Using Social Context for Person Re-Identification

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Liron Mizrahi
708810

Supervised by Dr Richard Klein

October 13, 2020

Ethics Clearance Number: H18/04/14

Acknowledgements

The support of the DST-NRF Centre of Excellence in Mathematical and Statistical Sciences (CoE-MaSS) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the CoE.

This work is based on the research supported in part by the National Research Foundation of South Africa (Grant Number: 118075) which provided computational, storage and video equipment for the project.

Abstract

Facial Recognition is currently a popular field in computer vision. It has many applications such as biometrics, Facebook's automatic photo tagging and automatic attendance monitoring. These systems all work in the same manner by creating a dataset of face images and training a deep learning model to recognise those faces. This is the standard way in which these systems are implemented and it is effective. However, there is usually more information in the domain besides just the facial features of subjects. For example, in Facebook's automatic photo tagging system, knowing who a subject of interest usually appears alongside in photos in addition to knowing the subject's visual appearance should provide more information to recognise them better.

To demonstrate this idea, a labelled dataset of students in lectures was created. With their informed consent, students were filmed in their lectures over a period of a few months. Because of the large number of students in the classes, an automatic labelling system using Augmented Reality Markers was developed. This allowed extracted face images to be quickly and easily labelled. In addition to the face images, the seat positions of the students were also captured. It has been shown that students tend to sit in preferred seats during lectures. We show that using the face images in conjunction with the student's seat position, the system is able to identify the student better. Several methods of injecting the seat information into the system were developed and analysed.

The facial recognition component was trained using a Convolutional Neural Network (CNN) with Triplet Loss and Quadruplet Loss. Triplet loss is a standard facial recognition framework. Quadruplet loss attempts to improve on triplet loss however we show that there are some issues with quadruplet loss. The results show that this approach is not worth the extra computation cost. A series of adaptive margin strategies for potentially improving these loss functions were developed.

The network creates low dimensional representations of the face images instead of performing the classification. Once these representations were generated, several classical classification techniques were used, such as Support Vector Machines, Logistic Regression, boosting methods and mixture models. These representations are forced to lie on the boundary of a hypersphere which makes the data curved. A mixture model which can fit to this spherical data was developed and if the data is curved enough, it outperforms standard mixture models.

Finally, we show that in the inclusion of the extra metadata can improve recognition accuracy. However, if the metadata is too consistent then simply classifying on the metadata alone would outperform classifying on the face images. The network trained with standard triplet loss performed the most consistently across several different classifiers.

Declaration

I, Liron Mizrahi, hereby declare the contents of this MSc dissertation to be my own work. This dissertation is submitted for the degree of Master of Science in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

Contents

List of Figures

Chapter 1

Introduction

Currently, facial recognition is a popular field in machine learning and is being implemented in a variety of domains. Facial recognition models are becoming increasingly accurate with Facebook's DeepFace being one of the most accurate. However, one similarity between most models is that they focus on modelling and recognising an individual without considering the social interactions between other people in the photo/video. There are cases where having prior knowledge of a scene can give better estimates as to what should appear in the scene. We show how some of these interactions can be modelled by exploring the domain of university students in lectures. There have been some studies which show how students tend to have certain predictable behaviours (Shernoff et al., 2017). One such behaviour is the seating positions of students. Anecdotally, students prefer to sit in select seating positions. This can be attributed to a range of factors such as familiarity, the lecturer's pitch, the size of the writing on the board, or simply because it was the first seat the student picked *etc.* These factors lead to students having consistent seating patterns which affect how other students choose seats. We show that by knowing these seating behaviours and combining them with a standard facial recognition system, we can achieve increased recognition accuracies as well as draw conclusions about the domain as a whole. To this effect we propose an end to end system to collect and process data, model the appearance and behaviours of the students and finally recover their identities.

This work is an extension to the Wits Intelligent Teaching System (WITS) (Klein and Celik, 2017). WITS is a smart lecture theatre which gives lecturers feedback on the efficacy of their lecturing style. It is a system designed to assist lecturers in the creation of better and more engaging lectures. Using various sensors such as video, thermal and infra-red cameras, and microphones, the system learns to use this data to recognise facial expressions, body posture and general behavioural engagement of students. After classes, the lecturer would also be able to get information about how the class reacted when different topics were covered. Alternatively, lecturers could use real time reports of the class to identify areas of students who are not engaging with the class or who seem distracted, and address those issues. The WITS system however does not take into account any temporal and spatial information between students. The proposed attendance recognition system is the first step to integrating temporal and spatial information into the WITS smart lecture room. A system like this would allow lecturers to take attendance of students in a non-invasive and automated manner. Besides automated attendance, this system would allow lecturers to track specific students' behaviours across lectures and understand their behaviours. This could be extended to detect

anomalies in students such as attending/participating differently from their usual behaviour.

Chapter 2 explains the background knowledge needed to understand this project. This includes the Inception series of convolutional neural networks typically used for modern vision tasks, the triplet loss framework for training the network as well as some classification techniques. Chapter 3 explores some prior work which relates to this project. Chapter 4 explains the data collection process and the data processing. Chapter 5 explains how the facial recognition component was trained. Chapter 6 explains how the seating behaviour is modelled. Chapter 7 discusses how to combine the facial recognition and the seating behaviour to make better predictions. Finally, chapter 8 discusses the results and observations of the project.

Chapter 2

Background Knowledge

2.1 Introduction

This chapter explains the techniques needed to understand the rest of the project. Facial recognition is a difficult task and so does not have an easy or exact solution. A neural network is trained to learn the faces of different students and identify them. This is achieved by using metric learning where Triplet loss is used to cluster similar looking faces while distancing dissimilar ones. By the end of training, the network should be able to learn good representations for faces which will create clear and well-defined cluster boundaries. The Inception V3 network is used for this work. This chapter discusses the history of the Inception networks and why V3 was chosen, a preprocessing technique that allows the training images to be upscaled to the same resolution while maintaining as much detail as possible, the Triplet loss framework for training as well as some classification techniques which are used once the network has been trained. The network will return learnt representations of images but will not perform the actual classification. Once the representations are obtained, methods such as Support Vector Machines, Logistic Regression and Nearest Neighbour are used for classification. More sophisticated methods such as Mixture Models and Boosting algorithms are used as well.

2.2 Inception V3 Network

Convolutional networks are the basis for most modern computer vision tasks. Deep convolutional networks became the standard for many benchmarks. Generally, the deeper the networks the better the performance. However, this also leads to a very high parameter count which increases training time and memory usage significantly. Before the introduction of the Inception networks, most convolutional networks stacked many convolutions followed by dense layers. This leads to high parameter count. The Inception networks attempt to reduce the parameter count while maintaining high accuracies. The Inception v1 (GoogLeNet) (Szegedy, Liu, et al., 2015) focused on 2 main problems. The first is deciding the right kernel sizes and the second is the issue of vanishing gradients. Larger kernels are better at locating bigger, global features while smaller kernels are better at locating smaller, local features. If objects of interest in the data have high location and size variance, then choosing a good kernel size is difficult. Their solution was to add kernels of

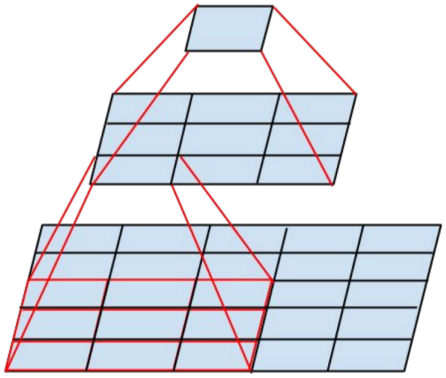


Figure 2.1: Using smaller kernels to replace a large kernel while maintaining the size of the receptive field (Szegedy, Vanhoucke, et al., 2016). For example, a 5×5 kernel can be replaced with 2 3×3 kernels. This reduces the parameter count and speeds up training.

different sizes in the same layer, called an *Inception module*. There is a 1×1, 3×3 and 5×5 kernel applied to the input as well as a 3×3 max pooling. The outputs of each kernel and the max pooling are concatenated and passed to the next layer. By doing this, the network learns which filter sizes are the most useful for any given input. The Inception module is further improved by reducing the amount of channels in the input before the kernels. This is done by performing a 1×1 convolution before the 3×3 and 5×5 kernels. In the case of the max pooling, the 1×1 is done afterwards. GoogLeNet has 9 Inception modules stacked sequentially and is 22 layers deep. As such it suffers from vanishing gradients. To help with this issue, 2 *auxiliary classifiers* are introduced in the middle of the network. The final loss that gets propagated is a weighted combination of the auxiliary losses and the actual loss of the network. This will increase the gradient signal that gets propagated. During inference, the auxiliary classifiers are not used.

The next iteration of the network, Inception v2 (Szegedy, Vanhoucke, et al., 2016), attempts to improve upon the previous version in 2 main ways. The first is reducing the amount of information loss from dimensionality reduction and the second is improving computational complexity. Computational complexity of the convolutions is improved by decomposing larger kernels into multiple smaller ones to have the same receptive field. For example, Fig 2.1 shows how a 5×5 kernel can be decomposed into 2 3×3 kernels. A 5×5 kernel has 25 parameters whereas 2 3×3 kernels have 18 parameters. Furthermore, it can be shown that any $n \times n$ kernel can be decomposed into an $n \times 1$ kernel followed by a $1 \times n$ kernel to have the same receptive field as an $n \times n$ kernel. A 3×3 kernel has 9 parameters and a 3×1 kernel followed by a 1×3 kernel has 6 parameters. When both types of decomposition are done for every large kernel in the network, the parameter count decreases significantly. Reducing the amount of information loss is done by making the modules wider instead of deeper. More kernels of different sizes are added in the same layer. If the modules were made deeper then the amount of dimensionality reduction would increase. It was found that it is important to balance both the width and the depth of the network. By combining more kernels with the convolution decomposition the Inception v2 network was faster to train and in most cases was more accurate.

The Inception v3 network was introduced in the same paper as Inception v2 (Szegedy, Van-

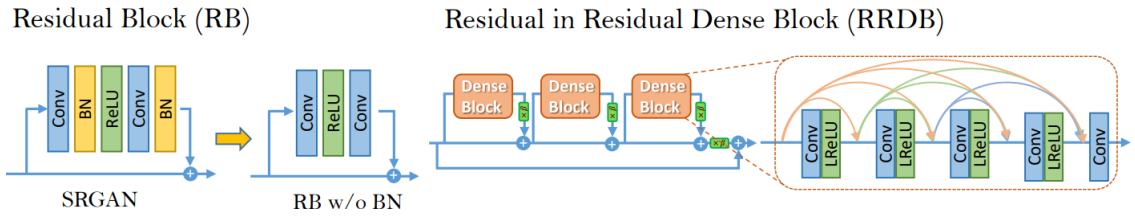


Figure 2.2: Residual Blocks in SRGAN compared to the Residual in Residual Dense Blocks in ESRGAN(Xintao Wang et al., 2018). The Batch Normalisation (BN) layers in the original Residual Blocks were removed and more residual connections are added.

houcke, et al., 2016). It uses all the above mentioned ideas as well as a few more. It was found that the auxiliary classifiers did not really contribute much to training until the end and act more as regularisers. To this end, batch normalisation layers were added to the auxiliary classifiers. An additional mechanism is introduced to regularise the classification layer. When the softmax of the logits (unnormalised outputs of the network) is taken, usually the largest logit becomes much larger than the other. This forces the network to be overly confident about the class. Even if the network assigns full probability to the correct class label, it may fail to generalise well. Instead of comparing the softmax probabilities to a one-hot encoded label, a soft label assignment can be used. For example, the correct class can be labelled 0.9 instead of 1 and the other classes can be labelled higher values besides 0. This is called *Label Smoothing Regularisation* (LSR). In conclusion, the Inception v3 network is faster and more accurate than its predecessors and so was chosen for this project.

2.3 Enhanced Super Resolution Generative Adversarial Network

The images that are passed into the Inception v3 network need to be the same size and have a resolution of at least 299×299. The images from the collected data have varying resolutions and usually less than 299×299. The images all need to be upscaled to the same resolution. This can easily be done by upscaling the images using bilinear or bicubic interpolation. Interpolating images in this manner often creates artefacts. Instead of interpolating to upscale images, a neural network can learn to upscale while reducing the amount of artefacts and preserving as much detail as possible. This was first done by the Super Resolution CNN (SRCNN) *(Dong et al., 2015)*. It was a 3 layer convolutional neural network that was given a downscaled image as input and the original image as the target. The network was made to optimise the Peak Signal-to-Noise Ratio (PSNR). A major issue with networks that optimise PSNR is that they tend to create very smoothed images with a lack of detail. GANs were introduced to super resolution in order to try to create more natural looking images. One of the most prominent GAN methods is the Super Resolution Generative Adversarial Network (SRGAN). It achieves more natural images by optimising a different loss function and using many residual connections. The loss function is a perceptual loss (Johnson, Alahi, and Fei-Fei, 2016) meaning that instead of basing the loss on the pixel values, it looks at the difference between extracted features of the final image. Using these 2 techniques SRGAN produces clearer and more detailed images especially when compared to PSNR based methods.

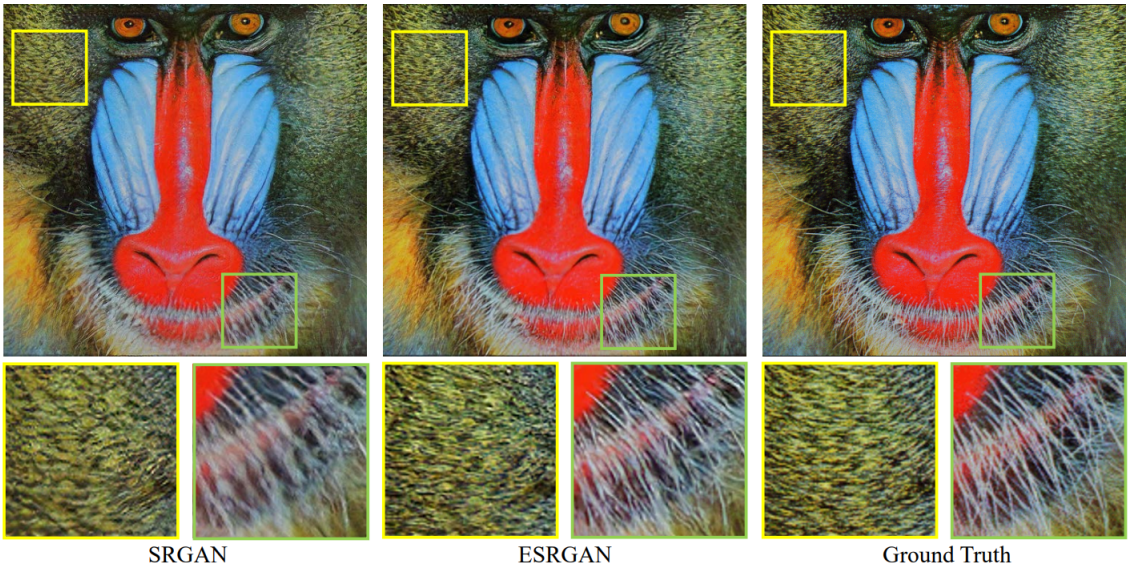


Figure 2.3: Learnt upscaling of SRGAN compared to ESRGAN and the ground truth (Xintao Wang et al., 2018). SRGAN produces good images however it fails to create clear high frequency details such as hair. ESRGAN produces clearer high frequency details.

