Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Håkon Wardeberg

Mesh-based 3D face recognition using Geometric Deep learning

Master's thesis in Computer Science
Supervisor: Theoharis Theoharis
Co-supervisor: Antonios Danelakis
June 2021

NTNU
Norwegian University of
Science and Technology

Håkon Wardeberg

Mesh-based 3D face recognition using Geometric Deep learning

Master's thesis in Computer Science
Supervisor: Theoharis Theoharis
Co-supervisor: Antonios Danelakis
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Face recognition has been a very active and challenging task in the Computer Vision field. Performing face recognition based on facial images can be tricky since images are illumination, scale, and pose variant. On the contrary, 3D scans are invariant in regards to the aforementioned aspects. Primarily, 3D face recognition has been using data-driven approaches, and in later years, projection-based deep learning techniques that project the 3D space to 2D. Such approaches allow the use of traditional convolutional neural network structures to work with 3D data. The problem with projection-based models is that they remove geometric relationships important for distinguishing faces and that the projection is expensive and slow. An alternative to this is using Geometric Deep Learning techniques that directly utilize the 3D geometry and data. Currently, very few approaches use any Geometric Deep Learning techniques. Such approaches will in theory perform better as they have greater freedom than a model using the 2D space. This Thesis proposes a fast mesh-based 3D face recognition approach that inputs meshes directly. To do this, we propose a new feature extraction network based on graph convolutions to extract face features. By using a siamese architecture to compare extracted facial features generated from 3D meshes, the model is able to do both face validation and identification. The proposed approach achieves close to state-of-the-art performance on three public 3D face benchmarks, i.e., a rank-1 identification rate of 90.2%, 90.1%, and 90.1%, and a verification rate at 0.1% false acceptance rate of 87.6%, 86.0%, and 99.55% on the BU-3DFE, Bosphorus, and FRGCv2 datasets, respectively. Further, our proposed approach only requires 8 milliseconds to identify a face among a gallery with 466 faces.

Sammendrag

Ansiktsgjenkjenning har vært en aktiv og utfordrende oppgave innen datasynfeltet. Å utføre ansiktsgjenkjenning basert på ansiktsbilder kan være vanskelig siden bildene er belysnings-, skala- og positurvariant. 3D-bilder har ikke disse problemene. Primært har 3D ansiktsgjenkjenning brukt datadrevne metoder, og i senere år, projeksjonsbaserte dyplæringsteknikker som projiserer 3D-rommet til 2D. Disse metodene tillater bruk av tradisjonelle kunstig nevrale nettverksstrukturer for å jobbe med 3D-data. Problemet med projeksjonsbaserte metoder er at de fjerner geometriske forhold som er viktige for å skille ansikter og at en projeksjon er kostbar og langsom prosess. Et alternativ til dette er å bruke Geometric Deep Learning teknikker som direkte bruker 3D geometri og data. Foreløpig bruker svært få metoder geometriske dyplæringsteknikker. Slike metoder vil i teorien fungere bedre ettersom de har større frihet enn en modell som bruker 2D-rommet. Denne masteroppgaven foreslår en rask maskebasert 3D ansiktsgjenkjenningsmetode som tar inn maskenett direkte. Ved å bruke en siamesisk arkitektur for å sammenligne ansiktstrekk laget fra 3D-bilder kan modellen gjøre både ansiktsgjenkjenning og validering. Den foreslåtte tilnærmingen oppnår god ytelse på tre offentlige 3D-ansikts benchmarks, dvs. en rang-1 identifikasjonsrate på 90,2%, 90,1% og 90,1%, og en verifiseringsrate med 0,1% feilakseptrate på 87,6%, 86,0% og 99,55% for henholdsvis BU-3DFE, Bosporus og FRGCv2 datasettene. Videre krever vår foreslåtte tilnærming bare 8 millisekunder for å identifisere et ansikt blant en gruppe med 466 ansikter.

