neural Network has to extract more features in order to discriminate the identities, and this is the really effective learning.

The following is an example of an implementation of a Triplet Loss function in Tensorflow. Keep in mind that tf.reduce$_m eanisusedsinceabatchofTripletsisprocessed.$

```
def TripletLoss(self) :

    anchor_feature = self.output3
    positive_feature = self.output2
    negative_feature = self.output1

    with tf.name_scope("triplet_loss"):
        pos_dis = tf.reduce_mean(tf.square(tf.subtract(anchor_feature,
        positive_feature)),1, keepdims=True)
        neg_dis = tf.reduce_mean(tf.square(tf.subtract(anchor_feature,
        negative_feature)),1, keepdims=True)

        res = tf.maximum(0., pos_dis + margin - neg_dis)
        loss = tf.reduce_mean(res)

        return loss
```

The following example has a dataset which is labeled in order to recognize images of males and females. We have to train a triplet net in order to be able to discriminate males from females with our CNN. Green dots stand for images of a male, and red for images of a female. The Triple CNN extracts a 2 dimensional representation of the image similarity in form of a position for each image. Points similar to each other should have a high similarity. When looking at epoch 1 we see that the embedding's of male and female are nearly randomly. But with further epochs the Triplet-Net learns to detect similarity better and points with a high similarity are closer to each other. Within a few epochs we can clearly observe a separation between embedding's for males and females.

As an optimizer for training we may use a simple gradient descent or Adam optimizer

```
optimizer = tf.train.AdamOptimizer(learning_rate = learningRate).minimize(model.loss)
```

The features described in the function are simply images which have been passed to our neural network

```
self.x1 = tf.placeholder(tf.float32, [None, 784])
self.x2 = tf.placeholder(tf.float32, [None, 784])
self.x3 = tf.placeholder(tf.float32, [None, 784])
self.y_ = tf.placeholder(tf.float32, [None,2],)

with tf.variable_scope("triplet") as scope:
    self.output1 = self.network(tf.reshape(self.x1,[batchSize,28,28,1]))
    scope.reuse_variables()
    self.output2 = self.network(tf.reshape(self.x2,[batchSize,28,28,1]))
    scope.reuse_variables()
    self.output3 = self.network(tf.reshape(self.x3,[batchSize,28,28,1]))
    self.loss = self.TripletLoss()
```