Enhanced Super Resolution Generative Adversarial Network (ESRGAN) attempts to improve upon SRGAN in 2 major ways. The first improvement focuses on the generator network architecture. SRGAN is made up of residual blocks. ESRGAN removes the batch normalisation layers and adds more residual connections in the block to create a Residual in Residual Dense Block (RRDB) as shown in Fig 2.2. The residual blocks in the original network are replaced with RRDBs. It was found that removing the batch normalisation layers reduced the complexity and allowed the network to generalise better. The extra residual connections were added to increase the capacity of each block. The next major improvement focuses on the discriminator. The discriminator in SRGAN produces a probability measure of whether the generated image is real. The discriminator in ESRGAN is modified to produce a probability measure of whether a given image looks more realistic than another. In doing this, the generator is able to learn from the gradients of both the generated images and the real images. ESRGAN produces clearer and more detailed images as seen in Fig 2.3.

2.4 Facial Recognition

The neural network architecture has been explained and by using ESRGAN the images can all be scaled to the same resolution while maintaining as much detail as possible. Now the facial recognition component can be learnt. Facial recognition is the field of identifying faces in an image or video. Facial recognition tries to find the identity of a face given many faces from other classes, unlike facial verification where one face is compared with another to see if it is the same face, *e.g.* Apple's FaceId. There are 2 major steps involved in this process, face detection followed by face

recognition. Face detection is the process of locating any face in a given video or image. A labelled dataset of all located faces is then created. Once the faces have been found then a recognition model can be trained to recognise those faces.

Originally, the detection step was done by manually selecting landmark points on faces, such as the nose, eyes, edges of the lips *etc.* This method was slow and expensive. Sirovich and Kirby then improved on this by using Principal Component Analysis to automatically extract facial features in a method called Eigenfaces (Sirovich and Kirby, 1987). Turk and Pentland then showed how Eigenfaces can be used for automatic face detection (Turk and Pentland, 1991). In 2001, the detection step was fully automated and ran in real time when the Viola Jones face detection algorithm (Viola and Jones, 2001) was created. With the detection step becoming fast and automated, there was a big focus on deep learning approaches to the recognition step. The obvious approach was to set up a deep network and the output layer would contain as many nodes as there were classes. The final layer would be a distribution over which class the face should belong to and the class with the highest probability would be chosen. An immediate problem with this approach arises when another class is added to the dataset. The entire network must be retrained because the final layer needs an extra output. Another issue arises where 2 unknown faces need to be compared with each other to check if they are from the same class. A solution to these problems is to instead learn a feature representation of each face. Triplet loss (Schroff, Kalenichenko, and Philbin, 2015) and quadruplet loss (W. Chen et al., 2017) (discussed in the next sections) are methods which learn an embedding space. The learned feature representations (called embeddings) are projected into this space where the distance between 2 embeddings is proportional to the dissimilarity between them. The further 2 embeddings are, the more different they are. Conversely, the closer 2 embeddings are, the more similar they are.

2.5 Triplet Loss

2.5.1 Defining Triplet Loss

As mentioned previously, triplet loss allows the network to learn an embedding space where the distance between points relates to some property of the images. In the context of facial recognition, the network should cluster images of the same identity while pushing images of different identities further away. Triplet loss enforces this idea with a triplet of images where 2 images are of the same identity and another image of a different identity. The network produces the embeddings for each image and triplet loss penalises the network if the distance between the 2 images of the same identity is larger than the distance to the image of the different identity.

This idea is formally shown below using the same terminology of Schroff *et al.* (2015):

- An anchor is a reference image,
- A positive image is an image which is of the **same** subject as the anchor,
- A negative image is an image which is of a **different** subject to the anchor.

The objective of the loss can be stated as trying to minimise the distance between the anchor and positive, while maximising the distance between the anchor and the negative. Formally this is shown as:

$$\|f(A) - f(P)\|^2 \leq \|f(A) - f(N)\|^2 \tag{2.5.1}$$

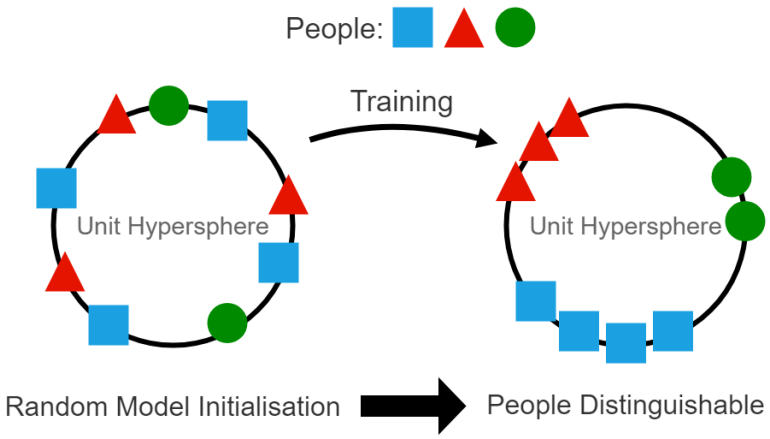


Figure 2.4: The objective of triplet loss (Amos, Ludwiczuk, Satyanarayanan, et al., 2016). Initially the model embeds the images randomly (*left*). As training occurs, the model learns to embed similar subjects close to each other while distancing different subjects (*right*).

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 \leq 0 \tag{2.5.2}$$

where:

- If X is an image then $f(X)$ is its embedding,
- A is the anchor,
- P is the positive,
- N is the negative,
- $\|\cdot\|$ is the ℓ^2-norm.

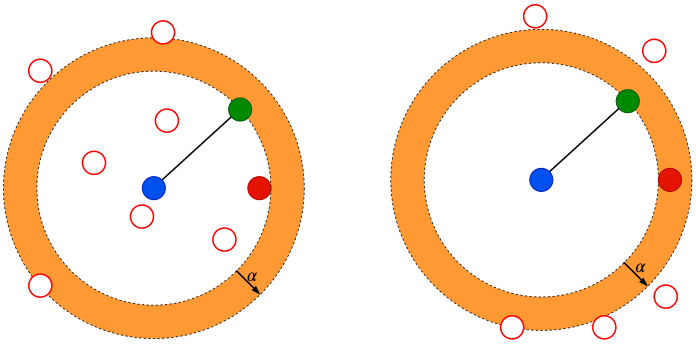
An important note is that the embedding is constrained to a unit hypersphere, i.e. $\|f(X)\| = 1$. This way the maximum distance between embeddings is always known. Figure 2.4 shows the overall objective of triplet loss.

The issue with this definition is that the embeddings can learn the trivial solution where $\|f(A) - f(P)\|^2 = 0$ and $\|f(A) - f(N)\|^2 = 0$. To prevent this from happening, an extra constraint must be enforced. A margin, α, is introduced. This margin forces the negative to be at least α more units away from the anchor than the positive is. The loss now becomes:

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha \leq 0 \tag{2.5.3}$$

If Eqn 2.5.3 holds then the distance between the anchor and positive is smaller than the distance between the anchor and the negative. Therefore the embedding is correct and there should be no contribution to the loss. Conversely, if Eqn 2.5.3 does not hold then the embedding is wrong and this becomes the loss. The loss can now be rewritten as:

$$max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0) \tag{2.5.4}$$



(a) There are negatives closer to the anchor and so the **easiest** negative is chosen.

(b) There are **no** negatives closer to the anchor and so the **hardest** negative is chosen. In this case the negative will usually lie within the margin (orange).

Figure 2.5: Illustration of semi-hard negative mining. For a given anchor (blue) and positive (green), the semi-hard negative is chosen (solid red) from all the other negatives (red rings)

The only case where the loss is negative is where the negative is further than the positive to the anchor. The maximum is taken because a negative loss means the network is training correctly. Considering all the triplets, the cost function is then finally defined as:

$$L_{trip} = \sum_{i=1}^{T} \left[\|f(A_i) - f(P_i)\|^2 - \|f(A_i) - f(N_i)\|^2 + \alpha \right]_+ \tag{2.5.5}$$

where:

- T is the number of all possible triplets,
- $[Z]_+ = max(Z, 0)$.

2.5.2 Triplet Selection

The biggest challenge with triplet loss is choosing the triplets to learn from. A large number of triplets would easily satisfy the loss constraint and would not contribute to the loss, therefore leading to slow convergence. It would be preferable to train on difficult examples. So A_i, P_i should be chosen such that they satisfy $\arg\max_{P_i} \|f(A_i) - f(P_i)\|^2$, and similarly for N_i and $\text{argmin}_{N_i} \|f(A_i) - f(N_i)\|^2$. This triplet selection scheme is called *hard-negative mining*. These triplets contain the hardest training examples because P_i is the furthest positive and N_i is the closest negative for a given A_i.

However, if the training examples are all too difficult then the network may reach bad local minima or may even collapse, *i.e.* $f(X) = 0$. The examples should be easy enough that the model does not collapse but still hard enough so that the model learns. Instead N_i can be chosen such

that if there are negatives closer to the anchor than the positive then the easiest negative is chosen. Similarly, if there are no negatives closer to the anchor than the positive then the hardest negatives are chosen. In the former case only 1 negative is chosen. In the latter case, several negatives can be chosen and these negatives will lie within the margin. This scheme is called *semi-hard negative mining*, shown in Fig 2.5.

In practice, images are passed into the neural network in batches and so the triplet selection scheme must be adapted for batches. There are 2 major batch triplet selection schemes (Hermans, Beyer, and Leibe, 2017), *batch-all* and *batch-hard*. In batch-all, for each anchor the semi-hard negatives in the batch are chosen. In batch-hard, the hardest negatives in the batch for each anchor are chosen. The choice of triplet selection is important for fast convergence. The batch-hard strategy has shown to be more effective for most cases, however the choice of scheme is still data dependent. Because of this, both schemes will be tested on the student data.

2.6 Quadruplet Loss

The quadruplet loss has the same aim as triplet loss which is to create an embedding space where similar subjects are clustered together and dissimilar ones are further apart. The triplet loss achieves this by taking a triplet of images and forcing the negative to be further from the anchor than the positive. However, it does not take into account the distance of the negative to other negatives. The quadruplet loss (W. Chen et al., 2017) is an extension to the triplet loss which takes into account the other negative classes. The objective of quadruplet loss is the same as that of triplet loss however it tries to achieve a lower intra-class variation and higher inter-class variation. Like triplet loss, quadruplet loss also enforces that the negative be further than the positive from the anchor. It also enforces that any 2 negatives from different classes are further from each other than the distance between the anchor and positive. It extends triplet loss by training on an extra negative image, *i.e.* it trains on 4-tuples $(A, P, N^{(1)}, N^{(2)})$, hence the name. Note that the 2 negatives are from different classes. Formally this is written as:

$$\begin{aligned} L_{quad} = &\sum_{i}^{T} \left[\|f(A_i) - f(P_i)\|^2 - \left\|f(A_i) - f(N_i^{(1)})\right\|^2 + \alpha_1 \right]_+ \\ &+ \sum_{i}^{T} \left[\|f(A_i) - f(P_i)\|^2 - \left\|f(N_i^{(2)}) - f(N_i^{(1)})\right\|^2 + \alpha_2 \right]_+ \\ &\text{with } S_A = S_P; S_{N^{(2)}} \neq S_{N^{(1)}}; S_A \neq S_{N^{(2)}}; S_A \neq S_{N^{(1)}} \end{aligned} \tag{2.6.1}$$

where:

- S_X is the class of image X,
- α_1 and α_2 are margins.

The first term is the triplet loss constraint, the second term refers to the distances between positive and negative pairs with respect to different input images. The addition of the second term ensures that the maximum intra-class distance must be smaller than the minimum inter-class distance, *i.e.* clusters are further apart than they are wide. When setting the margins, α_1 should be higher than α_2 to enforce a larger distance between the (A, P) and (A, N) pairs.

Additionally, the quadruplet loss offers 2 other improvements. The first relates to the distance metric. Standard Euclidean distance may not be able to capture the intricate relationships between embeddings. To this end, a learnt distance metric is proposed.

The second improvement concerns the choice of the margins α_1 and α_2. The choice of the margins is not obvious and is usually chosen empirically. A bad choice can lead to the model not learning correctly. Instead of choosing the margins manually, they can be set adaptively according to how well the model is performing. It is assumed that the distances between features of the same class and features of different classes are from 2 distributions, i.e. positive pair distance distribution and negative pair distance distribution. If these distributions are similar then the distances between any pair of embeddings is similar and a model would struggle to separate classes. On the other hand, if they are different then the model learns to cluster classes together. The average negative pair distance should be larger than the average positive pair distance. These distributions can be used to create an adaptive margin:

$$\alpha = w(\mu_N - \mu_p) \tag{2.6.2}$$

$$\alpha = w\left(\frac{1}{M_N}\sum_i g(N_i^{(2)}, N_i^{(1)}) - \frac{1}{M_P}\sum_i g(A_i, P_i)\right) \tag{2.6.3}$$

where:

- μ_N and μ_P are the means of the negative pair and positive pair distributions respectively,
- w is an importance coefficient,
- M_N and M_P are the number of negative and positive pairs respectively.

α in the above equations refers to either α_1 or α_2. α_1 will have a larger value for w than α_2. In the original paper, $w = 1.0$ and $w = 0.5$ for α_1 and α_2 respectively. This helps to satisfy the constraint that α_1 must be greater than α_2. In practice finding the average distance between the two distributions for each iteration is expensive. This would involve finding the average embedding for the images in each distribution for each iteration. For this reason the mean of each distribution in a batch is used instead.

Two different methods have been outlined to learn embedding spaces. Some of the issues with these approaches as well as some possible solutions have been discussed. Now that images can be transformed into an embedding space, the next task is to classify these embeddings to perform recognition. The next sections will discuss classification using several classical methods as well as introduce some variations of mixture models.

2.7 Classifiers

The network produces embeddings of the given images. The network does not perform the classification step. An external classifier is used instead. Nearest neighbour is usually used as the final classifier, because triplet and quadruplet loss were designed to optimise nearest neighbour classification. However, other classifiers are explored as well. The following will list some of the more common classifiers used and give a brief description of them. All of the following were implemented with scikit-learn (Pedregosa et al., 2011).

Nearest Neighbour is a simple classifier that takes in some points as training data. A new point is classified as the same class as the closest point to it from the training points. This method lends itself naturally to the embeddings as each individual should be separated into clearly separated clusters. The closest point to an embedding should be a point with similar features, *i.e.* an embedding of the same identity as the new point.

Naive Bayes is a classifier which relies on Bayes Theorem. Bayes Theorem gives a probability of an event occurring given other events that have occurred. Naive Bayes makes the assumption that all features are independent of each other as this makes computation easier. Bayes Theorem gives the following relationship for calculating the probability of class y given features $x_1, \ldots, x_n$:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(x_1, \ldots, x_n \mid y)P(y)}{P(x_1, \ldots, x_n)} \tag{2.7.1}$$

Using the assumption of independence $P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y)$, the relationship can be rewritten as:

$$P(y \mid x_1, \ldots, x_n) = \frac{\prod_{i=1}^{n} P(x_i \mid y)P(y)}{P(x_1, \ldots, x_n)} \tag{2.7.2}$$

The label for a new point, $\hat{y}$, is then found by:

$$\hat{y} = \arg\max_y \prod_{i=1}^{n} P(x_i \mid y)P(y) \tag{2.7.3}$$

Decision Trees partition the input space so that points from the same class are grouped together. The result is a tree of decisions which predict the class of a new point. The initial split on the data is chosen by looking at all the features and seeing which feature can explain the data the best based on some measure. The next split is then chosen by looking at the data that has not been explained yet and repeating the process until a series of splitting rules is created. Decision trees in scikit-learn are constructed using an algorithm called CART (Classification and Regression Trees). The measure CART uses for deciding which feature to create the split on is called the Gini Impurity. For a set consisting of C classes it is defined as:

$$I_G(p) = 1 - \sum_{c=1}^{C} {p_c}^2 \tag{2.7.4}$$

where p_c is the fraction of class c in the set. Gini Impurity measures the likelihood of a new point being labelled incorrectly given the observed data. First the Gini Impurity is measured for all features in the current split. The data is then split according to the feature that minimises the impurity. This process is then performed recursively on the now split data until a maximum tree depth is reached.

