Manu Varma

Kin Recognition Using Weighted Graph Embeddings

Computer Science Tripos – Part II

St John's College

April 1, 2021

Declaration

I, Manu Varma of St. John's College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed Manu Varma

Date April 1, 2021

Proforma

Name:	**Manu Varma**
College:	**St John's College**
Project Title:	**Kin Recognition Using Weighted Graph Embeddings**
Examination:	**Computer Science Tripos – Part II, June 2021**
Word Count:	**TBA**
Line Count:	**2907**
Project Originator:	**The Dissertation Author**
Supervisor:	**Daniel Bates**

Original Aims of the Project

Work Completed

Special Difficulties

Contents

List of Figures

Chapter 1

Introduction

Kinship recognition is the way of recognizing whether two given people are related to each other or not based on how they look. This is an evolutionary trait in humans as it is advantageous to inclusive fitness for an organism to be able to recognize which of their neighbors were close relatives [5]. Thus, it stands to reason that the ability to recognize kinship relationships has evolved in humans. In humans, specifically, facial resemblance is expected to serve as an indicator of kinship as we have seen that strangers are able to match photographs of mothers to their infants without any prior contact with any of the family [11].

Computational kinship recognition is the field of computationally figuring out kinship relationships between people without any prior knowledge of the family.

1.1 Problem Overview

There are multiple problems in the field of computational kin recognition, in which case I will focus on one of them. The first of which takes as input a pair of images and a proposed kinship relationship, for example father-daughter, and recognizes whether the relationship exists. These relationships tend to be more commonly parent-child relationships as opposed to sibling-sibling relationships, for example, which are less commonly used. A further extension of the main kin recognition problem is Tri-Subject Kinship which takes 3 images, two parents and a child, and determines if they are related or not. These problems are the ones that are being looked at in this dissertation.

However, some other major problems in the field include search-and-retrieval and family classification. Search-and-retrieval is the problem which takes an image of a person as input and searches through a database to find people with whom they are most likely to be related to. This outputs a list of these people that they could be related to. Family classification deals with a similar task of taking an image of a person as input and figuring out which family they may belong to.

1.2 Motivation

Accurate kinship recognition software has multiple applications in humanitarian issues. With regards to humanitarian issues, by using kin recognition, we can better recognize missing children and match them with their parents [15]. It can also be used for stopping human traffickers from claiming they are family members of the victim or to reunite families across refugee camps.

Furthermore, there are potential use-cases in social media and also poses privacy protections.

1.3 Related Work

One of the first papers in the field of computational kinship recognition used color, facial parts, facial distances and a Histogram of Gradients vector as the features for each image. Using these features, K-Nearest-Neighbors and an SVM were used in order to classify the image pairs into true and false parent-child pairs [4]. A classification accuracy of 70.67% was obtained overall and an SVM with a radial basis kernel obtained an accuracy of 68.6%.

Other approaches include using deep learning to solve the problem. One paper used a Siamese CNN approach by putting both of the face images in the pair through a SqueezeNet network which was trained on VGGFace2 which creates a feature vector for each image [12]. Using a similarity criterion, a new feature vector is created using the two feature vectors and a fully connected layer and a sigmoid activation function creates the predicted similarity score. This approach yielded an average test accuracy of 67.66% over all of the relationships that were in the dataset. Another paper that used a Siamese approach aimed to solve both the standard kinship verification problem and the Tri-Subject problem [18]. Both networks for the problems had three stages, a feature extraction stage, which used the ResNet50 or SENet50 models pretrained on VGGFace2 to extract the features from each face into a vector, a feature fusion stage which combines the feature vectors together, and a similarity quantization stage to get the similarity score. They then use a jury system to fuse together models which allows them to achieve an accuracy of 75.9%.

However, a prominent approach that is taken includes using metric learning to solve the problem. For example, the paper on Neighborhood Repulsed Metric Learning [10] uses a supervised metric learning approach. Using the approach, they aim to find a metric which minimizes the distance between the vectors of images that have a kinship relationship and maximize the distances between the vectors of pairs of images that don't have a kinship relationship. Furthermore, the paper that will be implemented in this dissertation, using Weighted Graph Embedding-Based Metric Learning, involves a metric learning approach to the problem [7].

1.4 Project Overview

The project deals with a 2019 method using Weighted Graph Embedding-Based Metric Learning [7], or WGEML for short, and aims to implement the paper and verify the accuracies that were obtained. The following is shown in the dissertation:

1. The algorithm, WGEML, is implemented and accuracies that are within an acceptable range are obtained, as shown in section 4.2.

2. Ablation studies on the face descriptors are performed and the results are in section 4.4.1.

3. We use the models that were created to discover biases in the datasets that were used and find out the impact that has overall in section 4.3.

4. We further test the face descriptors by replacing VGG with a smaller CNN in which the implementation is discussed in section 3.3.3.

Chapter 2 explains much of the needed technical background for the implementation, from what a Neural Network is and what metric learning is to each of the face descriptors used and what face descriptors are. Chapter 3 will then discuss the specifics of how WGEML and the extension that used a different network than VGG was implemented. Chapter 4 then discusses all of the results that were obtained from experimentation with WGEML and the implications of the results.

Chapter 2

Preparation

In this chapter, I discuss the necessary technical background information needed for the project in sections 2.1 through 2.5, then the requirements of the project are elaborated on in section 2.6 before the starting point and Software Engineering practices are discussed in section 2.7.

2.1 Convolutional Neural Networks

We must first discuss what a regular neural network is before going onto what a Convolutional Neural Network (CNN) is.

2.1.1 Convolutional Layers

2.2 Face Detection

Face detection is the task of finding a face within a given image and returning the set of faces found as seen in Figure 2.1. This differs from face recognition since the task is only to find the faces and not to recognize who the faces belong to.

Figure 2.1: Face detection example

2.3 Face Descriptors

We wish to be able to create a mapping from a colored image into a vector to make computations easier and to be able to determine similarity between images, which we do using the distance. These are called image descriptors for general images. When we try and map face images to vectors, we call these face descriptors. We want these face descriptor mappings to be able to match the same person in different poses and illuminant geometries to face descriptors that are close to each other in distance. Thus, these mappings try and capture color, texture, and shapes, for example. There are multiple face descriptors that can be made of a face image, each of which extracts different features from the face. We use the Local Binary Patterns, Histogram of Gradients, Scale-Invariant Feature Transform and VGG face descriptors to extract features from each face.

2.3.1 Local Binary Patterns

One such face descriptor is the Local Binary Patterns (LBP) visual descriptor which is adapted for faces. LBP is a texture descriptor [1] which means that the algorithm attempts to describe the texture of the image, rather than anything to do with the color. As such, given an image we want the descriptor of, we must first convert it to grayscale. Then, for each pixel in the image, a neighborhood of pixels is obtained from it. There are multiple ways to define this neighborhood, as shown in figure 2.2.

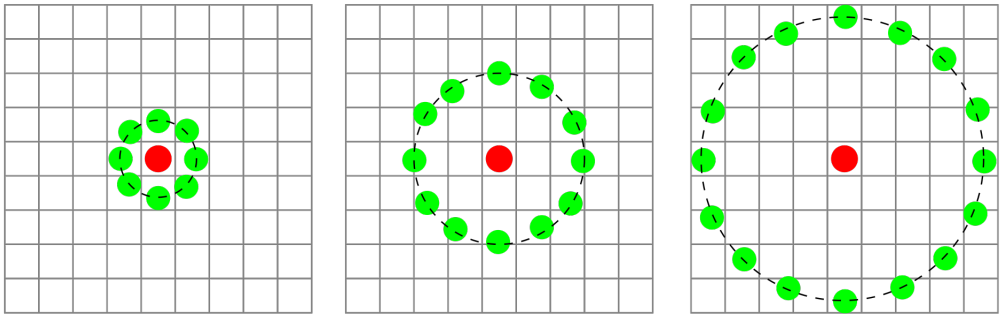

Figure 2.2: Potential neighborhoods of the pixel

The simplest neighborhood, which is the neighborhood used in this project, is the direct neighbors of the pixel, which is the first neighborhood in figure 2.2. However, on the edges, some of the neighbors won't exist, due to the fact that it's on the edge of the image. To mitigate this, in the implementation, I pad the grayscale image with 0s on the outside of the image such that each pixel in the image has the same number of neighbors. Once we have our neighborhood of the pixel, we compare the grayscale value of the main pixel with each of the grayscale values in the neighborhood. For each neighboring value, if it is greater than the main pixel, we make it a 1, otherwise we make it a 0. We are then able to create a binary number from the neighboring values. In practice, where the number is started from doesn't matter but in my implementation, the number starts from the left cell and goes counterclockwise. Applying this to figure 2.3, we get that our LBP value for this pixel would be 01111000_2 which, in decimal, is 120.