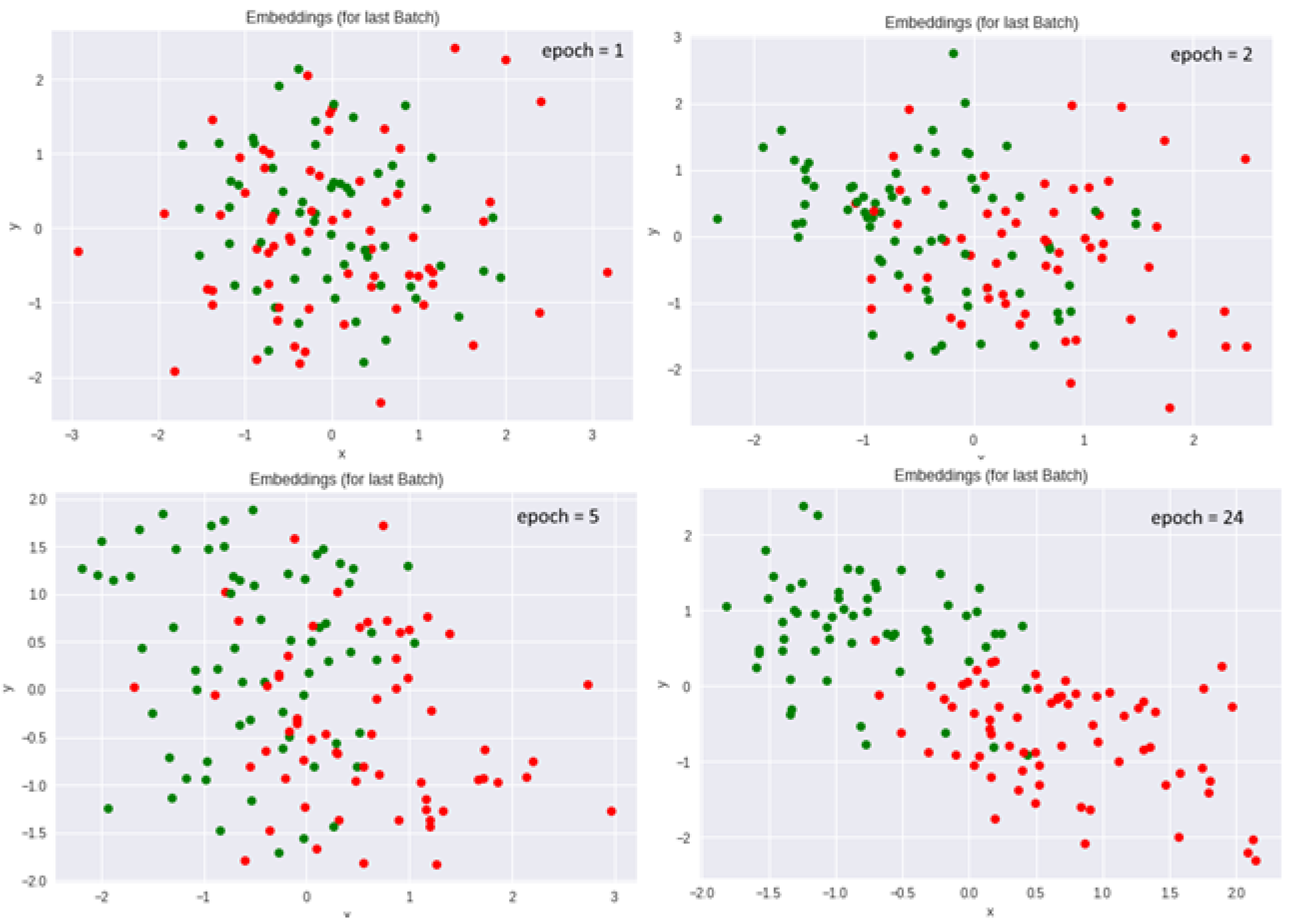


Figure 5.1: Embeddings for a two-class classification problem during training

After having trained the Triplet-Net it has obtained the ability to extract a metric for similarity of images. But how do we classify new images? When having a new point (blue) we have the choice to classify either by adding additional fully connected layers to the network or performing a KNN algorithm.

In the example of a new image (blue point in 2 dimensional embedding's) we can use a KNN (k-nearest neighbors) algorithm in order to classify the new point.

Lets choose k=5

Then we can look at the labels of the 5 nearest neighbors and by the majority of the labels we can decide upon which class the new images belong.

In the above case for a k of 5 we find 3 male points and 2 female points. Our code goes for the majority of points and classifies the new point as male. If k would have been chosen a bit higher the classification would be different. Here a code example of how we use KNN in order to classify a new image.

```
def Predict(self, input, k = KNN_k):              #Prediction using KNN

    neg_one = tf.constant(-1.0, dtype=tf.float32)
    distances =  tf.reduce_sum(tf.abs(tf.subtract(self.output3, input)), 1)
    neg_distances = tf.multiply(distances, neg_one)
    vals, indx = tf.nn.top_k(neg_distances, k)
    prediction = tf.gather(self.y_, indx)

    res = tf.reduce_sum(prediction,0)
    index = tf.argmax(res)

    return index
```

5.2 Batch All Triplet Loss

[AG17]

5.3 Hard Triplet Loss

5.4 Quadruplet Loss

The idea of a quadruplet loss was first introduced in [AHL17b] as a reaction to a common performance problem of Triplet Loss which suffer from a weak ability of generalization from training set to test set.

