LiU-ITN-TEK-A--21/047-SE

Automated Gait Analysis - Using Deep Metric Learning

Isak Engström

2021-06-21

Department of Science and Technology
Linköping University
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings universitet
601 74 Norrköping

LiU-ITN-TEK-A--21/047-SE

Automated Gait Analysis - Using Deep Metric Learning

The thesis work carried out in Medieteknik
at Tekniska högskolan at
Linköpings universitet

Isak Engström

Norrköping 2021-06-21

Department of Science and Technology
Linköping University
SE-601 74 Norrköping, Sweden

Institutionen för teknik och naturvetenskap
Linköpings universitet
601 74 Norrköping

Abstract

Sectors of security, safety, and defence require methods for identifying people on the individual level. Automation of these tasks has the potential of outperforming manual labor, as well as relieving workloads. The ever-extending surveillance camera networks, advances in *human pose estimation* from *monocular cameras*, together with the progress of *deep learning* techniques, pave the way for automated walking gait analysis as an identification method. This thesis investigates the use of *2D kinematic pose* sequences to represent gait, monocularly extracted from a limited dataset containing walking individuals captured from five camera views. The sequential information of the gait is captured using *recurrent neural networks*. Techniques in *deep metric learning* are applied to evaluate two network models, with contrasting output dimensionalities, against *deep-metric-*, and *non-deep-metric*-based *embedding spaces*. The results indicate that the gait representation, network designs, and network learning structure show promise when identifying individuals, scaling particularly well to unseen individuals. However, with the limited dataset, the network models performed best when the dataset included the labels from both the individuals and the camera views simultaneously, contrary to when the data only contained the labels from the individuals without the information of the camera views. For further investigations, an extension of the data would be required to evaluate the accuracy and effectiveness of these methods, for the re-identification task of each individual.

Acknowledgments

I wish to express my gratitude towards FOI for the opportunity to work with them and their resources during this thesis. Special thanks goes out to Henrik Petersson for the invaluable discussions, feedback and commitment. Further, I wish to extend thanks towards my supervisor Saghi Hajisharif, and examiner Ehsan Miandji, for the overall, and continuous support.

Contents

List of Figures

List of Tables

1 Introduction

Finding automated ways for identifying individuals is crucial in many sectors of security, safety and defence. Such sectors include criminal investigations, surveillance missions and restriction of access to certain areas, or information. Automation can have a range of benefits compared to manual solutions. Not only does it save time and resources, but it also has the potential to intercept details otherwise undetectable by humans, which ultimately lead to robust ways of identifying individuals.

Gait, the pattern of movement during locomotion [1], is an identifier thought to have the potential of being used in an automated capacity. There are many ways of measuring gait. Some use combinations of rare and intricate sensors to measure the pressure of steps and movements [2]. However, visual gait, which focuses on the visual perception of the locomotion, can make do with a simple, *monocular* camera observing the motion. The perceptions can either be measured manually or with automated techniques using a variety of *pose estimations*. Security related applications can potentially use these gait representations as identifiers, to recognize people and individuals. Therefore, this thesis is conducted together with the Swedish Defence Research Agency (FOI), to investigate visual gait by the *machine learning* (ML) technique known as *deep metric learning* (DML), evaluated by measuring comparable *embedding spaces*.

1.1 Motivation

The motivation behind using visual identifiers is that the data needed can be gathered at a distance, without the knowledge of the individual. Further, it has the potential of making use of surveillance cameras and vast networks of *Closed-circuit television* (CCTV). A suspect might leave a visual trace at a crime scene, which later can be matched on a completely different location and time. Multiple ways of recognising individuals exist today, where common approaches involve facial and clothing recognition. However, there are at times when these methods are not reliable. Low resolution cameras can have difficulty capturing faces with enough details. At other times, a person can avoid being recognised, by either covering the face or by changing clothes.

Gait has the potential of acting as a complement in these cases, as the locomotion of a person can be perceivable at lower resolutions or farther distances. Potentially, gait does not require data of a person from a specific angle, and can handle changes of clothing , even

if certain angles and clothing can be more difficult than others. Further, it is of interest to investigate how gait is perceived and identifiable when varying camera angles (or views) are used.

Applying machine learning techniques to automate solutions has shown promise in recent years, where the task can be learned through supplied data examples, instead of being explicitly programmed [3]. The specific method of using deep metric learning, which uses *artificial neural networks* (ANNs), has shown to be successfully used for person verification [4] through colour and texture features, as well as face verification [5]. This motivates the investigation of how gait features can be used to re-identify individuals using the deep metric learning method.

1.2 Aim

This thesis aims to continue the investigation of an automated Gait analysis solution of the human walk, in between different video materials. Focusing on how deep metric learning based methods compares to each other, and non-metric based learning, with the goal of unraveling necessities needed to design a model, able to re-identify individuals, as well as differentiating between individuals.

1.3 Research questions

The thesis strives to answer the following research questions, where the walking gait is extracted from 2D pose estimation, from a limited dataset consisting of monocular video sequences of walking individuals, where the models are either deep-metric-, or non-deep-metric-based:

1. How do the models compare to each other when trained on several camera views, using labels of individuals as the only metadata, for the scalable tasks of re-identifying individuals, as well as identifying new individuals?
2. To what extent can the network models handle different camera views?

1.4 Delimitations

This study focuses only on walking gait analysis. Gait is known for its dynamics, details such as speed differences and carried objects can change the gait of an individual significantly [6]. Therefore, this thesis will focus on gait where the walking speed is constant and no extra objects are carried. Thus, no *in-the-wild* data will be used when training the neural network. Further, it will focus solely on one model-based, *kinetic representation* of the human pose. Lastly, when evaluating the gait, the *keypoints* representing the hands, arms and face of the skeletal pose will be discarded.

2 Theory

The following chapter aims to present the theory of the thesis. As it aims to investigate deep learning based methods for achieving automation in the field of gait analysis, the background focuses on topics surrounding gait, as well as artificial neural networks. It begins with a description of the underlying mathematical concepts, followed by delving into the background on gait analysis, and the neural network based methods used to perform the analysis. Concluding the chapter, the theoretical background of metric learning will be presented.

2.1 Mathematical Background

In this section, general mathematical concepts are introduced that are used throughout the chapter.

2.1.1 Tensors

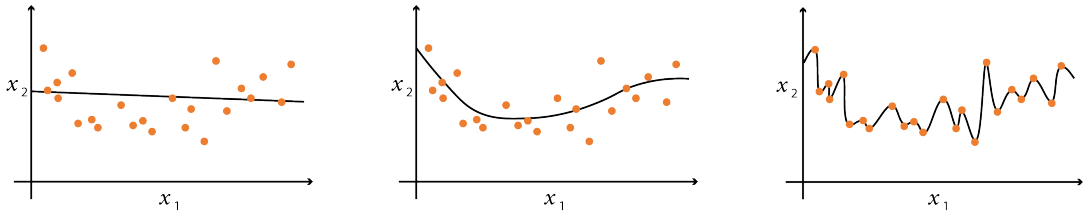
Mathematically, *tensors* are a generalized version of vectors. Similar to vectors, tensors have a magnitude, however, instead of only having one direction, they have $n \in \mathbf{R}_0^+$ directions. The value of n indicate the *order* or *rank* of a tensor. A zero-rank tensor is the same as a scalar value, whilst a vector is a one-rank tensor [7].

For this thesis, the tensor will be used in a simpler way, similar to how it is being used by the leading machine learning libraries [8], [9]. Here, the tensor is described as a N-dimensional array, where $N \in \mathbf{R}_0^+$. Therefore, scalars and vectors have the same ranks as before. Additionally, a matrix would have been of rank two.

2.1.2 Regression Models

Regression models are *statistical models*. Similarly, they aim to fit a set of N-dimensional data points to a mathematical function [10]. These functions can either be linear as seen in Fig. 2.1a, or non-linear, which is shown in Fig. 2.1b and 2.1c.

The goal of regression is to fit the data sufficiently, without *under-* or *overfitting* the model [10], [3]. These cases are seen in Fig. 2.1. The left instance is of a linear regression, which is not sufficient in capturing the distribution well, hence the data is underfitted. The right



(a) A linear Regression which is underfitted

(b) A non-linear regression with a robust fit

(c) A non-linear regression which is overfitted

Figure 2.1: Three versions of regression models of the same data distribution. The left instance underfits the distribution, the right instance overfits the distribution and the middle is able to represent the overall distribution in a more stable manner, a case of a robust fit.

instance is overfitted, as the model fits the individual data points overly well. In the middle is an instance which is able to capture the distribution well, which corresponds to a good fit.

2.2 Gait Analysis

Gait is described as the pattern of movement during locomotion [1]. The study of gait is not just relevant for the human walk, in fact, the *biometric identifier* dates back to 384-322 BCE with the work *On the Gait of Animals* by the Greek philosopher Aristotle. Here, many types of locomotion, in different types of environments are discussed.

The human walking gait is considered *bipedalism*, i.e., locomotion using two limbs. Gait started as an identifier solid enough to distinguish between species. However, it has also been established enough to be able to differentiate on an individual level, within the same species. This has been clear from the year 1839, with the first known case of gait as court evidence [11]. In modern days, gait has been used in forensics as evidence since the year 2000 [1]. Gait analysis has since then been part of several trials, and while some have been partially automated, many are conducted by measuring lengths and angles of limbs by hand [6].

Although gait has been used in court, its admissibility is still under debate. Gait is known for being dynamic. A walk can look drastically different depending on the speed and if something is carried. An individual has the ability to change their walk willingly and it is unclear to what extent identifiable characteristics could be extracted in such a case [6], [2]. Further studies are continuously being conducted, into both manual [12] and automated gait analysis solutions. The recent progression in deep learning shows promise in advancing the use of gait in forensics, as well as increasing the potential for automated solutions [13]. There are two main methods to consider when extracting imagery information from walking gait; the silhouette-based *Gait Energy Image* (GEI) [14], as well as the method based on *human pose estimation* (HPE) [15].

2.3 Gait Energy Image

A method for representing *spatio-temporal* gait, i.e., gait over a sequence of time instead of gait of a single pose, is gait energy image [14]. This method takes the silhouette representations of a gait sequence and concatenates them into one image. The GEI G is calculated for each image coordinate, as

$$G(x,y) = \frac{1}{N}\sum_{t=1}^{N} B_t(x,y), \tag{2.1}$$

where N is the sequence length of the walking cycle and x and y are image coordinates. Fig. 2.2

Figure 2.2: A representation of gait energy image (GEI), the concatenated image of the multitude of silhouette gait representations. Image source: *Gait energy image (GEI)* by Liu and Liu [16], CC BY 4.0.

2.4 Human Pose Estimation

Human pose estimation (HPE) aims to use sensors to extract instances of the human pose. HPE is a constantly progressing area and multiple surveys have been conducted since the early 2000s [17], [18], [19], [20], [21], [22], [23], [24], [25], [26]. The survey by Zheng et al. [27], released in 2021, provide a comprehensive overview of recent advances of HPE using deep learning.

An exact HPE method involves placing markers on a body about to be estimated. Calibrated sensors are able to read the position of the markers in three-dimensional space, leading to the ability to reconstruct a pose. Other sensors, such as *inertial measurement units* (IMUs), *radio frequency devices* and depth sensors, such as *depth cameras* and *Lidar* are also employed. However, the most common type of sensor used in practice is the monocular camera [27].

There are different ways of portraying the pose, both in two- and three-dimensional space. Among the frequent representations are *kinematic*, *planar* and *volumetric*. The volumetric model is a 3D depiction, which often strives to preserve the human shape as closely as possible. Therefore, it is often represented as a *mesh* following the contours of the body. The planar model can be regarded as a simpler version of the volumetric model. Instead of a detailed mesh, the planar representation subsists of primitive shapes or a silhouette, filling out the contours of the body. Lastly, the kinematic representation, which is the most common pose portrayal, is described by a graph structure. Points of interest, such as joints, heels and the nose, represent nodes, whilst the edges represent how they are connected. The kinematic pose is also referred to as *skeletal-based pose* [27]. The pose representations are depicted in Fig. 2.3.

Concerning imagery-based HPE, the process oftentimes differs when estimating poses with a single view, compared to multi-view *monocular* images and videos. For 2D HPE, a single view can be adequate to obtain a sufficient representation. However, as there is only one view observing the pose, *occlusions* is a well-known problem. Occlusions can stem either from other objects or from self-occlusions, ultimately hindering the pose from being fully estimated [15]. For 3D HPE, further problems, in the form of depth ambiguities, arise. To solve these ambiguities, single views can be complemented or exchanged by other sensors, such as depth cameras, Lidar. The issue with occlusions might nonetheless still be present [27].

A solution, which might deal with both the depth ambiguities and the occlusions, is a multi-view setup. Here, a stereoscopic composition has the ability to handle depth issues, while more precise placements of the multiple cameras may need to be considered to avoid possible occlusions. 3D HPE using a multi-view arrangement is a well-posed problem, and when a pose has been estimated in three dimensions, it is possible to project the pose down to a plane to receive a 2D estimation [27]. With the aforementioned problems, going from a 2D to a 3D HPE pose is a greater challenge. However, *2D to 3D lifting* has since 2016 become an

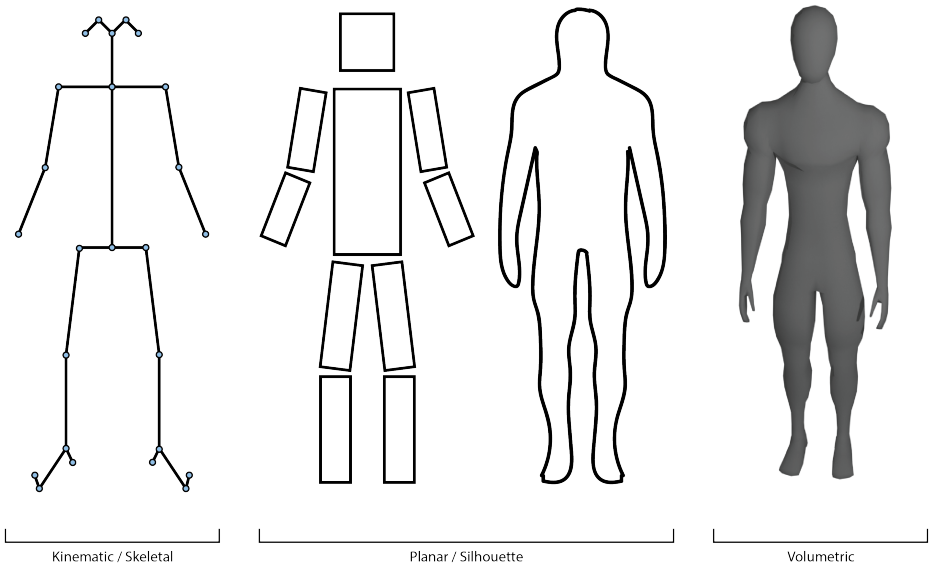


Figure 2.3: Frequent representations of Human Pose Estimation (HPE).

area of interest [28], [29], [30]. This method strives to lift a 2D kinematic pose into 3D, based exclusively on the graph representation of one pose or a sequence of poses.

2.4.1 2D Kinematic Pose Estimation

The imagery-based HPE task consists of several partial tasks, depending on which representation to be used. Focusing on the generally well-posed and common portrayal of 2D kinematic pose estimation, there are still a plethora of ways of obtaining a solution. As deep learning methods have shown to outperform more traditionally based computer vision methods, the following will be focusing on those techniques [27].

Firstly, the methods are categorized as *single-* and *multi-person* HPE. Single-person is the simpler of the two, and is further divided into regression, or body part detecting models. Regression based models are able to learn the task of HPE as a single *end-to-end* framework, with an image as input and kinematic pose as output [27]. The other method takes an image as input, and applies a body part detection algorithm on the input. This technique uses *heatmaps* to predict the likelihood of a part of the image containing a pose keypoint. The most likely keypoints are then stitched together to a pose using a body part associator, which is the output of the framework [31].

For estimation of multiple people in an image, the two main approaches are *top-down* and *bottom-up* [27]. Top-down initiates by using *object detection* to identify a single body followed by cropping person out. Then the same techniques used for single-person estimation can be applied to the cropped individual. This process is repeated for each detectable person in the frame. The bottom-up firstly identifies all of the keypoints in the frame, followed by pairing them to an individual. According to benchmarks made by Zheng et al. [27], of different multi-person estimators developed between 2017 and 2020, top-down outperforms bottom-up on the prevalent *COCO dataset* [32] using the *average precision* (AP) metric. Their results also show that the AP generally gets better by each year.

2.5 Deep Learning

A core concept of *artificial intelligence* (AI) is to be able to automate solutions. Instead of programming such behaviours directly, ML uses techniques to learn the behaviour from already

existing data. A sub-field of ML known as *deep learning* (DL) which has shown to be successful in learning behaviours uses *deep neural networks* (DNNs), a version of artificial neural networks [3].

2.5.1 Neural Networks

The idea of neural networks is influenced by the brain of mammals. ANNs are commonly described as *computational graphs* with layers of nodes, representing *neurons* [3]. An artificial neuron cell is oftentimes called a *perceptron* [33]. Although the words are interchangeable, the word perceptron is mostly used when looking at the innards of a single cell. The neurons are connected to neurons in other layers through edges, known as *weights*. The weights are the artificial equivalence of synapses in the brain.

A neural network has different layers. A commonality of all networks is the input and output layers. The hidden layers and their connections in-between the input and output layers are what perform the actual task. A network that does not form a cyclic connection between its neurons or layers, is known as a *feed-forward neural network*, which this section focuses on. A common cyclic neural network, which uses the concept of recurrence is described in Sec. 2.5.4. The purpose of the network is to have certain neurons in every layer activate, or fire, to produce an output depending on the input. To be able to produce the desired result, the network needs to be learned. This will be explained further in section 2.5.3. Firstly, the background of how information flows through the network will be clarified in what follows.

For a feed-forward neural network, the number of hidden layers defines how deep the neural network is, whereas a network is defined as *shallow* if it only has a few hidden layers. Fig. 2.4 shows a graphical depiction of a simple neural network with two hidden layers. x, h and y denote input, hidden, and output layers respectively. The hidden layers are numbered according to their superscript, whilst all layers have subscripts representing the neuron index for each layer. In this network, all neurons are connected to the neurons in the next layer through the weights. These types of layers are called *fully connected layers*, however, layers do not need not be fully connected. [3].