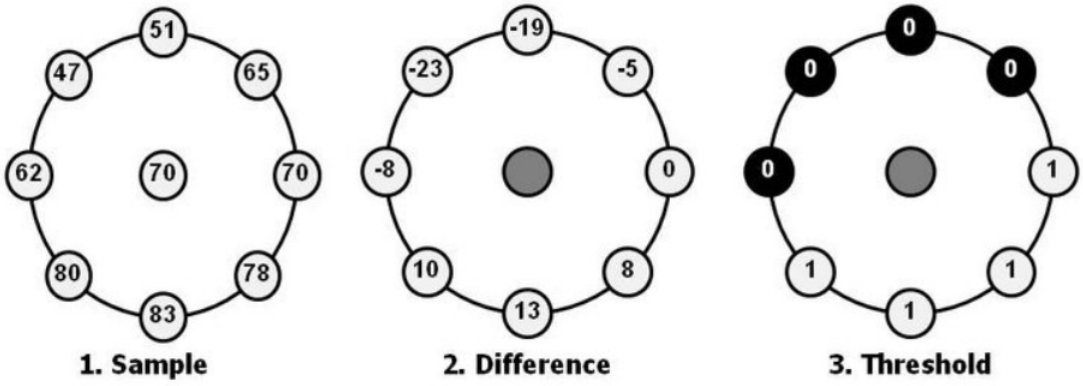

Figure 2.3: Example of LBP operator on a pixel

To summarize the LBP operator mathematically, given a coordinate in the image, (x, y) which has grayscale value g, a neighborhood of P points with radius R enumerated by g_p where

$p \in \{0, P-1\}$, we have that, [2]:

$$LBP_{P,R}(x,y) = \sum_{p=0}^{P-1} s(g_p - g)2^p$$

Where:

$$s(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Once we get the values of the pixel, an added extension, which we do for face description, is to check whether it is a *uniform value* which we define as a value such that, in binary, there are only, at most, 2 bitwise transitions when traversed circularly. For example, 11000111 is a uniform value since it only transitions from 1 to 0 and 0 to 1 but 11001000 isn't since it has 4 bitwise transitions. With this extension, we have 58 possible uniform values that a pixel's LBP value can be and an extra value for the LBP value not being uniform. In other words, there are 59 LBP values that a pixel in an image can take.

Once each pixel in the image has an associated LBP value, we can split up the image into blocks. For our implementation, since our input images have size 64×64, we split it up into non-overlapping blocks of size 8×8, of which there are 8×8 of. For each block, we obtain a histogram of the LBP values that were in the block, which gives us a 59-dimensional vector. To obtain the vector for the entire image, each of these vectors are appended together and a 3776-dimensional LBP face descriptor is obtained for our case.

2.3.2 Histogram of Gradients

Another face descriptor is Histogram of Gradients (HOG) [3]. Given a colored image, we can think of the image as a function, $I : \mathbb{N}_m \times \mathbb{N}_n \to \mathbb{R}^3$, which takes a coordinate in the image and returns a vector of values, which correspond to the red, green, and blue values of the pixel. This then means that we are able to calculate the gradient of the image. The gradient of an image can characterize local object appearance and shape due to the fact that the gradient of the image can be used to help find edges in the image. As such, it is useful to obtain such a histogram of gradients.

2.3.2.1 Calculating the Gradients

First, we need to calculate the gradients of the image. As the image function is discrete, in other words we cannot analytically find the gradients, we must find approximations to do so. This is done by convolving specific kernels on the image. What this means is that the kernel, K which is a matrix, is applied to each pixel in the image and its neighbors and the values are multiplied with the corresponding value in the kernel and all of the values are then summed up to create the new value for the convolved image. For example, if we had the kernel:

$$K = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

And the image:

$$I = \begin{bmatrix} 0 & 5 & 4 \\ 3 & 3 & 3 \\ 2 & 1 & 2 \end{bmatrix}$$

We get that the kernel convolved on the image is:

$$K * I = \begin{bmatrix} 0 \times 1 + 5 \times 2 + 4 \times 1 \\ 3 \times 1 + 3 \times 2 + 3 \times 1 \\ 2 \times 1 + 1 \times 2 + 2 \times 1 \end{bmatrix} = \begin{bmatrix} 14 \\ 12 \\ 6 \end{bmatrix}$$

We can then approximate the derivative of an image as a kernel being convolved on the image. To do so, we use the kernels:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$
$$G_y = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T$$

By convolving these kernels on the image, we are able to get an estimate for the derivative in the x direction and in the y direction, respectively. Abusing notation slightly where here the square of the matrix just means an element-wise square and division just means element-wise division, we get that the magnitude of the gradient is:

$$\sqrt{(G_x * I)^2 + (G_y * I)^2}$$

And that the angles at each point are:

$$\tan^{-1}((G_y * I)/(G_x * I))$$

However, there are two things that need to be addressed before we move on to the rest of the algorithm. Firstly, there are still 3 channels for the image, so we have obtained the gradient in the x and y direction for each pixel and each channel. As such, we define the gradient of each pixel to be the gradient which has the maximum magnitude among the 3 channels and the corresponding angle is used as well.

Furthermore, the angles returned range between $0°$ and $360°$. However, we require that the angles be unsigned, so the angle at each pixel, $\theta_{(x,y)}$, becomes:

$$\theta_{(x,y)} \texttt{ := } \theta_{(x,y)} \mod 180$$

2.3.2.2 Weighted Vote into Histogram

At this point, we have the magnitude of the gradient at each point in the image as well as the angle of the gradient. Now, similarly to LBP, we break up the image into blocks. We create a histogram for each of these blocks and append them together to make the face descriptor for the entire image. Unlike LBP however, we don't have a finite set of labels that each pixel can neatly fall into however, since neither the magnitude nor the angles are discrete. Thus, first, the labels of the histogram are going to be the angles of the gradients. The labels will then be:

$$[0, 20, 40, 60, 80, 100, 120, 140, 160]$$

However, these aren't blocks ranges of $[0, 20)$, for example. Instead, for each pixel in the block, we distribute the magnitude of the gradient of the pixel between the angles that the angle falls between. Given a pixel with magnitude m and angle θ, if $0 \equiv \theta \mod 20$ then:

$$\texttt{histogram[}\theta/20\texttt{] += } m$$

Otherwise, we have it that θ is between two angles, ϕ_1 and ϕ_2, both of which are divisible by 20, where $\phi_1 < \theta < \phi_2$. In this case, we weight the amount that we add to each label based on how far away θ is to the label. So, for the label ϕ_1, we have that we add to the label associated with ϕ_1 the value:

$$\frac{\phi_2 - \theta}{20} \times m$$

And, similarly for ϕ_2:

$$\frac{\theta - \phi_1}{20} \times m$$

We can see an example of this in figure 2.4.