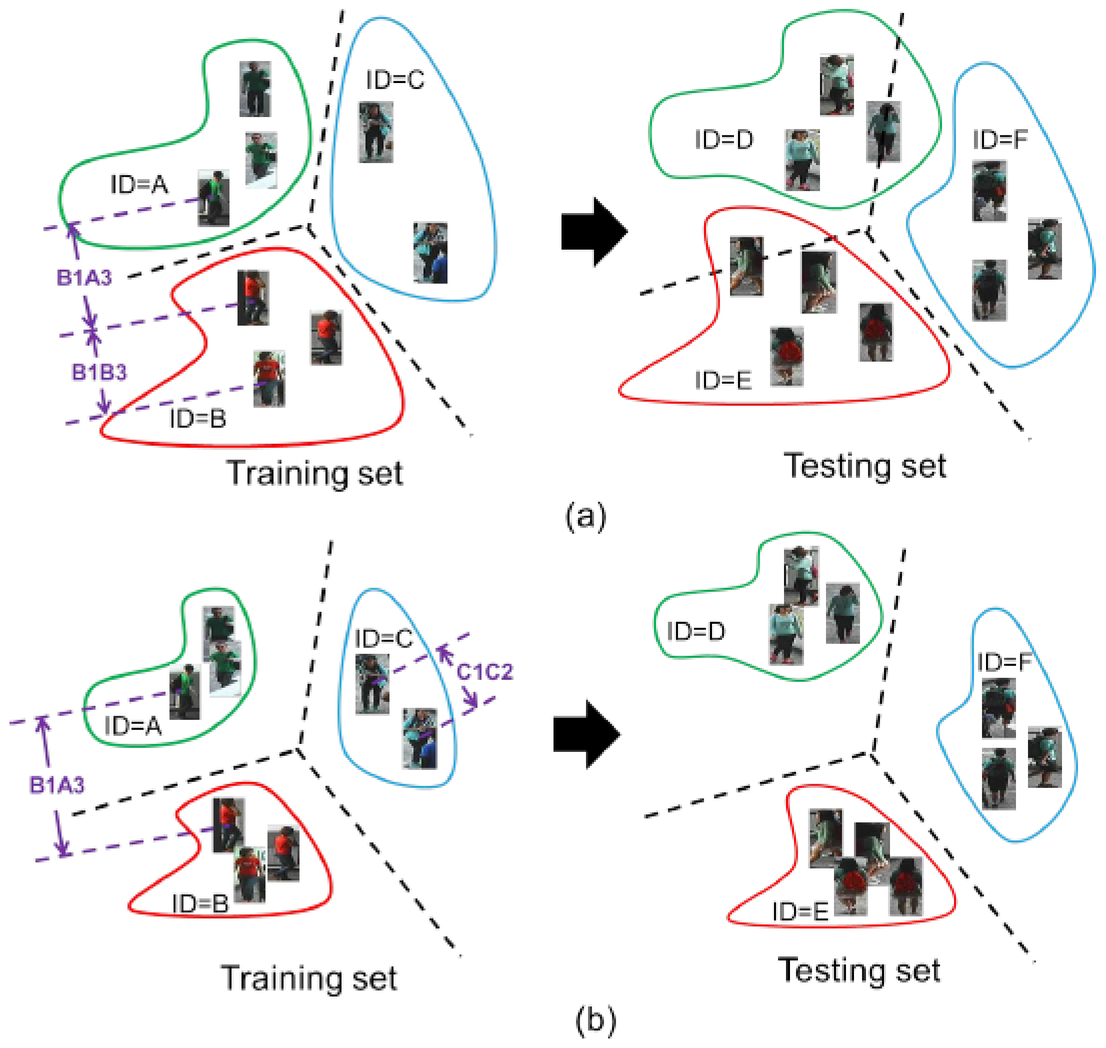


Figure 5.2: Quadruplet Loss [AHL17b]

In quadruplet loss a quadruplet of anchor, positive, and two negative images (a,p,n1,n2) is feed into a 4-stream network with the following loss:

$$L_q = \sum_{i,j,k}^{N} \|f_a - f_p\|_2^2 - \|f_a - f_{n1}\|_2^2 + \alpha 1 + \sum_{i,j,k}^{N} \|f_a - f_p\|_2^2 - \|f_a - f_{n2}\|_2^2 + \alpha 2$$

Unlike triplet loss two margins $\alpha 1$ and $\alpha 2$ where as generally $\alpha 1 > \alpha 2$

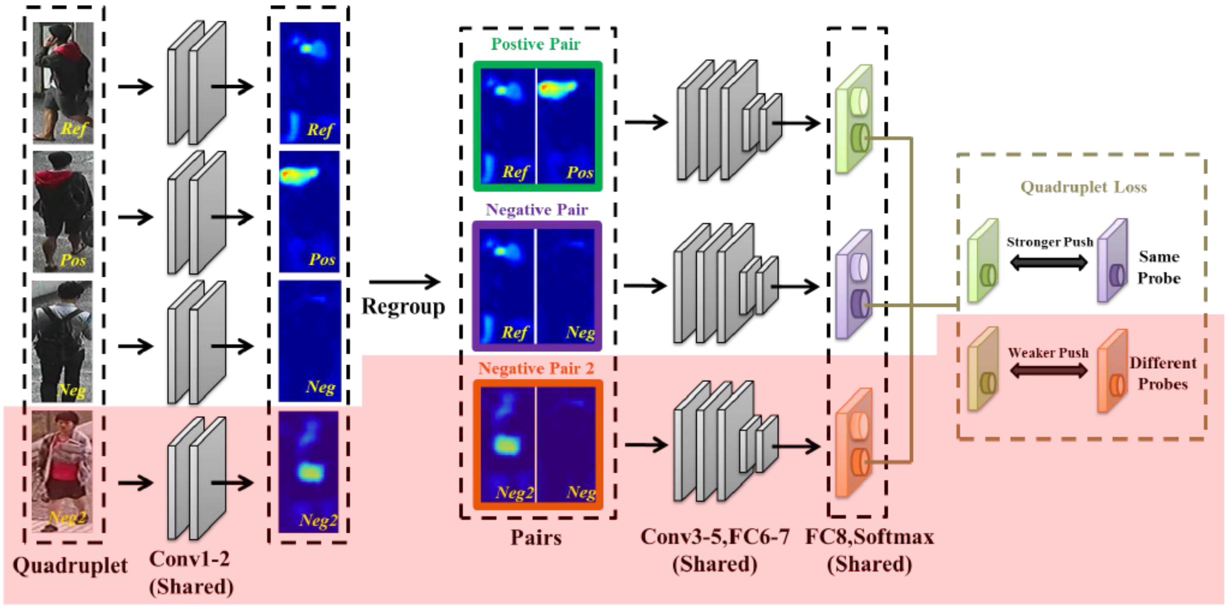


Figure 5.3: Architecture of a Quadruplet Network [AHL17b]