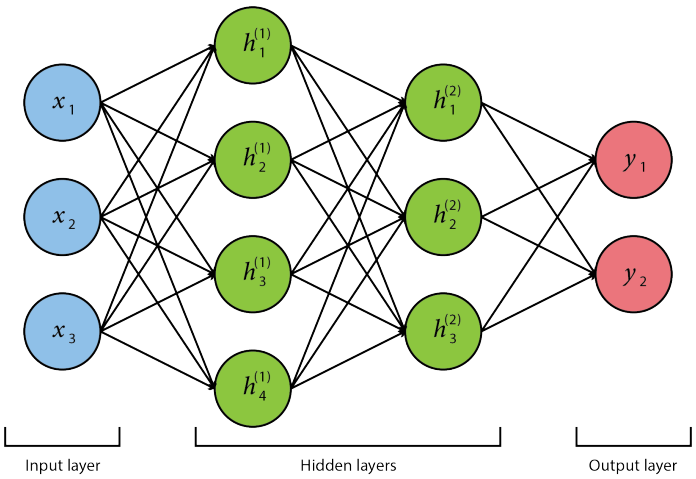


Figure 2.4: A graph representation of a neural network, with an input layer to the left, output layer at the right and two hidden layers in-between. All neurons are connected to each neuron in the following layer through the weights.

Fig. 2.5 highlight how the weights w are notated. The superscript indicates the weight layer, whilst the subscript indicates the starting and ending neuron index of said weight. Note that this figure only aims to indicate how weights are named. The passing of information

through the network is more complex than looking at a single path from the input to the output layer.

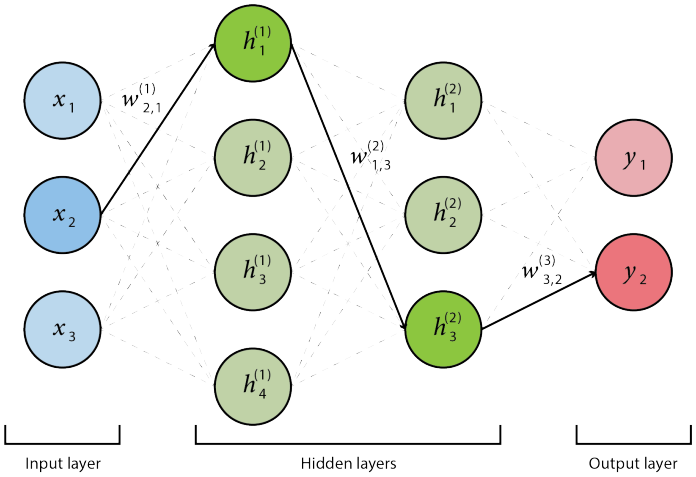


Figure 2.5: A highlight of some of the neurons and their connected weights in a simple neural network structure.

The pass through a perceptron is given by

$$\hat{y} = f(bw_b + \sum_{i=1}^{n} x_i w_i), \tag{2.2}$$

where the n inputs from the previous layer and the *bias* term, $b = 1$, are multiplied by their corresponding weights and summed over. The sum is then passed trough an *activation function* f. The bias and activation function are described in more details in Sec. 2.5.1. The output of the layer, $\hat{y}$, is the result of activation function, as well as the perceptron itself. This pass is shown in Fig. 2.6. Equation 2.2 can be rewritten in vector representation as

$$\hat{y} = f(w_b + \boldsymbol{X}^T \boldsymbol{W}) \text{ where } \boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \text{ and } \boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}, \tag{2.3}$$

which is a representation frequently used and optimised by machine learning libraries [33].

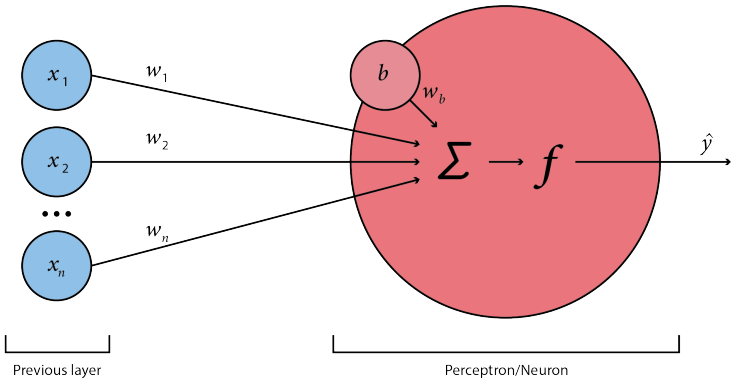


Figure 2.6: A zoomed in perceptron and its inputs from the previous layer.

Neural networks deal with numerical data. A useful way of passing the data along in the network is to use tensors, as described in 2.1.1. In 2.4, the deep learning methods for performing HPE uses images as input. For images with spatial and spectral resolutions larger than one, the data is often three-dimensional, which makes a three-rank tensor an apt option to use [3]. A video sequence of similarly described images would mean that the data could be passed as a four-rank tensor, as the data now contains multiple frames of images over the time domain. A non-cyclic network suitable for handling such sequences is further explained in Sec. 2.5.4.

Looking at how the data flows through the complete network, each numerical value in the input tensor is placed in a single neuron x_i, where i is the index for each neuron in the input tensor. These values are then passed to the first hidden layer $h_j^{(1)}$ by multiplying them with their corresponding weights $w_{i,j}^{(1)}$, where j is the index of each connected neuron in the hidden layer. The output of each neuron in the hidden layer is calculated according to Equations 2.2 or 2.3. This is then repeated for every layer until the output layer is reached [3].

Activation Functions

An activation function decides if a neuron should be activated or not, depending on the inputs of the mentioned neuron. The activation functions for DNNs are preferably non-linear. Fig. 2.7 shows four different activation functions, where the leftmost function is the linear *Identity function*, followed by the non-linear *Logistic*, *Hyperbolic tangent* (tanh), and *Rectified linear unit* (ReLU) functions. They are defined respectively as

$$f(x) = x, \tag{2.4}$$

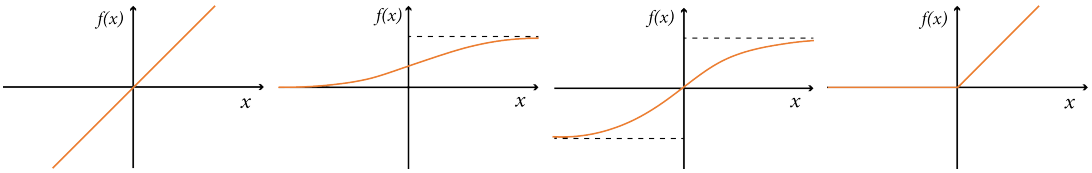
$$f(x) = \frac{1}{1 + \exp(-x)}, \tag{2.5}$$

$$f(x) = \tanh(x), \tag{2.6}$$

$$f(x) = \max(0, x), \tag{2.7}$$

where x is the weighted sums of the inputs and the bias and $f(x)$ is the results of the activation functions. The bias and its weight are used for shifting the output of the activation function horizontally. Both the Logistic and tanh functions are *Sigmoid functions*, that squishes the input to the interval $]0, 1[$ and $]-1, 1[$ respectively. A property of the Sigmoids is that larger negative and positive values end up close to the limits of their defining intervals. A notable difference between the two Sigmoids is the strength of the gradient. For weighted sums close to 0, tanh has a greater change in its gradient than the Logistic function [34]. The ReLU does not squish the values, instead it rectifies all negative inputs to zero, while a positive input equals the output of the identity function. The output of the ReLU will therefore be in the interval $[0, \infty[$ [3].

The reason for using non-linear activation functions is because linear activation functions for DNNs are problematic. If the network uses only linear activations across the layers, the result of the network would be combining several linear functions. The result of a linear combination is a linear function, which might not be able find a suitable fit. This would also mean that the potential of the complex layered structure of the network is wasted, instead, the same result could be reached using a single layer with some arbitrary set of neurons and weights. Another problem with linear activation functions is that its gradient is constant, which causes issues during the learning phase of a network [3].



(a) Identity (b) Logistic (c) Tanh (d) ReLU

Figure 2.7: Common activation functions, where b), c) and d) are used in DNNs.

2.5.2 Datasets and Network Iterations

Data samples are stored in datasets, either as raw samples or together with metadata, such as *class labels* or *ground truth*. There are many ways of splitting the data for deep learning tasks. However, a frequently used divide consists of *training*, *validation* and *test* splits. The training and validation sets are used during the *learning phase* and the test set is used during the *test phase*. A pass through a full set is referred to as an *epoch*, while a sub-set is called a *batch*.

2.5.3 The Learning Phase

At the root of teaching an ANN a specific task in a certain domain is to tweak the weights of the biases and incoming neurons, and to reach a state where the neurons fire in a manner that produces a satisfying result. However, tweaking the weights manually is cumbersome for advanced networks, as the number of weights can lie in the millions. Instead, machine learning is applied to allow the network to learn by itself, based on data. Contrary to the human brain, which oftentimes is able to learn tasks from a few examples, an ANN may need thousands, or even millions of examples to learn well [3]. However, when trained correctly, ANNs are capable of outperforming humans in many tasks, and doing this using unexplored details previously unfathomable by humans. Such areas include lip reading [35], *high-dynamic-range* image reconstruction [36], *natural language processing* [37] and face recognition [38], [39], to name a few.

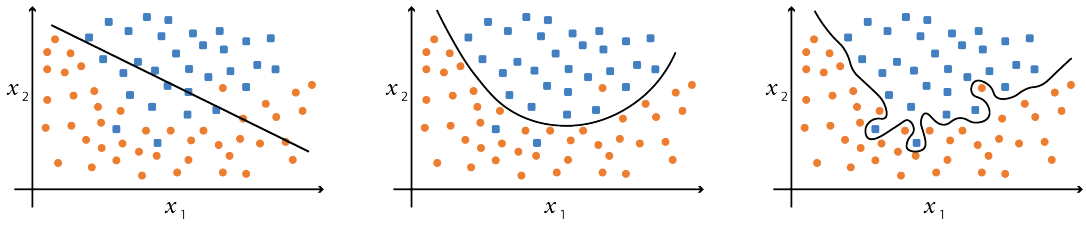
The learning phase is an *optimisation problem*, which can be split up into iterations of *forward* and *backward passes*. A pass forward consists of passing the inputs of a batch through the network to receive their respective outputs. These outputs are then sent into a *loss function*, which calculates the loss. The loss function can be described as the activation function of the output layer, aiming to express the difference between the actual, and wanted outputs. For an optimisation problem, the goal is to minimize the loss. This is done by an *optimisation algorithm*, which steps towards a local minimum of the model by updating the weights during the backward pass. Reaching the global minimum would be optimal for the training data, however, it is not possible to calculate directly [3]. Furthermore, reaching the global minimum might result in overfitting [40].

Classification Loss Functions

Classification loss functions are useful when the goal of a network is to classify the inputs. These use probability functions to predict which class an input belongs to. This is followed by a comparison with the actual class, provided by the annotated data. If the classification is correct, the responsible neurons will be updated to strengthen that network signal. If the classification is incorrect, this results in penalisation of the responsible weights. The output of a network model is not always suitable to indicate class labels (or class probabilities). To allow classification, the output of a model must, either directly or through transformation, be in the form of a *logits vector*. This vector of logit values corresponds to which class is the most probable output, where the highest logit value represents the most probable class. The transformation is often performed by adding a fully connected layer, at the end of a network

model, which linearly transforms the outputs to a logits vector. To more easily compare the probabilities of each class, logits vectors are oftentimes normalised [3].

Sigmoid functions, mentioned in Sec. 2.5.1, are apt choices for normalising the logits vector for *binary classification* probabilities. The logistic function utilises the predicted output in the interval $]0, 1[$, as the probability of belonging to one of the classes. A probability higher than the 0.5 limit favours one class, while a value lower than the limit favours the other class. For tanh, the threshold is 0 [34]. Similar to how regression is used to fit a regressor to a set of data points, as seen in Sec. 2.1.2, classification strives to find a function that separates a number of classes from each other. Fig. 2.8 depict three examples of classifier for binary data.



(a) Underfit (b) Robust fit (c) Overfit

Figure 2.8: Three versions of a binary classification on the same data points. (a) and (c) do not capture the distribution well, whilst (b) is able to capture it, without overfitting to outlying data.

For *multi-class classification*, a well known alternative for the logits vector normalisation is the *Softmax* function, a generalized Logistic function defined as

$$Softmax(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{C} \exp(x_j)}, \tag{2.8}$$

where C is the number of classes, which will also be the number of outputs of the network. $i, j \in [1, ..., C]$. The sum of the C $Softmax$ functions is defined as

$$\sum_{i=1}^{C} Softmax(\boldsymbol{x})_i = 1, \tag{2.9}$$

because of the fact that the probabilities always adds up to 1. The *LogSoftmax* function is the *logarithmic probability* version of the Softmax, formulated as follows

$$LogSoftmax(\boldsymbol{x})_i = log(Softmax(\boldsymbol{x})_i) = log(\frac{\exp(x_i)}{\sum_{j=1}^{C} \exp(x_j)}). \tag{2.10}$$

To finalise the loss function, there needs to be a function that allows the comparison with the actual class. An option is the *Negative Log Likelihood* (NLL) loss, which combines with a logarithmic probability $\log(p)$, such as LogSoftmax, to form *CrossEntropyLoss*, defined as

$$CrossEntropyLoss = -\sum_{i=1}^{C} y_i \log(p_i), \tag{2.11}$$

where y_i is 1 if the classification is correct, and 0 otherwise. In the popular ML library PyTorch, LogSoftmax is common enough to be the standard logarithmic probability function for CrossEntropyLoss [41]. This will be the standard for this thesis as well. In other words, the loss function is standardised to

$$CrossEntropyLoss = -\sum_{i=1}^{C} y_i LogSoftmax(\boldsymbol{x})_i. \tag{2.12}$$

Network Optimisation

With the loss calculated, the remaining step is to update the weights during the backward pass. This is achieved by taking a step towards the local minima of the loss fun. This is done by propagating backward through the network, referred to as *backpropagation*, calculating the negative gradients of the loss L with respect to the weights. With an arbitrary perceptron $a^{(k)}$ and weight $w^{(k)}$ in layer k, the gradient is described with partial derivatives as

$$\frac{\partial L}{\partial w^{(k)}} = \frac{\partial L}{\partial \hat{y}^{(k)}} * \frac{\partial \hat{y}^{(k)}}{\partial S^{(k)}} * \frac{\partial S^{(k)}}{\partial w^{(k)}}, \tag{2.13}$$

where $\hat{y}^{(k)}$ is the output and $S^{(k)}$ the summation of $a^{(k)}$. Similar to how the output of a perceptron depends on the connections from the previous layer during the forward pass, the gradient of a weight depends on the connections in the following layers, during the backward pass. As mentioned in Sec. 2.5.1, having a linear activation function that results in a constant gradient, would mean that the activation part of Eq. 2.13, would not carry any information. The updated weight is defined as

$$\hat{w}^{(k)} = w^{(k)} - \eta \frac{\partial L}{\partial w^{(k)}}, \tag{2.14}$$

where η is the *learning rate* (LR). LR determines how large the step is in the direction of the negative gradient. If the LR is small, it takes longer to update the weights. On the contrary, a large LR might result in an overshoot of local minima [3].

Optimisation Algorithms

There are a plethora of optimisation algorithms for performing the network optimisation. *Gradient descent* (GD), also known as *batch gradient descent*, performs backpropagation by calculating the gradients with a batch size equal to the training set size, before updating the weights. Although computationally heavy, it results in a precise step towards the local minima. *Stochastic gradient descent* (SGD) calculates the gradients with a batch size equal to one training sample, before updating the weights. SGD, although not as computationally heavy as GD, generally has an imprecise, or noisy, descent towards the local minima. *Mini-batch gradient descent* has a batch size somewhere in-between GD and SGD, resulting in lightweight computations compared to GD, as well as precise descent in comparison with SGD [3].

Furthermore, there are concepts of optimisation that resolve potential issues. Two of them are *momentum* and *adaptive learning rates*. Momentum, firstly proposed by Polyak [42] in 1964, counters potentially slow, and imprecise decent associated with small batch sizes by taking the previous gradients into account. Similar to the physical phenomenon of momentum, the result is a smoother change of direction. Adaptive learning rates involve having separate learning rates for the weights, allowing them to increase or decrease depending on how the partial derivatives of the loss changes [3]. This concept was first introduced in 1988 for GD by Jacobs [43]. Adaptive learning rates has in recent years been introduced for mini-batch gradient decent as well. These optimisers include AdaGrad by Duchi et al. [44] from 2011, ADADELTA by Zeiler [45] and RMSProp by Hinton [46], both from 2012, and lastly the Adam optimiser, by Kingma and Ba [47], first published in 2014. Evaluating the optimisers empirically, AdaGrad has shown to be prone to decrease the adaptive learning rate early. However, AdaGrad and the other aforementioned optimisers have demonstrated to be acceptable, depending on familiarity, the data, as well as the network [3].

2.5.4 Recurrent Neural Networks

The neural networks mentioned in Sec. 2.5.1 Neural network models are optimised to solve certain tasks, based on different types of data. The networks used for deep learning based HPE [27], mentioned in section 2.4, predominantly uses *convolutional neural networks* (CNNs)

[48]. Similar to how CNNs perform well for value grids, such as imagery data, *recurrent neural networks* (RNNs), introduced in 1986 by Rumelhart et al. [49], are adapted for sequential data. These include the genetic instruction structures DNA and RNA, text, video, as well as other temporal sequences, such as gait. A fundamental aspect of the RNN network structure is the idea of sharing parameters, such as weights, for every step in a sequence. This provides the ability to generalise a network for sequences of varying lengths, as well as sequences where information appears in a different order. An example of this would be the comparison between two sentences of varying lengths and structure, i.e. "I went to Greece in 2017" and "In 2017, I went with my family to Greece". Here, a wish would be for the network to extract relevant information, such as the year and place of travel. RNNs are designed to handle these cases [3]. Sequences do not always have to be dependent on time. However, for the sake of this thesis, which deals with temporal data, steps in time will be generalised and used interchangeably with sequential steps.