Decision trees are good at explaining data and are easily interpretable. However, they tend to overfit very quickly and often don't generalise very well.

Random Forests are ensembles of decision trees. They attempt to reduce the overfitting of a single decision tree by constructing several trees and averaging the predictions. Each individual tree is built using random samples in a bootstrapping fashion. For each node in the tree only a random subset of features are used to split the data, typically the square root of the number of features.

This process is repeated until a set number of trees are created. In order to classify a new point, the point is run through every individual tree and the final prediction is chosen by majority voting. The idea is to create several trees that explain subsets of data well and combine the predictions of each tree to make a more informed decision while reducing any biases.

Extreme Gradient Boosting (XGBoost) is very similar to a random forest in that they both construct an ensemble of trees and combine the decisions of each tree to make a prediction. The difference lies in the way they are trained. XGBoost uses a method called gradient boosting where an ensemble of weak classifiers is learnt. Typically the weak classifiers are CART decision trees with fixed sizes. Initially, a small decision tree is trained on the data. Each subsequent tree will be trained to predict the errors of the previous tree, also known as the *residuals*. If the The tree is then scaled down by some constant value. New trees are added sequentially until either the residuals become very small or a set number of trees has been constructed. By training on the residuals instead of the actual data, each tree guides the predictions in the correct direction, hence gradient boosting. To make a prediction for a new point, it is run through every tree and the output of each leaf is added together. XGBoost was designed for large complex datasets.

Logistic Regression is a classification method which assumes that the decision boundary is linear. It fits straight lines for every label while minimising the cross entropy loss of the ground truth and the predicted labels. It is a simple classifier and easily interpretable. In the binary case, for features $x_1, \ldots, x_n$ and weights $w_0, \ldots, w_n$ the model is defined as:

$$z = w_0 + \sum_{i=1}^{n} w_i x_i \tag{2.7.5}$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.7.6}$$

where $\hat{y}$ is the predicted class. The weights are learnt by gradient descent by minimising the cross entropy loss which is given by:

$$J(z) = -\frac{1}{m} \sum_{i=1}^{m} y_i \log \sigma(z) + (1 - y_i) \log(1 - \sigma(z)) \tag{2.7.7}$$

where m is the number of training samples and y_i is the class label for sample $\mathbf{x}_i$. This can be extended to a multiclass setting using a one-vs-all strategy, where a binary classifier is trained for each class label. The predicted class is then chosen from the classifier which returns the highest probability.

A **Support Vector Machine (SVM)** attempts to solve the same problem as logistic regression. It tries to fit a set of linear boundaries to make predictions. The main difference is that SVMs try to maximise the distance between the boundary and the nearest data point of every class. By doing this, the generalisation error should decrease.

In the binary case with a training set of points $(\mathbf{x}_i, y_i)$, where y_i is either 1 or -1, the goal is to find the hyperplane which maximises the distance between itself and the nearest point from each class. The equation of this hyperplane is given by $\mathbf{w} \cdot \mathbf{x} - b = 0$ where $\mathbf{w}$ is the normal vector to the hyperplane and b is the offset. If the data is linearly separable then two parallel hyperplanes can be defined which separate the two classes while also being as far apart as possible. These two hyperplanes are defined as $\mathbf{w} \cdot \mathbf{x} - b = 1$ where anything above this plane is from class 1, and $\mathbf{w} \cdot \mathbf{x} - b = -1$ where anything below this plane is from class -1. The distance between these two

planes is given by $\frac{2}{\|\mathbf{w}\|}$. Therefore to maximise the distance between the hyperplanes, $\|\mathbf{w}\|$ must be minimised. Additionally, training points cannot be in between the planes so two constraints must be placed, $\mathbf{w} \cdot \mathbf{x}_i - b \geq 1$ if $y_i = 1$ or $\mathbf{w} \cdot \mathbf{x}_i - b \leq -1$ if $y_i = -1$. More simply this can be rewritten as $y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 \quad \forall i \in [1, n]$. The optimization problem can now be defined as:

$$\text{Minimize } \|\mathbf{w}\| \text{ with respect to } y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 \quad \forall i \in [1, n]$$

Once $\mathbf{w}$ and b are found then classification can be performed by $\hat{y_i} = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i - b)$. Extending this to a multiclass setting is similar to logistic regression where a one-vs-all strategy is used.

The embeddings should be clustered well and so an SVM should be able to construct very clear decision boundaries. Only a linear SVM is used.

Additionally, a small **neural network** (NN) was also used as a classifier. It has 3 hidden layers with 128, 128 and 64 nodes respectively.

These classifiers have all shown to be effective for different types of data. However, the clustered nature of the data can be exploited. Mixture models will give a probabilistic way of modelling the clusters. This type of data is a natural fit for mixture models. Additionally, the embeddings produced by the network lie on the boundary of a hypersphere and so the curvature of the data should be accounted for. The next sections will discuss mixture models and 2 variations which should be able to model the data more effectively.

2.8 Gaussian Mixture Models

The embeddings created by triplet and quadruplet loss will form clusters. Specifically, embeddings of the same person will be clustered near each other. New embeddings are usually classified by a nearest neighbour comparison however, it would be more useful to have a way of classifying new embeddings given these clusters of embeddings. The easiest method would be to use k-means clustering however k-means assumes spherical covariance which is not the case for the embeddings. Gaussian mixture models (GMMs) provide a probabilistic way of clustering and classifying embeddings while not assuming spherical covariance.

GMMs are comprised of several Gaussian distributions which update their parameters in an unsupervised way to fit the data, similar to k-means clustering. Each Gaussian distribution has its own set of parameters, i.e. mean μ and covariance Σ. The optimal parameters are found using the Expectation Maximisation algorithm (EM). The EM algorithm has 2 steps, the E-step and the M-step. The 2 steps can be related to k-means. The E-step is the same as assigning points to clusters. But where k-means has a hard assignment, EM uses a soft, probabilistic assignment of points. The M-step updates the cluster centres based on the assignment of the points. However, the M-step also updates the covariances of the distributions. Before each step is explained, there are some terms that need to be introduced. Assuming there are K distributions each with their own set of parameters θ, then the probability of data point x being generated by distribution k is:

$$P(x|k, \theta) = \text{Norm}_x \left[\mu_k, \Sigma_k\right] \tag{2.8.1}$$

The probability of choosing cluster k is described by a categorical distribution given by:

$$P(k|\theta) = \text{Cat}_k \left[\lambda_1 \dots \lambda_K\right] \tag{2.8.2}$$

Given the definitions above, the 2 steps can be derived (Prince, 2012).

The expectation step (E-step) determines which points belong to which Gaussian distribution. For each data point x_i calculate the probability of it belonging to each Gaussian distribution. The probability of distribution k being chosen for a data point x_i using parameters θ is given by:

$$\begin{aligned} P(k|x_i,\theta) &= \frac{P(k,x_i,\theta)}{P(x_i,\theta)} \\ &= \frac{P(x_i|k,\theta)P(k,\theta)}{P(x_i,\theta)} \\ &= \frac{\lambda_k \text{Norm}_{x_i}\left[\mu_k,\Sigma_k\right]}{\sum_{j=1}^{K}\lambda_j \text{Norm}_{x_i}\left[\mu_j,\sigma_j\right]} \\ &= r_{ik} \end{aligned}$$

r_{ik} is the responsibility Gaussian k has for data point x_i.

The maximisation step (M-step) uses the assignment generated from the expectation step to update the model parameters. The parameters are updated as:

$$\hat{\theta} == \arg\max_{\theta}\left[\sum_{i=1}^{I}\sum_{k=1}^{K} r_{ik}\log\left[\lambda_k \text{Norm}_{x_i}\left[\mu_k,\Sigma_k\right]\right]\right] \tag{2.8.3}$$

The update rules for the parameters are found by finding the parameters that maximise the probability of seeing the responsibilities calculated in the E-step. This is done by differentiating Eq 2.8.3 with respect to the different parameters and equating to zero. The maximum of the log is taken as the maximum will still be the same and it makes the differentiating easier. There is still the constraint of $\sum_k^K \lambda_k = 1$ so a Lagrange multiplier must be added. The update rules become the following:

$$\begin{aligned} \lambda_k^{new} &= \frac{\sum_{i=1}^{I} r_{ik}}{\sum_{j=1}^{K}\sum_{i=1}^{I} r_{ij}} \\ \mu_k^{new} &= \left(\frac{1}{\sum_{i=1}^{I} r_{ik}}\right)\sum_{i=1}^{I} r_{ik}x_{ik} \\ \Sigma_k^{new} &= \left(\frac{1}{\sum_{i=1}^{I} r_{ik}}\right)\sum_{i=1}^{I} r_{ik}\left(x_{ik}-\mu_k^{new}\right)\left(x_{ik}-\mu_k^{new}\right)^T \end{aligned}$$

2.9 Bayesian Non-parametric Mixture Models

Gaussian mixture models are good for modelling clustered data. However, in most cases the number of clusters is completely unknown. One method to find the number of clusters is to fit several GMMs with different numbers of clusters and choose the mixture that describes the data the best. This method is not efficient and not guaranteed to give a good solution. Bayesian non-parametric Gaussian mixture models (BN-GMMs) give a principled and probabilistic way of choosing the correct number of clusters. The model can freely add or remove clusters as needed without specifying

a set number of clusters. The basis of BN-GMMs is a *Dirichlet Process* (DP), which provides distributions over the number of clusters, k. DPs have 2 parameters, α and G. α is called the *concentration parameter* and controls how spread out the draws are from the DP. G is called the *base distribution* and controls the shape of the draws from the DP. DPs take as input a distribution and output a new modified distribution based off of the input while α controls how similar the output should be to the input.

A common analogy for DPs is the *Chinese Restaurant Process* (CRP). The CRP is the algorithm for Bayesian Non-parametric clustering. This process follows the analogy of a Chinese restaurant where there is always space to fit new customers. When a customer enters they can choose to sit at an existing table with other customers, or they can sit at a new table. The probability of sitting at an existing table is proportional to the number of customers sitting at that table. This metaphor is performing clustering where the customers are the data points and the tables are the clusters. For a new customer, the probability of sitting at any table is proportional to $\frac{n_k}{n+\alpha}$, where n_k is the number of customers at table k and n is the total number of customers. The probability of sitting at a new table is proportional to $\frac{\alpha}{n+\alpha}$. The CRP gives a way of calculating whether new clusters should be added to mixture. However, adding more clusters would clearly fit the data better to the point where each datapoint would be in its own cluster. To counteract this, the CRP biases bigger clusters over many clusters. This way, a new data point would most likely join another cluster rather than creating its own new cluster. The BN-GMM was implemented using the bnpy (Bayesian nonparametric machine learning for python) library (Hughes and Sudderth, 2014).

2.10 Hypersphere Mixture Models

Mixture Models give a probabilistic way of clustering and classifying embeddings. A big drawback of GMMs and BN-GMMs is that they do not take into account the curvature of the emmbeddings. Recall from section 2.5 that the embeddings lie on the boundary of a unit hypersphere. Because of this, the data does not lie on a flat plane but instead curves around a unit hypersphere, *i.e.* for any embedding $\mathbf{x}$, $\|\mathbf{x}\| = 1$. When the GMM is fit, the means of each distribution will be inside the sphere instead of the mean of the data shown in Fig 2.6. This causes the covariance to grow larger in order to compensate for the shifted mean. Even if the mean of the distribution becomes the mean of the data, the covariance will still need to be large to cover all the points. This does not pose much of an issue at lower dimensions but has a significant impact at higher dimensions, which will be shown later.

Because of these issues, a Gaussian Mixture Model which can deal with the curvature was created. This Hypersphere Gaussian Mixture Model (HGMM) attempts to unravel the sphere so that it becomes flat. There is no perfect way of doing this unravelling because any flattening of a sphere will cause some points to be further apart than they originally were on the sphere. Because this is a supervised clustering method, the sphere will be unravelled at an appropriate point for each class. For each class, the unravelling point is its mean on the boundary of the hypersphere. Once the mean is found, the points are projected onto a flat hyperplane that is tangent to the hypersphere at the mean, called a *Tangent Space*. Now that the points lie on a plane, a Gaussian distribution can be fit. Figure 2.7 shows this process for a 2D sphere. The following definitions and a more sophisticated version of this process using Dirichlet Processes are outlined by Straub et al. (2015). Instead of using the Euclidean distance to find the mean, a more appropriate distance

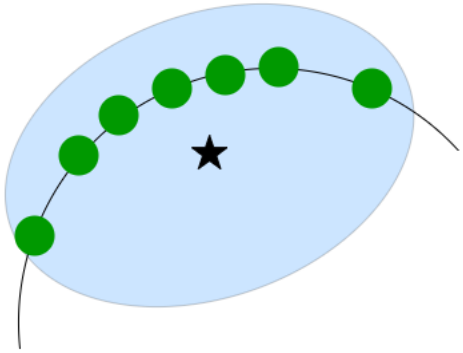


Figure 2.6: Embeddings lie on the boundary of a hypersphere. Regular Gaussian Mixture Models do not take into account the curvature of the data (green points). The mean of each distribution (star) will lie on the inside of the sphere instead of becoming the actual mean of the data. Because of the shifted mean, the covariance (blue) must grow to compensate which can produce wrong results.

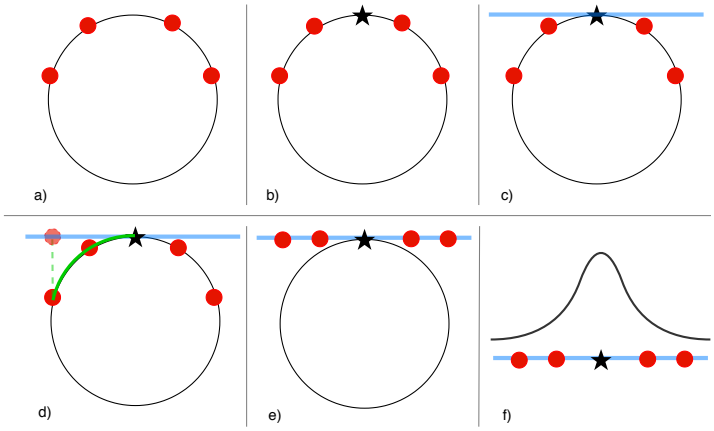


Figure 2.7: Process of fitting the hypersphere GMM. This process is applied to every class of embeddings separately. a) The embeddings (red points) lie on the boundary of the hypersphere. b) The Karcher mean (star) on the boundary is found by gradient descent. c) The tangent space (blue) is constructed at the mean and is tangent to the hypersphere at the mean. d) Each point is projected to the tangent space by the Riemannian Logarithm based at the mean. e) All the points now lie in the tangent space. At this point, Singular Value Decomposition (SVD), is used to reduce the dimensionality of the tangent space as its an N-1 dimensional hyperplane in N dimensional space. f) Once the points have been flattened out, a distribution can be fit.

would be to use the geodesic distance. This is simply the angle between 2 points and is defined by:

$$d_G(p, q) = \arccos(p^T q) \tag{2.10.1}$$

where p, q are points on the hypersphere. The mean on the boundary, also known as the *Karcher mean*, is defined as:

$$\hat{p} = \text{argmin}_p \sum_{i=1}^{N} d_G^2(p, x_i) \tag{2.10.2}$$

where $x_{1...N}$ are points on the hypersphere from some class. The Karcher mean is found through Gradient Descent. Once the Karcher mean, $\hat{p}$, is found the points are mapped to its tangent space, via the *Riemannian Logarithm*:

$$\text{Log}_{\hat{p}}(x) = (x - \hat{p}\cos\theta)\frac{\theta}{\sin\theta} \quad \left[\text{with } \lim_{\theta \to 0} \frac{\theta}{\sin\theta} = 1\right] \tag{2.10.3}$$

where x is a point on the hypersphere and $\theta = d_G(\hat{p}, x)$. The points now lie on a flat hyperplane, however the hyperplane can be described with one fewer dimension. Singular Value Decomposition (SVD) is used to reduce the dimensionality of the hyperplane. Finally, a Gaussian distribution is fit to the flattened points. To perform inference, a new point is projected to every class's tangent space and is evaluated by the corresponding distribution. The point is assigned to the distribution with the maximum log likelihood. The Hypersphere GMM was implemented using the geomstats python library (Miolane et al., 2018).