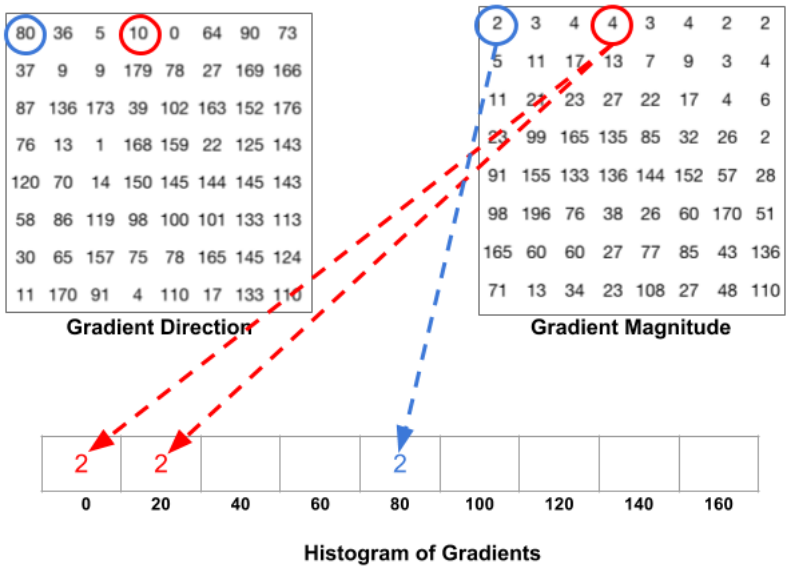

Figure 2.4: Creating a histogram for a block of the image

In other words, the closer the angle is to the label, the more of the magnitude is contributed to the label's histogram value. As a caveat, if $\phi_2 = 180°$, we treat ϕ_2 as $180°$ for the sake of this calculation but we add the value to the label 0 since $0 \equiv 180 \mod 180$. An example of this can be seen in figure 2.5.

We can do this for each pixel in the block and, thus, we are able to get a 9-dimensional vector for each block. In the project, the image is split up into 16×16 blocks of size 4×4 first and then 8×8 blocks of size 8×8 next and each of these blocks contributes a histogram to the overall vector which leads us with a face descriptor with dimension:

$$16 \times 16 \times 9 + 8 \times 8 \times 9 = 2880$$

2.3.3 Scale-Invariant Feature Transform

SIFT [8] is another way of obtaining face descriptors, although the original SIFT algorithm differs from how it is used in the project, though it is still important to briefly go over.

The first thing to do is to create octaves for the given image. An octave is a set of the given image being blurred multiple times. For example, in the first octave, the original image is the first image, and then the next image is blurred slightly which is then blurred further for the

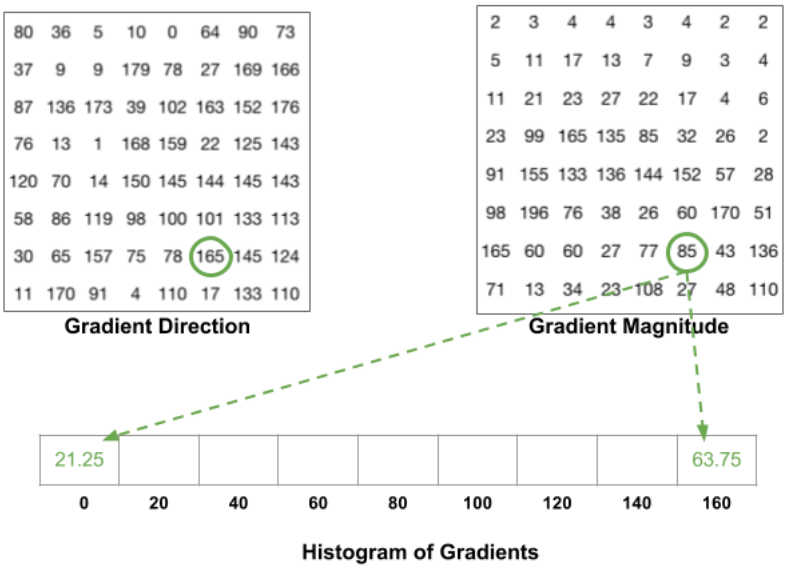

Figure 2.5: What happens in the case where $\phi_1 = 160$ and $\phi_2 = 180$

next image in the octave, etc. In the second octave, the image is halved in size and the same blurring effect happens. So, if the original image was of size 64×64, then the images in the second octave will be 32×32 and in the third it would be 16×16 and so on. A specified number of octaves and blurred images are used for the SIFT algorithm. The way that the image is blurred is as follows:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Where x, y is the coordinate in the image, I is the function mapping coordinates to the value of the image at that coordinate, σ is the amount of blurring, $*$ represents convolving G on the image and:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Which is the Gaussian blur.

From here, for each octave, a difference of Gaussians is created to help find the keypoints of the image. The difference of Gaussians is just the difference between the consecutive Gaussians. This can be visualized in figure 2.6.

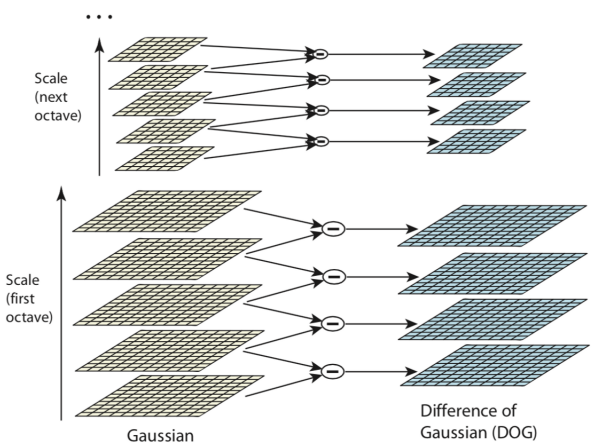

Figure 2.6: The Difference of Gaussians being created

This is used to approximate the Laplacian of Gaussians as the LoG helps find the edges of the image by blurring the image a bit and then finding the second order derivatives. It is first blurred as taking the Laplacian straight away would be sensitive to noise. However, this is computationally expensive. The Difference of Gaussians is a good approximation of the scale invariant Laplacian, $\sigma^2\nabla^2 G$.

From here we look for the keypoints in the image using the Difference of Gaussians. This is done by finding the local maxima and minima of the DoG. Once the extrema of the Difference of Gaussians are obtained, we need to refine the approximation of the keypoint because the actual keypoint is more likely to be between pixels. Thus, we can use a Taylor expansion around the proposed keypoint of the scale-space function $D(\sigma, x, y)$ where σ is the blur level in the DoG. This Taylor expansion looks like:

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^\mathbf{T}\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

Where D is the value of D at the proposed keypoint, $\mathbf{x} = (\sigma, x, y)^T$, $\frac{\partial D}{\partial \mathbf{x}}^T = (\frac{\partial D}{\partial \sigma}, \frac{\partial D}{\partial x}, \frac{\partial D}{\partial y})$. Letting this equal 0, we get that the offset from our keypoint is:

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1}\frac{\partial D}{\partial \mathbf{x}}$$

If our offset is greater than 0.5 in any dimension, then we want to try again since that means it's closer to another sample point. We keep trying again until we get an offset which is close to the sample point. We then find the value at the subpixel extrema:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D}{\partial \mathbf{x}}^T \hat{\mathbf{x}}$$

And if the value of the extrema is less than 0.03×255, then we throw it out since it is an unstable extrema and has low contrast.

Furthermore, we also eliminate any keypoints that are potentially on an edge. We can do this by looking at the Hessian of the keypoint. We have that the Hessian is:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix}$$

To determine whether something is a corner, we care about the eigenvalues of $\mathbf{H}$, or more specifically, the ratio between the eigenvalues. Using the trace and determinant of $\mathbf{H}$, we can find that:

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(r+1)^2}{r}$$

Where r is the ratio of the eigenvalues. Thus, letting the maximum ratio that the eigenvalues are allowed to be at be $r_0 = 10$, we just need to find if:

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r_0+1)^2}{r_0}$$

If it is, then this is a proper keypoint. If it isn't then this is an edge so we can discard it.

We now have each of the keypoints which are scale-invariant so we need to make it rotation-invariant as well. This is done by assigning an orientation to the keypoints. A neighborhood is taken around the keypoint in which the gradient is obtained for each pixel in the area and

a histogram of these gradients is created based on the angle, this time in bins of 0 to 10, 10 to 20, etc. until 350 to 360. The peak in this histogram is calculated and taken and any peak above 80% of it is also considered to calculate the orientation.

Now, for each keypoint, we have the location, scale and orientation so we can finally create the descriptor. A 16×16 window is obtained around the keypoint and divided into 16 blocks. For each block, a histogram of the gradients is taken with 8 bins which are then appended together which gives a 128-dimensional vector for the keypoint. The SIFT descriptor of the image is then the vectors for each keypoint appended together to create a $128n$-dimensional vector where n is the number of keypoints in the image.

The actual implementation of SIFT that is used differs from the original version which is talked about in section 3.3.1.