RNNs can be described as a computational graphs, as seen in Sec. 2.5.1, with the added recurrence aspect, as seen in Fig. 2.9. Here, the graph is unfolded to display how the RNN structure reoccurs for every time step t. The box-like structure in the figure is the *hidden cell* of the RNN, which recurrently passes the *hidden state* to the next time step, a cyclic behaviour that differs from feed-forward neural networks. The weight matrices, as described in Eq. 2.3, are the same for every time step t as well. $\boldsymbol{W_{xh}}$ is the weight matrix from the input x to hidden cell h, $\boldsymbol{W_{hh}}$ contains the hidden state passed along the sequence steps and $\boldsymbol{W_{hy}}$ consists of the weights from a hidden state to an output $\hat{y}$.

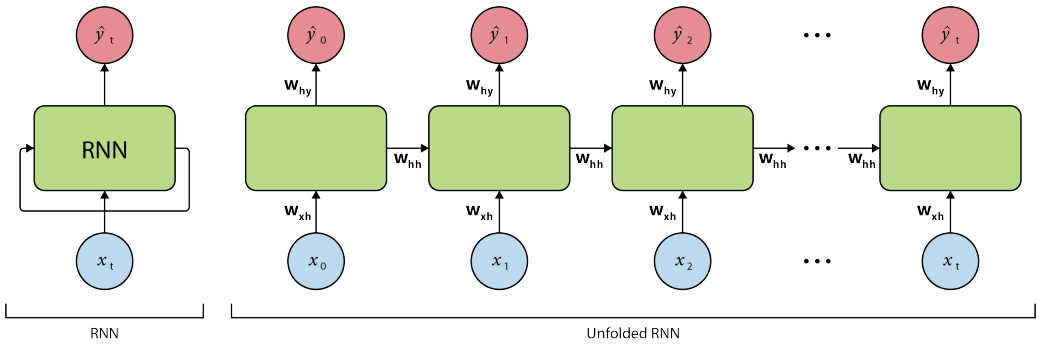


Figure 2.9: A recurrent graph and its unfolded representation. Figure adapted from Amini & Soleimany [33].

The networks described in Sec. 2.5.1 had design patterns to fit their static inputs and outputs, known as *one to one*. RNNs are not limited in that regard and are able to adapt to patterns such as; multiple temporal inputs to one output, known as *many to one*, from one temporal input to multiple outputs, denoted as *one to many*, as well as multiple temporal inputs and outputs, i.e. *many to many* [3]. These patterns are visualised in Fig. 2.10.

The hidden cell structure differs depending on which RNN type is used. Fig. 2.11 show the innards of a hidden cell of a regular RNN, with input x_t, output y_t, hidden state from the previous time step h_{t-1} and outgoing state h_t. Here tanh is used as a non-linear activation function, as described in Sec. 2.5.1. Although tanh is depicted here, the RNN is not limited to this non-linearity, other sigmoid functions or ReLU are viable options [3].

Similar to how the feed-forward networks, depicted in Sec. 2.5.1, had a number of hidden neurons in each hidden layer, the RNN has a number of *hidden units*. The number of hidden units defines how many separate features the network is able to extract from the features in the input layer. These features that the network extracts are oftentimes abstract to humans. Using gait, outlined in Sec. 2.2, as an example, humans might find it intuitive to characterise an individual by the way a certain joint bends when walking. An RNN is not bound to any such criterion and could possibly find features, unfathomable to humans. Furthermore,

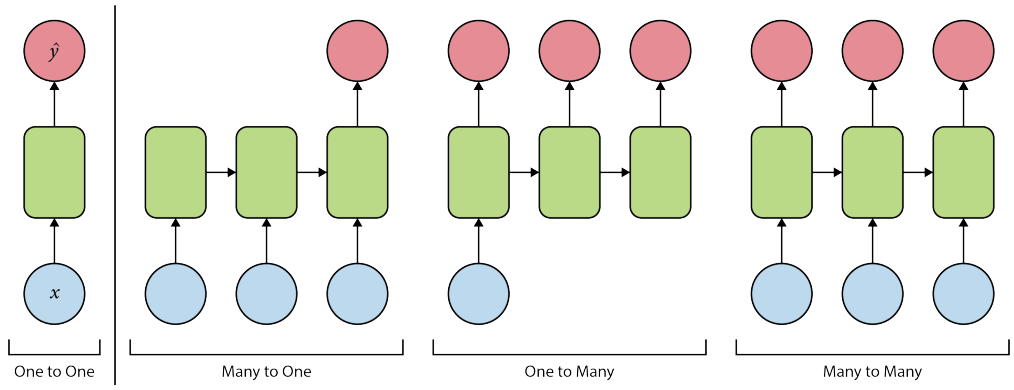


Figure 2.10: Design patterns for neural networks. The leftmost pattern depicts a network with static input and output, whereas the remaining patterns depict specialised cases for recurrent neural networks. Figure adapted from Amini & Soleimany [33].

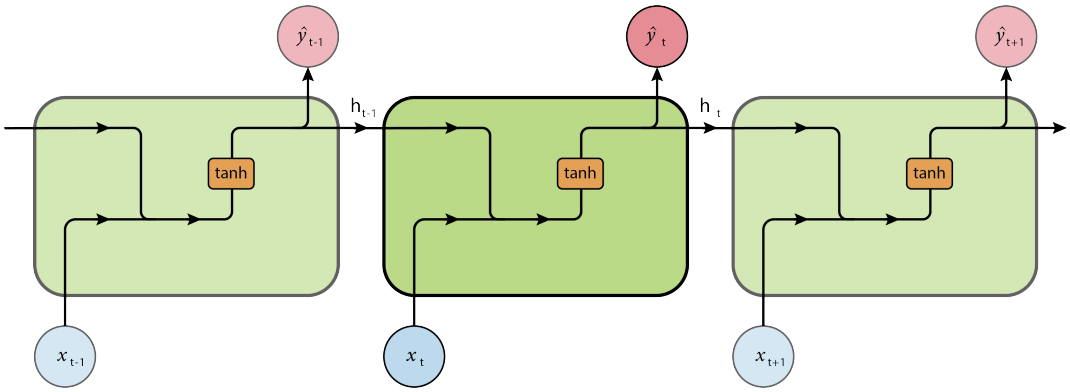


Figure 2.11: The original RNN cell structure. Figure adapted from Amini & Soleimany [33].

RNN cells have the capacity of being stacked, similar to how feed-forward networks can have multiple hidden layers [3]. In such a case, the output $y_t^{(k-1)}$ from RNN cell $k-1$ is passed to the input $x_t^{(k)}$ of RNN cell k, for every time step t. Therefore, backpropagation occurs backward through the stacked cells. However, the loss needs to be propagated backward through the hidden states as well. This is referred to as *backpropagation through time*, and adds further complexity to the backward pass, compared to the feed-forward network [3].

2.5.5 Gated Cells and Gradient Clipping

The regular RNN structure does not always scale well to longer sequences, i.e., it does not remember information from far back in the sequence. As the loss calculated by the loss function has to be propagated through time, longer sequences would result in a longer propagation compared to shorter sequences. The issues that may arise are called the *vanishing gradient problem* and the *exploding gradient problem*. If the error gradient is multiplied by multiple values smaller than 1, the propagation becomes smaller and smaller. Thus, resulting in not carrying the relevant information along the full sequence, i.e., the gradient vanishes. Similarly, multiplying the error gradient with several values larger than 1 results in a larger and larger propagation, i.e., the gradient explodes. To counter these issues, *gated cells* where introduced [3]. The two most common gated RNNs are *long short-term memory* (LSTM), introduced in 1997 by Hochreiter and Schmidhuber [50], as well as *gated recurrent unit* (GRU), firstly introduced in 2014 by Bahdanau et al. [51].

The cell structure of the gated cells is more complex than the regular RNN cell structure. Instead of calculating the weighted sum from the previous hidden state h_{t-1} and the current input x_t, followed by an activation function, the gated cells control the flow of information through the network using gates. These consist of a combination of activation functions, additions and point-wise multiplications, in order to forget irrelevant information from the previous state, store relevant information from the current input, update the current state, and lastly, to produce the current output y_t [33]. LSTMs have three gates and a separate *cell state* c_t, different from the hidden state h_t, whilst the GRUs have two gates and no separate cell state. The cell state of the LSTM allows for backpropagation through time, without exposing the hidden state, diminishing the risk of vanishing or exploding gradients. The GRU still exposes the hidden state during backpropagation, however, its gates have the functionality of withstanding the vanishing and exploding gradients as well [52]. The LSTM and GRU cells are depicted in Fig. 2.12 and Fig. 2.13 respectively. σ and tanh represent the logistic and tanh sigmoid functions, and mathematical expressions are presented in the circles of the cells.

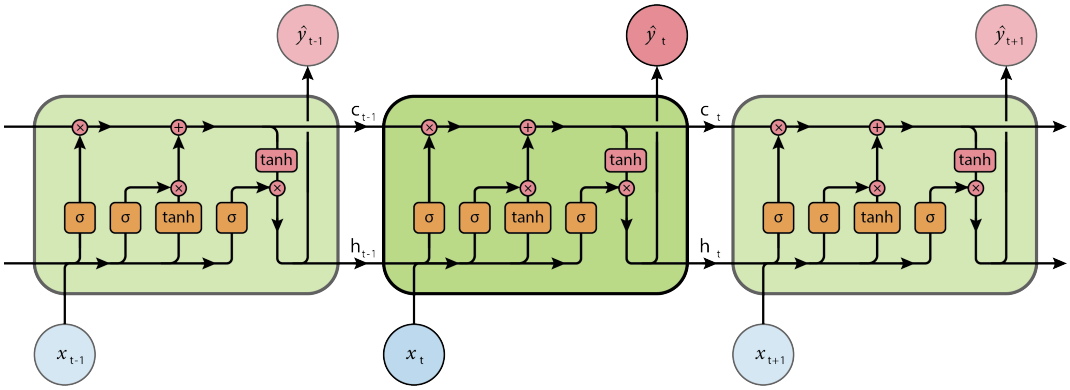


Figure 2.12: The LSTM cell structure. Figure adapted from Amini & Soleimany [33].

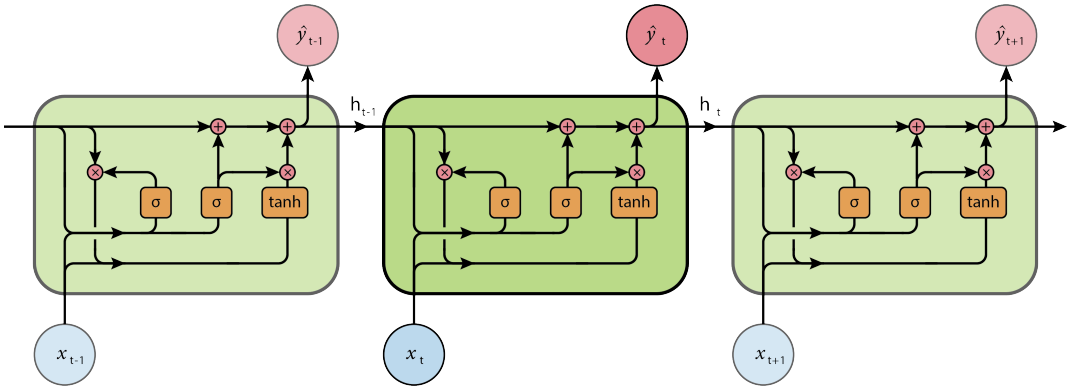


Figure 2.13: The GRU cell structure. Figure inspired by Amini & Soleimany [33].

Another way to diminish the exploding gradient problem is to introduce *gradient clipping* during the last step of the backward pass. The method either clips the gradients g, used for updating the weights by its norm $\|g\|$ as

$$g \longleftarrow \frac{gv}{\|g\|}, \text{ if } \|g\| > v, \tag{2.15}$$

for threshold v [53], or by clipping the gradient of the weights in a batch individually [54].

2.5.6 Bidirectional Recurrent Neural Networks

RNNs determine their outputs depending on past and present data. However, if the sequences exist beforehand, i.e., they are stored and not dependent on live input, the concept of *bidirectionality* can be introduced to learn from future data. This involves using two equal RNNs, one which processes the inputs from t_1 to t_n and the other processing the inputs from t_n to t_1, where n is the length of the sequence. The outputs of the two RNNs are then combined to form a final output at each time step. A bidirectional RNN is depicted in Fig. 2.14. The concept of bidirectionality is available for regular RNN cells, as well as LSTMs and GRUs [3]. During the thesis, these bidirectional variants are denoted as B-RNN, B-LSTM, and B-GRU.

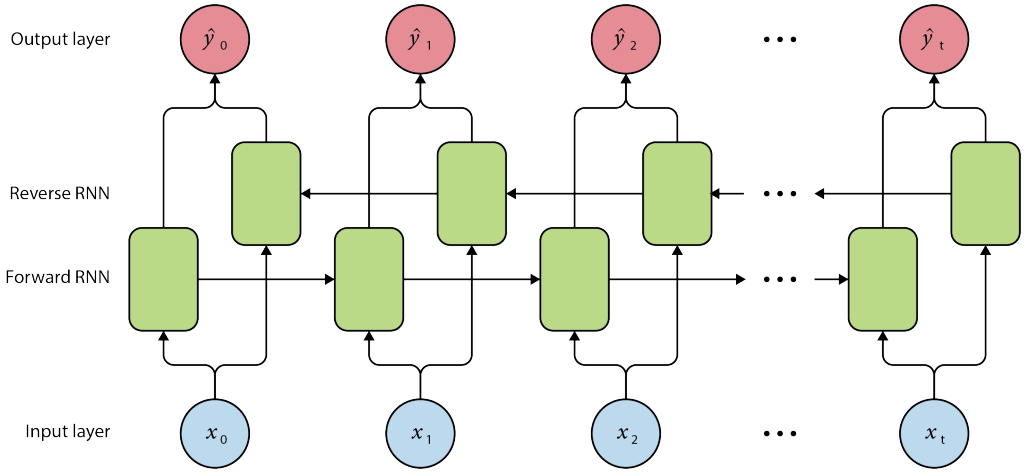


Figure 2.14: A bidirectional RNN structure. Figure inspired by Amini & Soleimany [33].

2.5.7 Transformers

Recently, in 2017, new types of network models known as *transformers* were introduced by Vaswani et al. [55]. The transformer is, similarly to RNNs, designed to deal with sequential data and has shown to scale well to longer sequences, as it does not suffer from the vanishing or exploding gradient problems. Furthermore, they have been shown to outperform RNNs, mainly in the area of *language modeling* [56]. Transformers are able to process a full sequence at a time, compared to RNNs which either processes it iteratively from one direction for a standard RNN, or two directions in the bidirectional case.

2.5.8 Dropout Layers

A problem in neural networks is *co-adaption* of the hidden units of the network. This problem refers to the fact that hidden units rely on each other to be able to detect features of interest. Preferably, they would be able to detect these features independently from one another, as co-adapted units oftentimes result in overfitting [3]. *Dropout*, introduced in 2014 by Srivastava et al. [57], can be introduced as a layer in a neural network to help prevent this co-adaptiveness. Such a layer drops connections at random during training, resulting in less co-adaption of the hidden units of the network. Dropout not only helps the network generalise, it can also prevent in-efficient computations.

2.5.9 Hyperparameters

On the topic of DL and ANNs, there is oftentimes unclarity regarding the use of the words parameter and *hyperparameter*. To clear matters up for this thesis, parameters refer to the

weights of inputs and biases. Hyperparameters are the initialisation settings for the network, such as the number of epochs, batches and network layers, learning rate, loss function, and optimizer. The hyperparameters are not learned when training a network and can differ quite significantly from case to case. Different hyperparameters can be evaluated together to find a suitable combination. A frequently used method for finding apt settings is *grid search*, which simply tests every combination between a given set of hyperparameters [3].

2.6 Deep Metric Learning

A core concept when working with samples of data is to be able to use mathematical models to tell information about the data. Regression and classification, introduced in sections 2.1.2 and 2.5.3 respectively, do this by using a regressor to represent a distribution of data, as well as using a classifier to determine which known distribution a data sample belongs to. Another core concept is to be able to measure how similar data samples are to each other. This is closely related to classification, as certain classifiers, such as *open-set* classification [58], use similarities to predict which class a sample belongs to. However, generalising by the similarities, other information of interest can be extracted, such as retrieving the most similar data samples to an input. This is also used for person re-identification and face verification, where individuals are seldom assigned to classes. Further, similarities are frequently used in *unsupervised* and *semi-unsupervised learning* to limit the need for labeled data [59].

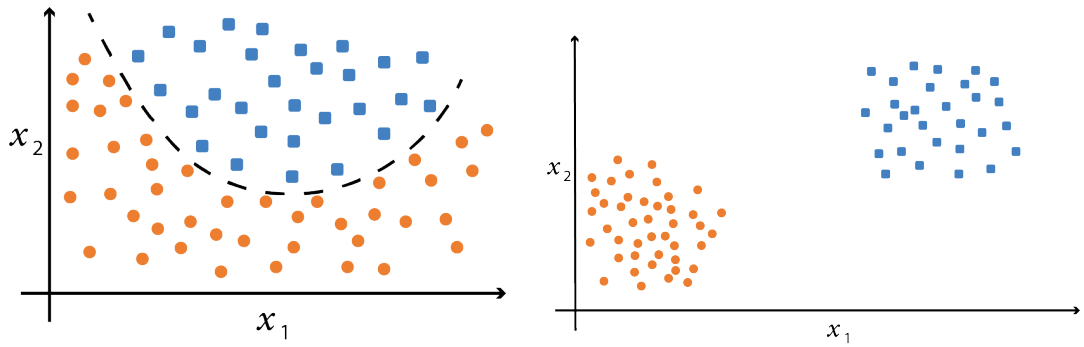
Metric learning uses measurable distances to be able to learn similarities. Furthermore, the concept of *deep metric learning* (DML) involves using deep neural networks to learn to distinguish between similar and dissimilar data samples by distance, in a measurable *feature space*, known as an embedding space. The feature spaces can either be the direct output of the neural networks, or be the result of a *dimensionality reduced* network output [60]. Among the reduction techniques is *principle component analysis* (PCA) [61].