Chapter 3

Related Work

The idea of using social context and other metadata from images for recognition has been done before. Many different strategies for different tasks have been explored. Most of the work done in literature focuses on using the metadata for a very specific task, such as gender recognition and detecting whether 2 subjects are friends or not. The work done by Gallagher and T. Chen (2009) focuses on the task of gender recognition from appearance and context. They use the idea that people in images space themselves according to social cues such as their relationships, culture and gender. It was found that men tended to be situated more on the left, right and top borders of the image, children tended to be at the bottom and women preferred the middle. First the faces in the image are detected then an Active Shape Model is fit. From the Active Shape Model, several features are extracted such as the position of the face, the relative positions of the other faces and some visual features such as the space between the eyes. Additionally, a Minimum Weighted Spanning tree is constructed to identify the structure of the group in the image. However, they do not explicitly model the social relationships themselves or perform facial recognition.

Other methods that relate more to this project are those that use visual features to perform person identification while also using other sources of metadata. A popular method to incorporate the metadata is by using a Probabilistic Graphical Model. The work done by Stone *et al.* (Stone, Zickler, and Darrell, 2008) uses social context to automatically tag individuals in Facebook images. Individuals in Facebook images tend to appear alongside their friends. By modelling the social network of users, a more informed automatic tagging can be created. For every image, a graphical model is defined where there is a node for every face and every pair of faces is connected by an edge. The parameters are learnt in order to maximise the probability of seeing the individual in the image while also appearing with a set of friends. Similar work done by Wang *et al.* (G. Wang et al., 2010) attempts to identify individuals in images given their facial features, birth dates and pairwise social connection (sibling, mother-child *etc.*). A graphical model is constructed using the visual features as well as the social features. Graphical models are a natural fit to this type of data and perform well in this context. However, they require many training examples. From the student data that was collected, there is essentially 1 datapoint for the social network per lecture which makes these approaches too data hungry for this work. The application of PGMs to the current problem should be considered after the collection of a larger data set spanning more lectures.

Xiaolong Wang et al. (2017) also focus on automatic tagging of images, specifically whether an image is a family photo or not. Their assumption on whether an image is a family image is based on the distribution of the people, the similarity in looks of the people and the environment in which the photo was taken. Their work is split into three parts. The first deals with the distribution of people in the image. Family in photos tend to have a wider distribution of heights and be closer together. For example, there will be a big height difference between parents and their children. Photos of friends generally will not have such big variations in height. in order to capture this information, faces are located in the image and each face location becomes the vertex of a polygon. This polygon is invariant of rotation and translation and so should be distinct for each photo. This type of descriptor gives information about the global distribution of people rather than a descriptor for each individual. The second part deals with the similarity of the people in the photo. Initially, a standard CNN with convolutional layers followed by linear layers is trained using a separate dataset. Then each face in the photo is passed through the CNN and the output from the last linear layer is used as the descriptor for each respective face. The third part deals with the environment in which the photo was taken. The places which friends visit are different compared to the places that family visit. An ensemble of SVMs was trained to classify the type of environment using a second separate dataset. The output from each SVM is concatenated into a large feature descriptor for the image. The final classification is performed by another SVM trained on the three extracted descriptors. This type of work differs from this project in that it does not perform any learning of individuals across images and instead treats every image separately. This system also relies on a lot of external data to be trained.

Sapkota et al. (2013) also propose a system which uses an ensemble of various features to make predictions. In this case, their features focus on facial appearance and the co-occurrences of the subjects in the dataset. Additional metadata such as age, expression and gender are also incorporated into the system by training separate models for each attribute using external datasets. The facial appearance features used are local binary pattern and Gabor features. An ensemble of SVMs is trained on these features. The social features are incorporated by creating a long feature vector for each individual which is comprised of the predicted age, gender and expression, the positions of each individual, a histogram of global age distributions and a histogram of global gender distributions. The co-occurrences of each individual are incorporated in the same manner as G. Wang et al. (2010). Another ensemble of SVMs is trained on the social features. The probability scores are obtained from each SVM ensemble and a third ensemble of SVMs is trained to create a final prediction for each individual. The results showed that training an ensemble with the social features included outperformed an ensemble without the social features. This system relies on large amounts of external data. This system is similar to the one proposed by (Xiaolong Wang et al., 2017) but also includes some social information which more closely resembles this project.

Chapter 4

Data Collection

4.1 Introduction

Chapters 2 and 3 presented a number of different techniques which can be used for facial recognition. This chapter will explain the data that was used and the collection process. Currently, there are no datasets which contain face images along with social metadata over a long period of time. Therefore data had to be manually collected. Various first year Computer Science lectures at the University of the Witwatersrand, Johannesburg were filmed during 2018 and 2019. In 2018, there were 5 recordings, each around 5 minutes with 68 subjects. The 2018 lectures were filmed with a high resolution camera (4000×3000 pixels) but with a low framerate (3 FPS). In 2019, there were 10 recordings ranging from 10 to 30 minutes with 66 different subjects. The 2019 lectures were filmed using a 1080p camera but a higher framerate (30 FPS). Because there are many first year students, manually labelling the videos would be a long and expensive task. In order to try remove the need for manual labelling, an automatic labelling system was developed. The following sections will explain the automatic labelling as well as the opt-in process followed by the students.

4.2 Collection Process

Students who opted-in to be in the recordings were given an Augmented Reality (AR) marker. These AR markers were chosen because they are simple, easily generated and easily detectable. Additionally, each marker can store a unique number. When a marker is detected in the footage, a bounding box around it is made and is given the same label as the number of the marker. AR marker detection was done using the OpenCV Aruco library. Any area of the footage where a marker is not found is heavily blurred as shown in Fig 4.1. Some students did not wish to be part of the dataset and so this was done to remove them. This is in line with informed consent and the ethics protocol approved for this project by the ethics committee (ethics clearance number: H18/04/14).

There are several issues with the AR markers. There are a limited number of them however this can be alleviated by using more than 1 set of markers. The markers are sometimes hidden behind other students. There may be some frames of the recording where the marker is seen but sometimes markers are completely hidden. In this case, not much can be done as this means the student will

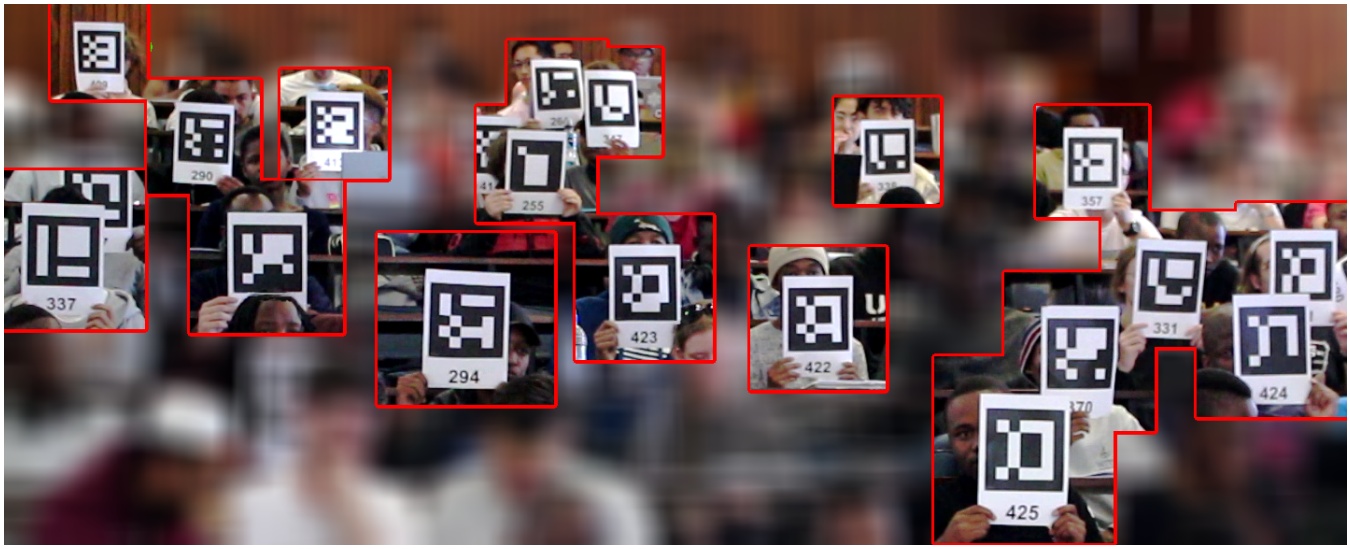


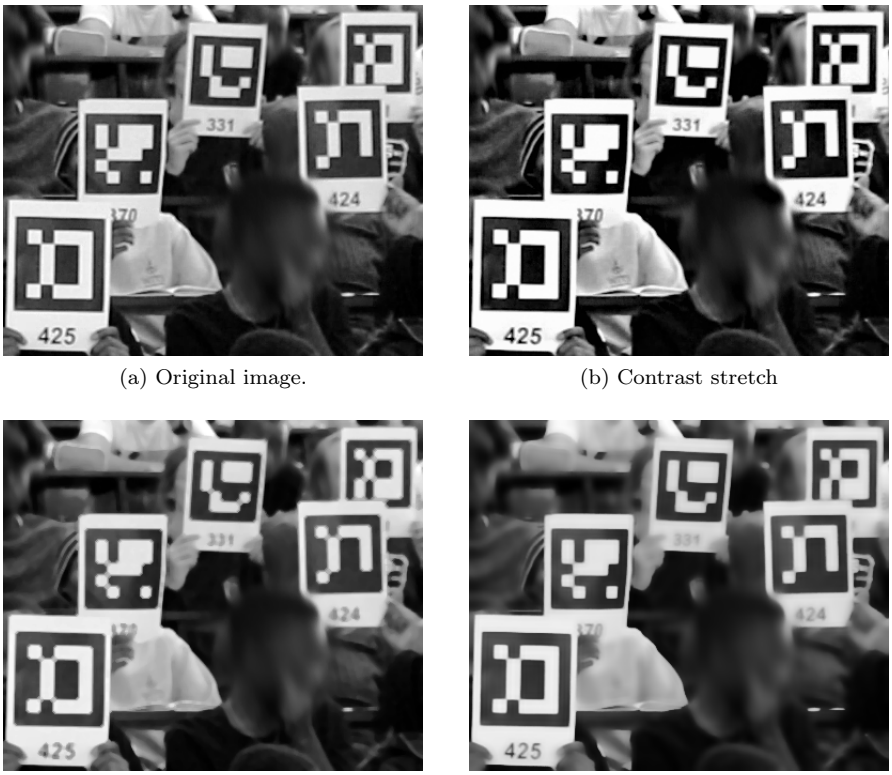
Figure 4.1: Students holding unique AR markers. Areas of the image where markers are not found are heavily blurred. Areas where markers are found are kept.

be hidden in the recording as well. Markers at the back of the venue are difficult to detect but some preprocessing can help to identify them better. Students would often lose or swap their markers with other students. In this case, nothing can be done except to manually update the labels of the students. Sometimes the detection library would return false positives, i.e. it would return a marker where one does not exist. These false markers always had strange proportions and so they could be ignored by checking the ratio of 2 sides to see if it had a square shape. The following section explains the preprocessing done to the footage, the face detection and social meta-data collection.

4.3 Processing Pipeline

Different preprocessing techniques helped to identify markers at different angles and sizes. Each frame undergoes a contrast stretch, bilateral filter and a median blur separately and then marker detection is performed. Marker detection is also performed on the original frame. Fig 4.2 shows the effects of each filter on the markers. The union of all detected markers from the 4 images is taken as the final marker positions. The bilateral filter is an edge preserving, smoothing filter. It generally improved marker detection across the entire image but especially on curved and angled markers. This may be because the filter helped emphasise the edges and also made the white spaces clearer. The contrast stretch helped detect markers on the sides of the image. The lights in the lecture venues are directly overhead with less lighting on the sides. The contrast stretch would help brighten the sides of the image and make the markers more identifiable. The median blur also helped to generally improve marker detection. The cameras used created a significant amount of salt and pepper noise. The median blur helped remove the noise without modifying the structure of markers too much.

Once the markers have been detected and the bounding boxes are created, the face detection



(a) Original image.

(b) Contrast stretch

(c) Median blur

(d) Bilateral filter

Figure 4.2: Different filters applied to a marker image.

step is performed. Face detection was performed using OpenCV's builtin haar cascade classifier (Viola and Jones, 2001). The face is saved along with the label of its corresponding AR marker. The xy pixel position of the face is also recorded. This gives an estimated seat position for the student. Each video was cropped to reduce the file size so the seat positions were then normalized by its corresponding video's width and height. This normalises the seat position to a pair of values between 0 and 1. The seat positions also give an indication of which students sit next to which, giving a social network structure. Once the face images have been saved they are upscaled using a super resolution network, specifically ESRGAN (Xintao Wang et al., 2018). As discussed in section 2.3, ESRGAN is a GAN which learns to upscale images rather than using linear or bicubic interpolation which can introduce artefacts. The face images are all varying in resolution and need to be a standard size to be passed into the neural network. Bicubic upscaling produced grainy images. Upscaling via the ESRGAN produced clearer images with better detail and less artefacting.

4.4 Conclusion

This chapter presented the data collection methodology based on informed consent, as well as the preprocessing steps to improve the quality of the images. The 2018 recordings had a total of 13256 images with 68 different individuals. The 2019 recordings had a total of 440657 images with 66 different individuals. The next chapter explains the facial recognition process, the different models used for classification, and finally the results.

Chapter 5

Training The Facial Recognition Component

5.1 Introduction

This chapter discusses the different methods used to train the facial recognition component. The first section discusses the methods used to train the network while the second compares the results of all the different methods.

5.2 Models and Training

The facial recognition was trained using the Inception v3 (Szegedy, Vanhoucke, et al., 2016) network using the triplet and quadruplet losses. The Inception v3 network was chosen because it is a deep network but still has a reasonable number of trainable parameters when compared to networks like VGG and Alexnet. The face images are passed through the network to obtain embeddings which are then passed to the loss function to train the network.

Triplet selection: When the network was trained using triplet loss, both the batch-all and batch-hard strategies were tested. Typically, the batch-hard strategy has been shown to perform better (Hermans, Beyer, and Leibe, 2017) because it selects the hardest training examples within a batch. The batch-hard examples will select the largest anchor-positive distance pairs and the smallest anchor-negative distance pairs. Batch-all selects examples which are still hard but are easy enough to let the network train. Batch-all selects the hardest examples with respect to the margin.

Embedding size: The network was trained with different dimensional embeddings. As the embedding dimensionality increases, there would be more space for the embeddings to separate and the accuracy should increase. Once there is enough space for all the embeddings, increasing the dimensionality further should have little to no effect on the accuracy.

Margin values: The next biggest factor would be the margins. The margins dictate how far each class should be from each other. The network was trained using different margin values while keeping the embedding dimensionality constant. As the margin increases, this increases the inter-class variance, and the accuracy of the classifiers would be expected to increase accordingly. This should occur until the margin becomes too large, in which case, the accuracy would start to

decrease as the margin would be too large to accommodate all the classes while still giving each class sufficient space to cluster properly.