2.3.4 VGG

Another way of getting face descriptors for an image is to use the VGG network. The original VGG network [16] is a CNN which takes in a 224×224 colored image and outputs a probability vector of size 1000 for the ImageNet classes. In other words, if `out` is the output of the model, `out[i]` corresponds with the probability that the $\texttt{i}^{\text{th}}$ ImageNet class is the class of the input image. There are 5 configurations of the network, which have varying depth and, as a result, more parameters. The architecture of the second deepest configuration is configuration D in Figure 2.7.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.7: The different configurations that VGG can have. Image Reproduced from Simonyan et. al (2015) [16]

By training this model on the ImageNet dataset, the model is able to get a top-5 classification error of 7.5%, which means that for 7.5% of the entries in the test set, none of the top 5 classes

that the image could be were the actual image.

However, in order to use the model for face description, we must change the output layer and the training dataset [13]. Instead of training on general objects in an image, we train the model on faces, specifically celebrity faces. The celebrities are curated to a list of 2622 people in which 2000 images are obtained for each celebrity and the images are then curated. The curated set of images constitutes the training set for the VGG model for faces.

The VGG model for faces is then the model described above with the softmax layer having dimension of 2622. The model is then trained with the dataset we described which allows us to get rid of the softmax layer and use the output of the last fully-connected layer as our 4096-dimensional face descriptor for the image.

2.4 Metric Learning

We wish to measure the similarity between a pair of faces in order to determine whether they are related or not. One way to approach this is to use similarity learning, which is a type of approach to the problem in which the goal is to learn a similarity function in order to measure how similar two objects are. However, we often use distance to help measure the similarity between data points. By learning what this distance function ought to be, we would be able to find a better similarity function so finding this distance function is the goal of metric learning [17]. To go more in depth, we must first define a few concepts.

Given a non-empty set A, we define a *distance* over A as a function $d : A \times A \to \mathbb{R}$ such that the following holds:

1. $\forall x, y \in A$, $d(x, y) \geq 0$ and $d(x, y) = 0$ if and only if $x = y$.
2. **Symmetry**: $\forall x, y \in A$, $d(x, y) = d(y, x)$
3. **Triangle Inequality**: The triangle inequality must hold. That is, $\forall x, y, z \in A$:

$$d(x, y) + d(y, z) \geq d(x, z)$$

Given the definition of a distance, we define a *Mahalanobis distance* that corresponds to the positive semidefinite matrix M to be a distance function, $d_M : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, where d is the number of dimensions the input vector has, which is defined as:

$$d_M(x, y) = \sqrt{(x - y)^T M (x - y)}$$

If M is positive semidefinite, then M must be decomposable into $B^T B$ where B is a real matrix. Thus, we can also write the Mahalanobis distance as:

$$d_M(x, y) = \sqrt{(x - y)^T M (x - y)} = \sqrt{(x - y)^T B^T B (x - y)} = \sqrt{(Bx - By)^T (Bx - By)}$$

Thus, given a dataset $\mathcal{X} = \{x_1, ..., x_n\} \subset \mathbb{R}^d$ which has the sets:

$$S = \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \mid x_i \text{ and } x_j \text{ are similar}\}$$
$$D = \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \mid x_i \text{ and } x_j \text{ are not similar}\}$$

We wish to find the distance metric M such that we can minimize some loss function, l:

$$\min_M l(d_M, S, D)$$

In other words, in metric learning, we wish to find out what the matrix M should be defined as in order to make data points which are similar closer together and data points which aren't similar further away, which is encapsulated by the loss function. We can visualize this in figure 2.8.

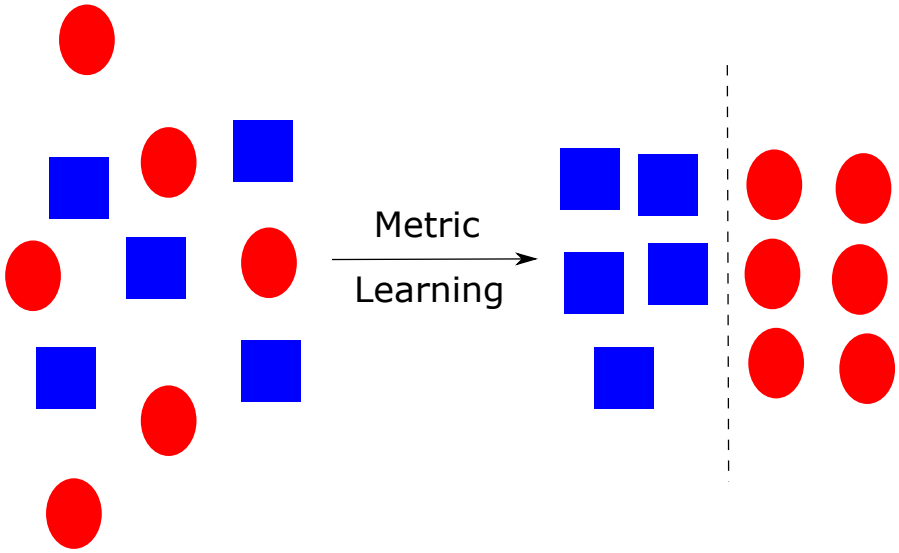

Figure 2.8: A high-level view of metric learning

2.5 Side Algorithms Used (Temp Name)

Aside from the main algorithms that will be used which were described earlier, a couple of high-profile algorithms were used that deserve to be explained, although they are used only a few times at most in very specific places.

2.5.1 K-Nearest Neighbors

K-Nearest Neighbors is an algorithm that finds the K nearest neighbors for each datum in a given set of data. Given a list of datapoints in Euclidean space, we wish to find, for each datapoint, the K closest datapoints to it.

2.5.2 Principal Component Analysis

Principal Component Analysis (PCA) [6] is a statistical tool that we can use to reduce the dimensionality of a dataset while maintaining as much of the variability as possible. The algorithm identifies *principal components* which are the directions of maximum variance in the dataset, which can be visualized in figure 2.9.

Let p be the number of variables that are being measured in the dataset, which in our case is the dimension of the face descriptor. Let $\mathbf{X}$ be the matrix $n \times p$ data matrix that is composed of the p n-dimensional vectors $\mathbf{x_1}, ..., \mathbf{x_p}$ in which x_i represents the n observations of the ith variable. The goal of PCA is then to create a $p \times k$ matrix $\mathbf{W}$, where $k \leq p$, which maps each row-vector of $\mathbf{X}$ from the original p-dimensional feature space to a new k-dimensional feature subspace such that the variability of the dataset is kept in tact, as much as possible.

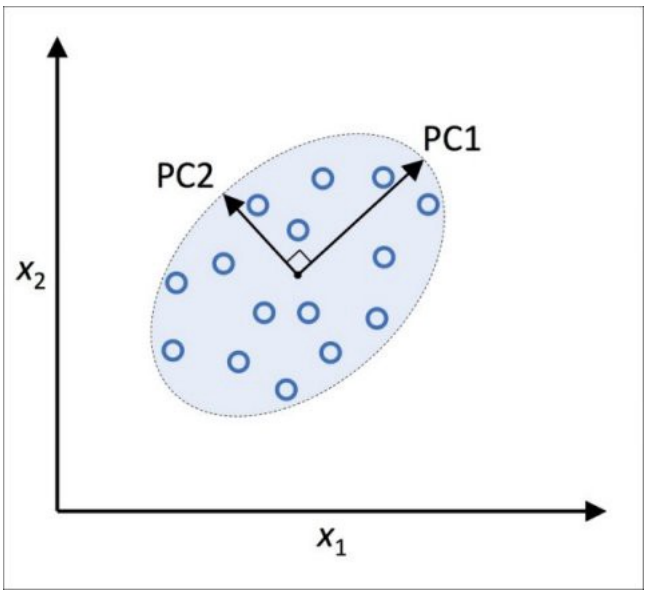

Figure 2.9: Principal components of an arbitrary 2 dimensional dataset

First, the dataset must be standardized which means that the mean of each feature must be 0, which we denote as the matrix $\mathbf{X}'$ and the column vectors as $\mathbf{x'_1}, ..., \mathbf{x'_p}$. From there, the covariance matrix is obtained. This matrix is the following:

$$\Sigma = \begin{bmatrix} Cov(\mathbf{x'_1}, \mathbf{x'_1}) & ... & Cov(\mathbf{x'_1}, \mathbf{x'_p}) \\ \vdots & \ddots & \vdots \\ Cov(\mathbf{x'_p}, \mathbf{x'_1}) & ... & Cov(\mathbf{x'_p}, \mathbf{x'_p}) \end{bmatrix}$$