For a network to learn to produce an embedding space, *embedding losses* are used. In cases of *supervised learning*, where metadata such as labels, are available, the concept of these losses involves comparing an *anchor* sample to *positive* samples with the same label, as well as to *negative* samples with differing labels. When learning a network to produce such a DML embedding space, penalisation occurs if the anchor is far from the positive sample or close to the negative sample [60]. Comparing embedding losses to the classification loss introduced in Sec. 2.5.3, the classification loss does not explicitly favour distances in an embedding space. Instead, it is enough for the classifier model to find separability. This notion is characterised in Fig. 2.15. However, even if classification loss does not favour distance, it is still possible to produce such an embedding space implicitly. Once an embedding space is learned, clustering, as well as *nearest neighbour* techniques can be used to evaluate the embeddings.

DML has gained traction during recent years, with papers claiming large performance boosts for new losses, compared to traditional embedding losses. However, a paper by Musgrave et al. [60] found several inconsistencies in their methodologies. After reproducing the experiments of the papers with corrected methodologies, it was found that their techniques resulted in marginal performance boosts, at most.

2.6.1 Embedding Losses

Here, losses based on the embeddings are described. A commonality of the losses is that they either form anchor-positive and anchor-negative pairs, or anchor-positive-negative triplets from a batch of data samples. An important aspect during the learning phase of the neural network, is that the weights are shared for the samples in a pair or triplet network, so that the weights remain the same for a pair or triplet [62].



(a) The result of a classification loss, which finds a separability between the two classes, represented by the dotted line.

(b) Depiction of the results of an embedding loss, which favours similar samples to be close to each other, while far from dissimilar samples.

Figure 2.15: Two-dimensional embedding space results of different loss functions, using toy data.

Contrastive Loss

The *contrastive loss* uses a *siamese* [63] network structure for sharing the weights between pairs of samples. The loss is defined as

$$L_c = \begin{cases} d(s_a, s_p), & \text{if anchor-positive pair} \\ \max(0, m - d(s_a, s_n)), & \text{otherwise} \end{cases}, \tag{2.16}$$

where m is a margin, s represent the input to the embedding loss function, the subscripts a, p and n indicate either an anchor, positive or negative input. Finally, d is a distance function, such as the Euclidean distance, also known as the $L2$ distance [64]. The siamese network structure is depicted in the middle part of Fig. 2.16.

Triplet Loss

The *triplet loss* uses a triplet network for sharing the weights between triplets of samples. The loss function is describes as

$$L_t = \max(0, m + d(s_a, s_p) - d(s_a, s_n)), \tag{2.17}$$

with the same notations as Eq. 2.16 [65]. The triplet network structure is represented in the right part of Fig. 2.16, and is comparable to the single and siamese networks, located in the left and middle parts of the figure respectively.

2.7 Evaluation Background

This section presents the theoretical background of the evaluation scores and visualisation techniques considered during the thesis. For this thesis, a *query* is defined as the data sample to be evaluated. When using embedding losses, this query is the same as the anchor.

2.7.1 Classification Scores

Working with classification task, the fundamental task is the *binary classification*, which deals with two classes. A query can successfully be classified as belonging to a class and not belonging to a class. These cases are referred to as *true positive* (TP) and *true negative* (TN), respectively. The other two possible cases are wrongfully classifying our query to the class,

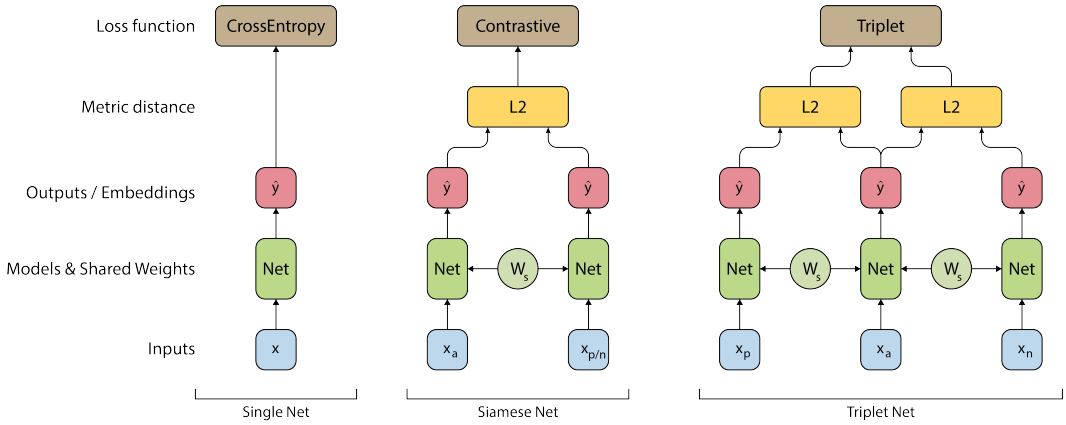


Figure 2.16: Schematics of the single, siamese and triplet networks.

as well as wrongfully classifying it to not belong to the class, accordingly known as *false positive* (FP) and *false negative* (FN). These binary predictions are used to calculate binary *accuracy*, *precision*, *recall* and *F-* scores [3], defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.18}$$

$$Precision = \frac{TP}{TP + FP}, \tag{2.19}$$

$$Recall = \frac{TP}{TP + FN}, \tag{2.20}$$

$$F\text{-}score = 2 * \frac{Precision * Recall}{Precision + Recall}. \tag{2.21}$$

Scaling the task to multi-class classification the same metrics; TP, TN, FP and FN are used to calculate the classification scores. However, instead of measuring the negative as one class, it is seen as all the classes the query does not belong to, i.e., for each positive class, there are $N - 1$ negative classes, where N is the number of classes. The precision, recall, and F- scores can be calculated separately for each class. The total TP, FP, and FN can also be summed together for all classes to calculate the *micro scores*. However, this results in precision, recall and F-score being equal to each other. This is known as the *multi-class accuracy*.

2.7.2 Embedding Scores

Firstly, three accuracy scores used in the thesis will be introduced, followed by a score that measures how well the distribution of data in an embedding space is clustered.

Precision at k

The *k-nearest neighbour* classifier is used to examine the k nearest data samples to a query sample in an embedding space. The query is then classified as the class which occurs the most times, in the k closest cases. The *Precision at k* (P@k) is another name for this metric, which is used in this thesis [66]. This is a commonly used metric for evaluation of DML, however, it is quite unstable and does not tell much about the distribution of the samples in the embedding space [60].

R-Precision

R-Precision (RP) uses the k-nearest neighbour algorithm to tell more about how a query and the samples with the same label as the query are distributed. Firstly, all R samples of the same label as the query is counted using existing metadata. Then, the R nearest neighbours are located, and out of those R neighbours, r is the number of neighbours with the same label as the query. RP is then defined as [66]

$$RP = \frac{r}{R}. \tag{2.22}$$

Mean Average Precision at R

Mean average precision at R (MAP@R) is an extension of the R-precision metric introduced by Musgrave et al. [60], defined as follows

$$MAP@R = \frac{1}{R}\sum_{k=1}^{R} P(k), \tag{2.23}$$

where R is defined equally as for Eq. 2.22, and $P(k)$ is defined as

$$P(k) = \begin{cases} \text{precision at } k, & \text{if correct at } k \\ 0, & \text{otherwise} \end{cases}. \tag{2.24}$$

The MAP@R, compared to RP, is able to capture the placement of the correct retrievals, i.e., the score will be better if many samples of the same label occur at lower indices of k, compared to higher ones [60].

Silhouette Coefficient

The *Silhouette Coefficient* (SC) was introduced in 1987 by Rousseeuw [67], and is defined as

$$SC = \frac{b-a}{\max(a,b)}, \tag{2.25}$$

where a and b are the mean Euclidean distances between a query and samples of the same class, and between a query and the next nearest cluster respectively. The result of SC is bound to be in the interval $]-1, 1[$. The score reaches closer to 1 if the clusters are dense and separated from each other, a score around 0 indicates overlapping clusters, and finally, scores closer to -1 correspond to badly formed clusters.

2.7.3 Embedding Visualisation

PCA mentioned in Sec. 2.6, can with its ability to reduce dimensions be used as a visualisation tool. This is due to the fact that feature spaces of high dimensionality oftentimes can be reduced, while still maintaining much of the information of the initial feature space. PCA works by finding linear combinations from the feature space, that represent the highest *variance* of the initial data [68].

3 Method

This chapter details the implementation and assessment protocols used in the thesis. First, a description of the necessary resources is provided. Subsequently, it is explained how HPE was extracted and pre-processed from a dataset in order to represent gait. Lastly, the used deep learning strategies are described, followed by the explanation of how they were evaluated for the automation of gait analysis.

3.1 Computer Hardware and Software

For the conduction of this study, a computer running the Linux distribution, Ubuntu 20.04 operating system (OS), was used. The notable hardware components consisted mainly of the GPU, an Nvidia GeForce RTX 2070 Super with 8 gigabytes of memory, an Intel i7-7800X 3.50GHz CPU with 12 cores, and 16 gigabytes of RAM.

The main programming language used during the thesis was Python, version 3.9 [69]. It was chosen because of its fast prototyping capabilities, as well as support for advantageous frameworks, libraries and toolkits. These include the machine learning library PyTorch 1.7.1 [70], the computer vision library OpenCV 4.5 [71], the HPE extraction library OpenPose[1] 1.7.0 [72] and lastly the visualisation toolkit TensorBoard [73]. Support for these libraries exists for the C++ language [74] as well. Although C++ is compiled and therefore, generally faster computationally than the interpreted Python language, the speed of C++ was determined to be outweighed by the ease of development using Python. The last notable libraries, specific to Python, were the PyTorch Metric Learning (PML) library[2], version 0.9.98 [60] and the machine learning library Scikit-learn, version 0.24 [75].

3.2 The Dataset

The data used had been gathered by FOI for the purpose of performing walking gait analysis and was compiled into a dataset. This dataset will be referred to as the *FOI Gait Dataset*[3] in this thesis. It consists of video material of individuals walking on a treadmill, at a fixed walking

[1]The OpenPose repository: https://github.com/CMU-Perceptual-Computing-Lab/openpose
[2]The PML repository: https://github.com/KevinMusgrave/pytorch-metric-learning
[3]The FOI Gait Dataset is not publicly available.

speed and flat angle. Fig. 3.1 is a display of the experimental setup. Describing the setup from a *top-down perspective*, the treadmill was placed at the origin of a semicircle, with five cameras $a)$ to $e)$ placed at the arc of the semicircle. The angles they cover go from directly behind, around the right side, to directly in front of the treadmill, with cameras at approximately 45° apart. Four of the cameras were placed close to pelvis height, while the remaining camera, the one at 45° from the front angle, was raised higher, to simulate the height of the surveillance camera. The cameras were aimed towards the space above the treadmill, to cover the full body of the individuals when they were walking. Fig. 3.2 and Fig. 3.3 contain still frames from the walk of Individual 1, from session 1 and session 2 respectively. Fig. 3.4 display the only session for Individual 3. Cameras a, b, c and e are placed at pelvis height, while camera d is raised higher. Appendix A consists of still frames from every individual, session, and view.

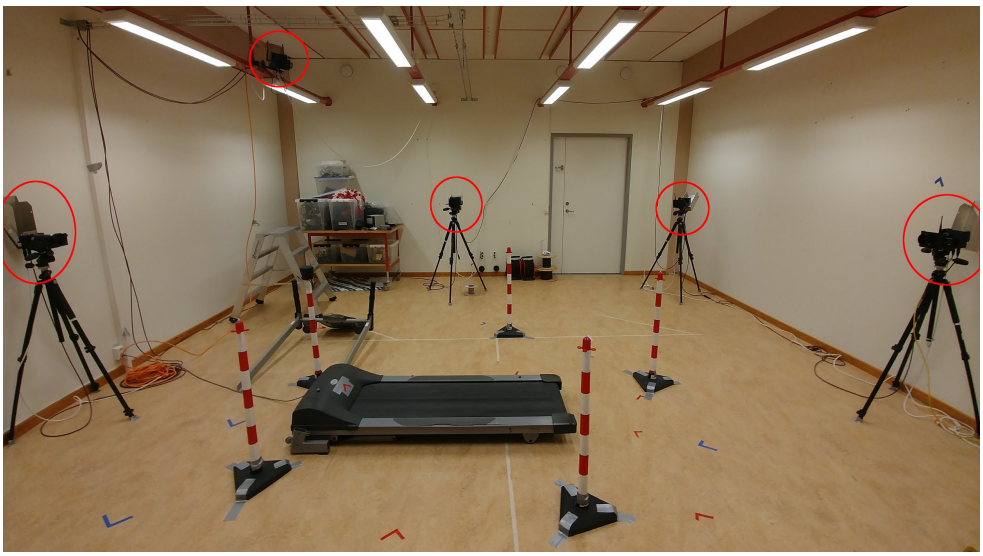
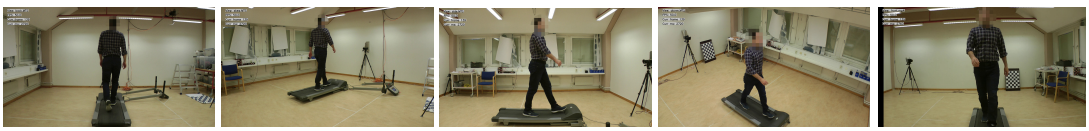


Figure 3.1: The setup for collecting the FOI Gait Dataset. The cameras are highlighted in red. Starting from the left, they are named e, d, c, b and a.

The dataset contains 10 individuals walking on the treadmill for 10 to 12 minutes per session, covered by all five camera views simultaneously. Two of the individuals recorded two sessions each, while the remaining people walked for one session each. The individuals that were recorded for two sessions, did this on different days with altered clothing. The



(a) Back (b) Skewed back (c) Side (d) Skewed front (e) Front

Figure 3.2: Still frames from session 1 of Individual 1.

(a) Back (b) Skewed back (c) Side (d) Skewed front (e) Front

Figure 3.3: Still frames from session 2 of Individual 1.

(a) Back (b) Skewed back (c) Side (d) Skewed front (e) Front

Figure 3.4: Still frames from session 1 of Individual 3.

cameras were set to record in colour at 50 frames per second (fps) at a resolution of 1920 x 1080 pixels. The recordings could be started and ended simultaneously through a controller, with a latency of only a few frames in-between the views. A synchronisation signal was placed in the audio track of the recordings, allowing the exact alignment of the frames. This is useful for multi-view HPE, mentioned in Sec. 2.4. However, the synchronisation signal was not utilised during this thesis, as the focus is single view HPE, described further in Sec. 3.3. The different sessions were not cut to be of the same lengths. Still, the views from each session were trimmed to be of the same length, as described in Table 3.1.

Table 3.1: A description of collected data of the FOI Gait Dataset.

Individuals	Sessions	Number of Views	Support (Frames/View)	Used for
Individual 1	Session 1	all 5	32 450	Training / Evaluation
	Session 2	all 5	32 282	Evaluation
Individual 2	Session 1	all 5	31 298	Training / Evaluation
	Session 2	all 5	35 642	Evaluation
Individual 3	Session 1	all 5	31 994	Training / Evaluation
Individual 4	Session 1	all 5	32 066	Training / Evaluation
Individual 5	Session 1	all 5	32 090	Training / Evaluation
Individual 6	Session 1	all 5	31 970	Training / Evaluation
Individual 7	Session 1	all 5	31 658	Training / Evaluation
Individual 8	Session 1	all 5	32 738	Training / Evaluation
Individual 9	Session 1	all 5	32 162	Training / Evaluation
Individual 10	Session 1	all 5	32 354	Training / Evaluation

3.3 Pose Extraction

To perform the gait analysis with the use of deep metric learning, the two options for gait representation were gait energy image and human pose estimation. Although GEI efficiently represents a full gait cycle as a single image, it might lose some temporal information once concatenated. By using HPE, the temporal information is still held in the individual frames of the sequence. A well designed HPE method can also, to an extent, bypass the inability of capturing a change to baggy clothes, which a silhouette-based method would suffer from. Furthermore, as a previous master thesis had been conducted at FOI in 2020 by Persson [76], using HPE. It was of interest to continue the investigations conducted, therefore, HPE was chosen. The specific representation used was the bottom-up, multi-person and kinematic HPE library OpenPose [72]. The reason for choosing OpenPose was based on several factors. It is a library able to run close to real time, for a multi-person case. Further, OpenPose includes an extension able to estimate feet, hand and face keypoints. Where feet are of particular interest, as it provides relevant information for the purpose of walking gait analysis. Noteworthy, however, is the fact that OpenPose is relatively old and rated among the worst pose estimators during the benchmarks on the COCO dataset by Zheng et al. [27]. Another aspect considered when initially choosing OpenPose, was its active maintainers and community. However, active support for the library seems to have dwindled during the time of the thesis.

OpenPose was chosen nonetheless, together with the Body-25 kinematic pose representation, which is seen in the leftmost part of Fig. 3.5. OpenCV was used together with OpenPose

for the extraction. Although OpenPose is considered one of the faster pose extractors [27], it would be a bottleneck to extract the poses from the video every time a walking sequence was needed. Therefore, the library was used to extract the kinematic pose for all the frames in the dataset, and then stored them in a lookup table, together with look up keys, such as individual, session and view indices. Poses could then be looked up in sequences of varying lengths by using the keys.