Preface

This project is a continuation of a specialization project by Wardeberg [1]. As the specialization project is not publicly available and this Thesis is a continuation of the same research, some chapters and relevant background material were reused with various degrees of modification. This is considered standard practice at NTNU when the master's thesis is a continuation of the specialization project. It is worth mentioning that no code was reused from the specialization project. All adapter chapters are listed below:

- **Section 2.1 – 3D data representations** This section was based on the specialization project but re-written and expanded.
- **Section 2.2.1 – Signal Convolution operator** This section was adapted with minor modifications.
- **Section 2.2.2 – Image Convolution operator** This section was adapted with minor modifications.
- **Section 2.2.4 – Fully Connected Layers** This section was based on the specialization project but re-written and expanded.
- **Section 2.2.6 – Geometric Deep Learning** This section was adapted with minor modifications.
- **Section 2.2.6.2 – Graph Convolution Network (GCN)** This section was adapted with minor modifications.
- **Section 2.2.7.1 to 2.2.7.2** The sections were adapted with minor modifications.
- **Section 2.2.8 – Training** This section was adapted with major modifications.
- **Section 3.1.2 to 3.1.4** The sections were adapted with major modifications.

Acknowledgements

I would like to express my gratitude to my supervisors Antonios Danelakis and Theoharis Theoharis. They have both been there when I have had questions, and have guided me throughout the project.

I would like to thank my family for always being there.

And lastly, I would also like to thank old and new friends. As a wise one once said: "This is your university careers magnum opus. Take some pride."

Dum panem est, spes est.

Panem et panem

Contents

Figures

Tables

Chapter 1

Introduction

Face recognition is a prevalent and well-established technology widely used in non-intrusive biometrics and for general identification [2]. Today, most face recognition is done using 2D images and deep learning techniques [3], while the 3D field has predominantly been based on data-driven approaches [4]. Lately, multiple papers have proposed methods for using deep learning techniques on 3D scans, steadily improving the speed and accuracy.

The problem with 3D face recognition is that the 3D field has had a lot less research compared to the 2D face recognition field, making 2D generally preferred [3]. This has been because of the incapability of typical deep learning networks to process 3D data. However, recently more geometric deep learning techniques have been proposed that generalize neural network methods to work on non-euclidean structures such as graphs and manifolds [5]. This enables better and more efficient networks.

Regarding face recognition, there are multiple unique challenges and properties for 2D and 3D face recognition.

1.1 Challenges with 2D face recognition

The largest challenge with 2D face recognition and corresponding techniques is that images are illumination, scale- and rotation variant, are non-robust to poses, and have issues with self-occlusion, meaning that parts of the face may occlude itself. [3, 6, 7]. This means that a change in any of these factors, like lightning, will change the look of the image, making it harder to do accurate facial recognition.

1.2 Challenges with 3D face recognition

3D facial recognition has another set of challenges. Firstly, the biggest problem with 3D tasks is that available datasets used for training are relatively small and limited [3]. There exist datasets used in 2D facial recognition which contains over

200 million images [8], while the largest 3D face dataset contains only 13,450 scans [9].

The 3D machine learning field is also a lot less matured compared to the 2D field [3]. 3D data techniques are still being developed and are often much more computationally expensive compared to their analogous 2D counterpart [10]. The 3D format also poses problems for neural networks [5]. Most scans have different resolutions meaning that networks must be robust to variable-sized inputs. For point clouds, the number of points may vary, and for graphs, they are often irregular with a variable amount of vertices and edges.

1.3 3D face recognition vs. 2D face recognition

There are multiple reasons why 3D face recognition may, in some cases, be better than 2D face recognition. 3D scans are relatively invariant to illumination, scaling, and rotation and are more robust to poses and expressions [3, 6]. In theory, 3D scans also provide more information, like geometric depth and relationships, enabling better and more secure face recognition compared to using images [3, 6].

On the other side, there are multiple advantages to using 2D images. Firstly, there exist several larger datasets usable for 2D face recognition. This is needed for networks to generalize well and helps to prove the face recognition efficiency as testing sets can be larger. 2D neural networks also do not have the same variant data issue 3D networks has, as image scaling and cropping can be used to re-scale images with minimal information loss.

1.4 Recognition & Verification

There are two main types of face recognition systems: face recognition, often called identification, and face verification [3].

Face verification is the process of verifying if a reference identity is equal to a single unknown identity. The process returns a similarity score, a single number that represents the likeness of the pair. An example of a use case is to run verification on a person and their passport to see if they are the same identity. This process is a one-to-one process.

Face identification is the process of matching an unknown face to a set of known faces. An example would be for an office security system to validate if a person is among the allowed users. This process is a one-to-many process. Face identification can be done via verification by using the verification process against all known identities. The problem with this approach is that a single verification can be expensive and that the amount of verifications needed is increased linearly with the size of the known identities set.