Quadruplet loss: Additionally, the network was also trained with quadruplet loss. The margins in quadruplet loss are designed to be adaptive. The original adaptive margin strategy (Eqn 2.6.3) caused the network to always collapse (embeddings became 0) and so several different adaptive margin methods were used. In every strategy α_1 was always double the value of α_2 as the anchor-positive condition should be more important than the negative-negative condition. The first strategy used was to take the max of the newly calculated margin and some small positive number. The original method allowed the margins to become negative, especially at the beginning of training, which should not be valid for the margins. This approach would keep the margins strictly positive. The second strategy was to manually grow the margins logarithmically after every epoch. The margins were expected to be small at the beginning of training and grow slower over time until finally plateauing. α_1 and α_2 were grown between $[0.1, 1.0]$ and $[0.05, 0.5]$ respectively. The fourth strategy was to grow the margins linearly after every epoch between the same values as the previous strategy. Linear growth was tried because the logarithmic growth of the margins may initially prove to be too difficult for the network to accommodate. When using linear growth the α_1 values are defined as:

$$\alpha_1^{(e)} = a + \left(e \times \frac{b+a}{E} \right) \tag{5.2.1}$$

where:

- $\alpha_1^{(e)}$ is the value of α_1 at epoch e,
- a is the start point *i.e.* 0.1,
- b is the end point *i.e.* 1.0,
- e is the current epoch number,
- and E is the total number of epochs.

To create the logarithmic scale of the α_1 values, first a linear scale is created then each element is used as a power of 10. Let L be the linear scale created by the above process, then the α_1 values are defined as:

$$\alpha_1^{(t)} = 10^{L^{(t)}} \tag{5.2.2}$$

where $L^{(t)}$ is the value of the linear scale at time step t. In this case, because the scale is used as a power 10 the start and end are $a = -1$ and $b = 0$ respectively. This is because the α_1 values should be in the range $[0.1, 1.0]$ and $10^{-1} = 0.1$ and $10^0 = 1$.

The fifth strategy was to adapt the margin based on the difference between the anchor-positive and anchor-negative distributions. The KL divergence of the top ten largest anchor-positive distances and the ten smallest anchor-negative distances were used. The KL Divergence for probability distributions P and Q is defined as:

$$\mathrm{KL}(P||Q) = \sum_x P(x) \ln \left(\frac{P(x)}{Q(x)} \right) \tag{5.2.3}$$

The KL divergence measures how different distribution P is from Q. Generally, $\mathrm{KL}(P||Q)$ does not equal $\mathrm{KL}(Q||P)$. In order to make the measure symmetric, the average of them is taken. Let $P(x)$ be the anchor-positive distribution and $Q(x)$ be the anchor-negative distribution. Then the adaptive margin for each batch is defined as:

$$\alpha_1 = \frac{1}{2}(\mathrm{KL}(P||Q) + \mathrm{KL}(Q||P)) \tag{5.2.4}$$

The number of samples from both P and Q are required to be equal but there are different amounts of anchor-positive and anchor-negative pairs in a batch. The ten extreme values are chosen in order for the divergence to be calculated. At the beginning of training, the 2 distributions would be similar and so the KL divergence would be small. But, as training converges the distributions would be very different and the KL divergence would become larger. In the final strategy, the network was initially trained using triplet loss and then finally trained using quadruplet loss. The main issue with the adaptive margins is that the network takes too long to start producing good embeddings and so the adaptive margins were hindering the network's training. The goal of this strategy was to try get the network to a good initialisation in order for the original adaptive margin strategy to adapt better.

Distance metric: Besides the margin values, another hyperparameter of the loss functions is the distance metric used. These loss functions typically use Euclidean distance as the distance metric. Two other distance metrics were applied, namely *great circle distance* and a *cosine dissimilarity distance*. The embeddings lie on a hypersphere so great circle distance is an appropriate distance metric to use. It is defined as:

$$\begin{aligned} d_{GC}(p,q) &= r \times \theta \\ &= r \times \arccos\left(\frac{p \cdot q}{r^2}\right) \\ &= \arccos(p \cdot q) \end{aligned}$$

where:

- $\vec{p}, \vec{q}$ are unit vectors on the boundary of a hypersphere,
- θ is the angle between $\vec{p}$ and $\vec{q}$,
- r is the radius of the hypersphere, *i.e.* $r = 1$.

Regular cosine similarity distance is a measure of similarity, meaning the higher the cosine similarity the more similar the vectors. The maximum value of cosine similarity is 1, so by subtracting the cosine similarity measure from 1 the result is a dissimilarity measure. It is defined as:

$$\begin{aligned} d_{CS}(p,q) &= 1 - \frac{p \cdot q}{\|p\| \, \|q\|} \\ &= 1 - p \cdot q \end{aligned}$$

where p, q are unit vectors on the boundary of a hypersphere. Since p, q are unit vectors, $\|p\| = 1$ and $\|q\| = 1$.

Holdout training: The neural networks were trained using 2 different sets of training data. In the first scheme, the first 70% of each label was kept for training and the last 30% of each label was

kept for testing. The data is ordered chronologically meaning older images appeared before newer ones. In terms of splitting the dataset this means that older images were used for training while newer ones were used for testing. This training scheme was expected to perform well as there were many examples of each student and in most cases there were examples similar to that of the testing data. In order to test whether the network learnt to recognise students in completely different settings, several networks were trained in a cross validation style. One network was trained on all the lectures except the first, but was tested on the first lecture. The second network was trained on all the lectures except the second, but was tested on the second lecture *etc.* This training scheme was expected to perform much worse than the first scheme. In this case, the networks were trained on many students but only tested on the students that appeared in the testing lecture as not every student appears in every lecture. Additionally, the networks were tested on lighting conditions and new student appearances that did not appear in the training set. Every experiment used the 70%/30% split data except where specified otherwise.

Once the embeddings have been learnt using all the discussed methods, classification can be done. Many different classification methods were used, including the GMM, Bayesian Non-parametric GMM, Hypersphere GMM, XGBoost, Support Vector Machine, Decision Tree, Random Forest, Naive Bayes, Nearest Neighbour, Logistic Regression and a small Neural Network.

5.3 Results

Triplet selection: Two separate networks were trained, one with batch-hard and the other with batch-all strategies. In most literature, batch-hard performs better but it is still very data dependent. Using the student facial data, the batch-all strategy produced better results as shown in Fig 5.1. The batch-all strategy outperformed the batch-hard strategy for every classifier. The batch-all strategy had a maximum accuracy of 89.82% while the batch-hard strategy had a maximum accuracy of 82.23%. The training examples selected by batch-all were too difficult for this specific data. Batch-all selected easier examples but still hard enough for the network to learn properly. This could mean that the hard negative images chosen were too similar to the positives compared to the semi-hard negatives. As such the batch-all strategy was used as the triplet selection strategy for all subsequent experiments.

Embedding size: The first experiment focused on the embedding dimension. The embeddings were generated with different embedding sizes using Triplet loss with a constant margin $\alpha = 0.5$. As the embedding size increases, the accuracy generally increases. The increased accuracy is due to having more space to separate classes more clearly. However, once the embedding size becomes too large the accuracy starts to decrease. Figure 5.2 shows the general trend for several classifiers. The reason for this is the data becomes more and more curved as the embedding size increases. Classifiers that cannot fit well to spherical data will not be as accurate. This is especially evident in mixture models. The GMM and BN-GMM do not perform well with large embedding sizes, however the Hypersphere GMM is able to fit the data much better at higher dimensions. Classifiers like Nearest Neighbour do not suffer as much from the increased curvature. The ensemble methods cope well with large embedding sizes. This is because they can continuously add weak classifiers to account for the spherical data. The best embedding size was found to be 64 for most classifiers.

Margin values: The next experiment focused on the margin values. The embeddings were generated using Triplet loss and an embedding size of 64 and increasing α values for Triplet loss. Figure 5.3 shows the results. As the margins increase, the accuracy generally increases as well. This is due to a bigger margin forcing classes to be further from each other. The further each

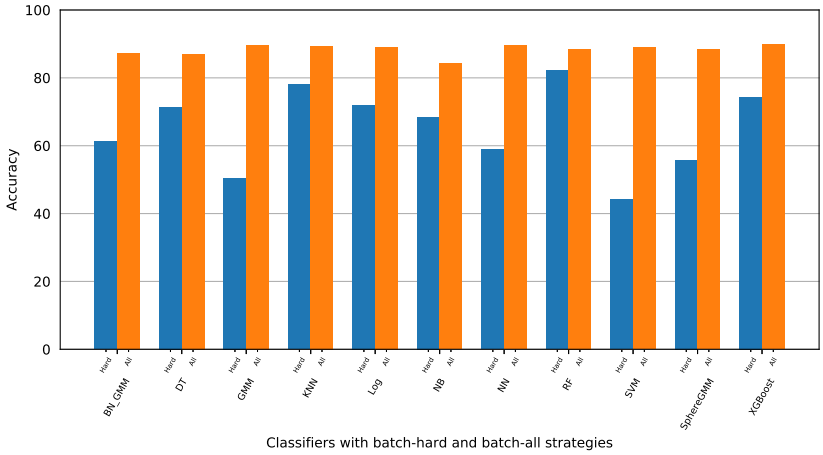


Figure 5.1: The accuracy of different classifiers using different triplet mining strategies.

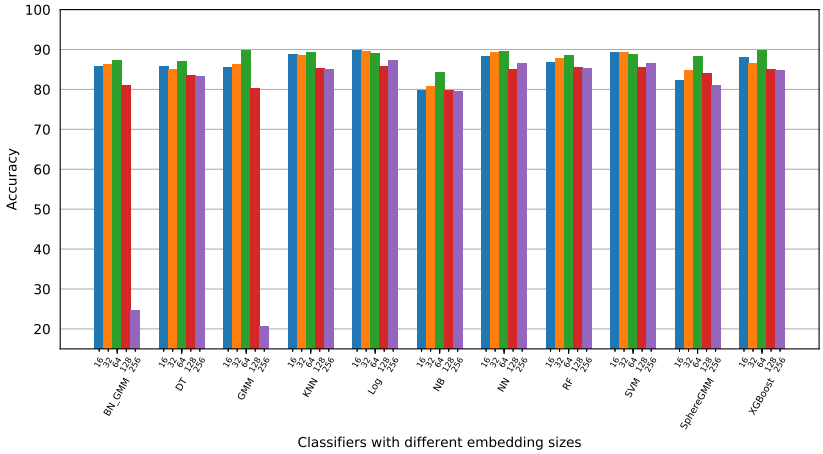


Figure 5.2: The accuracy of different classifiers using different embedding sizes. Generally, the accuracy increases as the embedding size increases. Once the embedding size becomes too large, accuracy starts to decrease. The GMM and BN-GMM struggle to fit the high dimensional data but the Hypersphere GMM is able to cope with it much better.

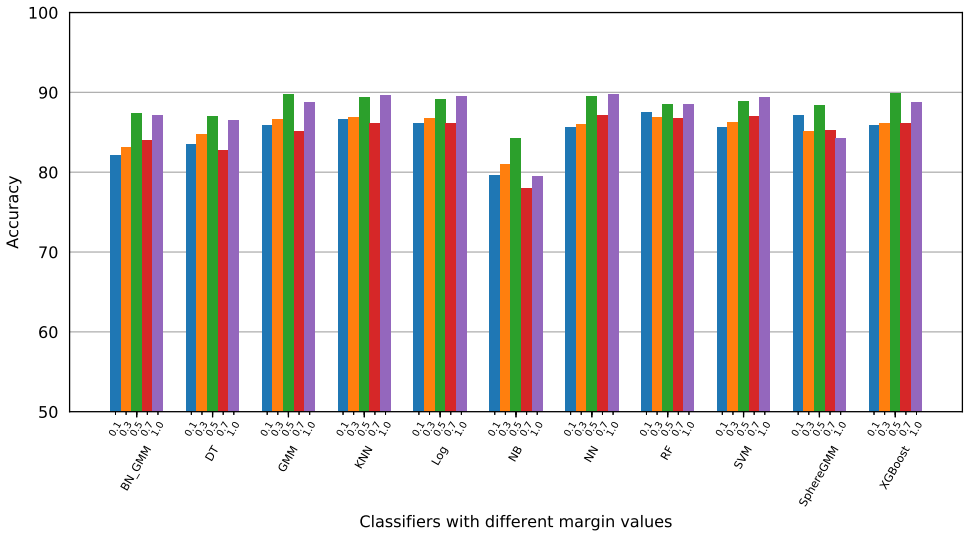


Figure 5.3: The accuracy of different classifiers using different margin values. The accuracy generally increases as the margin increases. If the margin is too small then the data cannot be separated properly. If the margin is too large then the classes may not have sufficient space to fulfil the Triplet loss criteria. A margin of $\alpha = 0.5$ had the highest accuracy for most of the classifiers.

class is from each other, the better the classes are separated and the new points can be classified better. Once the margin becomes too large the classes cannot be separated clearly as there may not be enough space or there are too many classes. Some classifiers benefit slightly from a large margin but most classifiers perform best with a margin which is neither too easy nor too hard. The maximum Euclidean distance any two embeddings on a unit hypersphere can have from each other is 2. However, the distances calculated in triplet loss are squared so the maximum distance is 4. A margin of 1 is still feasible in this setting but the results show that simply increasing the margin does not result in higher accuracies. The best margin value was found to be $\alpha = 0.5$ and so will be used in all subsequent experiments.

Quadruplet loss: One of the proposed improvements of Quadruplet loss over Triplet loss is adaptive margins. The original adaptive margin strategy always caused the networks to collapse ($\alpha = 0$) and so other strategies were used *i.e.* taking the max of the margin and a small positive number, KL divergence of the anchor-positive and anchor-negative distributions and growing the margins logarithmically/linearly. Fig 5.4 shows the results. Growing the margins linearly produced the best results. Growing the margins logarithmically produced the worst results for most classifiers. This shows that the rate of growth, especially at the beginning of training cannot be too fast. If it is then the network will fail to produce good embeddings. Using the max strategy performs better than keeping the margins constant. This shows that the original adaptive margin strategy can make the margins too small and even negative. If the margins become too small then the network can place negatives only slightly further than positives to the anchor. The max strategy stops the margins from becoming too small. Setting the margins to the KL divergence of the anchor-positive and anchor-negative distributions should allow the margins to capture how well the network was

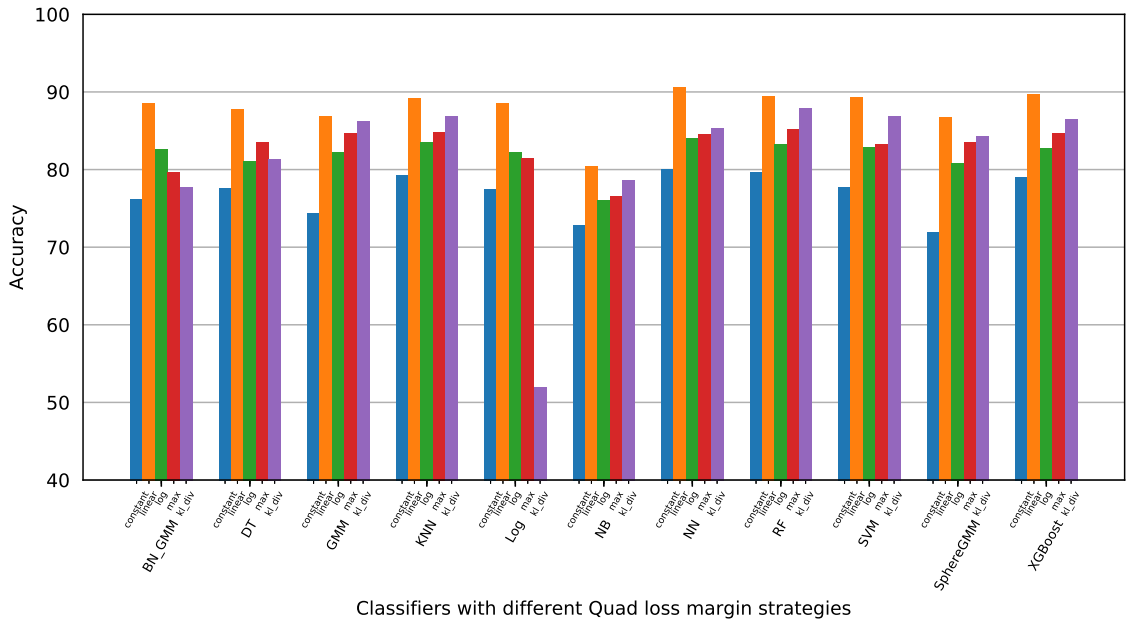


Figure 5.4: Quadruplet loss with constant margins (in blue), setting $\alpha_1 = 1.0$ and $\alpha_2 = 0.5$, performed the worst of the strategies. Keeping the margins at set values does not allow the network to adapt to harder examples. Linear growth of the margins (in orange) performed the best. The rate of growth was not too great to hinder the network from learning properly. Logarithmic growth (in green) grew the margins too quickly at the beginning of training and the network could not adapt properly. Taking the maximum of the adaptive margins and 0.1 (in red) allowed the network to learn using the original adaptive margin strategy. However, the margins may not have grown at a fast enough rate. The KL divergence strategy had varying performance (in purple).