The table 5.1 compares the current results for different Triplet-Loss approaches according to [AHL17a]

Accuracy after 1 rank	
Method	rank-1
Triplet	59.23
Triplet + OHM	36.6
Batch hard	83.51
Batch all	79.24
Batch all(BAnot 0)	83.65
Lifted 3-pos	82.71
Lifted 1-pos	81.35

Table 5.1: Accuracy after one epoch for different methods of Triplet-Loss

Chapter 6

Proposed Hard Triplet KNN Algorithm

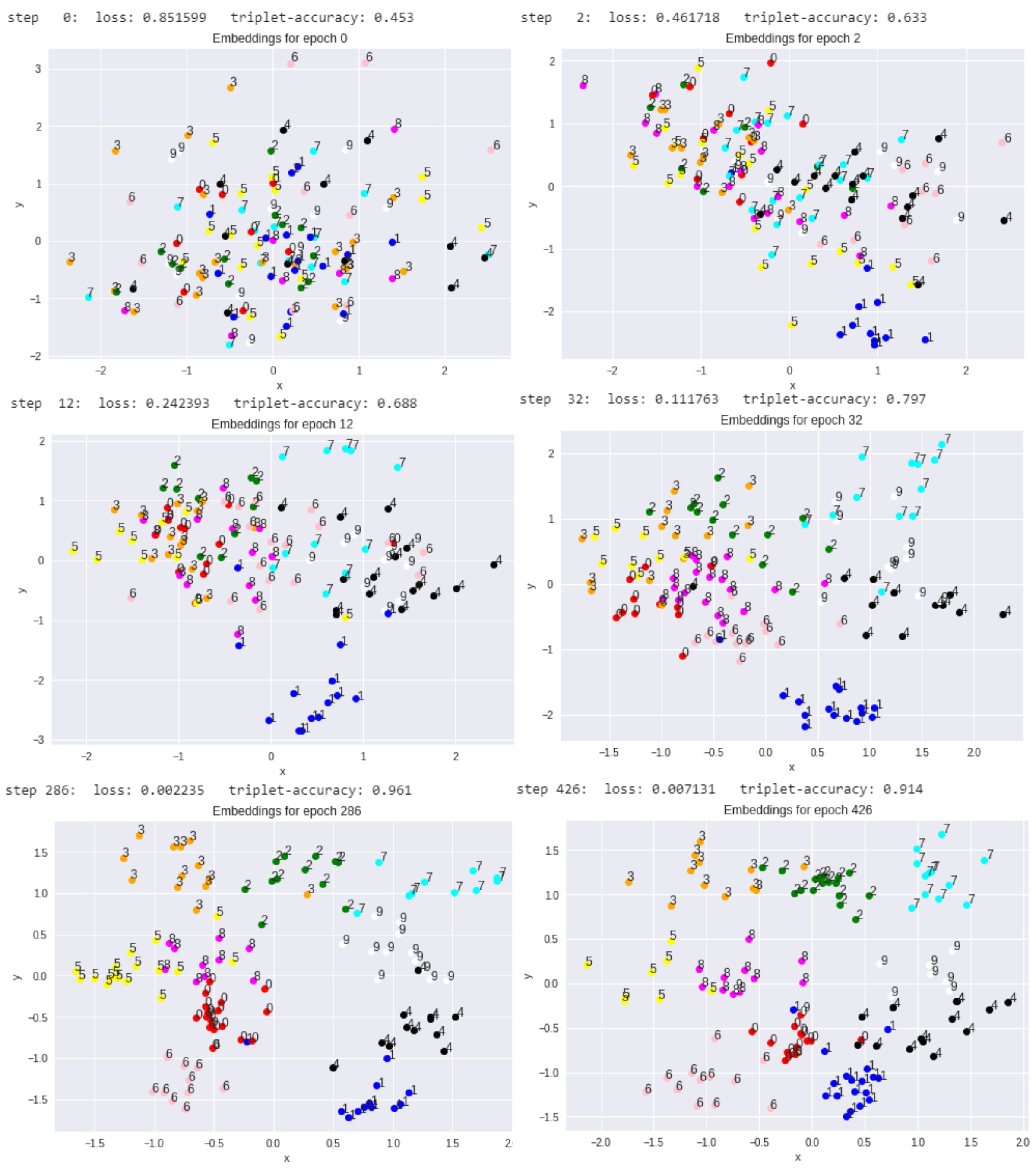
6.1 Disadvantage of current Methods for Triplet Loss