1.5 This Thesis

The goal of this Thesis is to investigate if a mesh-based approach is a feasible method for making an efficient and robust 3D face recognition algorithm. To the best of our knowledge, this is the first work that uses mesh convolutions for obtaining features on 3D face scans.

By directly using the 3D data, the network should be able to perform quick and robust face recognition as no pre-processing is needed, and the original spatial and geometrical structures will be preserved in the data. Furthermore, there already exist multiple graph convolutions, which can be used on the mesh. The edges in a mesh may also allow for efficient traversal of features through the structure, enabling the model to train on very little data.

Our implementation of the proposed method is available over at
https://github.com/hakonw/3D-Facial-mesh-recognition hash:406e5ae.

Chapter 2

Theory

This chapter contains relevant theory and related works important for the project and methodology later in this Thesis.

2.1 3D data representations

There are multiple ways of representing three-dimensional data. The most common are the discrete extrinsic representations like voxels, point clouds, and meshes. These representations, also called euclidean representations, use parametrizations or coordinates external to the shape [11]. There exist other approaches like intrinsic representations and parametric objects, but they are less relevant for this Thesis.

2.1.1 Voxel grid

One way to represent 3D models is via a voxel grid. Here, a signal like a 3D object is voxelized into a defined grid, similar to how a 2D shape is restricted and rasterized into the image dimensions. On a voxel grid, standard convolutional neural network mechanisms can be applied and utilized [12]. Instead of a 2D filter, a 3D filter will be moved over the x, y, and z dimensions activation on volumetrically similar structures to the filter. Similarly, as a pixel p_i of a rasterized two-dimensional image (RGB) can be expressed in $p_i \in \mathbb{N}^{2\times3}$ where 3 is the dept of the image, the voxel grid data structure can is represented in $\mathbf{V} \in \mathbb{N}^{3\times F}$, where F is the amount of features per voxel. The data can then simply be stored in a matrix of shape $W \times H \times D \times F$. An example is a cubic space with a height, depth, width of 30, and 3 values in each voxel can be stored in a matrix of shape $30 \times 30 \times 30 \times 3$.

Voxel grids also separate between hollow and filled models. Scans taken with a regular 3D camera will be hollow as the camera cannot see the inside of the object while results from simulations often are filled.

There are multiple problems with voxel grids. If the model is hollow, most of the grid will be empty or in the same state as its neighbor. This makes the representation more inefficient and wasteful.

Another problem is spatial resolution. Volumetric representations often use small dimensions because of the inefficiency of the representation. Wu *et al.* [13] proposed a method for classifying 3D objects using a voxel grid of $30 \times 30 \times 30$. Small face details are important for separating faces making small resolution voxel grids not suitable. This issue can be avoided by using a larger resolution grid that captures these details.

With a defined grid, rigid transformations create significant changes and artifacts. This is true for both low-resolution 2D images and voxel grids. A good example of the artifacts can be seen with the rotation of a box or square. The edges will become jagged, and it will be harder to see the outline of the box. The significant changes from transformations are a highly undesirable effect. To mitigate the effect, higher resolution grids can be utilized, which might not always be optimal.

2.1.2 Point-based representations

Point-based representations rely on points to create discreet approximations of the surface of an object [14]. The most rudimentary technique is the point cloud. A point cloud is an unordered point set $\{x_1, x_2, ..., x_n\}$ with $x_i \in \mathbb{R}^3$ that is sampled from original signal [15]. If each point holds more information than its position, the feature matrix becomes $\mathbf{X} \in \mathbb{R}^{3 \times F}$. These sets are the raw data generated from 3D scanners. The resolution can also be locally variant, where more detailed sections of a model have more data.

A mesh is another form, where instead of only storing a list of points, the relationships between the points are described in terms of edges and faces. When talking about meshes, a vertex is used to describe the reference to a point, as each point is used in multiple edges and faces. A mesh can be seen as a graph that includes face information, as they are represented in the same way in memory. Even though a mesh is parameterized to a euclidean space, the edges and faces make the shape a non-euclidean manifold. This means that regular convolutions can not easily be applied on the structure.

Point-based representations have the property of being robust to transformations. As the points are defined in $\mathbb{R}$ the limiting factor is the precision of the storage medium format.

The spatial resolution of edges and vertices in meshes is dependent on the subdividing or discretization of the original signal. Large flat areas can be expressed with few vertices and edges, allowing for more efficient storage than a volumetric representation.