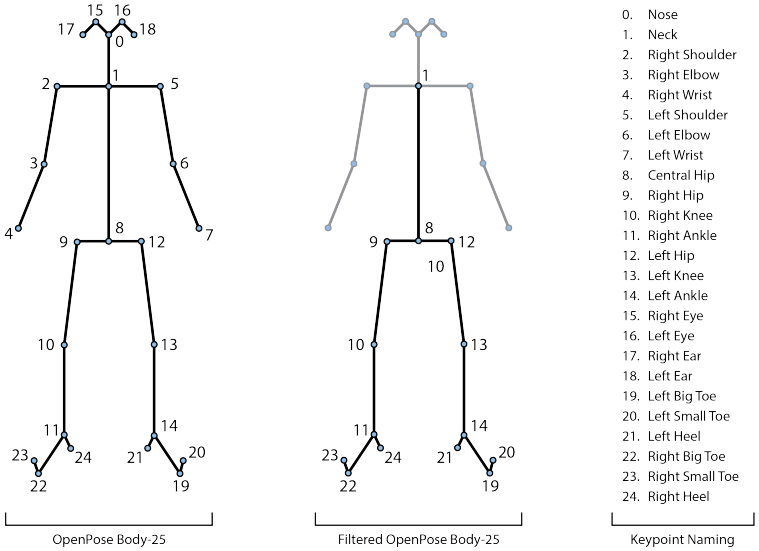
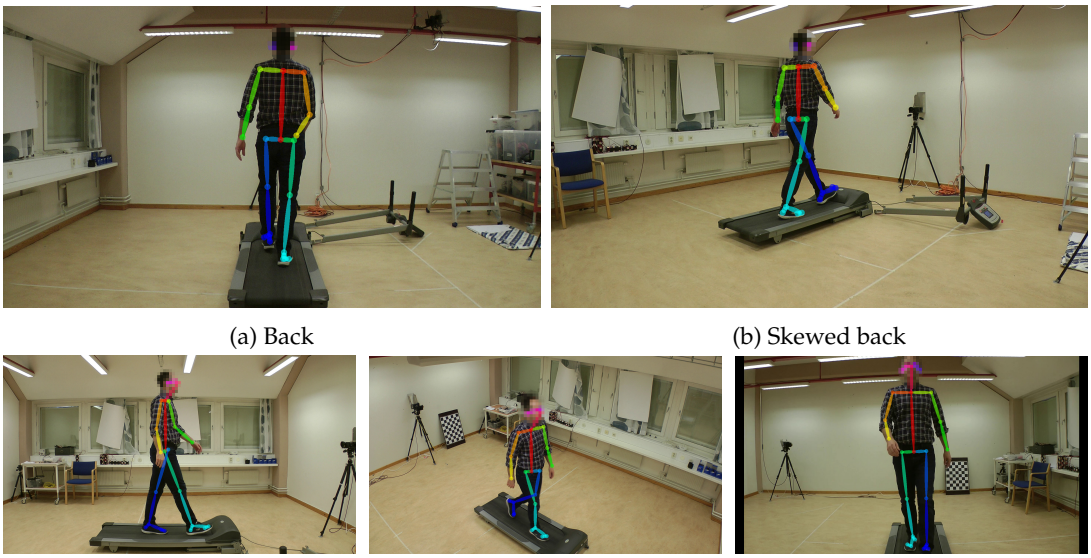


Figure 3.5: The BODY-25 kinematic pose representation of OpenPose with the names of the keypoints. The filtered version was used during the thesis.

Notable settings chosen for the extraction were the *net resolution* and the *render threshold*, set to 656 x 368 and 0.5 respectively. The net resolution was chosen to be of the same aspect ratio as the image frames of the dataset, and as close to the original frame resolution as possible. According to the maintainers of OpenPose, this increases the ability of OpenPose to extract correct keypoints. A higher resolution would be preferable, however, it was limited by the memory usage. The render threshold is in the range $[0, 1]$ and limits the confidence of the keypoint extractions. The threshold is a measure of how sure the library needs to be. If the threshold is low, OpenPose makes guesses for keypoints of occluded joints, as well as for other unsure areas. For higher thresholds, OpenPose needs to be confident that a possible keypoint is actually a keypoint, before extracting it [72]. For the 2D pose representation, OpenPose outputs horizontal and vertical screen coordinates (x, y) measured from the top left of the image frame, for every keypoint. If the library is not able to detect the placement of a keypoint, either directly or through guessing, the coordinates of said keypoint are set to (0, 0). The skeletal pose extracted with these settings is seen in Fig. 3.6.

3.4 Pose Processing

Prior to feeding sequences of poses to the network, they were processed. OpenPose had a tendency to perceive false poses from other objects in certain frames. These were filtered out using a novel method of tracking one of the keypoints in-between the previous and the current frame. The central hip joint, seen in Fig. 3.5, was chosen, as it was in the central part of the pose, as well as never undetected by OpenPose. With k_{t-1} representing the correct central hip keypoint from the previous frame and $k_t^{(i)}$, $i \in [1, ..., n]$ are all the detected poses in the current frame, where n is the number of detected poses. The Euclidean distance was measured between the coordinates of k_{t-1} and all $k_t^{(i)}$ keypoints, and the pose in the current



(a) Back (b) Skewed back

(c) Side (d) Skewed front (e) Front

Figure 3.6: Still frames of the extracted HPE with OpenPose, from session 1 of Individual 1.

frame that resulted in the shortest distance was kept, and the remaining were discarded. As the falsely detected poses from inanimate objects were rare it was sufficient enough to not perform an initial measure for the keypoint in the first frame k_0. A more robust method would have been needed if this occurred often.

With the false detections removed, the following processes were applied to the correctly detected poses for each frame

1. Change the origin of the pose.
2. Normalise the pose.
3. Filter out certain keypoints.
4. Reshape the keypoints and their coordinates.

These were then kept constant throughout the thesis. The first process changed the origin coordinates of the poses from the upper left corner of the screen to one of the keypoints. The reason for this was to make the coordinates invariant of where in the frame they walked. One person might have walked further back on the treadmill than the rest, and if no change of origin was performed, such a distance could skew the results from the neural network. The central hip joint was chosen as the origin, as it is a relatively steady part of the body.

The normalisation processing followed the change of origin. Every pose was normalised to the same height, i.e., normalised in the y direction, as well as scaled accordingly in the x direction. A consequence of this is that the network is not able to use height to differentiate between the individuals in the dataset. However, proportions between limbs are still kept intact, which may be important for recognising the gait of an individual. The main reason for applying a normalisation, however, is to handle changes in distance and camera resolution. A goal of the network is to be able to recognise a person, close to the camera, as well as far away. Further, cameras can be set to capture at varying resolutions, and a normalisation diminishes this variation.

The third step involved filtering out certain keypoints from the BODY-25 representation, and the result of this filtration is depicted in Fig. 3.5. The reason behind this was to remove the sporadic movements and rotations of the arms and head and focusing on investigating

the ability to of recognising gait based on the central and lower parts of the human body. After the filtering step, 14 of the 25 keypoints remained.

The fourth and final process was to reshape the features in the two-rank tensor of the keypoints and their x and y coordinates to a one-rank tensor, as follows

$$\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_{14} \\ y_{14} \end{bmatrix} \longleftarrow \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{14} & y_{14} \end{bmatrix}, \tag{3.1}$$

i.e., it was reshaped from a tensor of shape 14 x 2, to shape 28 x 1. These one-rank tensors from one frame then formed sequences together with features from subsequent frames, altogether, representing a single sample.

3.5 The Network Learning and Testing Implementation

The extracted and processed poses could be passed in sequences of varying lengths to a network model as input. The networks were implemented in Python with PyTorch and PML. The first step involved determining the sequence length, which was used for all samples for the network. Following that, the dataset was split into training, validation and testing at random. If a short sequence length was chosen, this would result in more samples being allocated into the three splits. However, the longer the sequence length, the fewer samples would be available. The size of each split was dependant on the evaluation task, and is further described under each section outlining the tasks.

The learning phase consisted of passing the training and validation data to the network during several epochs. For each epoch, the data was split into multiple mini-batches with newly shuffled samples. During training, a mini-batch was passed forward and backward, followed by an update of the weights before the next mini-batch was iterated over. When all the batches had been passed through the network, the validation mini-batches were passed through the evaluation step, resulting in evaluation scores. These scores were then used to monitor the performance of the network. If the network became stagnant, i.e., if the validation scores did not increase from epoch to epoch, firstly, the learning rate was stepped by multiplying it by 0.1. After a number of learning rate steps without an increase in validation performance, the learning phase would break early to prevent overfitting. Every time the validation performance was increased, the state of the model would be saved. The best performing model state was then used to evaluate the network using the fully unseen test split, which resulted in the final score of that network run.

An upper limit for the number of epochs was set to 250, however, this limit was never reached, as the learning phase was manually designed to break early. Different network configurations learned slower and validation scores could fluctuate, therefore, grid search was used to decide after how many un-increasing validations the network would either step the learning rate, or break early. The Adam optimizer was used for all investigations during the thesis. The learning rate was initialised to 5e-4 for all runs and stepped consecutively to 5e-6 and 5e-7, followed lastly, by breaking. TensorBoard was used to continuously monitor the performance of the network, by plotting the loss, as well as using its projector tool to visualise the embedding space using PCA. Finally, to add further reliability to the numerical results presented in this thesis, each evaluation ran ten times. The results shown are the mean of those runs, together with the 95% confidence interval deviation.

3.6 Network Design

When choosing a network design, an important aspect was the ability to deal with sequential data. The two most promising models were RNNs and transformers. However, as transformers are relatively new compared to RNNs, their presence in established literature is sparse. This led to not realising the promising potential of transformers until late in the thesis process. Therefore, RNNs were chosen as the model. According to the theory, the specific RNN models GRU and LSTM models and their bidirectional variants showed promise for dealing with longer sequences. The previous thesis by Persson [76] used a model with four stacked B-LSTM cells with 1024 hidden units, and dropout added after the final layer. Furthermore, a sequence length of 350 was used. However, it is unclear if other network configurations and sequence lengths were evaluated. Therefore, for this thesis, grid search was used to first evaluate the performance of different network configurations, together with sequence lengths. It was found that a model with two stacked RNN cells with 512 hidden units, with a dropout layer in-between the RNN cells was able to capture the gait from the data when evaluated as a classification task. For this classification task, a final layer applying a linear transformation from the 512 dimensional feature space, the result of the 512 hidden units, to a logits vector space, to be able to use a classification loss. The dimension chosen for the logits space was 10. The dropout layer was set to drop half of the connections at random.

The performance of different RNNs and sequence lengths were tested by evaluating the RNN, GRU, and LSTM networks and their bidirectional counterparts against six different sequence lengths, ranging from 25 frames to 800 frames, i.e., half a second to 16 seconds at 50 fps. The results are presented in Appendix B. The first sessions of all individuals were used to conduct this evaluation, using the training, validation and testing split of 70%/15%/15%, with a batch size of 32. The multi-class accuracy was used to measure the performance. The network type that performed best was the GRU and B-GRU. As the bidirectional version is more complex, yet only resulted in a marginal increase in accuracy at best, for certain sequence lengths, the network used for the continuation of the study was chosen to be the unidirectional GRU, together with the sequence length of 100 frames.

3.7 Deep Metric Learning Evaluation

The embedding losses and scores used for DML were implemented using the PML library. The library contains the implementations of the Contrastive and Triplet losses, as well as the metric scores P@k, RP and MAP@R. For P@k, k was set to 1, meaning that this score classifies the query with the same label as the closest sample, using the Euclidean distance. For the SC score, the implementation in Scikit-learn was used. In this thesis two different embedding spaces were used. As mentioned in Sec. 2.6, dimension can be reduced using methods such as PCA. However, this was not used during the thesis. Instead, direct outputs of network models were used, *10-dim* and *512-dim*. The 10-dim is the same model that was employed when evaluating the network designed, described in Sec. 3.6. The 512-dim is the same model, however, without the final linear transformation layer, i.e., the direct output from the 512 hidden units of the GRU. Fig. 3.7 show the two network models.

The batch size was set to 32 for these evaluations. The Contrastive and Triplet losses were evaluated against the CrossEntropy loss in the 10 dimensional embedding space. The CrossEntropy loss is not able to be evaluated for the 512 dimensional embedding space, as it would need a transformation to logits space before being able to produce a reliable predictive result for the classes. The 512 dimensional embedding space was mainly used to evaluate how a reduced embedding space, such as the 10 dimensional one, could compare to the original high dimensional output feature space. Both the Contrastive and Triplet losses have the margin hyperparameter to set. A grid search was conducted, and it was found that a margin of 0.1 performed the best out of the evaluated margins. Therefore, the margin was set to this value for the deep metric learning evaluation.

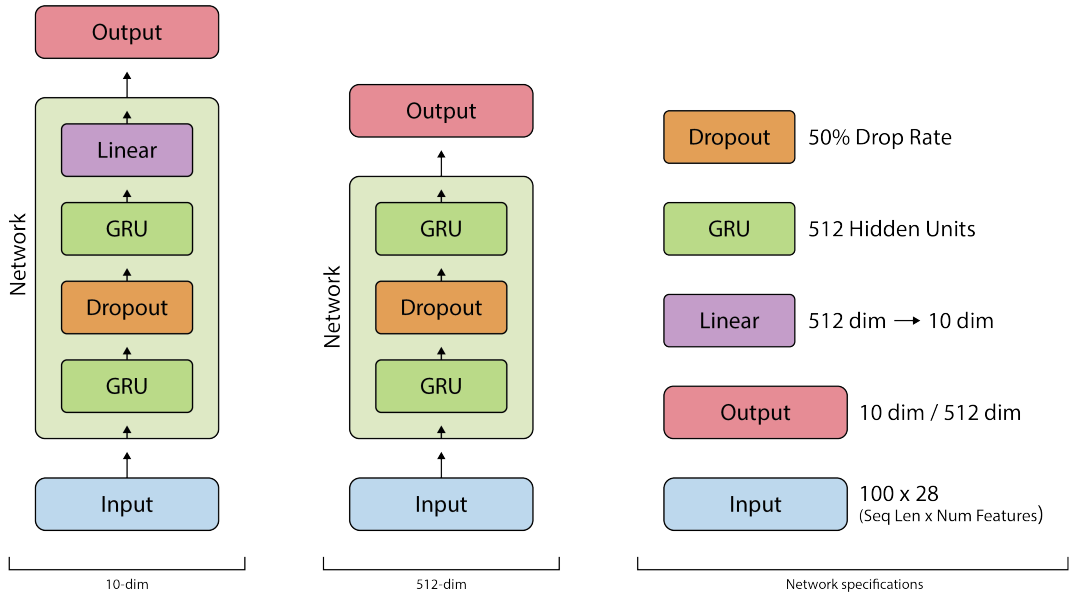


Figure 3.7: The two network models used during the thesis.

3.7.1 Re-Identifying Seen Sessions

The re-identification of seen sessions utilises the first session of the ten individuals. The same splits were used for the network design evaluation, i.e., 70%/15%/15%. For evaluation, the training data was fed into the network and mapped into the embedding space. A query from the validation and test splits were then mapped and compared to the training samples one at a time to obtain the embedding scores.

3.7.2 Re-Identifying Unseen Sessions

For the re-identification of unseen sessions, the first sessions of the ten individuals were trained and evaluated upon. The second sessions of the first and second individuals were then used during testing. The training and validation splits from the first sessions were set to 85%/15%, while the testing split consisted of 100% of the samples in the second sessions. When evaluating, the same process as for re-identifying the seen sessions was used to map the training samples and compare them to validation and testing queries.

3.7.3 Identifying Unseen Individuals

To evaluate if the network model was able to adapt to unseen individuals, seven individuals were used for training and validation. When the network had finished learning, the model was saved and the training and validation splits were discarded. Then, the samples from the three unseen individuals were split into two subsets. One subset was first mapped to the embedding space, while the other split was used as queries. The learning phase training and validation splits were set to 85%/15% for the seven individuals. The mapping and query splits used for testing were set to 70%/30%. For each run, new randomised individual splits were chosen, instead of having the same split for the ten runs.

Separating the Views

To this point, the network was fed sequences from the five different views, with the individual label as the sole metadata. Evaluation of the separate camera views was performed to tell if gait is difficult to extract from certain views, as well as to analyse if the separations performed

better than for the combined views. This was investigated using the same strategies described for evaluating the adaption to unseen individuals. The results from the separate views are presented in Sec. 4.1.3. These results indicate that the separate views are prone to generalise better for unseen individuals, compared to when they are combined.

3.8 The Augmented View Dependency

Ideally, the goal would be to have one network model that works for all camera views. However, if it is not able to generalise well to unseen data, it might not be useful to use such a model at all. An experiment was therefore conducted using the triplet loss, to analyse if it was possible to guide the network in its learning process. A hypothesis formed around creating a model that could differentiate between the views, as well as the individuals. The first step was to see if the network was able to identify the different camera views, by forming clusters in the embedding space based on the camera views. This was done by labeling the data samples with their corresponding view instead of their individual label. The network was able to learn this clustering after only one or two epochs, which indicated that this task was not excessively difficult, given the available data and current model. The next step of the hypothesis was to add a second triplet loss function to compute the loss for individual labels. The network was set to compute both losses before performing the backpropagation. The result was a network model that learned to differentiate between separate views and individuals. The same margin was used for the triplet loss when it came to the individuals, i.e., the margin was set to 0.1. The margin used for the view loss went through a grid search and a margin of 0.5 gave the best performance and was chosen. To differentiate between this learning method and the previous one using the triplet loss function, this network that measures both view and individual labels, is denoted as *Dependency-Triplet* (D-Triplet).

To assess the new proposed method, which had an added view dependency, the same evaluations were conducted, as for the original deep metric analysis of Sec. 3.7. The reason was to be able to compare the methods. However, as this new model was intended to handle all of the views at a time, no evaluation was performed on the separate views. Furthermore, only the triplet loss was used for this method, and the batch size was set to 64 instead of 32. In order to compare the proposed model to the original method, the network was re-run with the regular triplet loss and a batch size of 64. Finally, the investigation was extended to also evaluate the adaption to unseen individuals, with five learning individuals and five testing individuals.

3.9 The Pipeline

Fig. 3.8 show one of the many finalised pipelines used during the thesis. This setup was used to form an embedding space using a triplet network, the triplet loss and the 10-dim network model. The pipeline is split into three main parts, at the top is the extraction and processing of the FOI Gait Dataset, in the middle is the training of the model, and at the bottom is the evaluation of the trained model. Lastly, the outputs of the model are directly represented in an embedding space.