Where:

$$\Sigma_{ij} = Cov(\mathbf{x'_i}, \mathbf{x'_j}) = \frac{1}{n}(\mathbf{x'_i} \cdot \mathbf{x'_j})$$

Since we have that the mean of both are 0 now, after standardization. Thus:

$$\Sigma = \frac{1}{N}\mathbf{X'}\mathbf{X'}^T$$

We then find the eigenvalues and eigenvectors of Σ, of which we get p of them. The eigenvalues get sorted from largest to smallest and, thus, the first principal component is the eigenvector that corresponds to the largest eigenvalue, the second principal component is the eigenvector with the second largest eigenvalue, etc. Denoting $\mathbf{w_i}$ as the ith principal component, we can make the matrix $\mathbf{W}$ as:

$$\mathbf{W} = \begin{bmatrix} \mathbf{w_1} & ... & \mathbf{w_k} \end{bmatrix}$$

Which is a $p \times k$ matrix. Thus, to reduce the dimensions of the original dataset, we can create the matrix:

$$\mathbf{T} = \mathbf{XW}$$

In which the ith row of $\mathbf{T}$ corresponds with the ith observation of $\mathbf{X}$ and this vector has is k-dimensional where $k \leq p$.

2.6 Requirements Analysis

The main requirement of the project is stated in Appendix A which is to replicate the results from Liang et al. (2019) [7] within a 15% error range or to reject the results. In other words, the project should implement the Weighted Graph Embedding-Based Metric Learning (WGEML) algorithm for Kin Recognition. The core project can be broken down into the following requirements:

- **Face Detection**: Given an image, the faces in the image must be identified and saved as a 64×64 image on disk.
- **Face Descriptors**: The face descriptors, LBP, HOG, SIFT and VGG, must be implemented such that, given an image, each of these face descriptors should be able to be obtained from them.
- **WGEML**: The WGEML algorithm must be implemented such that, given a relationship, the positive pairs of faces, the negative pairs of faces, if any, it returns the distance metric matrix for each face descriptor used and how much each face descriptor should be weighted in the prediction step.
- **Prediction**: Given the model obtained from the WGEML algorithm, a pair of images, and a relationship, it must return how similar the images are and whether the people in the images have the kin relationship given.
- **End-to-end Replication**: Running the project end-to-end on the given datasets either gives similar results to the original paper or rejects the original results.

The extensions can be summarized as follows:

- Replace the VGG face descriptor with a smaller network to see how it affects accuracies.
- Look at how biases in the datasets affect the results and identify these biases.
- Use different combinations of face descriptors for WGEML to see how it affects accuracy of the model.

Analysis There were two main phases for the project, the phase to finish the core requirements and one for the extensions. Each of the first 4 core requirements were split up into their own modules and thus constituted its own sub-phase in the main part of the project. They were also linear in that each requirement depended on the output of the last modules. The last requirement constituted of a medley of scripts which were used to integrate the functions that were created for each of the requirements with the datasets, which required a data preparation module.

2.7 Software Engineering Practices

2.7.1 Starting Point

Before I started my project, I had knowledge in Python and surface-level knowledge of Keras and Tensorflow. I had also worked with face recognition before. Furthermore, I had used

Numpy many times before, so I had enough knowledge about it to use it comfortably in my project. However, I hadn't ventured into computer vision before the project aside from knowing very generally what a face descriptor is.

2.7.2 Tools Used

The project was written in Python 3.6. In order to separate the environment that the project needs with my local environment, I used a virtual environment to contain the installations of the required libraries. I also had a requirements.txt file to contain the names and versions of the packages that were used. I also used the following libraries:

1. **Numpy**: Using numpy helped increase performance of many parallel computations, such as matrix multiplication, and provided a simple interface to do these with.

2. **Keras and Tensorflow**: I used Keras to create the VGG model and the CifarNet model. We then use it to train the CifarNet model and then make predictions for both models. The weights of VGG were already pre-computed so no training was necessary for VGG.

3. **OpenCV**: I used OpenCV for general image processing, such as converting an image into grayscale or loading in images, and for face detection.

4. **Sklearn**: This library provided an implementation for K-Nearest Neighbors and PCA.

5. **Scipy**: This library provided a function that solved the general eigenvalue problem, given two numpy arrays which was used in the main WGEML algorithm.

Finally, Git and Github were used for version control and backups. Branches were created for each feature that was to be added to the project and I merged them into the master branch once enough testing was done, whether it was unit testing or integration testing.

2.7.3 Datasets

I used the KinFaceW-I, KinFaceW-II, and TSKinFace datasets to train the model.

The KinFaceW datasets [9] [10] are composed of face images from the internet which include public figures as well as their parents or children. Both datasets contain no restriction on pose, lighting background, expression, age, ethnicity, or partial occlusion. The main difference between the datasets is that the pairs of face images that have a kin relationship in KinFaceW-I are from different images whereas, in KinFaceW-II, they are from the same image. In terms of the specifics of the datasets, they both support four kin relationships, Father-Son (FS), Father-Daughter (FD), Mother-Son (MS), and Mother-Daughter (MD). Each face image are images of size 64×64. The datasets also contain pre-computed 5 folds for cross-validation for each relationship which contained the names of the pairs of images that had the relationship and those that didn't, henceforth positive and negative pairs respectively. Finally, the dataset has 2 main settings which are used: the restricted setting in which only positive pairs of images are used and the unrestricted setting in which negative pairs of images are used.

The TSKinFace dataset [14] contains images for the relationships, Father-Mother-Son (FMS), Father-Mother-Daughter (FMD), and Father-Mother-Son-Daughter (FMSD). The dataset contained folders for each relationship and a positive set comprised of the images

in which the names of the images had the form "[relationship]-N-[member].jpg" in which "relationship" was the relationship, N was a consistent number and "member" refers to which member of the relationship they were. For example, "FMS-10-F.jpg", "FMS-10-M.jpg", and "FMS-10-S.jpg" would form a positive triplet. The dataset only contained the images in this form so negative pairs of images and the folds for cross-validation had to be computed in the project.

I examine the effects of the WGEML algorithm on the FS, FD, MS, MD, FMS and FMD relationships, as defined above.

2.7.4 Testing

Throughout the project, I created unit tests for any modules and for each function in the module. I would only end up merging a feature branch into the master branch if all of the unit tests passed for that feature.

Along with this, I did integration tests in the form of feeding the scripts small, controlled inputs to see if the scripts would output what was expected. These scripts each used functions from different modules and did a part of the workflow.

Finally, an end-to-end test was done once each script was tested individually which came in the form of trying the small, controlled input for the first script and then running it through each of the scripts successively.

2.7.5 Licensing

The SIFT algorithm had been patented in the US. However, the patent expired last year, and the patent was only to protect stop commercial use of the algorithm, whereas this is an academic use of the algorithm.

Furthermore, I am able to use VGG for non-commercial purposes under the Creative Commons Attribution License.

Chapter 3

Implementation

In this chapter, the repository is examined in detail in section 3.1, how the project is run from end-to-end is discussed in section 3.6, and each of the modules are discussed in sections 3.3, 3.4, 3.5, and 3.2. Finally, the extension to replace VGG with another face descriptor is discussed in section 3.3.3.

3.1 Repository Overview

The repository is shown in Figure 3.1

Figure 3.1: Folder Structure of the project

3.2 Data Preparation

3.2.1 Cross-Validation

We split the datasets for each relationship into 5 folds each in which each fold within a relationship has around the same amount of pairs. Each training set composed of 4 of the folds and the last fold was designated as the test set, which composed of 5 training/testing set combinations. For the KinFaceW datasets, as the images were already split into folds by the dataset, that was used in the project, whereas with TSKinFace, it had to be generated randomly.

3.2.2 Positive and Negative Pairs

Within each dataset, there are pairs of images which either have the kin relationship or don't, which we call positive and negative pairs respectively.