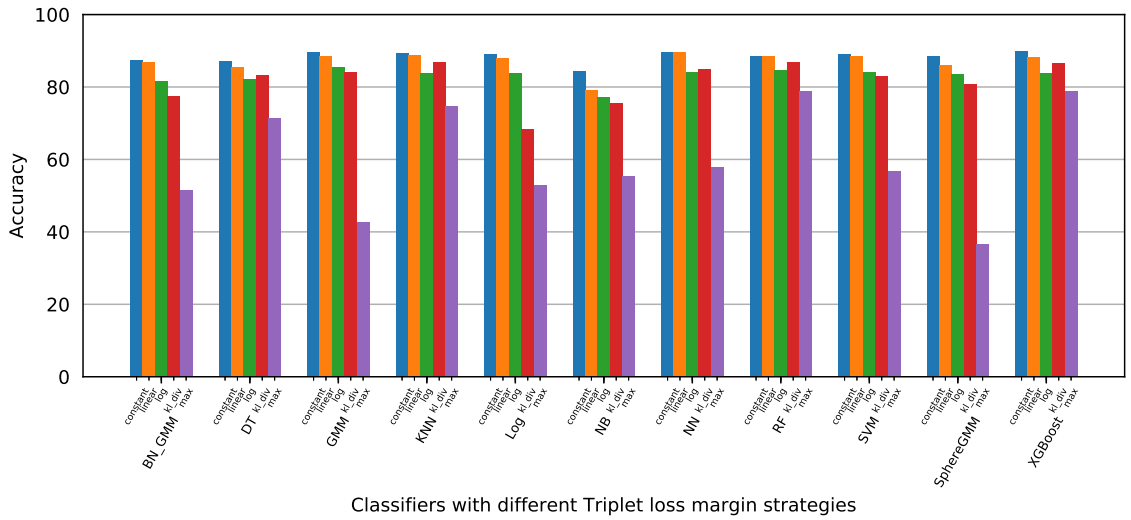


Figure 5.5: The accuracy of different classifiers using different margin growth strategies for Triplet loss. Having a constant margin gives the best accuracies. Growing the margin linearly performs slightly worse. Unlike Quadruplet loss, Triplet loss does not get the same boost in accuracy by growing the margin or adaptively setting it using KL divergence.

separating the different classes. However, it is dependent on the examples in the batch and the batch size. The KL divergence may have set the margin too small at some iterations.

The same adaptive margin strategies were tested on triplet loss as shown in Fig 5.5. Triplet loss does not benefit from the strategies in the same way that quadruplet loss does. The constant margin performed the best while the other strategies performed worse. Linear growth still performed better than logarithmic growth. KL divergence had varying performance. The max strategy performed the worst.

Distance metric: The common distance metric for Triplet loss is Euclidean distance. Three networks were trained using 2 other distance metrics, *i.e* cosine dissimilarity and great circle distance. The embeddings were generated using an embedding size of 64 and a margin value of $\alpha = 0.5$. The results are shown in Fig 5.6. Euclidean distance performed the best. cosine dissimilarity distance performed slightly worse than Euclidean for most classifiers. Great circle distance performed the worst. A potential reason for this may be that great circle distance cannot distinguish distances of close points. The inverse cosine is numerically unstable when comparing the values of two very close points. An alternate formulation of the great circle distance may produce better results.

Holdout training: Networks were trained using the whole training set except for 1 lecture. Then the networks were tested on the specific lecture they were not trained on. This experiment was to test whether the networks were truly learning the students appearances and not overfitting. The accuracies for these networks are expected to be significantly lower than previous training methods. The accuracies can only be calculated using subjects which appear in both the testing lecture and the training lectures. Figures 5.7 (a)-(g) show the results of holdout testing for the 2019 lectures. Each network was trained using different subsets of students. For each year, the networks were trained on $\approx$ 60 labels which is a 1.6% chance of guessing correctly. While the networks

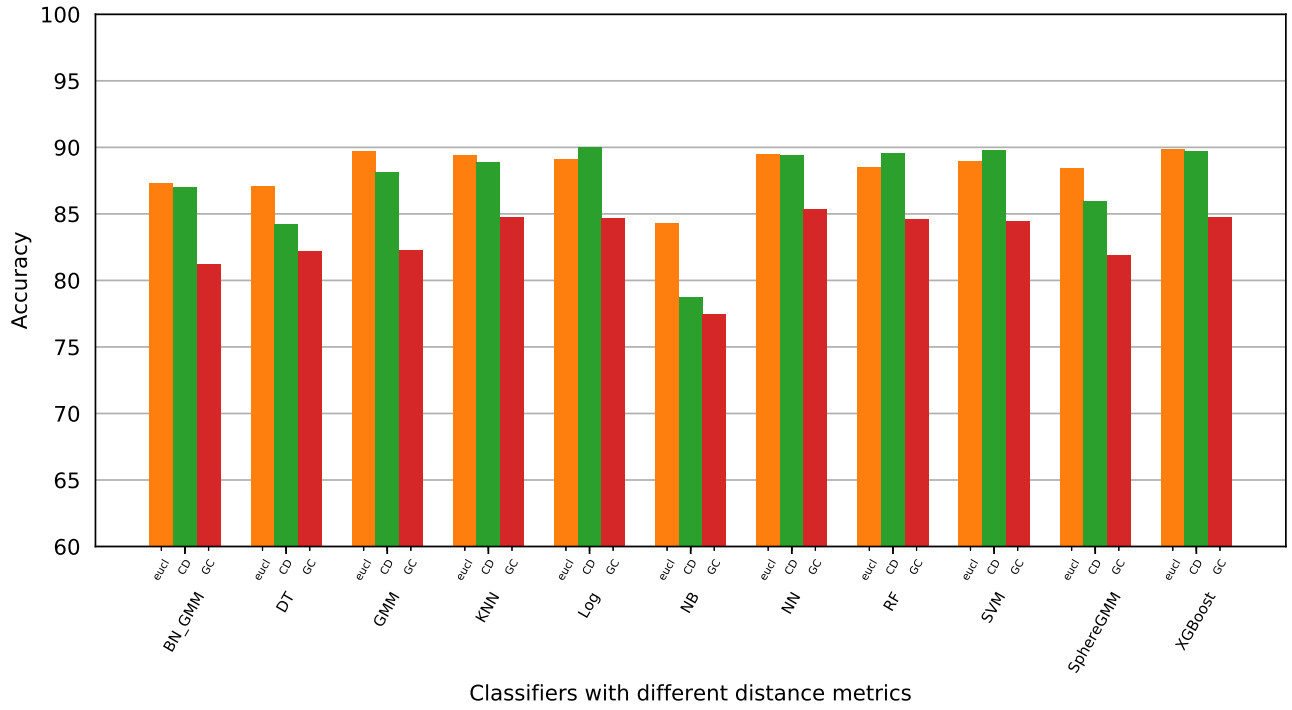
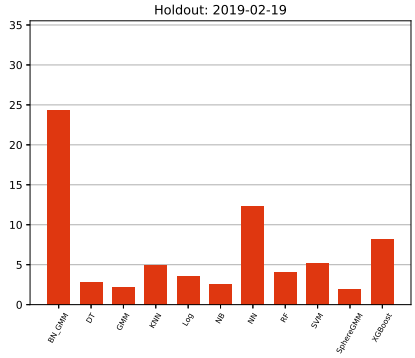
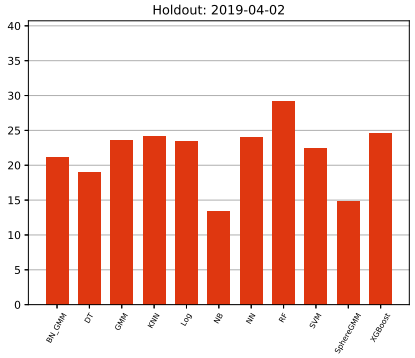


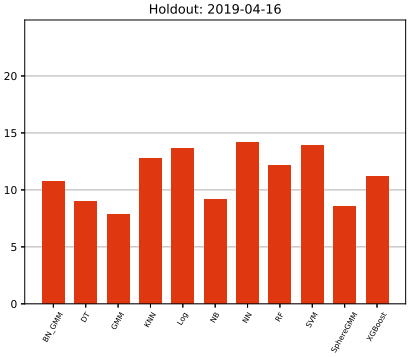
Figure 5.6: The accuracy of different classifiers using different distance metrics when training Triplet loss. Three different metrics were tested, euclidean distance (eucl), cosine dissimilarity (CD) and great circle (GC). Euclidean distance captures the distances of the embeddings better than the other two distance metrics. Great circle distance has the lowest accuracy for every classifier.



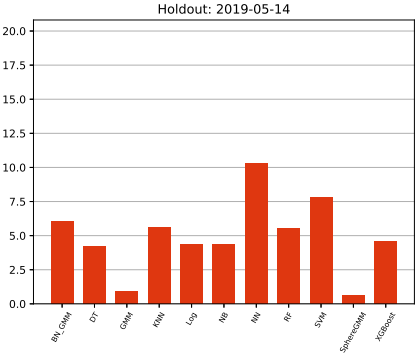
(a) Holdout testing on the 2019-02-19 lecture.



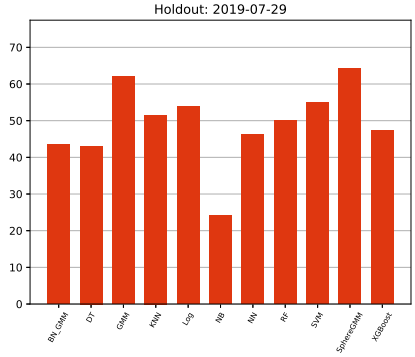
(b) Holdout testing on the 2019-04-02 lecture.



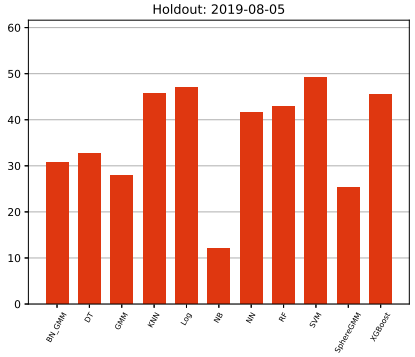
(c) Holdout testing on the 2019-04-16 lecture.



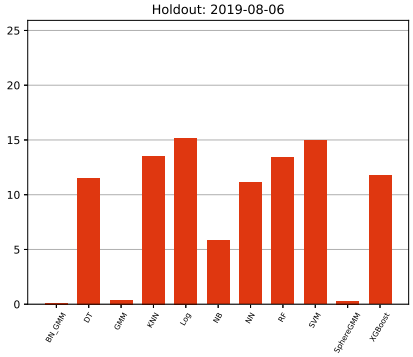
(d) Holdout testing on the 2019-05-14 lecture.



(e) Holdout testing on the 2019-07-29 lecture.



(f) Holdout testing on the 2019-08-05 lecture.



(g) Holdout testing on the 2019-08-06 lecture.

Figure 5.7: Results of holdout testing on various 2019 lectures.

performed much worse than using regular training, they still performed much better than random. This indicates that the networks can still recognise students (even partially) when not trained on examples of students in different clothing and lighting conditions. The low accuracies in Fig 5.7 (a) are attributed to it being the first recording for 2019. In that recording, the camera was positioned to try record as many students as possible. However, this setup resulted in poor image quality. The recordings that followed focused on smaller areas. The other low accuracies could indicate that more data is needed in order to generalise better. Figure 5.7 (c) and (d) have the same lecture venue but it is a different venue from the rest of the recordings. The model is overfitting to the other venues which results in worse performance for the 2019-04-16 and 2019-05-14 lectures.

5.4 Conclusion

This chapter presented the methods for training the facial recognition component as well as the results. It was found that using triplet loss with an embedding dimensionality of 64 and a constant margin of 0.5 performed the most consistently. And so this will be the final configuration of the network. The following chapter explains the seat behaviour modelling process and then results.

Chapter 6

Modelling Seating Behaviour

6.1 Introduction

Besides the learnt facial embeddings, the seating behaviour of the students were also used to classify students. This chapter explores the seating behaviour of the students. The first section discusses the various models used while the second discusses the results.

6.2 Models

The accuracy of the classifiers using the 2018 data is expected to be higher than the accuracy of the classifiers using the 2019 data. The 2018 recordings were more consistent. The same lecture venue was used for all the recordings. Also, the same camera and same lighting conditions were used in every recording. In the 2019 seat data, the students sat in more varied seating positions, making classification much harder. Two different cameras were used as well. The cameras were in different positions and the lighting conditions changed significantly from video to video. For these reasons, the accuracy for the 2019 data is expected to be significantly lower than the 2018 data. Three sets of experiments were done. The classifiers were trained using just the 2018 data, just the 2019 data and the whole dataset in separate experiments.

As with the face embeddings many different classification methods were used, including the BN-GMM, Hypersphere GMM, XGBoost, Support Vector Machine, Decision Tree, Random Forest, Naive Bayes, Nearest Neighbour, Logistic Regression and a small Neural Network. Students typically sit in familiar positions and with familiar people over time. This makes each student's seat data multi-modal. GMM's, BN-GMM and the ensemble methods should perform the best on this type of data as they are able to capture these patterns.