A major influence of the performance of a Triplet Loss network is beyond the Loss function also the way in which we pick the Triplets. Normally at each epoch Triplets are picked randomly. That means a random point is picked, and then a while loop is executed as till an element with the same label has been found. Then another while loops goes over random elements till a element with a different label is found. In this way we may get a Triplet (a,p,n) with the labels (2,2,9).

```
# Create random Triplet and assign binary Label
def GetTriplet(mnist) :

  ran = np.random.randint(0,mnist.train.labels.shape[0], 1)
  lab = mnist.train.labels[ran]

  sames = GetDifferentIdentityImage(mnist, lab)
```



```
step    0:  loss: 0.851599    triplet-accuracy: 0.453
step    2:  loss: 0.461718    triplet-accuracy: 0.633
step   12:  loss: 0.242393    triplet-accuracy: 0.688
step   32:  loss: 0.111763    triplet-accuracy: 0.797
step 286:  loss: 0.002235    triplet-accuracy: 0.961
step 426:  loss: 0.007131    triplet-accuracy: 0.914
```

Figure 6.1: Triplet-Loss with random Triplet picking in each epoch

```
  return np.array([ran, sames[0], sames[1], sames[2]])


#Creates batch of shape (128,4) where as 128 is batch size and 4 stands for a triplet pl

def CreateTripletBatch(mnist) :

  Triplet_Set = []
  for i in range(batchSize) :

    Triplet_Set.append(GetTriplet(mnist))


  return np.array(Triplet_Set)
```

The Batch All approach generates every possible Batch, which is a quite cost intensive process.

The mayor problem with the random picking of Triplets is that the neural network doesn't **learn** from a Triplet which is well sorted, that means where the condition of:

$$\|f_a - f_p\|_2^2 - \|f_a - f_n\|_2^2 + \alpha \leq 0$$

is already fulfilled. Meaning a point is in the right cluster already.

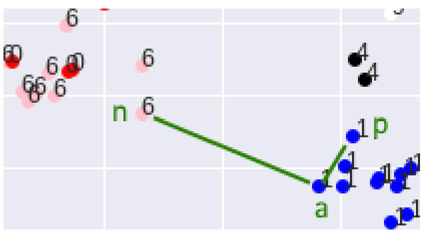


Figure 6.2: The Triplet condition is already fulfilled, points are in their respective cluster. The neural network can't improve by using this Triplet

But taking a point which is not to be found within it's cluster we can construct a Triple which doesn't fulfilled the condition of:

$$\|f_a - f_p\|_2^2 - \|f_a - f_n\|_2^2 + \alpha \leq 0$$

And by such a Triplet the neural network can improve, since the cluster is not well sorted yet.

The logical consequence is that when picking Triplets we just have to choose Triplets where the anchor doesn't belong to a cluster of the same color such as in

in Figure 6.4 we can see a point which is to be found of his cluster. So this is a good point to create a Triplet, which the neural network really can learn from. The goal is to create a maximum loss in the loss function for all Triplets chosen.

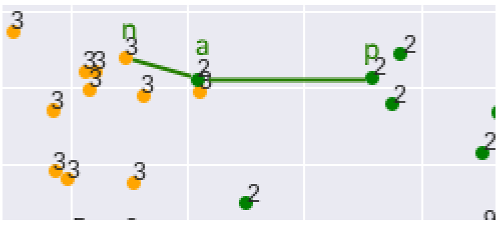


Figure 6.3: The Triplet condition is not fulfilled, points is in a foreign cluster. The neural network can improve by using this Triplet

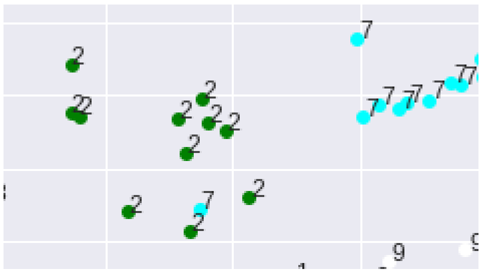


Figure 6.4: A point of Label 7 is not in the right cluster, so it is a good point to create a Triplet

6.2 Proposed Algorithm for Triplet Creation

We will need to identify points like in Figure 6.4 which are not in their cluster in the beginning of each epoch in order to create the Triplets. This can be done best by using the knn (k-nearest-neighbours) - algorithm. We may pick a random point, calculate his nearest neighbours and then decide if the point qualifies as not beein in his cluster.

The proposed algorithm will at the beginning of each epoch create a set of Triplets in order to determine the loss. To the end of the algorithm no Triplets should be found anymore which don't fulfill the condition of:

$$\|f_a - f_p\|_2^2 - \|f_a - f_n\|_2^2 + \alpha \leq 0$$

Here the proposed steps:

1. Choose a random point **a** from the batch.
2. Use the knn-algorithm with k=2 to determine the labels of **a**'s 2 nearest neighbours.
    - If all neighbours are identical (e.g. (2,2,2)), then don't create a Triplet for **a**

- If neigbours not identical (e.g. (2,4,4)) then create Triplet for **a** in such a way, that **a** and the first of the nearest neighbours which was not identical is the negative **n**, then use a while till you find a point with the same label as **a** (which probably will be in the right cluster) and use it as a positive example **p**. This creating the triplet (**a**, **p**,**n**)

3. Use created Triplets as input for the Triple-Network and calculate the loss function

As indicated above the number of Triplets created in every batch will decrease in every epoch.