KinFaceW had these negative and positive pairs prepared in the `mat` files that came with the dataset. However, as TSKinFace only came with the images, only the positive images could be found from them. Thus, negative pairs for the dataset had to be created randomly. For each relationship, the number of negative pairs was set to be equal to the number of positive pairs in the corresponding set, which was 404 for the training set and 101 for the test set. Then, an two images were picked out randomly such that they didn't belong to the same positive pair.

Finally, each of the KinFaceW datasets had a setting, restricted or unrestricted, in which the restricted setting meant that no negative pairs are used in the training splits and unrestricted

means that they are used [10]. TSKinFace didn't have this type of setting, however, but the differences in whether negative pairs were or weren't used in the training process was still explored.

3.2.3 PCA

In order to save on computation time for the training process as well as on disk space, PCA is used in order to reduce the dimensions of each of the face descriptors. In the implementation, after each image had its face descriptors obtained for a dataset, for each type of face descriptor, PCA is used to reduce the dimensions to 100 through the method defined in Section 2.5.2.

3.2.4 Saving Results to Disk

Running the project end-to-end each time would take too long each time and, in order to better explore some of the results that were taken, the pipeline needed to be split into modules. This ended up being the preprocessing stage, the training stage and then the testing stage. These stages are explained in further detail in section 3.6. What was saved to disk is as follows, split by stage:

1. *Preprocessing*:
    - **Face Descriptors**: Under the corresponding dataset folder in the data folder, the each face descriptor for each image is stored as a pickle file which contains a map from the image name to the PCA reduced face descriptor. The files are split up by face descriptor type so `VGG_face_descriptors.pkl` would correspond to the VGG face descriptors of each image.
    - **TSKinFace Splits and Negative Pairs**: Since the TSKinFace splits and negative pairs are created randomly and both are used in the training and testing step, it is imperative that these are saved on disk since trying to recompute them would result in different splits and negative pairs.
2. *Training*:
    - **WGEML Output**: The output of the training is saved on disk for each fold of the relationship. It is saved to the `data` folder under the corresponding dataset and setting. Each file is called `[relationship]_out.pkl` which is under the folder path `data/[dataset]/WGEML_out/[setting]/[relationship]_out.pkl`.
3. *Testing*:
    - **Ablation Studies Results**: The accuracies obtained from the ablation studies are stored in a CSV for each dataset and setting configuration which is stored in the path `out/ablation_studies/[dataset]_[setting].csv`.
    - **Pairwise Accuracies**: Similarly, the accuracies obtained from changing which dataset the test data comes from for each model is saved as a CSV to the path `out/pairwise_accs/[dataset]_[setting].csv`.

3.3 Face Descriptors

Each of the LBP and HOG face descriptors were implemented exactly as mentioned in section 2.3 directly without the use of any libraries other than Numpy and OpenCV. However,

3.3.1 SIFT Implementation

The major difference between the original version of SIFT described in [8] and what was needed was that keypoints of each image weren't calculated. Instead, the descriptor that was obtained from each keypoint, which is described in section 2.3.3, is instead obtained from dividing the image into 7×7 overlapping patches on a 16×16 grid as shown in figure 3.2.

Figure 3.2: The overlapping patches

This leads to there being a 128-dimensional vector for each patch which leads to a $128 \times 7 \times 7 = 6272$-dimensional vector for each image. This also helps to standardize the dimension of the vector for each image as different images can have a different number of keypoints which would lead to different dimensionalities of the vectors.

Due to the differences, this version of SIFT had to be implemented directly rather than using a library function like OpenCV's SIFT implementation.

3.3.2 VGG Implementation

The original VGG network that was trained on ImageNet had 138 million parameters [16] which would then increase if we were to increase the number of outputs of the softmax layer for the face network. This would take too long to train in practice and the pretrained weights were uploaded to their site[1]. Therefore, it was easier to download the weights from the website and reformat it as needed.

Firstly, due to the fact that the weights were used 566MB, this wasn't something that could just be uploaded to GitHub. Thus, when predicting, I made sure to check if there was a weights file with a specific name in the `src/face_descriptors` folder path. If there wasn't, then the weights would be downloaded from the site. Due to the fact that I was using the Torch weights but I was implementing this in Keras, I needed to reshape a few of the fully-connected layers. For example, one layer had the shape $(7, 7, 512, 4096)$ for the weights but for Keras, it needed to be in the shape $(7 \times 7 \times 512, 4096)$. This was easily solved by using the `np.reshape` function for each of the fully-connected layers and these slightly modified weights were saved to an `h5` file on disk.

From there, the weights would be on disk so, to get a face descriptor for a set of images, I would create the VGG face model with the softmax layer, then load the weights into the model before returning the model without the softmax layer. This model then was used to get the face descriptors for the set of images given. Due to the fact that VGG takes images of size 224×224, the images needed to be resized to 224×224 before it could be used as an input into the model. Thus, the model took a `numpy` array of size $(n, 224, 224, 3)$ and outputted a list of face descriptors which had size $(n, 4096)$.

[1] `https://www.robots.ox.ac.uk/~vgg/software/vgg_face/`

3.3.3 CifarNet Extension

3.4 WGEML

The main algorithm used in the project is Weighted Graph Embedding-Based Metric Learning (WGEML) [7]. The algorithm is a form of metric learning which takes in, as input, a training positive set for each face descriptor, $\mathcal{S}^p = \{(\mathbf{x}_i^p, \mathbf{y}_i^p) \mid 1 \leq i \leq N\}$ where N is the number of image pairs in the set and a negative pair set $\mathcal{D}^p = \{(\mathbf{x}_i^p, \mathbf{y}_j^p) \mid 1 \leq i \leq N, j \neq i\}$. Let there be M face descriptors for each image in the set. It also takes in a tuning parameter r and a neighborhood size K as input but those have been experimentally found to be 5 for each, each of which will be discussed later in this section.

3.4.1 Problem

Given these inputs, the goal is to find the distance metrics and weights for:

$$\begin{aligned} d^2(\mathbf{x}_i, \mathbf{y}_i) &= \sum_{p=1}^{M} w_p (\mathbf{x}_i^p - \mathbf{y}_i^p)^T \mathbf{A}_p (\mathbf{x}_i^p - \mathbf{y}_i^p) \\ &= \sum_{p=1}^{M} w_p d^2_{\mathbf{A}_p}(\mathbf{x}_i^p, \mathbf{y}_j^p) \end{aligned}$$

Where $\mathbf{x}_i^p$ represents the pth face descriptor of the image $\mathbf{x}_i$, w_p is a weight and $\mathbf{A}_p$ is a $D \times D$ semidefinite positive matrix, where D is the dimensionality of the corresponding face descriptor. This distance function finds the distance between each pair of face descriptors and weights them accordingly. We wish to find $\mathbf{A}_p$ such that the between-class variance is maximized and the within-class variance is minimized. This can be formalized as the optimization problem:

$$\begin{aligned} \max_{\mathbf{A},\mathbf{w}} \mathcal{F} = \sum_{p=1}^{M} w_p^r [(\frac{1}{2}(\frac{1}{NK} \sum_{i=1}^{N} \sum_{n_1=1}^{K} d^2_{\mathbf{A}_p}(\mathbf{x}_i^p, \mathbf{y}_{in_1}^p) + \frac{1}{NK} \sum_{i=1}^{N} \sum_{n_2=1}^{K} d^2_{\mathbf{A}_p}(\mathbf{x}_{in_2}^p, \mathbf{y}_i^p)) \\ + \frac{1}{N} \sum_{i=1, j \neq i}^{N} d^2_{\mathbf{A}_p}(\mathbf{x}_i^p, \mathbf{y}_j^p)) / \frac{1}{N} \sum_{i=1}^{N} d^2_{\mathbf{A}_p}(\mathbf{x}_i^p, \mathbf{y}_i^p)] \end{aligned}$$

$$s.t. \sum_{p=1}^{M} w_p = 1$$

$$\forall p \in \{1, ..., M\}, w_p \geq 0$$

Where $\mathbf{y}_{in_1}^p$ is the n_1th nearest neighbor of $\mathbf{y}_i$ and similarly for $\mathbf{x}_{in_2}$. We have w_p^r in order to avoid over-fitting and use information from each face descriptor. This optimization function tries to minimizes the denominator which, in turn, pulls the samples that have the kin relationship together and maximizes the numerator which means that the pairs which don't have the kin relationship are pushed further away from each other as well as those of their neighbors.