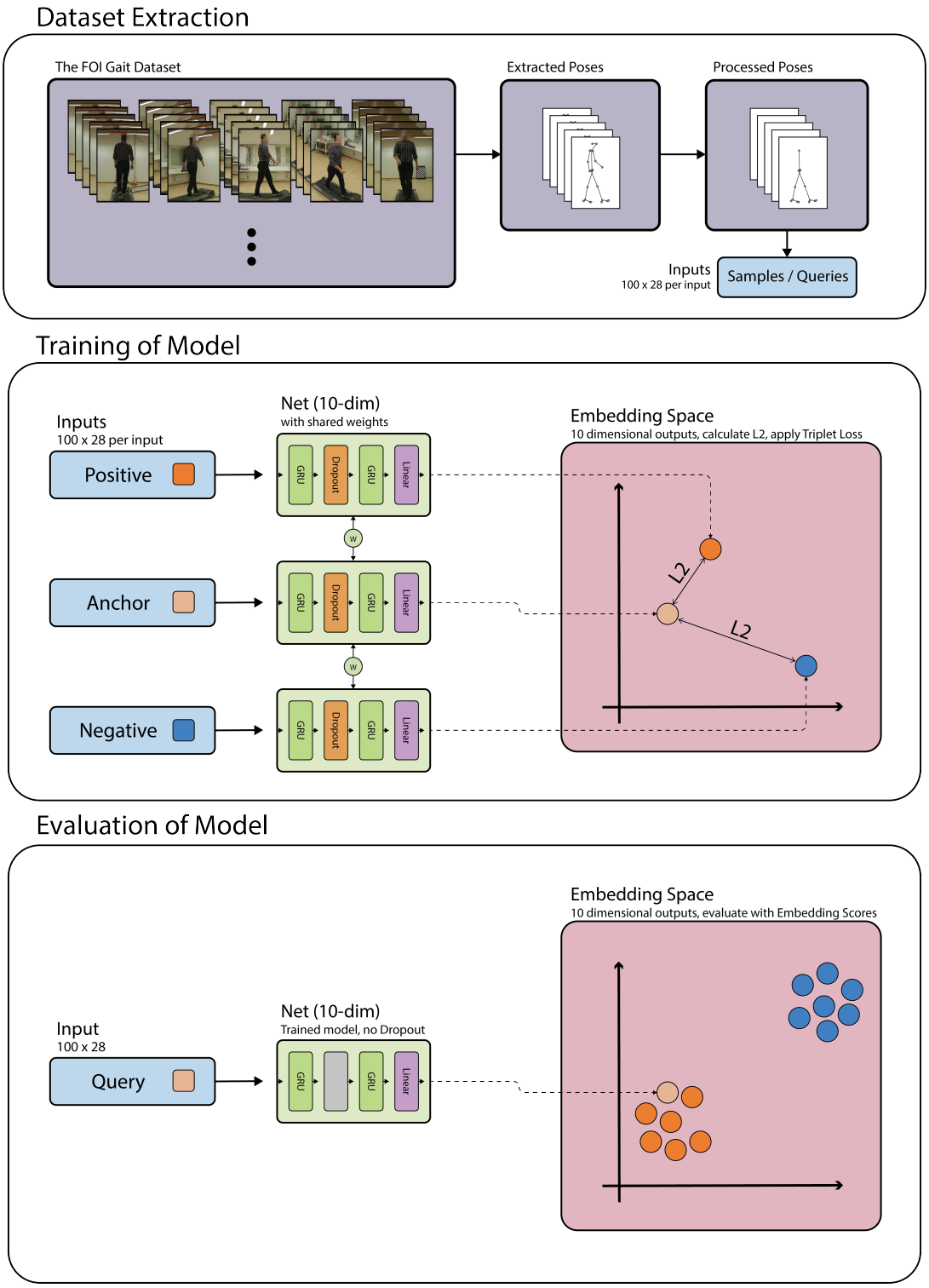


Figure 3.8: A visualisation of the full pipeline for a triplet network setup.

4 Results

In this chapter, the outcome of using deep metric learning to re-identify the seen and unseen walking gait sessions, as well as the results of the generalisability of the network for unseen individuals, with combined and separate views are presented. Lastly, the results of the proposed method are shown, where the view dependency is augmented using a second loss function. The numerical values presented in the tables of this chapter are presented as the mean of 10 network runs, together with the 95% confidence interval deviation. The best scores in the tables are highlighted using a bold font. To see the results for the recurrent neural network designs for different sequence lengths, see Appendix B.

4.1 Deep Metric Learning Evaluation

In this section, the results for the re-identification of sessions and identifying of new individuals are presented using deep metric learning and embedding space analysis. Three losses are evaluated; CrossEntropy, Contrastive, and Triplet, for the 512 dimensional and the linearly reduced 10 dimensional embedding spaces. The embedding scores P@1, RP, MAP@R, and SC were used. Every table represent results from using all five views simultaneously, with the exception of Table 4.4, where the views are separated. As the 512 dimensional space is not a logits space, the CrossEntropy classification loss is not able to learn and is left out of the tables. Therefore, the deep-metric-based Contrastive and Triplet losses are only comparable to the non-deep-metric-based CrossEntropy loss in the 10 dimensional space.

4.1.1 Re-Identifying Seen Sessions

The results for re-identifying the seen sessions are shown in Table 4.1. The testing sequences used for the evaluation were still unseen by the network, however, they are perceived to be similar to the learning sequences.

Table 4.1: The embedding scores from learning and testing on the first sessions.

	10-dim				512-dim			
	P@1 (%)	**RP (%)**	**MAP@R (%)**	**SC**	**P@1 (%)**	**RP (%)**	**MAP@R (%)**	**SC**
CrossEntropy	**99.82 ± 0.04**	**99.67 ± 0.05**	**99.62 ± 0.06**	0.67 ± 0.00	-	-	-	-
Contrastive	99.48 ± 0.21	98.06 ± 0.68	97.75 ± 0.83	0.76 ± 0.03	97.97 ± 0.54	87.17 ± 4.60	85.32 ± 5.54	0.61 ± 0.09
Triplet	99.17 ± 0.14	98.76 ± 0.21	98.54 ± 0.26	**0.76 ± 0.01**	**99.18 ± 0.17**	**98.96 ± 0.15**	**98.89 ± 0.17**	**0.63 ± 0.01**

4.1.2 Re-Identifying Unseen Session

Table 4.2 show the results from evaluation of re-identification on the second sessions of individuals one and two, which were completely unseen during the learning phase of the network.

Table 4.2: The embedding scores from learning on the first sessions and testing on the second sessions.

	10-dim				512-dim			
	P@1 (%)	**RP (%)**	**MAP@R (%)**	**SC**	**P@1 (%)**	**RP (%)**	**MAP@R (%)**	**SC**
CrossEntropy	68.56 ± 2.41	69.81 ± 2.45	66.65 ± 2.56	**0.48 ± 0.01**	-	-	-	-
Contrastive	**72.37 ± 4.48**	**73.14 ± 3.97**	**69.31 ± 4.64**	0.47 ± 0.04	63.36 ± 3.40	58.83 ± 6.68	54.29 ± 7.85	0.33 ± 0.07
Triplet	65.10 ± 2.92	65.76 ± 3.05	62.62 ± 3.16	0.43 ± 0.03	**63.68 ± 1.81**	**66.03 ± 1.99**	**64.75 ± 2.06**	**0.35 ± 0.01**

4.1.3 Identifying Unseen Individuals

The results from learning on seven individuals and testing on three unseen individuals, with all of the camera views, is presented in Table 4.3. The individual split was randomised for all ten runs.

Table 4.3: The embedding scores from learning on seven individuals and testing on three individuals.

	10-dim				512-dim			
	P@1 (%)	**RP (%)**	**MAP@R (%)**	**SC**	**P@1 (%)**	**RP (%)**	**MAP@R (%)**	**SC**
CrossEntropy	**93.36 ± 1.13**	**59.69 ± 1.96**	**47.97 ± 2.51**	**0.20 ± 0.02**	-	-	-	-
Contrastive	82.21 ± 4.70	51.99 ± 5.60	36.92 ± 7.05	0.13 ± 0.06	70.85 ± 10.64	49.53 ± 6.14	32.56 ± 8.26	0.11 ± 0.09
Triplet	91.82 ± 3.25	54.60 ± 4.68	40.98 ± 5.60	0.15 ± 0.04	**95.93 ± 2.44**	**54.84 ± 6.28**	**41.71 ± 8.62**	**0.11 ± 0.05**

Separating the Views

The identifying of unseen individuals is shown in Table 4.4, however, the camera views were separated and run separately. 10 runs were conducted for each view, with randomised individual splits.

Table 4.4: The embedding scores from learning on seven individuals and testing on three individuals. Individual views.

		10-dim				512-dim			
		P@1 (%)	**RP (%)**	**MAP@R (%)**	**SC**	**P@1 (%)**	**RP (%)**	**MAP@R (%)**	**SC**
CrossEntropy	**a)**	91.16 ± 4.78	70.97 ± 9.45	62.36 ± 12.46	0.35 ± 0.10				
	b)	89.79 ± 7.09	73.48 ± 9.78	65.11 ± 14.06	0.40 ± 0.13				
	c)	87.17 ± 9.69	67.69 ± 12.04	57.93 ± 16.44	0.30 ± 0.16	-	-	-	-
	d)	**97.05 ± 2.58**	**81.59 ± 6.30**	**77.87 ± 8.18**	**0.50 ± 0.10**				
	e)	91.11 ± 6.96	73.53 ± 11.89	66.20 ± 15.82	0.41 ± 0.16				
Contrastive	**a)**	98.08 ± 1.85	90.67 ± 6.69	87.82 ± 9.00	0.65 ± 0.11	95.77 ± 2.51	77.74 ± 9.25	70.41 ± 12.63	0.46 ± 0.13
	b)	97.04 ± 3.59	90.16 ± 5.62	87.37 ± 7.90	0.65 ± 0.09	92.35 ± 6.30	78.79 ± 9.64	71.54 ± 13.80	0.48 ± 0.14
	c)	97.43 ± 1.89	92.12 ± 5.15	**89.83 ± 6.95**	**0.68 ± 0.09**	97.17 ± 2.44	80.23 ± 10.29	75.6 ± 13.27	0.51 ± 0.14
	d)	**98.81 ± 1.06**	**91.23 ± 4.78**	89.47 ± 6.07	0.65 ± 0.09	94.21 ± 4.86	73.05 ± 6.35	67.63 ± 8.50	0.41 ± 0.12
	e)	94.92 ± 6.06	85.84 ± 9.13	81.04 ± 12.86	0.54 ± 0.13	**97.66 ± 1.87**	**89.12 ± 6.98**	**86.16 ± 9.01**	**0.61 ± 0.10**
Triplet	**a)**	96.51 ± 4.06	87.68 ± 9.69	83.74 ± 13.37	0.58 ± 0.14	98.50 ± 1.62	88.31 ± 8.52	85.09 ± 11.23	0.57 ± 0.13
	b)	97.23 ± 3.18	88.35 ± 7.64	85.16 ± 10.20	0.59 ± 0.12	96.96 ± 4.12	89.71 ± 6.51	86.32 ± 9.43	0.61 ± 0.09
	c)	97.23 ± 3.18	88.35 ± 7.64	85.16 ± 10.20	0.59 ± 0.12	97.62 ± 1.90	88.95 ± 7.38	86.06 ± 9.68	0.54 ± 0.11
	d)	**99.93 ± 0.16**	**97.41 ± 1.55**	**97.10 ± 1.73**	**0.73 ± 0.05**	**99.19 ± 1.48**	**95.81 ± 4.93**	**94.85 ± 6.55**	**0.66 ± 0.08**
	e)	99.20 ± 1.47	94.70 ± 3.78	93.54 ± 5.05	0.62 ± 0.06	97.56 ± 2.66	87.28 ± 8.98	83.27 ± 12.35	0.49 ± 0.10

4.2 The Augmented View Dependency

This section presents the result from the proposed network learning structure with two triplet losses. One loss function measures the loss according to the views of samples, while the

other measures the loss in-between individuals. All of the results presented here are conducted with a batch size of 64 instead of 32, with a margin of 0.5 for the loss measuring the view dependency, and 0.1 for the other one. This proposed triplet variant is compared to the regular triplet loss, which only checks the individual dependency. The embedding spaces and score metrics are the same as described in Sec. 4.1. Table 4.5 present the scores on the re-identification of the seen sessions, while Table 4.6 show the results from testing on the unseen sessions of individuals one and two. Appendix C show visualisation of the samples from the resulting 10-dimensional embedding spaces presented in Table 4.5.

Two different tests were conducted for identifying unseen individuals. Table 4.7 present the results from a learning and test individual splits of seven to three, while Table 4.8 show a five to five split. The ten runs conducted for each split applied randomised individual splits.

Table 4.5: Comparing the triplet loss with different amounts metadata. Embedding scores from learning on the first sessions and testing on the second sessions.

	10-dim				512-dim			
	P@1 (%)	**RP (%)**	**MAP@R (%)**	**SC**	**P@1 (%)**	**RP (%)**	**MAP@R (%)**	**SC**
Triplet	99.70 ± 0.05	99.63 ± 0.04	99.58 ± 0.04	**0.80 ± 0.00**	99.6 ± 0.10	99.52 ± 0.10	99.48 ± 0.10	0.65 ± 0.01
D-Triplet	**99.98 ± 0.02**	**99.93 ± 0.02**	**99.91 ± 0.03**	0.79 ± 0.00	**99.99 ± 0.01**	**99.94 ± 0.02**	**99.93 ± 0.03**	**0.73 ± 0.00**

Table 4.6: Comparing the triplet loss with different amounts metadata. Embedding scores from learning on the first sessions and testing on the second sessions.

	10-dim				512-dim			
	P@1 (%)	**RP (%)**	**MAP@R (%)**	**SC**	**P@1 (%)**	**RP (%)**	**MAP@R (%)**	**SC**
Triplet	74.84 ± 2.23	74.84 ± 1.95	72.38 ± 2.00	0.48 ± 0.02	69.07 ± 2.24	69.71 ± 2.10	**68.48 ± 2.16**	0.37 ± 0.02
D-Triplet	**77.51 ± 2.70**	**78.93 ± 2.45**	**74.58 ± 2.72**	**0.84 ± 0.01**	**70.07 ± 1.41**	**70.61 ± 1.71**	67.79 ± 1.55	**0.77 ± 0.01**

Table 4.7: Comparing the triplet loss with different amounts metadata. Embedding scores from learning on seven individuals and testing on three individuals.

	10-dim				512-dim			
	P@1 (%)	**RP (%)**	**MAP@R (%)**	**SC**	**P@1 (%)**	**RP (%)**	**MAP@R (%)**	**SC**
Triplet	93.60 ± 4.03	58.02 ± 7.23	46.54 ± 9.69	0.20 ± 0.07	97.65 ± 1.50	58.33 ± 5.54	46.48 ± 7.05	0.15 ± 0.05
D-Triplet	**99.22 ± 0.54**	**93.34 ± 1.27**	**92.02 ± 1.66**	**0.62 ± 0.03**	**99.80 ± 0.10**	**96.39 ± 1.60**	**95.74 ± 1.96**	**0.62 ± 0.04**

Table 4.8: Comparing the triplet loss with different amounts metadata. Embedding scores from learning on five individuals and testing on five individuals.

	10-dim				512-dim			
	P@1 (%)	**RP (%)**	**MAP@R (%)**	**SC**	**P@1 (%)**	**RP (%)**	**MAP@R (%)**	**SC**
Triplet	80.16 ± 5.09	39.73 ± 4.60	24.07 ± 4.77	0.02 ± 0.04	89.27 ± 3.16	37.33 ± 3.75	23.80 ± 4.25	0.02 ± 0.03
D-Triplet	**95.98 ± 1.83**	**81.55 ± 5.26**	**76.28 ± 6.89**	**0.46 ± 0.06**	**98.10 ± 1.47**	**84.17 ± 5.05**	**80.20 ± 6.78**	**0.43 ± 0.06**

5 Discussion

This chapter encompasses a discussion about the results and the methods of the thesis, followed by outlining the thesis in a wider context, together with societal and ethical aspects correlated with automated recognition and the admissibility of Gait.

5.1 Results

The results show promise for a continued investigation of pose estimation based walking gait analysis. However, not every result was as expected and they are worthy of discussion.

5.1.1 Network Models

The gated recurrent cells outperformed the regular RNN for sequences of increasing length. This result was expected, as the cells were specifically designed for that purpose. However, unanticipated was the fact that bidirectionality did not noticeably increase the performance. A potential explanation could be the relative predictability of moving keypoints, i.e., future values of the sequence may not be needed for the network model to be able to figure out where a keypoint will be traversed to next. To test this further, it would be of interest to try transformers, as they are perceived to surpass RNNs when capturing sequential dependencies. If a transformer does not improve the results, this could indicate that a unidirectional, gated recurrent cell is sufficient in capturing kinematic pose estimated gait.

5.1.2 Embedding Spaces Evaluation Scores

When training a network to map values to an embedding space, the use of embedding losses was presumed to outperform a classification loss, such as CrossEntropy. However, CrossEntropy was shown to outperform the triplet loss in every comparable evaluation when, all views were used simultaneously, in the 10 dimensional embedding space, for the P@1, RP and MAP@R scores. The classification loss also surpassed the Contrastive loss for these scores, except in the case of re-identifying individuals from unseen sessions. The Silhouette Coefficient, which measures how well formed clusters of the same labels are, was the highest for the CrossEntropy loss for unseen individuals and second, unseen sessions, while highest for the embedding losses when tested on the first sessions. This indicates that the classification

loss was able to outperform the embedding losses when clustering, even if clustering is not the main task of classification losses.

It is of interest to discuss why an embedding loss, designed to create an embedding space, is worse than a classification loss. The reason might be that two samples with different labels, captured from the same view, are more similar than two samples of the same label, captured from different views, when looking at sequences of 2D kinematic poses. For a case like this, it might be straightforward to find a class separation, with a relatively well formed embedding space as a byproduct. Attempting to map such an embedding space directly might be more difficult and require a larger dataset.

This argument is further strengthened when analysing the camera views separately, where the embedding losses outperform the classification loss for every score. Here, the distance is more intuitively measured, at least when assuming the difference in gait for the individuals in the dataset. Further, when using the augmented view dependency metadata, which helps carry this intuition to a model trained on all views, the model is able to outperform all the previous models.