6.3 Hard Triplet Mining

The idea of finding triplets with embeddings which are not within their cluster is known as Triplet Mining. And other then the above proposed algorithm there is another one described in [Moi]

In KNN algorithm it is necessary to calculate the pairwise distances of all points. A distance matrix can be created.

Instead of going over KNN to find hard Triplets, we can also use the distance matrix directly and get hard positives and hard negatives.

All valid pairs remain in the matrix, that means distances of the images of the same identity. The other values are set to 0.

And then taking the maximum from each row of the remaining matrix. The hardest negative can be found by

In order to get the hardest negative we compute the minimum distance of each row.

$$tripletLoss = max(hardPositive - hardNegative + margin, 0)$$

It seems that the algorithm described in [Moi] is faster.

Chapter 7

Proposed Gravity Loss Function based on Center of Gravity of a cluster

7.1 Proposed Gravity Loss Function

Let be x_i the embeddings for a cluster of the same identity with the center of gravity R and the closest center of gravity of another cluster $\bar{R}$ (determined by obtaining the center of gravity for each cluster and building a distance matrix δ_{ij} of the centers. Then taking the minimum of the row of the current center.) The average distance of embeddings from their center of gravity for must be smaller then the halve distance between R and $\bar{R}$ plus a margin.

$$\frac{1}{N}\sum_{n=1}^{N}\|R-x_i\|_2^2+\alpha \leq \left\|R-\bar{R}\right\|_2^2$$

we then can shift the distance of centers to the left:

$$\frac{1}{N}\sum_{n=1}^{N}\|R-x_i\|_2^2-\frac{1}{2}\left\|R-\bar{R}\right\|_2^2+\alpha \leq 0$$

And for usage with an gradient decent optimization function we have to formulate our equation as minimization problem using maximum function:

$$L(x_i, R, \bar{R}) = max(\frac{1}{N}\sum_{n=1}^{N}\|R - x_i\|_2^2 - \frac{1}{2}\left\|R - \bar{R}\right\|_2^2 + \alpha, 0)$$