6.3 Results

When classifying the 2018 seating data, as shown in Fig 6.1, the accuracies were very high especially for XGBoost. This shows that the students tended to sit in very similar seats throughout the lectures. Two other factors which contributed to the high accuracies were the camera position and the constant lecture venue. The camera was always placed in the same position and a similar angle

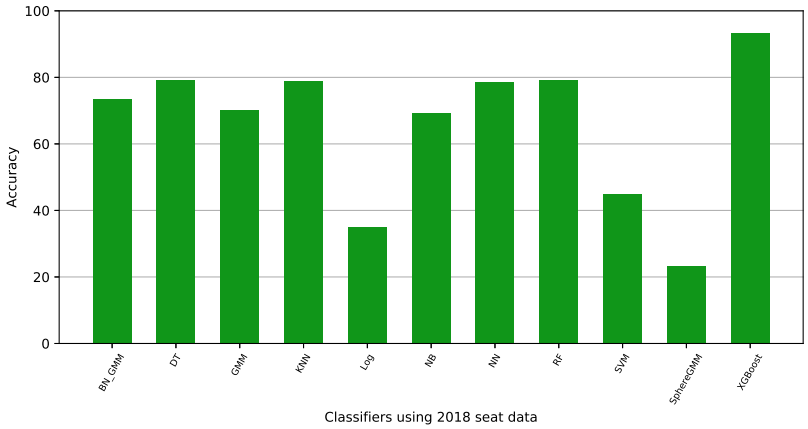


Figure 6.1: Results of classifiers using only the 2018 seat data. In this case, XGBoost performs extremely well achieving 96% accuracy. The high accuracies show that the students tended to sit in familiar seats. The Hypersphere GMM performs poorly because the data is not on a hypersphere.

every lecture. Additionally, the lecture venue did not change so the lighting conditions remained similar in all the videos and the students had settled into familiar seats. In some sense this also means that the classifiers have possibly overfit to the venue. When classifying the 2019 seating data, as shown in Fig 6.2, the accuracies were not as high as 2018. This shows that the students sat in more varied seating positions compared to 2018. However, they still had regular enough seating positions to identify them correctly. XGBoost still performed the best of all the classifiers. The lower accuracies are mostly due to the camera moving position often and the change in lecture venue that occurred in the 2019 lectures. The Hypersphere GMM did not perform well with this data because the data does not live on a hypersphere. Logistic regression and SVM also did not perform well due to the decision boundaries not being linear. The classifiers performed moderately well when using all the data, as shown in Fig 6.3. The accuracies when using all of the data lies in between the 2018 and 2019 results. The two sets of data balanced each other out by having students with very regular seats and students with less regular seats.

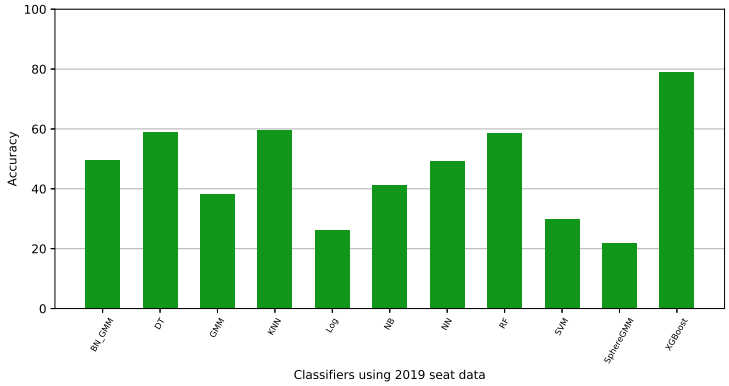


Figure 6.2: Results of classifiers using only the 2019 seat data. The maximum accuracy was 78% using XGBoost. The accuracies are still high but much lower than that of the 2018 data. This shows that in 2019 the students sat in more varied seating positions which makes classification harder.

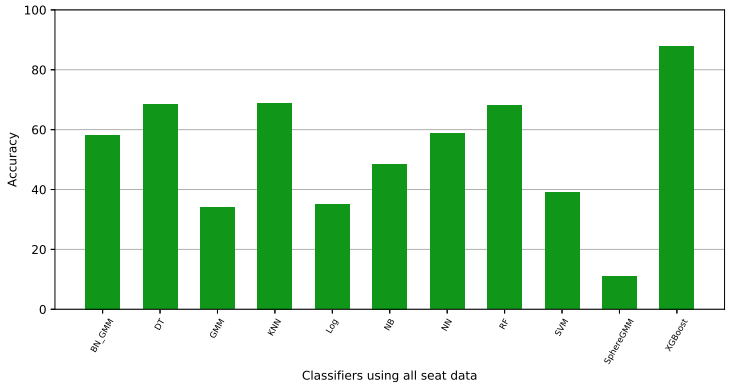


Figure 6.3: Results of classifiers using all the seat data. The maximum accuracy was 88% using XGBoost. The accuracies lie in between those of the 2018 and 2019 results.

Chapter 7

Combining Facial Recognition and Seating Classifiers

7.1 Introduction

Chapter 5 discussed several methods of training the facial recognition component while chapter 6 discussed modelling the seating behaviour of students. The first section explores several methods of combining the two sources of information while the second discusses the results.

7.2 Models

The models for the facial recognition and the seat recognition performed fairly well. The classifiers when paired with standard triplet loss performed the most consistently with accuracies between 80% and 90%. However, the maximum accuracy was a small neural network paired with quadruplet loss and the linear growth margin strategy, obtaining an accuracy of 90.6%. The seating classifiers had varying performance but XGBoost performed the best with 88% accuracy. The next step is to combine the two sets of information in order to make better classifications. This section discusses the different methods used to try incorporate the seat information into the facial recognition model.

Concatenation: The first method was to concatenate the embeddings and the corresponding seat together and then train the classifiers using this extended data. It was expected that adding the seat position to the embedding would help in differentiating similar embeddings of students better. Similar embeddings of images means that the network placed them in the same area in the embedding space. This means that the network thought that the images contained the same subject. If there are different students with similar facial embeddings then seat positions might be able to disambiguate them. This is assuming that the seat positions are different for each student.

Seat injection: The second method was to add the seat information while training the embeddings. The network was given a batch of face images as well as the seat positions after the convolutional layers. By injecting the seat positions right before the embeddings are created the network should learn more meaningful embeddings. This should in turn separate the classes more clearly and help to distinguish similar embeddings of different students.

Learnt priors: The third method is different in that the seat data is not explicitly introduced

with the facial information. In this method the focus is on the priors of the mixture models. Recall from chapter 2 the definition of a mixture model:

$$\begin{aligned} P(x|\theta) &= \sum_{k=1}^{K} P(x|k,\theta)P(k|\theta) \\ &= \sum_{k=1}^{K} \lambda_k \text{Norm}_{x_i}\left[\mu_k, \Sigma_k\right] \end{aligned}$$

$P(k|\theta)$ is the prior of the mixtures. The prior dictates the weightings for each mixture and is described by a Categorical distribution $\text{Cat}_k\left[\lambda_1 \dots \lambda_K\right]$. Three different priors are used to weight the mixtures in different ways. The first is a flat prior where all classes are equally as likely *i.e.* $\lambda_k = \frac{1}{K}$ for K classes. The second is a naive prior where each class is only as likely as they appear in the training set *i.e.* $\lambda_k = \frac{n_k}{N}$ where n_k is the number of examples from class k and N is the total number of examples in the dataset. The third is a learnt prior which is constructed by training a small neural network on the seat information. The predictions from the network for each class are used as the priors for the corresponding mixture. Then for a new embedding, its corresponding seat information is fed through the network and the probabilities of the predicted identities of the network are used as the prior of the mixture model when performing inference. This method should weight the mixtures better and provide better classifications, provided that the network is sufficiently accurate.

7.3 Results

Concatenation: The first method concatenated the embeddings and the seat data together. This simple method was able to provide an increase in accuracy as shown in Fig 7.1. Even when the seat classifier was not completely informative on its own, like in the case of the HypersphereGMM, the combined classifier can still benefit from the extra metadata. The increase in accuracy was very similar for every classifier. One of the reasons for this is that the embedding feature vector is 64 dimensional while the seat feature vector is only 2 dimensional. The main influence will be the embeddings while the seats will help disambiguate close predictions. And so the combined classifiers should be similar to the embedding-only classifiers. The Naive Bayes classifier performed the worst with 88.02% accuracy and the best performing was the Neural Network with 92.05% accuracy. Overall, this method performed well and each classifier gained a small boost in accuracy.

In the case where the seat classifier is more accurate than the embedding classifier, the combined classifier will not benefit as much from the seat data. Figure 7.2 shows this where the seat accuracy is higher than the embedding accuracy using XGBoost and the 2018 data. In this case, the combined classifier is not as accurate as just using the seats. This could be because the seat information alone is a much better discriminator than using the embeddings. When they are concatenated the high dimensional embedding features dominates the much lower 2 dimensional seat features. To gain some insight into how the combined classifier performs with different ratios of seat and embedding accuracies, Gaussian noise was added to the seat data. The Gaussian noise was 2 dimensional, had a mean of 0 and a changing covariance. This covariance ranged from no noise to a covariance equivalent to 2 rows to the front or back and 5 seats to the left and right of the student of interest.

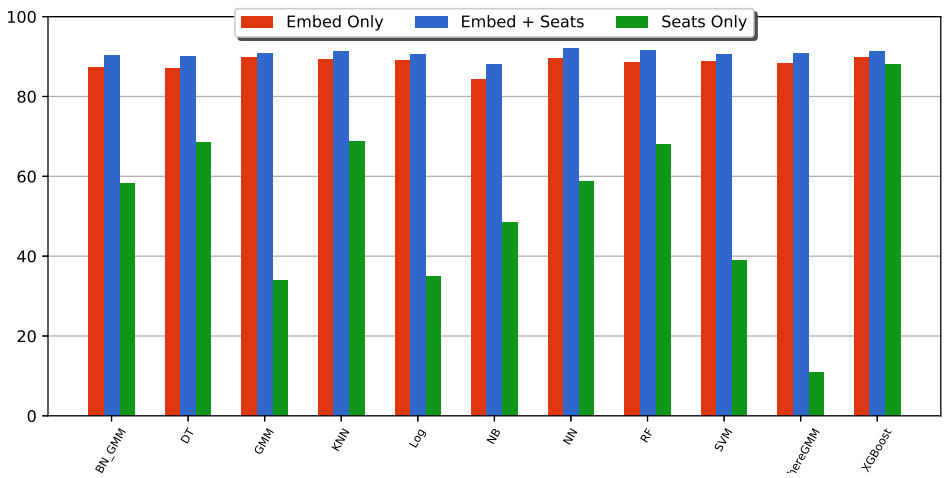


Figure 7.1: In every case the classifiers benefit from the added seat data, even if it is a slight benefit. The added seat data helped to distinguish similar embeddings of different students.

The actual covariance values differed per recording because recordings had different resolutions. The seat accuracy drops significantly after only a small portion of noise is introduced. This shows that XGBoost was overfitting to the data to some degree. However, as the noise increases to the point where the seat accuracy drops to below 20%, the combined classifier maintains a slightly higher accuracy than the embedding only classifier. This experiment shows that as long as the metadata that is included into the system is not too regular *i.e.* students sit in the same seats too often, then the combined classifier can benefit from both sources of information. If the metadata is too regular then it means that the metadata is a better discriminator than the main feature set. The metadata should be able to be used in conjunction with the main features rather than hinder them.

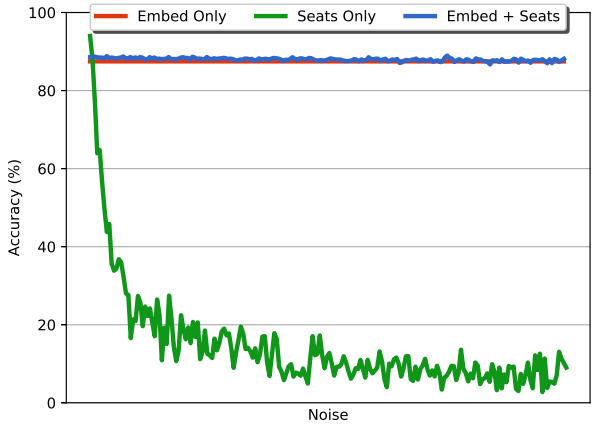


Figure 7.2: The XGBoost accuracy for the 2018 seat data was very high while the accuracy for the embeddings was not as high. This prevents the classifier from benefiting from the combined data. Increasing 2D Gaussian noise was added to the seats over time ranging from no noise to the equivalent of 2 rows front/back and 5 seats left/right. The sharp drop in accuracy shows that XGBoost was overfitting to the seats. However, as the seats became more noisy, the combined classifier was still able to be as accurate as just using the embeddings.

(a) Example from 2018 data. Originally, the embeddings were a distance of 0.0195 apart. After seat injection, they are now a distance of 0.2211 apart.

(b) Example from 2018 data. Originally, the embeddings were a distance of 0.0785 apart. After seat injection, they are now a distance of 0.2988 apart.

Figure 7.3: Examples of seat injection helping to disambiguate similar facial embeddings. The face embeddings were originally close together and less than the average embedding distance. After retraining the network and injecting the seat data, the face embeddings are now further apart and have a distance greater than the average embedding distance.

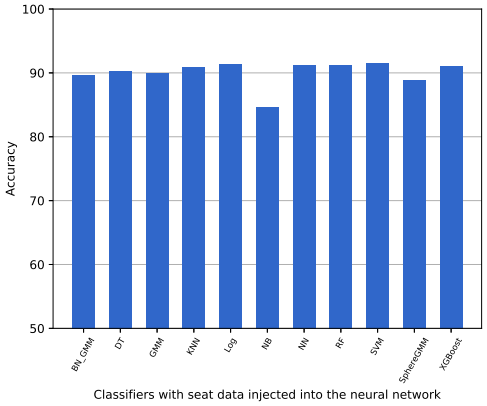


Figure 7.4: Results of classifiers when seat data was injected into the neural network. This allowed the network to distinguish between similar embeddings of different students. This method increases the overall classification accuracy with most of the classifiers having accuracies over 90%.

Seat injection: The second method was to inject the seat data into the neural network after the convolutional layers. The embeddings learnt by the network are now a mix of the facial features and the seat data. Figure 7.4 shows the results. The accuracies are slightly lower than just concatenating the embeddings and seat data however still higher than not incorporating the seat data at all. The worst performing classifier was Naive Bayes with 84.59% accuracy and the best performing was SVM with 91.55% accuracy. Figure 7.3 shows how injecting seat data into the models can help disambiguate similar embeddings of face images. Face embeddings of similar yet different faces can be mapped close together. By adding the seat data to the embeddings, the similar embeddings can be disambiguated and are mapped further apart.

Learnt priors: The third method focused on the priors of the mixture models. There are 3 mixture models used, *i.e.* Gaussian Mixture Model, Bayesian Nonparametric Mixture Model and the Hypersphere Gaussian Mixture Model. Figure 7.5 shows the results of the different priors on the 2018 data. The learnt priors performed significantly better the other priors. The learnt priors take in extra information so it would be expected to perform better than conventional priors. Figure 7.6 shows the results of the different priors on the 2019 data. In this case, the learnt priors are not vastly different from the other priors. Figure 7.7 shows the results of the different priors on the entire seat dataset. The learnt priors still outperform the other priors.

Out of all the methods to combine the seat data and the embeddings, the concatenation method performed the best with the neural network classifier. The seat injection method performed almost as well with a maximum of 91.5% with the SVM classifier. In every case there was a boost in accuracy when using both sources of information rather than just one source. Concatenation performed the best generally across all the classifiers. Concatenation forces the data to not lie on a hypersphere anymore, however that is not an issue for any of the classifiers except for the HypersphereGMM. To counteract this, the HypersphereGMM normalises the data first before it performs classification. The seat injection may hold more potential when there is more metadata in the system. As more metadata is concatenated, the feature set will grow larger. However, with the injection method the output feature set will always remain the same dimensionality. There may come a point where concatenating too much data results in less accuracy gains while also

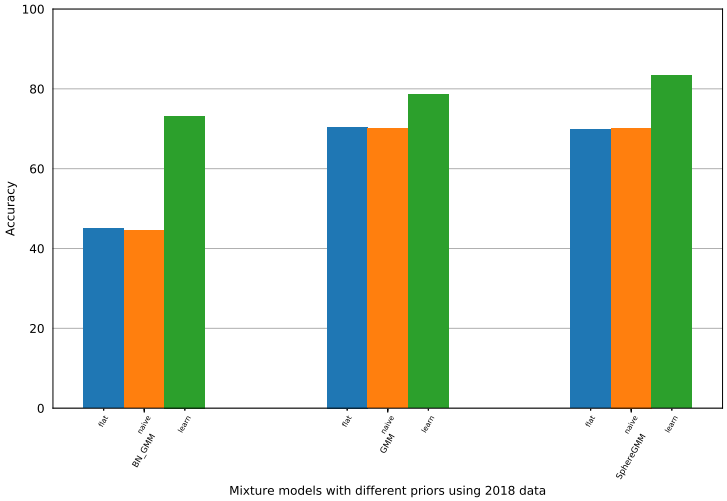


Figure 7.5: Using only the 2018 data, the mixture models were able to get a significant boost in accuracy from the learnt priors. The flat prior and naive prior had very similar performance.

leading to slow convergence. On the hand, reducing too much metadata to a set dimensionality with the injection method may also result in less accuracy gains. The learnt prior method performed unexpectedly well. The flat and naive priors performed exactly the same and so that was expected for the learnt priors as well. It seemed as though the mixture models stopped benefiting from the priors very quickly because of the amount of data. However, with the learnt priors, they were able to give the mixture models a good initialisation. More accurate learnt priors using more sources of metadata could result in even bigger accuracy gains. Each method has its own advantages but currently the method which produced the best results is concatenation.