Re-Identification of Individuals

According to the results from the embedding space evaluations, re-identifying individuals based on sessions seen during training is not a difficult task. The models can perform well on test data from the same session as the training data. However, the models that evaluate the same sessions correctly are not well generalised to the unseen sessions of individuals one and two, which were captured on different dates with different clothing. One explanation for this could be that OpenPose extracts different pose estimations, as the clothing is different. The most likely reason is that the gaits of the individuals are slightly different from session to session. An overfitted model might then perceive these as sequences from separate individuals. The final reason could be that gait is not a robust enough technique that can be used to always re-identify individuals. This would need to be evaluated further, with an extended dataset with several sessions for individuals.

Identifying Unseen Individuals

The results from evaluating the model on unseen individuals indicate that the models had some issues generalising. However, when either evaluating the separate views, or with the augmented view dependency, the models were able to generalise quite well. The best results were received from the D-Triplet loss in the 512 embedding space, for both three and five unseen individuals.

5.2 Method

Here, some of the methods used during the thesis are discussed, focusing on their credibility and usefulness.

5.2.1 Human Pose Estimation for Gait Representation

Using HPE to represent gait can be precarious if the pose estimator itself is not able to generalise to new individuals. Otherwise, the pose representation used for the gait of a person might be highly influenced by the individuals seen during the training of the pose estimator. This could cause some ethical concern as well and is discussed further in Sec. 5.3. It would also be beneficial to benchmark this representation against other methods, such as Gait Energy Images, as well as other gait datasets.

5.2.2 Network Settings and Hyperparameter Initialisation

When using deep learning, it is inevitable to discuss the initialisation of settings and hyperparameters. Grid search was used to test many variants and combinations of settings and hyperparameters. However, this does not mean that the best combinations were found. Furthermore, multi-class classification was used to investigate the network types and sequence lengths used for the deep metric learning task. Ideally, the networks and sequence lengths would be determined on the deep metric learning task instead. However, the classification task could be performed with greater speed, which was of essence during this thesis with limited time resources.

5.2.3 Scalability of the Deep Metric Model

A goal of a deep metric based model used for identification tasks is to scale, and not be limited to the queries and their labels seen during the training of the network. Although the D-Triplet performed well on the available data, it will not necessarily scale to an extension of the FOI Gait Dataset, or other datasets. This method was created with knowledge about the data, e.g. that the dataset only contains five camera views. Applying the D-Triplet method to a dataset with additional camera views, with less distance in-between them, might result in worse performance than using the regular method, where the view dependency is not augmented. However, even if D-Triplet is unable to scale, it does show that a dependency on views is learnable, and can be an important feature to learn with this type of data. Furthermore, it is not certain if a model trained with individuals walking in place, on a treadmill, will adapt to in-the-wild data. To determine whether the models are scalable, more data is naturally needed.

5.3 The Work in a Wider Context

Automated solutions for the identification of individuals is a concerning issue worthy of discussion, as the purpose under which these methods are deployed may border to, or fall into, unethical territories. The thesis was conducted at the Swedish Defence Research Agency, and the Total Defence of Sweden falls under strict laws of how authorities with surveillance responsibility (*Bevakningsansvariga myndigheter*) [77] are able to conduct themselves. However, critical situations, such as wartime, can arise where authorities (not limited to Sweden) encroach on situations where methods, otherwise deemed unethical, are deployed [78]. Automated approaches to identify individuals are not only limited to authorities, anyone with the hardware and knowledge can develop such methods, and use them maliciously. Although unlikely, it cannot be excluded that this thesis investigating gait analysis will contribute to such automation methods.

An automated identification method using video surveillance can, however, if used correctly, help increase the privacy and integrity of individuals. If an authority with surveillance responsibility [77] is looking for a suspect, a manual approach might involve operatives looking through video footage of several individuals. An automated approach, could scan and limit the number of cases with innocent individuals that need to be examined, and therefore increase privacy.

The Admissibility of Gait

As mentioned in Sec. 1.4, the gait of an individual can vary significantly depending on the situation, which includes changes in walking speed and angle, clothing, and carried objects [6]. Further, this dynamic behaviour, and the unclarities surrounding the uniqueness when using gait as a biometric identifier, have resulted in the admissibility of gait as forensic evidence to be under critique. However, in a case where gait is not fully admissible, it may still be able to be used as supportive evidence [2]. Furthermore, an automated or semi-automated

solution can still ease the workload for the otherwise manual labor of tracing the movements of suspects. Akin to how another indicator, such as clothing colour, can reduce the cases needing examination, gait can indicate which cases are most likely be from the same suspect.

6 Conclusion

Altogether, the conducted thesis furthered the investigation of using gait features to re-identify individuals and identify new people. Deep metric learning based methods were used to design suitable neural network structures, using metadata consisting of labeled individuals. The added view dependency, which introduced further metadata in the form of labeled views, indicates the importance of understanding camera view differences when using the evaluated methods.

This chapter concludes the thesis by firstly answering the research questions, followed by a section outlining future works suggestions.

6.1 Research Questions

With the walking gait being extracted from 2D kinematic pose estimation, from a limited dataset consisting of monocular video sequences of walking individuals, where the models are either deep-metric-, or non-deep-metric-based, the research questions and their answers are concluded as follows:

1. **How do the models compare to each other when trained on several camera views, using labels of individuals as the only metadata, for the scalable tasks of re-identifying individuals, as well as identifying new individuals?**

   A network model designed for classification is able to outperform a deep metric-based model when scaling to new individuals, as well as re-identifying individuals based on the same walking session they were trained on . However, the deep metric-based model using the Contrastive loss generalises better when re-identifying individuals in-between walking sessions.

2. **To what extent can the network models handle different camera views?**

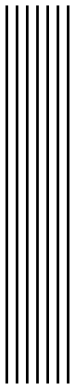
   With the limited dataset, a network trained using labels of individuals as the only metadata, the model is unable to generalise well. However, when models are designed and tested using the views separately, they generalise better. Further, when metadata of the camera views is supplied together with the labels of the individuals, to a model designed and tested on all views simultaneously, it is able to generalise better than the models lacking the camera view metadata.

6.2 Future Work

The method of using human pose estimation to represent gait, together with neural network models show promise. Deep metric learning and an analysis of embedding spaces are suitable tools for evaluating these models. However, further work is still needed to decide if gait can be used as a unique identifier for individuals, as well as concluding if network models are able to generalise to unseen walking sessions and individuals.

Future work involves gathering more data, both for learning, and evaluating the network models. Interesting data involves more individuals, walking for several sessions, covered from more camera angles. Sessions could consist of individuals walking at many speeds, wearing different types of clothes, as well as a plethora of other varying factors. Such sessions have the possibility of contributing to a better network model, and the study of the admissibility of gait. Suggested future work would also evaluate the models on in-the-wild data, to realise if similar methods are usable in real world scenarios.

Finally, further recommended subsequent studies could evaluate the use of transformers in a comparison to recurrent neural networks when human pose estimation is used, as well as comparing human pose estimation to other gait representations, such as gait energy image.



Bibliography

[1] Ivan Birch, Wesley Vernon, Jeremy Walker, and Maria Young. "Terminology and forensic gait analysis". In: *Science & Justice* 55.4 (2015), pp. 279–284. ISSN: 1355-0306. DOI: https://doi.org/10.1016/j.scijus.2015.03.002. URL: https://www.sciencedirect.com/science/article/pii/S1355030615000349.

[2] Ioana Macoveciuc, Carolyn J. Rando, and Hervé Borrion. "Forensic Gait Analysis and Recognition: Standards of Evidence Admissibility". In: *Journal of Forensic Sciences* 64.5 (2019), pp. 1294–1303. DOI: https://doi.org/10.1111/1556-4029.14036. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1556-4029.14036. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1556-4029.14036.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[4] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z. Li. "Deep Metric Learning for Person Re-identification". In: *2014 22nd International Conference on Pattern Recognition*. 2014, pp. 34–39. DOI: 10.1109/ICPR.2014.16.

[5] Junlin Hu, Jiwen Lu, and Yap-Peng Tan. "Discriminative Deep Metric Learning for Face Verification in the Wild". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1875–1882. DOI: 10.1109/CVPR.2014.242.

[6] Nina M van Mastrigt, Kevin Celie, Arjan L Mieremet, Arnout C C Ruifrok, and Zeno Geradts. "Critical review of the use and scientific basis of forensic gait analysis". In: *Forensic sciences research* 3.3 (2018), pp. 183–193. ISSN: 2096-1790. DOI: 10.1080/20961790.2018.1503579. URL: https://europepmc.org/articles/PMC6201773.

[7] Joseph C. Kolecki. *An Introduction To Tensors for Students of Physics and Engineering*. https://www.grc.nasa.gov/www/k-12/Numbers/Math/documents/Tensors_TM2002211716.pdf. 2002.

[8] *TORCH.TENSOR*. https://pytorch.org/docs/stable/tensors.html. Accessed: 2021-05-05.

[9] *Introduction to Tensors*. https://www.tensorflow.org/guide/tensor. Accessed: 2021-05-05.

[10] Norman R Draper and Harry Smith. *Applied regression analysis*. Vol. 326. John Wiley & Sons, 1998.

[11] Michael Nirenberg, Wesley Vernon, and Ivan Birch. "A review of the historical use and criticisms of gait analysis evidence". In: *Science & Justice* 58 (Mar. 2018). DOI: 10.1016/j.scijus.2018.03.002.

[12] Ivan Birch, Louis Raymond, Anastasia Christou, Milan Angelo Fernando, Nigel Harrison, and Flo Paul. "The identification of individuals by observational gait analysis using closed circuit television footage". In: *Science & Justice* 53.3 (2013), pp. 339–342. ISSN: 1355-0306. DOI: https://doi.org/10.1016/j.scijus.2013.04.005. URL: https://www.sciencedirect.com/science/article/pii/S1355030613000476.

[13] A. S. Alharthi, S. U. Yunas, and K. B. Ozanyan. "Deep Learning for Monitoring of Human Gait: A Review". In: *IEEE Sensors Journal* 19.21 (2019), pp. 9575–9591. DOI: 10.1109/JSEN.2019.2928777.

[14] Ju Han and Bir Bhanu. "Individual Recognition Using Gait Energy Image". In: *IEEE transactions on pattern analysis and machine intelligence* 28 (Mar. 2006), pp. 316–22. DOI: 10.1109/TPAMI.2006.38.

[15] Richard Szeliski. *Computer Vision: Algorithms and Applications, 2nd ed.* http://szeliski.org/Book/. Springer, Mar. 2021.

[16] Xiaoyang Liu and Jinqiang Liu. "Gait Recognition Method of Underground Coal Mine Personnel Based on Densely Connected Convolution Network and Stacked Convolutional Autoencoder". In: *Entropy* 22 (June 2020), p. 695. DOI: 10.3390/e22060695.

[17] Thomas B. Moeslund and Erik Granum. "A Survey of Computer Vision-Based Human Motion Capture". In: *Computer Vision and Image Understanding* 81.3 (2001), pp. 231–268. ISSN: 1077-3142. DOI: https://doi.org/10.1006/cviu.2000.0897. URL: https://www.sciencedirect.com/science/article/pii/S107731420090897X.

[18] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. "A survey of advances in vision-based human motion capture and analysis". In: *Computer Vision and Image Understanding* 104.2 (2006). Special Issue on Modeling People: Vision-based understanding of a person's shape, appearance, movement and behaviour, pp. 90–126. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2006.08.002. URL: https://www.sciencedirect.com/science/article/pii/S1077314206001263.

[19] Ronald Poppe. "Vision-based human motion analysis: An overview". In: *Computer Vision and Image Understanding* 108.1 (2007). Special Issue on Vision for Human-Computer Interaction, pp. 4–18. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2006.10.016. URL: https://www.sciencedirect.com/science/article/pii/S1077314206002293.

[20] Xiaofei Ji and Honghai Liu. "Advances in View-Invariant Human Motion Analysis: A Review". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.1 (2010), pp. 13–24. DOI: 10.1109/TSMCC.2009.2027608.

[21] Michael B. Holte, Cuong Tran, Mohan M. Trivedi, and Thomas B. Moeslund. "Human Pose Estimation and Activity Recognition From Multi-View Videos: Comparative Explorations of Recent Developments". In: *IEEE Journal of Selected Topics in Signal Processing* 6.5 (2012), pp. 538–552. DOI: 10.1109/JSTSP.2012.2196975.

[22] Zhao Liu, Jianke Zhu, Jiajun Bu, and Chun Chen. "A survey of human pose estimation: The body parts parsing based methods". In: *Journal of Visual Communication and Image Representation* 32 (2015), pp. 10–19. ISSN: 1047-3203. DOI: https://doi.org/10.1016/j.jvcir.2015.06.013. URL: https://www.sciencedirect.com/science/article/pii/S1047320315001121.

[23] Wenjuan Gong, Xuena Zhang, J. González, Andrews Sobral, T. Bouwmans, C. Tu, and E. Zahzah. "Human Pose Estimation from Monocular Images: A Comprehensive Survey". In: *Sensors (Basel, Switzerland)* 16 (2016).

[24] Nikolaos Sarafianos, Bogdan Boteanu, Bogdan Ionescu, and Ioannis A. Kakadiaris. "3D Human pose estimation: A review of the literature and analysis of covariates". In: *Computer Vision and Image Understanding* 152 (2016), pp. 1–20. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2016.09.002. URL: https://www.sciencedirect.com/science/article/pii/S1077314216301369.

[25] Yucheng Chen, Yingli Tian, and Mingyi He. "Monocular human pose estimation: A survey of deep learning-based methods". In: *Computer Vision and Image Understanding* 192 (Mar. 2020), p. 102897. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2019.102897. URL: http://dx.doi.org/10.1016/j.cviu.2019.102897.

[26] Tewodros Legesse Munea, Yalew Zelalem Jembre, Halefom Tekle Weldegebriel, Longbiao Chen, Chenxi Huang, and Chenhui Yang. "The Progress of Human Pose Estimation: A Survey and Taxonomy of Models Applied in 2D Human Pose Estimation". In: *IEEE Access* 8 (2020), pp. 133330–133348. DOI: 10.1109/ACCESS.2020.3010248.

[27] Ce Zheng, Wenhan Wu, Taojiannan Yang, Sijie Zhu, Chen Chen, Ruixu Liu, Ju Shen, Nasser Kehtarnavaz, and Mubarak Shah. *Deep Learning-Based Human Pose Estimation: A Survey*. 2021. arXiv: 2012.13392 [cs.CV].

[28] Ching-Hang Chen and Deva Ramanan. *3D Human Pose Estimation = 2D Pose Estimation + Matching*. 2017. arXiv: 1612.06524 [cs.CV].

[29] Hongsuk Choi, Gyeongsik Moon, and Kyoung Mu Lee. *Pose2Mesh: Graph Convolutional Network for 3D Human Pose and Mesh Recovery from a 2D Human Pose*. 2020. arXiv: 2008.09047 [cs.CV].

[30] Denis Tome, Chris Russell, and Lourdes Agapito. "Lifting from the Deep: Convolutional 3D Pose Estimation from a Single Image". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017). DOI: 10.1109/cvpr.2017.603. URL: http://dx.doi.org/10.1109/CVPR.2017.603.

[31] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler. *Efficient Object Localization Using Convolutional Networks*. 2015. arXiv: 1411.4280 [cs.CV].

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[33] Alexander Amini and Ava Soleimany. *MIT Introduction to Deep Learning | 6.S191*. http://introtodeeplearning.com/. 2021.

[34] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert" Müller. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3. URL: https://doi.org/10.1007/978-3-642-35289-8_3.

[35] Yannis M. Assael, Brendan Shillingford, Shimon Whiteson, and Nando de Freitas. *LipNet: End-to-End Sentence-level Lipreading*. 2016. arXiv: 1611.01599 [cs.LG].

[36] Gabriel Eilertsen, Joel Kronander, Gyorgy Denes, Rafał K. Mantiuk, and Jonas Unger. "HDR image reconstruction from a single exposure using deep CNNs". In: *ACM Transactions on Graphics* 36.6 (Nov. 2017), pp. 1–15. ISSN: 1557-7368. DOI: 10.1145/3130800.3130816. URL: http://dx.doi.org/10.1145/3130800.3130816.

[37] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[38] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. "DeepFace: Closing the Gap to Human-Level Performance in Face Verification". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1701–1708. DOI: 10.1109/CVPR.2014.220.

[39] Yi Sun, Xiaogang Wang, and Xiaoou Tang. "Deep Learning Face Representation from Predicting 10,000 Classes". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1891–1898. DOI: 10.1109/CVPR.2014.244.

[40] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. *The Loss Surfaces of Multilayer Networks*. 2015. arXiv: 1412.0233 [cs.LG].

[41] *CrossEntropyLoss*. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. Accessed: 2021-05-16.

[42] B.T. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(64)90137-5. URL: https://www.sciencedirect.com/science/article/pii/0041555364901375.

[43] Robert A. Jacobs. "Increased rates of convergence through learning rate adaptation". In: *Neural Networks* 1.4 (1988), pp. 295–307. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(88)90003-2. URL: https://www.sciencedirect.com/science/article/pii/0893608088900032.

[44] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12.null (July 2011), pp. 2121–2159. ISSN: 1532-4435.

[45] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. arXiv: 1212.5701 [cs.LG].

[46] G. Hinton. *RMSProp: Normalize the gradient*. https://www.cs.toronto.edu/~hinton/coursera_lectures.html. Accessed: 2021-05-26. 2012.

[47] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[48] Yann Lecun and F. Fogelman Soulie. "Modeles connexionnistes de l'apprentissage". English (US). In: *Intellectica* special issue apprentissage et machine (Mar. 1987).

[49] D. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[50] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[51] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[52] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].

[53] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. 2013. arXiv: 1211.5063 [cs.LG].

[54] Tomás Mikolov. *STATISTICAL LANGUAGE MODELS BASED ON NEURAL NETWORKS*. 2012.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[56] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

[57] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[58] Chuanxing Geng, Sheng-Jun Huang, and Songcan Chen. "Recent Advances in Open Set Recognition: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1. ISSN: 1939-3539. DOI: 10.1109/tpami.2020.2981604. URL: http://dx.doi.org/10.1109/TPAMI.2020.2981604.