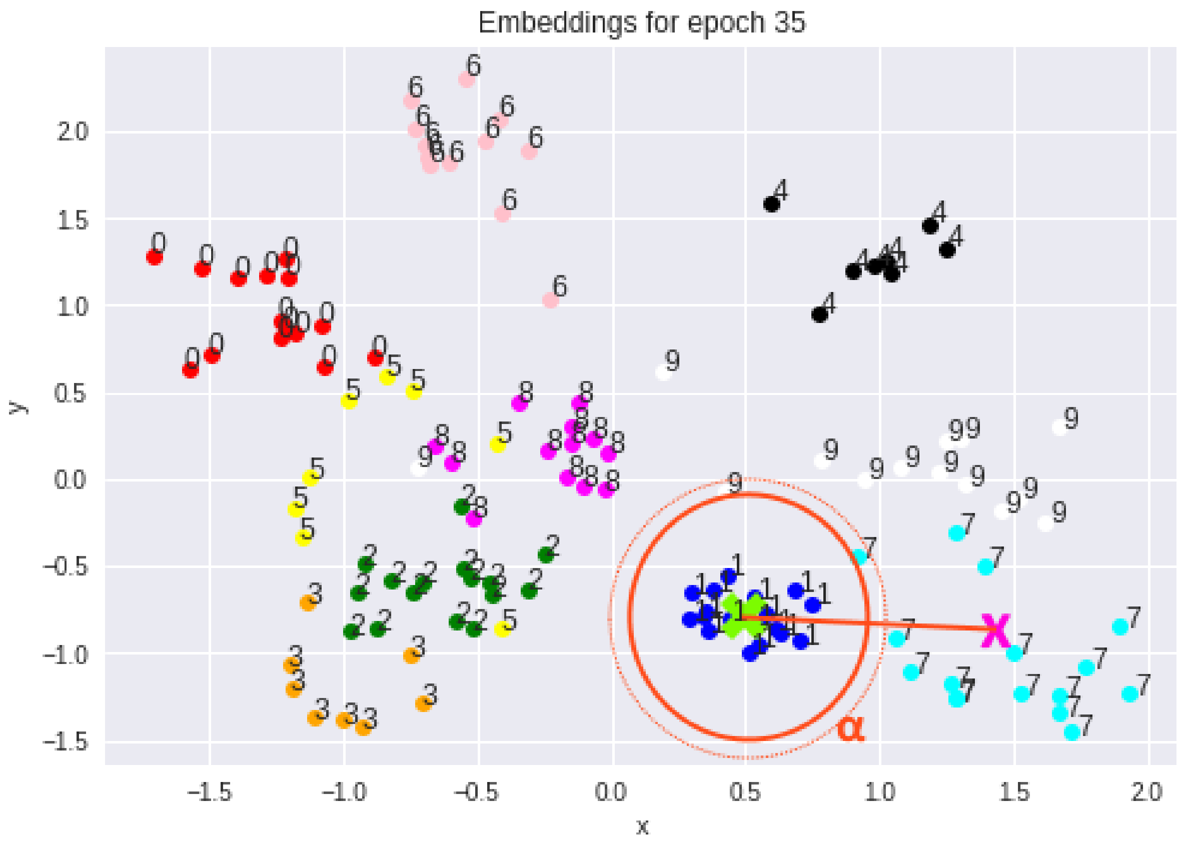


Figure 7.1: Gravity Loss

Another way to even improve the loss function would be adding an aspect which pushes the centers of gravity away from each other in order to avoid building of unsymmetrical clusters.

We could determine a constant minimum distance which and determine the difference between this distance to the actual distance and punish the network for having unequal distances.

The elements of the distance matrix δ should be as close as possible to the constant distance c defined, so that $\delta_{ij} \approx c$

and we could ad the following factor to the loss function:

$$\alpha_2 \left\|\delta_{ij} - c\right\|_2^2$$

in a way that α_2 would be a parameter in order to estimate the influence of the loss of distances between centers. So the overall Loss function which cares to provide equal distances betwen centers of gravity could be:

$$L(x_i, R, \bar{R}) = max(\frac{1}{N}\sum_{n=1}^{N} \|R - x_i\|_2^2 - \frac{1}{2}\left\|R - \bar{R}\right\|_2^2 + \alpha + \alpha_2 \|\delta_{ij} - c\|_2^2, 0)$$

Implementation of our Gravity loss funciton

```
def ProposedLoss(self, labels, embeddings):   #KNN accuracy

    #1. Get all embeddings for a label
    #2. Calculate Center of Gravity

    lab_hot = tf.one_hot(tf.cast(labels,tf.int32),10)   #10 is number of labels
    res = tf.boolean_mask(embeddings, lab_hot[:,1])

    centerOfMass = tf.reduce_mean(tf.cast(res, tf.float32), axis=0, keep_dims=None)

    #2.5 create all COM's
```

Chapter 8

Detection Networks

8.1 Detection in Human Re-Id

The task of object detection is generally considered to be more complex then classification alone. With the ability to classify an object comes the necessity to detect possible objects. This of course can be solved by applying an old fashioned person detection algorithm on a video or real-time video stream and extract possible candidates for persons first, and then send each person through an convolutional neural network in order to classify the person. In a camera view of a public place such as a train station or airport the detection algorithm might suggest us hundreds of persons per frame, which all would have to be send through our classification convolutional neural network. It is not hard to see that this would be quite cost intense. We will see in 8.2 that a algorithm like R-CNN does exactly this. After proposing a number of so called region proposals each proposed object (about 2000) is send through our classification convolutional neural network. The Fast R-CNN described in 8.3 makes this process less complicated by sending all proposals only through one convolutional neural network. The Faster R-CNN and SSD (Single Shoot Multibox Detection) described in 8.4 and in are able to detect objects which were trained in the containing (mostly vgg16/vgg19) network.

In figure 8.1 we see that a library is able to detect and suggest regions for persons in a frame. This comes with the challenge to classify a person or recognize that the person is not yet in