3.4.2 Approach

Now that the problem is defined, we break down $\mathbf{A}_p = \mathbf{U}_p\mathbf{U}_p^T$ since $\mathbf{A}_p$ is symmetric and positive semidefinite. $\mathbf{U}_p$ has size $D \times d$ such that $d \ll D$. Thus, the optimization problem can be rewritten as:

$$\max_{\mathbf{U},\mathbf{w}} = \sum_{p=1}^{M} w_p^r \frac{tr[\mathbf{U}_p^T(\frac{1}{2}(\mathbf{D}_{1p} + \mathbf{D}_{2p}) + \mathbf{D}_p)\mathbf{U}_p]}{tr[\mathbf{U}_p^T\mathbf{S}_p\mathbf{U}_p]}$$

Such that $\mathbf{U}_P^T\mathbf{U}_p = I$, $\sum_{p=1}^{M} w_p = 1$ and $\forall p \in \{1, ..., M\}, w_p \geq 0$, where:

$$\mathbf{S}_p = \frac{1}{N} \sum_{(\mathbf{x}_i^p, \mathbf{y}_i^p) \in \mathcal{S}^p} (\mathbf{x}_i^p - \mathbf{y}_i^p)(\mathbf{x}_i^p - \mathbf{y}_i^p)^T$$

$$\mathbf{D}_p = \frac{1}{N} \sum_{(\mathbf{x}_i^p, \mathbf{y}_j^p) \in \mathcal{D}^p} (\mathbf{x}_i^p - \mathbf{y}_j^p)(\mathbf{x}_i^p - \mathbf{y}_j^p)^T$$

$$\mathbf{D}_{1p} = \frac{1}{NK} \sum_{\substack{(\mathbf{x}_i^p, \mathbf{y}_i^p) \in \mathcal{S}^p \\ \mathbf{y}_k^p \in \mathcal{N}_K(\mathbf{y}_i^p)}} (\mathbf{x}_i^p - \mathbf{y}_k^p)(\mathbf{x}_i^p - \mathbf{y}_k^p)^T$$

$$\mathbf{D}_{2p} = \frac{1}{NK} \sum_{\substack{(\mathbf{x}_i^p, \mathbf{y}_i^p) \in \mathcal{S}^p \\ \mathbf{x}_k^p \in \mathcal{N}_K(\mathbf{x}_i^p)}} (\mathbf{x}_k^p - \mathbf{y}_i^p)(\mathbf{x}_k^p - \mathbf{y}_i^p)^T$$

Where $\mathcal{N}_K(\mathbf{x}_i^p)$ represents the K nearest neighbors of $\mathbf{x}_i^p$. This is when K-nearest neighbors is used and K is set to 5 in the experiments. In order to solve this, we first let $\mathbf{w}$ be constant in order to solve $\mathbf{U}$ and then use that to find $\mathbf{w}$. By letting $\mathbf{w}$ be constant, the problem is reduced to:

$$\max_{\mathbf{U}_P^T\mathbf{U}_p = I} = \frac{tr[\mathbf{U}_p^T(\frac{1}{2}(\mathbf{D}_{1p} + \mathbf{D}_{2p}) + \mathbf{D}_p)\mathbf{U}_p]}{tr[\mathbf{U}_p^T\mathbf{S}_p\mathbf{U}_p]}$$

For each $p \in \{1, ..., M\}$.

The overall algorithm is written in psuedocode in Algorithm 1 from [7]:

Algorithm 1 WGEML

Inputs:

Outputs:

```
1: procedure MyProcedure
2:     stringlen ← length of string
3:     i ← patlen
4:     if i > stringlen then return false
5:     j ← patlen
6:     if string(i) = path(j) then
7:         j ← j − 1.
8:         i ← i − 1.
9:         close;
10:    i ← i + max(delta1(string(i)), delta2(j)).
```

3.5 Prediction

Given the matrices U_p and weights w_p from WGEML for all face descriptors where U_p corresponds to the p'th face descriptor, we can use this to predict whether a given pair of images are of the kin relationship specified. First, we transform each of the faces into their M face descriptors and use PCA to reduce the dimensions to the proper dimensionality. We then have $\mathbf{x} = \{x_p \mid 1 \leq p \leq M\}$ and $\mathbf{y} = \{y_p \mid 1 \leq p \leq M\}$. Then, the similarity of these images can be calculated as follows, letting $A_p = U_p U_p^T$:

$$sim(\mathbf{x}, \mathbf{y}) = \sum_{p=1}^{M} \frac{w_p}{2} \left(\frac{x_p^T A_p y_p}{\sqrt{x_p^T A_p x_p} \sqrt{y_p^T A_p y_p}} + 1 \right)$$

This ends up finding a weighted cosine similarity for a non-Euclidean space, since our metric here is A_p, for each face descriptor that is used. The function sim ranges from 0 to 1 as needed of a similarity function. This differs from the version mentioned in [7] which was:

$$sim(\mathbf{x}, \mathbf{y}) = \sum_{p=1}^{M} \frac{w_p}{2} \left(\frac{x_p^T U_p^T U_p y_p}{\sqrt{x_p^T U_p^T U_p x_p} \sqrt{y_p^T U_p^T U_p y_p}} + 1 \right)$$

However, as mentioned in section 3.4, the model was trained such that $U_p^T U_p = I$ which would make this boil down to a weighted cosine similarity which wouldn't make much use of the metric learning. As such, this was changed up to use A_p in the implementation rather than $U_p^T U_p$.

In order to then determine whether two images are of the kin relationship specified, the similarity must then be compared to a threshold, θ. If $sim(\mathbf{x}, \mathbf{y}) \geq \theta$, then the pair is labelled as having that relationship and, otherwise, it isn't. As this value θ wasn't mentioned in [7], it had to be found experimentally. By ranging over samples from 0 to 1, it was found that $\theta = 0.6$ yielded the best results for the accuracies and seemed the most similar to what the original paper had.

3.6 Overview of Workflow

There are 3 main stages to running the project, which are, for each dataset, the preprocessing stage, the training stage and then the testing stage. The important outputs of each of these stages are saved onto disk so that each stage doesn't have to be run right after another.

3.6.1 Preprocessing

Given a dataset, we wish to precompute the face descriptors for each image and set up the 5-fold cross validation for TSKinFace.

3.6.2 Training

Once all of the face descriptors for each dataset is computed and, for each relationship in the dataset and for each setting (restricted or unrestricted), the cross-validation folds are created, the training set is properly made from the data and WGEML, which is described in section 3.4, is run. The output of WGEML is then saved onto disk, which are the metrics and weights

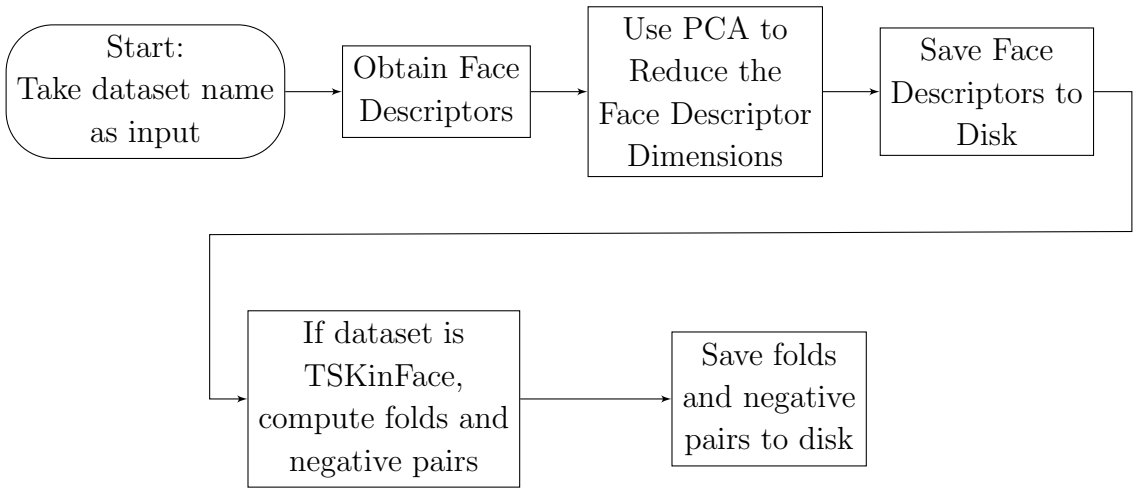

Figure 3.3: The preprocessing pipeline

for each face descriptor for each fold. In the end, this is done for each relationship, setting and fold, so for KinFaceW-I, for example, there would be $4 \times 2 \times 5 = 40$ models saved.