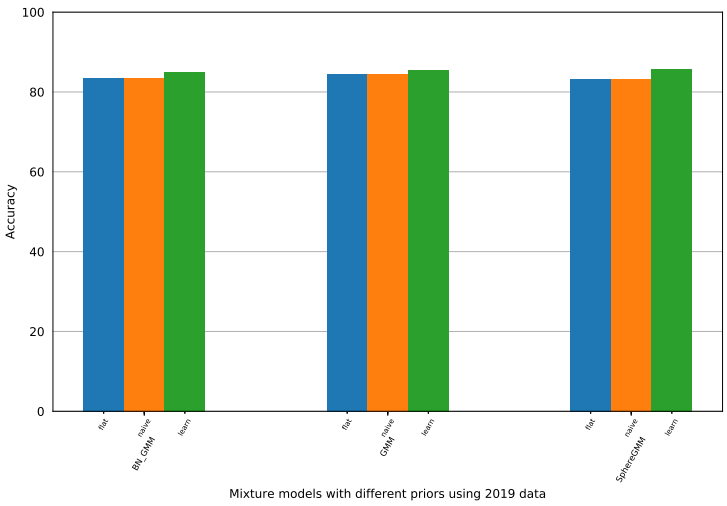


Figure 7.6: Using only the 2019 data, the learnt priors did not have a significant impact on the accuracy. However, it still performed better than the other priors.

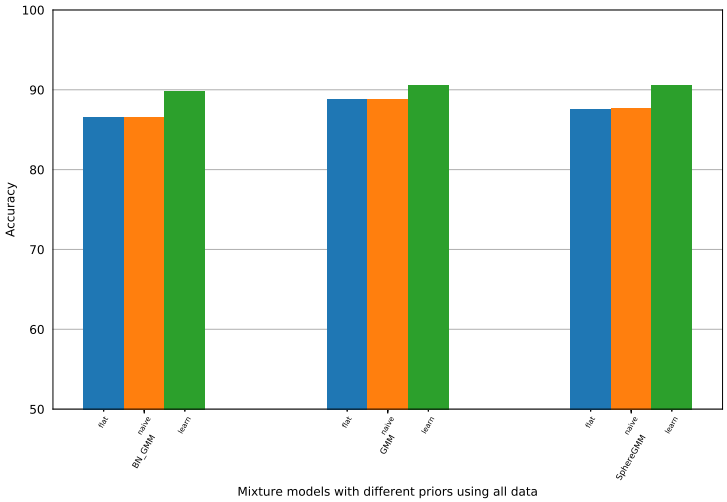


Figure 7.7: Using all of the data, the learnt priors had a moderate impact on the accuracy. However it still performed better than the other priors.

7.4 Exploration of Seating Behaviour

Originally a Probabilistic Graphical Model (PGM) was going to be used to classify students based on their seat positions, specifically a Markov Random Field. The original idea was to create a node at every student location and pass the triplet loss embedding and seat position as features. A PGM would be able to capture the intricate relationships between the students. The main issue with this is that each recording essentially gives 1 datapoint of seat positions. Each year has 8-10 recordings. That would not be enough data to sufficiently train a PGM. Additionally, most of the students only appear in 1 recording which means nearest neighbour data also cannot be used. It would be inconsistent across lectures. Instead of using a PGM, an analysis of the different seat behaviours was done for both years.

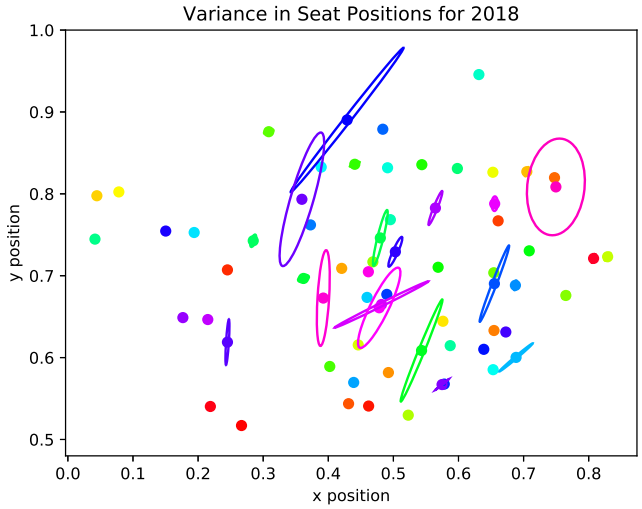


Figure 7.8: The average seating positions of students from the 2018 recordings. Each point is the average position of each student. The variance for each student is also shown.

Figure 7.8 shows the average seat positions for students and their variances. The students with higher variances tended to sit in more varied vertical regions. A potential reason for this could be because the 2018 venue was deep and more narrow. Students may have found a good viewing angle of the board and tried to keep that same angle throughout the year. The seats become closer towards the back of the venues so a small change in the x or y values near the top can actually correspond to a very big change in seat. Because of this it is harder to assign meaning to the x and y values than in 2019. In this case, 0.1 on the x-axis corresponds to about 3 seats and 0.1 on the y-axis corresponds to about 3 or 4 rows. Since the variances of the students positions were not high or overlapping much, it can be concluded that the students tended to sit in similar positions with only a few students sitting in very different positions over time. Moreover, students prioritised sitting in similar columns. This could mean that students prioritise certain viewing angles of the board over how far they are from the board. This in turn means that students that sit on one side of the venue tended to stay on that side for most of their lectures.

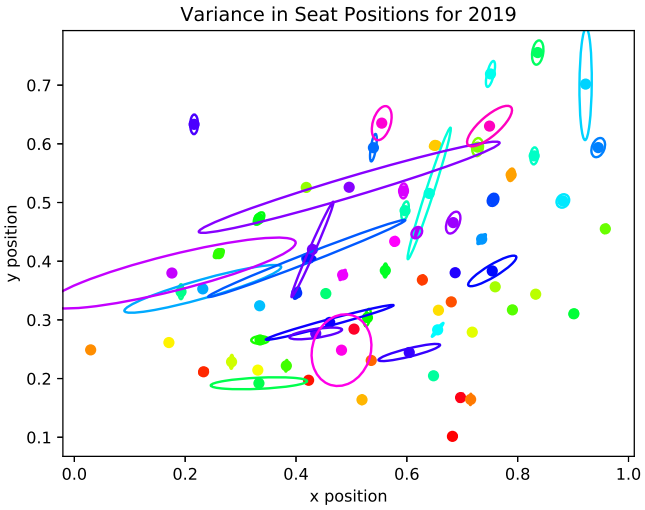


Figure 7.9: The average seating positions of students from the 2019 recordings. Each point is the average position of each student. The variance for each student is also shown.

The students in the 2019 recordings, shown in Fig 7.9, have very varied seating positions. The large and overlapping variances indicate that the students in 2019 sat in much more varied seating positions compared to 2018. The 2019 lectures were recorded in two different lecture venues. The venues in 2019 are narrower but much wider than the venues in 2018. The varying seats could be attributed to having been recorded in two different lecture venues as well as inconsistent camera positions across the venues. The higher consistency of the 2018 lectures could show that students are consistent within a single lecture venue but not between lecture venues. This could also show that students prefer to sit in similar seats in each lecture venue. This would indicate that having a model per lecture venue might produce the best results. Many of the students with high variances tended to sit within larger x position ranges. This could be due to friend groups which liked to sit along rows instead of sitting in small clusters.

Chapter 8

Conclusion

An end-to-end system has been proposed for the task of re-identifying students across lectures using not only their facial features but also their seating behaviours. Students were recorded with informed consent across various lectures over a long period of time where their facial images and seating positions were captured.

Using the facial images, a facial recognition network was trained with the triplet loss and quadruplet loss frameworks. Many variations of the standard triplet and quadruplet loss configurations were explored. Using this data, the batch-all triplet mining strategy performed better than the batch-hard strategy. Most applications in literature perform better using the batch-hard strategy but the strategies have varying performance across different dataset types. The embedding dimensionality used has a big impact on the performance. Typically, an embedding dimensionality of 128 is used, however with this data a dimensionality of 64 was found to be the best. Using ensemble methods, the HypersphereGMM or Nearest Neighbour allows for higher dimensionality embeddings to be used as they can still fit to high dimensional spherical data. The margin size had less of an impact on the performance than the embedding dimensionality. Larger margins tend to produce the best results up to a point. A margin of 0.5 is commonly used in literature and it was also the best performing margin size. Quadruplet loss attempts to improve over triplet loss by using adaptive margins. The original adaptive margins always became ≤ 0 and so the networks always collapsed. Additional adaptive strategies were developed in order for the network to train correctly with quadruplet loss. Using constant margins performed the worst for all classifiers and served as a good baseline. Linear growth of the margins performed the best across every classifier. The KL divergence strategy performed similarly for some classifiers. Interestingly, the same adaptive strategies did not perform well for triplet loss. In the case of quadruplet loss, the adaptive margins are essential despite the increased computation. Overall, the most consistent configuration across all the classifiers was triplet loss with 64 dimensional embeddings, a constant margin of 0.5 and the Euclidean distance metric. The highest performing configuration was quadruplet loss with 64 dimensional embeddings and the linear growth margin strategy using a secondary neural network for classification. The extra computation needed for quadruplet loss is not justified by its performance on this data.

After the facial recognition component was trained, the seating behaviour component was trained. The same classifiers used for the facial recognition component were used in the seating behaviour component. When using only the 2018 data, the results were very high. This showed

that the students preferred to sit in familiar seats over time. When using the 2019 data, the results were much lower. The 2019 recordings took place in multiple venues compared to just the singular venue in the 2018 data. This could show that students have regular seating habits for a lecture venue but not across different venues. In both cases, XGBoost performed the best.

Finally, the facial recognition and seating behaviour components were combined. Three different methods were discussed. Concatenation was the simplest method and also performed the best. The seat injection method performed slightly worse however, it may be able to outperform concatenation with increased metadata. The learnt prior method performed surprisingly well given that other priors for the mixture models provided no benefit. The reliability of the learnt priors relies on the accuracy of the neural network used to create them. A larger network with more sources of metadata could provide a significant performance increase. If a system like this were to be implemented as a production system, a network with standard triplet loss and simple concatenation would provide the best results.

Chapter 9

Future Work

This work was limited in terms of the amount of data per student. It would be beneficial to have more images per student in different settings. Additionally, more students appearing in multiple recordings as well as more recordings of students would create more nearest neighbour data. That way, nearest neighbour data can be included and more sophisticated methods can be trained, such as PGMs. Recordings of students in different lecture venues would help make the facial recognition more robust. The seat data was inconsistent for the 2019 recordings. Having a fixed camera position and using the same camera will help minimise any inconsistencies. Another issue with the seating data is that the further back the seats, the closer they are in the images. To help mitigate this, a homography can be applied to the venues first to fix the perspective. The methods for combining the information proved successful however the results may differ if more sources of metadata were to be collected.

Bibliography

Amos, Brandon, Bartosz Ludwiczuk, Mahadev Satyanarayanan, et al. (2016). "Openface: A general-purpose face recognition library with mobile applications". In: *CMU School of Computer Science* 6.

Chen, Weihua et al. (2017). "Beyond triplet loss: a deep quadruplet network for person re-identification". In: *CoRR* abs/1704.01719. arXiv: 1704.01719. URL: http://arxiv.org/abs/1704.01719.

Dong, Chao et al. (2015). "Image super-resolution using deep convolutional networks". In: *IEEE transactions on pattern analysis and machine intelligence* 38.2, pp. 295–307.

Gallagher, Andrew C and Tsuhan Chen (2009). "Understanding images of groups of people". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 256–263.

Hermans, Alexander, Lucas Beyer, and Bastian Leibe (2017). "In defense of the triplet loss for person re-identification". In: *arXiv preprint arXiv:1703.07737*.

Hughes, Michael C and Erik B Sudderth (2014). "Bnpy: Reliable and scalable variational inference for bayesian nonparametric models". In: *Proceedings of the NIPS Probabilistic Programming Workshop, Montreal, QC, Canada*, pp. 8–13.

Johnson, Justin, Alexandre Alahi, and Li Fei-Fei (2016). "Perceptual losses for real-time style transfer and super-resolution". In: *European conference on computer vision*. Springer, pp. 694–711.

Klein, Richard and Turgay Celik (2017). "The Wits Intelligent Teaching System: Detecting student engagement during lectures using convolutional neural networks". In: *2017 IEEE international conference on image processing (ICIP)*. IEEE, pp. 2856–2860.

Miolane, Nina et al. (2018). "geomstats: a Python Package for Riemannian Geometry in Machine Learning". In: *arXiv preprint arXiv:1805.08308*.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Prince, S.J.D. (2012). *Computer Vision: Models Learning and Inference*. Cambridge University Press.

Sapkota, Archana et al. (2013). "Appearance, context and co-occurrence ensembles for identity recognition in personal photo collections". In: *2013 IEEE Sixth International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, pp. 1–8.

Schroff, Florian, Dmitry Kalenichenko, and James Philbin (2015). "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *CoRR* abs/1503.03832. arXiv: 1503.03832. URL: http://arxiv.org/abs/1503.03832.

Shernoff, David J. et al. (2017). "Separate worlds: The influence of seating location on student engagement, classroom experience, and performance in the large university lecture hall". In: *Journal of Environmental Psychology* 49, pp. 55–64. ISSN: 0272-4944. DOI: https://doi.org/

10.1016/j.jenvp.2016.12.002. URL: http://www.sciencedirect.com/science/article/pii/S0272494416301086.

Sirovich, Lawrence and Michael Kirby (1987). "Low-dimensional procedure for the characterization of human faces". In: *Josa a* 4.3, pp. 519–524.

Stone, Zak, Todd Zickler, and Trevor Darrell (2008). "Autotagging facebook: Social network context improves photo annotation". In: *2008 IEEE computer society conference on computer vision and pattern recognition workshops*. IEEE, pp. 1–8.

Straub, Julian et al. (2015). "A Dirichlet process mixture model for spherical data". In: *Artificial Intelligence and Statistics*, pp. 930–938.

Szegedy, Christian, Wei Liu, et al. (June 2015). "Going Deeper With Convolutions". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Szegedy, Christian, Vincent Vanhoucke, et al. (2016). "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

Turk, Matthew and Alex Pentland (1991). "Eigenfaces for recognition". In: *Journal of cognitive neuroscience* 3.1, pp. 71–86.

Viola, Paul and Michael Jones (2001). *Rapid object detection using a boosted cascade of simple features*.

Wang, Gang et al. (2010). "Seeing people in social context: Recognizing people and social relationships". In: *European conference on computer vision*. Springer, pp. 169–182.

Wang, Xiaolong et al. (2017). "Leveraging multiple cues for recognizing family photos". In: *Image and Vision Computing* 58, pp. 61–75.

Wang, Xintao et al. (Sept. 2018). "ESRGAN: Enhanced super-resolution generative adversarial networks". In: *The European Conference on Computer Vision Workshops (ECCVW)*.