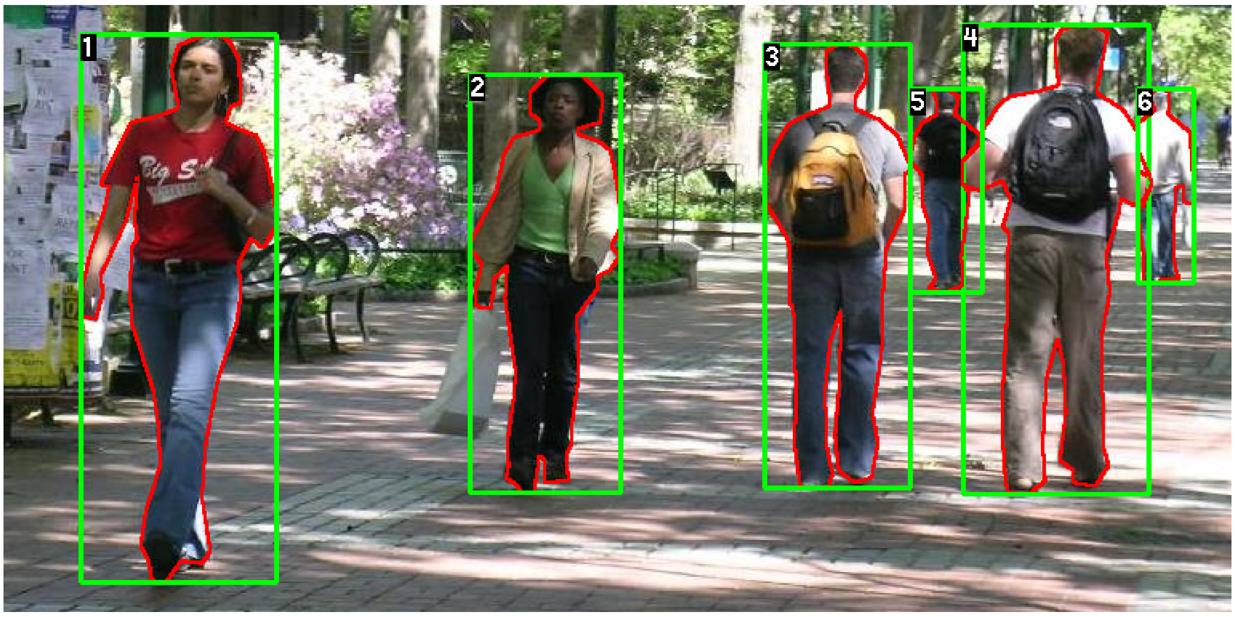


Figure 8.1: A simple Person Detection Algorithm returning a list of bounding boxes which contain persons as in Dataset: [LW07]

the data set and being able to add the person to our data set. A complete work flow has to be constructed and questions like what happens when a persons changes his appearance as defined in the Re-Id task by e.g. simply taking of his jacket. A work flow has to be able to replace the images of the identity by new images.

After a short introduction of the various Region-CNN algorithms we want to ask the question if we may combine the idea of a SDD network which is able to detect and classify several persons on an image together with the idea of multiple stream networks. Or short, may we train a SSD architecture with Triplet loss in the CNN.

8.2 R-CNN

8.3 Fast R-CNN

8.4 Faster R-CNN

8.5 SSD

The basic SSD architecture described in figure 8.2 replaces fully connected layers of the VGG16/19 architecure and extracts region proposals in convolution.

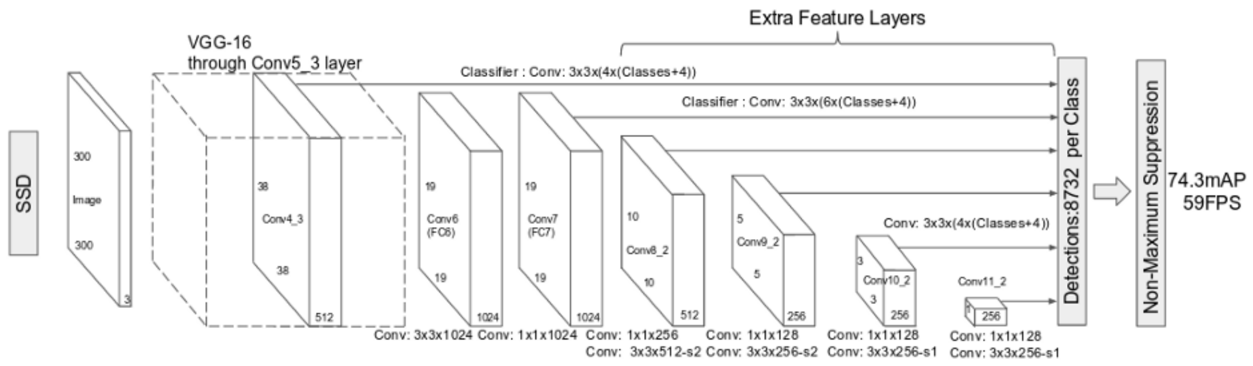


Figure 8.2: The basic SSD architecture as described in: [WL16]

8.6 Mask R-CNN

8.7 Multiple Stream Detection Networks

Bibliography

[AG17] Amita Kapoor Antonio Gulli. *TensorFlow 1.x Deep Learning Cookbook.* Packt, 2017.

[AHL17a] Lucas Beyer Alexander Hermans and Bastian Leibe. In defense of the triplet loss for person re-identification. *Researchgate*, 2017.

[AHL17b] Lucas Beyer Alexander Hermans and Bastian Leibe. Weihua chen, xiaotang chen, jianguo zhang, kaiqi huang. *CVPR*, 2017.

[Li] Li. Cuhk01 datasets.

[LW07] Gang Song I-fan Shen Liming Wang, Jianbo Shi. Object detection combining recognition and segmentation. *ACCV*, 2007.

[LZT] L. Tian S. Wang J. Wang L. Zheng, L. Shen and Q. Tian. Scalable person re-identification: A benchmark. *ICCV year =*.

[Moi] Olivier Moindrot. Hardtripletloss.

[WL16] Dumitru Erhan Christian Szegedy Scott Reed Cheng-Yang Fu Alexander C. Berg Wei Liu, Dragomir Anguelov. Ssd: Single shot multibox detector. *ECCV*, 2016.