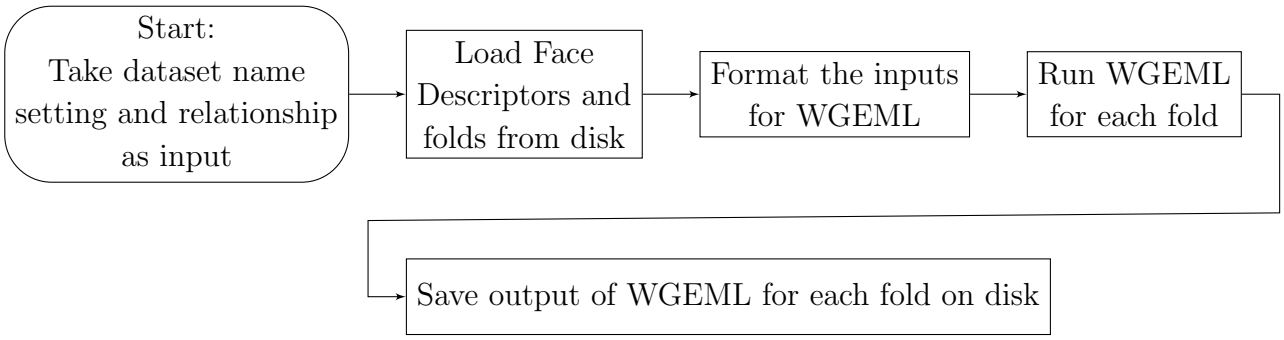

Figure 3.4: The training pipeline

3.6.3 Testing

Given a dataset, setting and relationship, the folds and the corresponding models are loaded from on disk and, for each test set, the corresponding model is used to predict whether the images in the test set are of the given relationship or not. The accuracy is then recorded and the average accuracy is outputted for the given configuration along with each of the individual accuracies, as shown in Figure 3.5.

```
(venv) mv465@idun:~/KinRecognition$ CUDA_VISIBLE_DEVICES=0 make runPredictionKFW1Unrestricted
python3 -m src.scripts.testing "KinFaceW-I" "fs" "unrestricted"
KinFaceW-I-fs-unrestricted: [0.7581 0.8387 0.8387 0.8065 0.7656]
KinFaceW-I-fs-unrestricted: 0.8015
python3 -m src.scripts.testing "KinFaceW-I" "fd" "unrestricted"
KinFaceW-I-fd-unrestricted: [0.7037 0.7407 0.7593 0.6296 0.75  ]
KinFaceW-I-fd-unrestricted: 0.7167
python3 -m src.scripts.testing "KinFaceW-I" "ms" "unrestricted"
KinFaceW-I-ms-unrestricted: [0.8913 0.8043 0.6522 0.6957 0.7708]
KinFaceW-I-ms-unrestricted: 0.7629
python3 -m src.scripts.testing "KinFaceW-I" "md" "unrestricted"
KinFaceW-I-md-unrestricted: [0.84   0.72   0.82   0.84   0.7407]
KinFaceW-I-md-unrestricted: 0.7921
```

Figure 3.5: A sample output of the testing stage on KinFaceW-I unrestricted

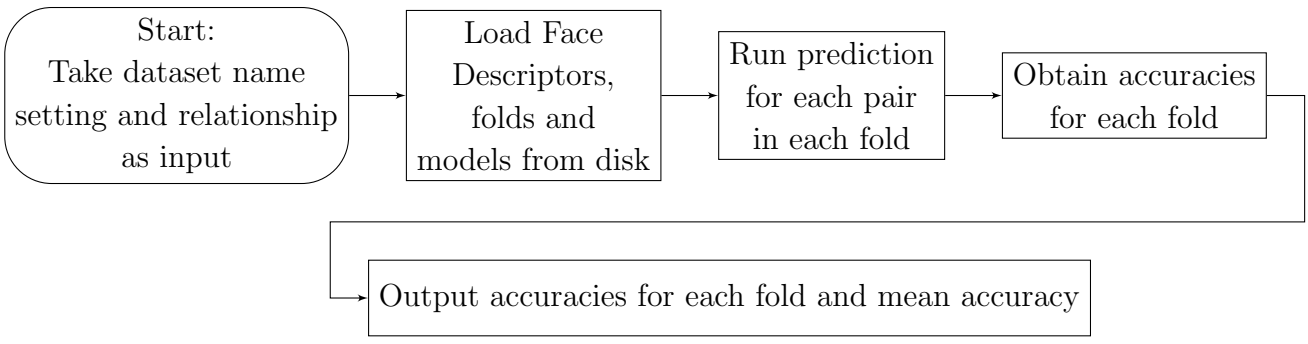

Figure 3.6: The testing pipeline

Chapter 4

Evaluation

4.1 Success Criterion

4.2 Original Results

4.3 Potential Biases in Datasets

4.4 Ablation Studies

4.4.1 Blocking Face Descriptors

4.4.2 Blocking Parts of Images

4.5 Replacing VGG

4.6 Unit Tests

Chapter 5

Conclusion

5.1 Achievements

5.2 Lessons Learnt

5.3 Future Work

Bibliography

[1] T. Ahonen, A. Hadid, and M. Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):2037–2041, 2006.

[2] Laleh Armi and Shervan Fekri-Ershad. Texture image analysis and texture classification methods - a review, 2019.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[4] R. Fang, K. D. Tang, N. Snavely, and T. Chen. Towards computational models of kinship verification. In *2010 IEEE International Conference on Image Processing*, pages 1577–1580, 2010.

[5] W.D. Hamilton. The genetical evolution of social behaviour. ii. *Journal of Theoretical Biology*, 7(1):17 – 52, 1964.

[6] I.T. Jolliffe. *Principal component analysis*. Springer-Verlag New York, 2002.

[7] J. Liang, Q. Hu, C. Dang, and W. Zuo. Weighted graph embedding-based metric learning for kinship verification. *IEEE Transactions on Image Processing*, 28(3):1149–1162, 2019.

[8] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91110, November 2004.

[9] J. Lu, J. Hu, X. Zhou, Y. Shang, Y. Tan, and G. Wang. Neighborhood repulsed metric learning for kinship verification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2594–2601, 2012.

[10] J. Lu, X. Zhou, Y. Tan, Y. Shang, and J. Zhou. Neighborhood repulsed metric learning for kinship verification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(2):331–345, 2014.

[11] Jill M. Mateo. Perspectives: Hamilton's legacy: Mechanisms of kin recognition in humans. *Ethology*, 121(5):419–427, 2015.

[12] A. Nandy and S. S. Mondal. Kinship verification using deep siamese convolutional neural network. In *2019 14th IEEE International Conference on Automatic Face Gesture Recognition (FG 2019)*, pages 1–5, 2019.

[13] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In Mark W. Jones Xianghua Xie and Gary K. L. Tam, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 41.1–41.12. BMVA Press, 2015.

[14] Xiaoqian Qin, Xiaoyang Tan, and Songcan Chen. Tri-subject kinship verification: Understanding the core of a family, 2015.

[15] Joseph P Robinson, Ming Shao, and Yun Fu. Visual kinship recognition: A decade in the making, 2020.

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[17] Juan Luis Surez-Daz, Salvador Garca, and Francisco Herrera. A tutorial on distance metric learning: Mathematical foundations, algorithms, experimental analysis, prospects and challenges (with appendices on mathematical background and detailed algorithms explanation), 2020.

[18] Jun Yu, Mengyan Li, Xinlong Hao, and Guochen Xie. Deep fusion siamese network for automatic kinship verification. *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020)*, pages 892–899, 2020.

Appendix A

Project Proposal