[59] Aurélien Bellet, Amaury Habrard, and Marc Sebban. *A Survey on Metric Learning for Feature Vectors and Structured Data*. 2014. arXiv: 1306.6709 [cs.LG].

[60] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. *A Metric Learning Reality Check*. 2020. arXiv: 2003.08505 [cs.CV].

[61] Imola K Fodor. *A survey of dimension reduction techniques*. Tech. rep. Citeseer, 2002.

[62] Elad Hoffer and Nir Ailon. *Deep metric learning using Triplet network*. 2018. arXiv: 1412.6622 [cs.LG].

[63] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. "Signature Verification Using a "Siamese" Time Delay Neural Network". In: *Proceedings of the 6th International Conference on Neural Information Processing Systems*. NIPS'93. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1993, pp. 737–744.

[64] R. Hadsell, S. Chopra, and Y. LeCun. "Dimensionality Reduction by Learning an Invariant Mapping". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. 2006, pp. 1735–1742. DOI: 10.1109/CVPR.2006.100.

[65] Kilian Q. Weinberger and Lawrence K. Saul. "Distance Metric Learning for Large Margin Nearest Neighbor Classification". In: *Journal of Machine Learning Research* 10.9 (2009), pp. 207–244. URL: http://jmlr.org/papers/v10/weinberger09a.html.

[66] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008. ISBN: 0521865719.

[67] Peter J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. URL: https://www.sciencedirect.com/science/article/pii/0377042787901257.

[68] Ian Jolliffe. "Principal Component Analysis". In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1094–1096. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_455. URL: https://doi.org/10.1007/978-3-642-04898-2_455.

[69] *Python*. https://docs.python.org/3/. Accessed: 2021-05-03.

[70] *PyTorch Documentation*. https://pytorch.org/docs/1.7.1/. Accessed: 2021-05-03.

[71] *OpenCV*. https://opencv.org/. Accessed: 2021-05-03.

[72] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[73] *TensorFlow's visualization toolkit*. https://www.tensorflow.org/tensorboard. Accessed: 2021-05-04.

[74] *C++ documentation*. https://www.cplusplus.com/. Accessed: 2021-05-04.

[75] *scikit-learn, Machine Learning in Python*. https://scikit-learn.org/stable/. Accessed: 2021-05-04.

[76] Martin Persson. "Automatic gait recognition: using deep learning". MA thesis. Linköping University, Computer Vision Laboratory, 2020, p. 40.

[77] *Bevakningsansvariga myndigheter*. https://www.msb.se/sv/amnesomraden/krisberedskap--civilt-forsvar/ansvar-och-roller/. Accessed: 2021-06-08.

[78] *Förordning (2015:1052) om krisberedskap och bevakningsansvariga myndigheters åtgärder vid höjd beredskap*. https://lagen.nu/2015:1052#P15S1. Accessed: 2021-06-08. Dec. 17, 2015.

A The Dataset in Detail

This appendix contains figures A.1 - A.10, which are still frames from Individual 1 - Individual 10. Individual 1 and 2 have two recorded sessions, seen in *a* and *b* of their respective figures. The remaining subjects recorded one session each, therefore, they do not have any sub-figures *a* or *b*.

(a) Session 1

(b) Session 2

Figure A.1: Still frames from Individual 1.

(a) Session 1

(b) Session 2

Figure A.2: Still frames from Individual 2.

Figure A.3: Still frames of Individual 3.

Figure A.4: Still frames of Individual 4.

Figure A.5: Still frames of Individual 5.

Figure A.6: Still frames of Individual 6.

Figure A.7: Still frames of Individual 7.

Figure A.8: Still frames of Individual 8.

Figure A.9: Still frames of Individual 9.

Figure A.10: Still frames of Individual 10.

B Network Design Results

The following appendix present additional results from the network design evaluation.

B.1 Effect of Sequence Length on Accuracy

This section contain the results from the investigation of RNN models and sequence lengths. Table B.1 show the multi-class accuracy for the six different models and six varying sequence lengths. These results are visualised in Fig. B.1 and Fig. B.2, where the second figure is cropped to the interval of $[0.9, 1.0]$ on the y-axis to show the difference in accuracy more easily.

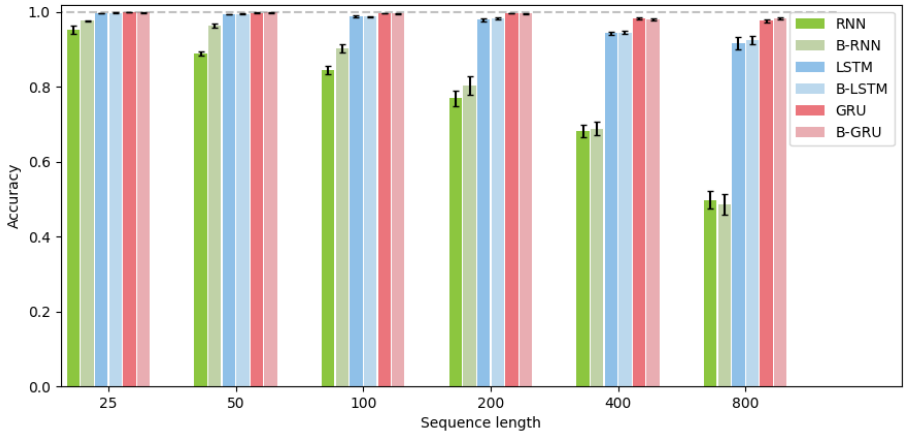


Figure B.1: Multi-class accuracy of the classification networks tested on the FOI Gait Dataset, split into different sequence lengths using all available samples. An increase of sequence length implies fewer dataset samples in total.

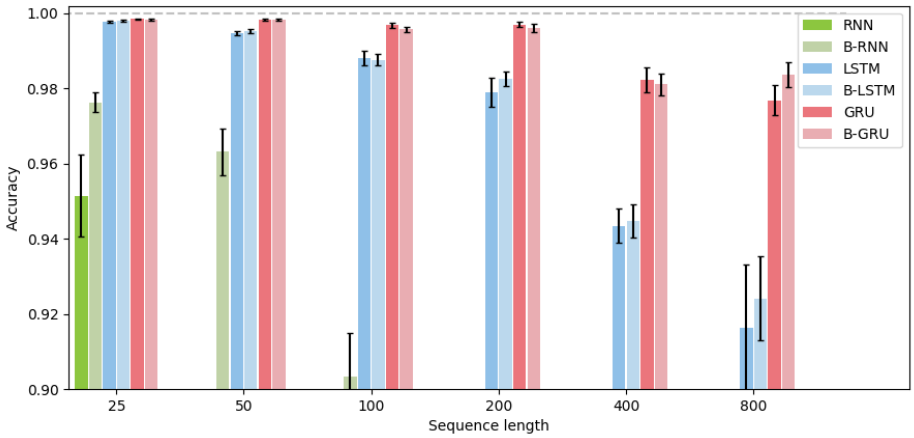


Figure B.2: Multi-class accuracy in the interval $[0.9, 1.0]$ of the classification networks tested on the FOI Gait Dataset, split into different sequence lengths with optimal number of samples. An increase of sequence length implies fewer dataset samples in total.

Table B.1: The multi-class accuracy (%) of different network designs and sequence lengths.

	Sequence lengths					
	25	**50**	**100**	**200**	**400**	**800**
RNN	95.14 ± 1.09	88.98 ± 0.53	84.51 ± 1.22	76.97 ± 2.04	68.23 ± 1.57	49.83 ± 2.39
B-RNN	97.63 ± 0.25	96.32 ± 0.62	90.33 ± 1.17	80.35 ± 2.59	68.80 ± 1.80	48.71 ± 2.71
LSTM	99.77 ± 0.03	99.47 ± 0.06	98.81 ± 0.20	97.90 ± 0.39	94.35 ± 0.46	91.63 ± 1.69
B-LSTM	99.80 ± 0.02	99.53 ± 0.06	98.76 ± 0.15	98.27 ± 0.19	94.49 ± 0.44	92.41 ± 1.12
GRU	**99.84 ± 0.02**	99.82 ± 0.03	**99.68 ± 0.07**	**99.71 ± 0.07**	**98.23 ± 0.32**	97.68 ± 0.39
B-GRU	99.83 ± 0.02	**99.83 ± 0.03**	99.57 ± 0.07	99.60 ± 0.11	98.11 ± 0.29	**98.37 ± 0.34**

B.2 Compensating for Training-set Sample Size

The dataset of the HPE walking gait representations were limited to the same amount of samples. The longest sequence length evaluated was 800 frames. This would mean that a 32 000 frame long walking session would result in 40 sequence samples. Normally, such a session would result in 1 280 samples, if sequences were 25 frames long. However, when compensating for training set sample size, the 25 frames long sequences were limited to the same amount of samples as the 800 frames long sequence, i.e. 40 sequence samples. The same principle was applied to the remaining sequence lengths The results are presented numerically in Table B.2 and visualised in Fig. B.3, for six different RNNs and six sequence lenghts. The results show that the GRU and its bidirectional version outperformed the other cells, for every tested sequence length.

Table B.2: The multi-class accuracy (%) when compensating for training set sample size.

	Sequence lengths					
	25	**50**	**100**	**200**	**400**	**800**
RNN	52.79 ± 1.16	55.30 ± 1.73	52.93 ± 2.20	51.31 ± 4.25	54.19 ± 4.00	49.83 ± 2.39
B-RNN	53.00 ± 3.93	55.57 ± 3.79	52.12 ± 4.09	50.17 ± 1.22	50.14 ± 2.95	48.71 ± 2.71
LSTM	74.28 ± 2.97	80.87 ± 3.08	86.33 ± 0.23	87.27 ± 1.86	87.97 ± 0.82	91.63 ± 1.69
B-LSTM	74.21 ± 1.61	81.01 ± 1.46	86.53 ± 2.25	87.88 ± 2.03	89.39 ± 1.47	92.41 ± 1.12
GRU	77.24 ± 2.59	87.32 ± 3.05	**92.46 ± 1.34**	94.75 ± 1.34	95.14 ± 1.05	97.68 ± 0.39
B-GRU	**80.81 ± 1.42**	**88.93 ± 0.93**	92.12 ± 0.48	**95.96 ± 0.51**	**96.01 ± 1.20**	**98.37 ± 0.34**

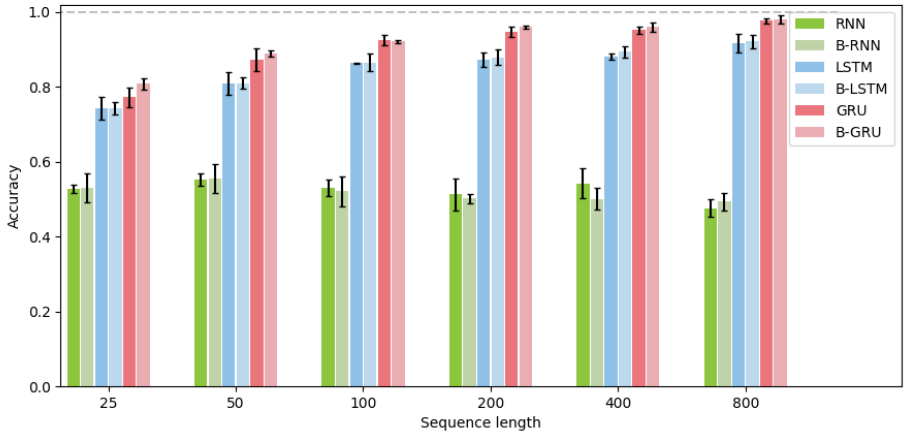


Figure B.3: Multi-class accuracy of the classification networks tested on the FOI Gait Dataset, split into different sequence lengths with equal number of samples. The number of dataset samples are equalised for the sequence lengths.

C Embedding Space Visualisations

In this appendix, two examples of scatter plots created by TensorBoard are presented. These visualise the 10 dimensional embedding spaces using PCA to plot the three dimensions representing the highest amount of variance. However these three dimensions are then projected to the two-dimensional screen, and the information is thus quite limited in how much variance can be represented through still images. Although the figures are limited, the general clustering structures can still be visualised.

The colours of TensorBoard plots are ambiguous, as they are not unique for each label and are re-used. Therefore, the numbering of each sample is more reliable. In the figures, the samples have numbers in the tens and ones positions. The tens position indicate the camera view, spanning from 0 to 4, representing cameras $a)$ to $e)$ respectively. The number in the ones position depicts the label of the individuals, however, the numbers are shifted by one in the plot, e.g., Individual 1 has the value of 0, Individual 2 has the value of 1, and Individual 10 has value 9.

The figures are of the case of training and evaluating on the same sessions, i.e., they correlate to 10-*dim* in Table 4.5, where Fig. C.1 is the result of using the Triplet loss, while Fig. C.2 is the result of the D-Triplet. Both the colouring and numbering can be difficult to distinguish in the figures. However, as mentioned, the structure can be described. As partially seen in Fig. C.1, ten clusters are formed. These mainly contain samples from one individual each, with all camera views of said individual in the same cluster. To the contrary, Fig. C.2, has five main clusters representing samples of the five camera views. The samples of the ten individuals then form sub-clusters in each of the camera clusters. The two original three-dimensional plots represent 67.4% and 65.6% of the variance, before being projected to the screen.

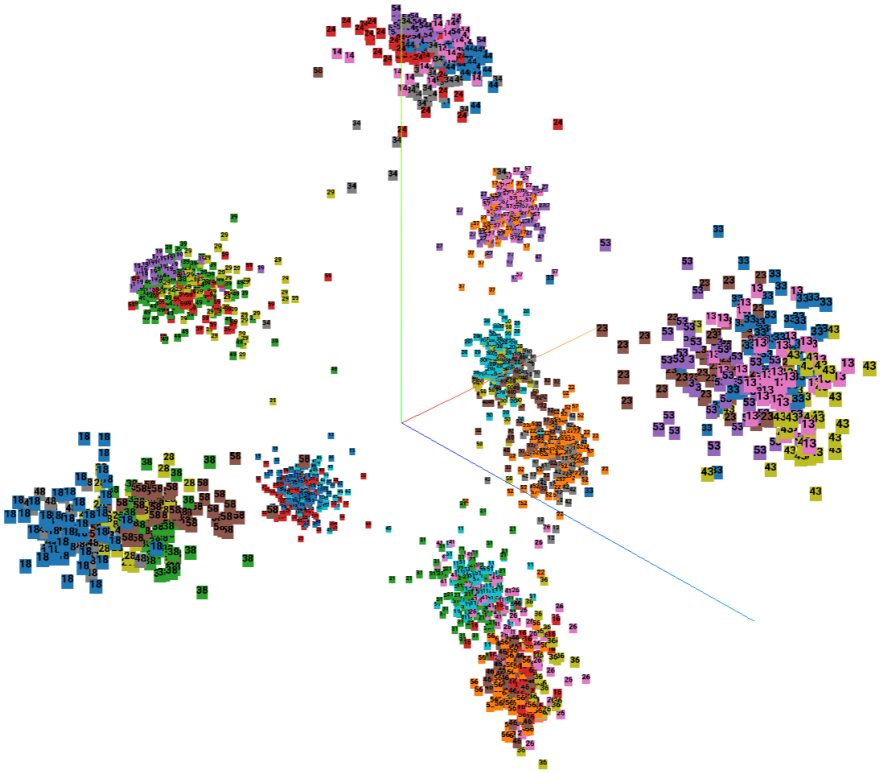


Figure C.1: A PCA visualisation of the embedding space using the Triplet loss.

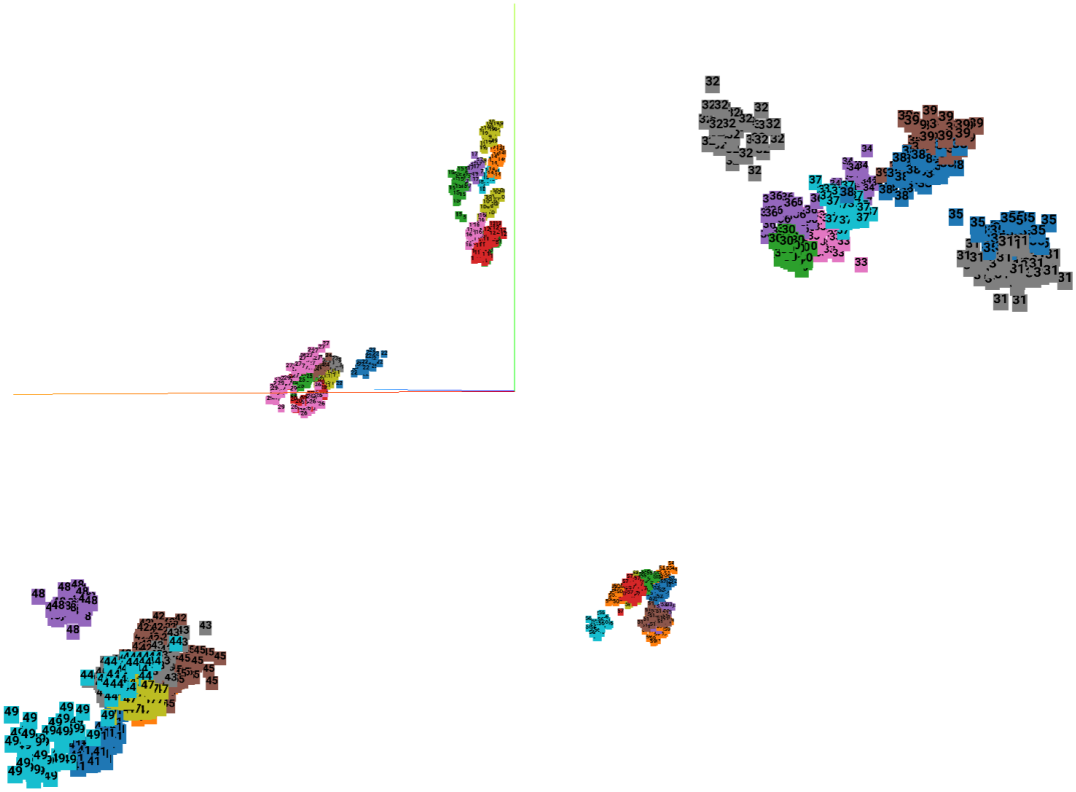


Figure C.2: A PCA visualisation of the embedding space using the D-Triplet loss.