The largest problem with point-based representations regarding machine learning is that convolution-like operators are harder to define. Another issue with point-based representations is that the number of vertices, edges, and faces can vary. Unless the model is limited, the neural networks must work on an unknown amount of data, often irregular and unsorted. The unknown size makes combining a neural network that works on point clouds and meshes with a layer that requires pre-defined parameters harder.

The internal construction of a mesh may also vary. How the edges are built up is an important property as the same structure can be represented in multiple ways. It varies based on the triangulation algorithm used and if quads is used instead of triangles. Bouritsas *et al.* [16], used a spiral convolution and emphasized that consistent orderings across meshes were important when using ordering-sensitive operators.

2.2 Neural network

This section details theory and methods specifically important for the neural network techniques used in this Thesis.

2.2.1 Signal Convolution operator

A convolution is a mathematical operation that expresses how two functions modify each other [17]. The mathematical definition is shown in Equation 2.1.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.1}$$

The spatial domain convolution can be expressed discretely, which gives Equation 2.2.

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau) \tag{2.2}$$

For the convolutions relevant to this project, the range of the input functions are limited over a finite range, and the equation becomes Equation 2.3. When using the convolutional operator on 2D domains like images, the function f becomes the image, g becomes a two-dimensional input called the kernel, and k becomes the set of all pixels.

The equation can be expanded and expressed as Equation 2.4 to take multiple arguments and iterate over both axes of an image.

$$(f * g)(t) = \sum_{k} f(k)g(t-k) \tag{2.3}$$

$$(f * g)(i,j) = \sum_{m}\sum_{n} f(m,n)g(i-m,j-n) \tag{2.4}$$

2.2.2 Image Convolution operator

The convolution operator applied to images can be explained more intuitively than Equation 2.4. The operation can be seen as a frame sliding over an image, creating a new value based on the sum of the element-wise multiplication between

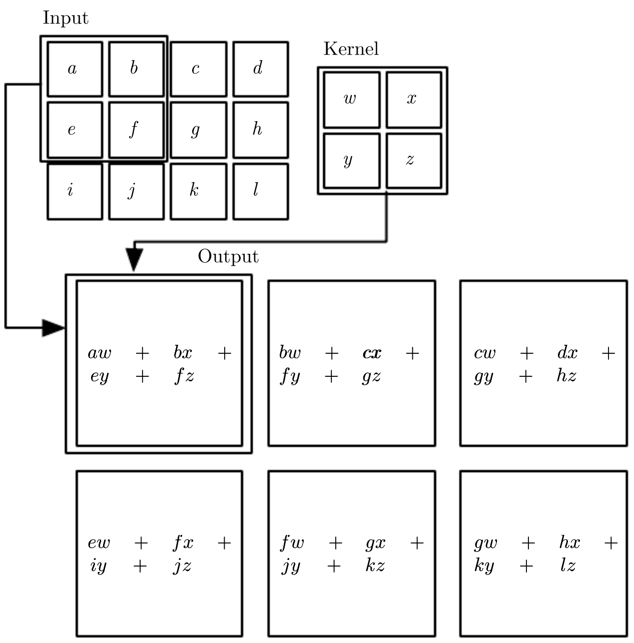


Figure 2.1: Figure from Goodfellow *et al.* [17, p. 334] showing an example of cross-correlation.

the image and kernel. In neural layers, instead of convolution, cross-correlation is used, where the only difference is that the filter is flipped as it saves a few operations [17]. The use of convolution or cross-correlation does not impact the statistical power of the layer. An example showing cross-correlation is presented in Figure 2.1.

There are multiple reasons why convolutional layers are a powerful tool in machine learning when dealing with spatial-structured data. According to Goodfellow *et al.* [17], three important properties are sparse connectivity, parameter sharing, and translation equivalence.

Sparse connectivity means that each output is dependent on a small number of inputs. Compared to a dense network where each output is a function of each input, a sparse network will have fewer connections and therefore need fewer mathematical operations. The sparse connectivity is created when the kernel is multiple times smaller than the input.

Another property described by Goodfellow *et al.* [17] is that of translation equivalence[1]. Translation equivalence means that any changes in the input will cause the same translation in the output. An example of where this is important is with an edge-detection kernel. Here, a simple kernel maps the same feature from the input and translates the output features corresponding to the input.

As the same kernel is used over the entire image, convolutional layers also

[1]Not to be confused with translation invariant, where the property of the output being invariant to translation.

have the property of parameter sharing. This significantly reduces the number of weights and features required by the network. Also, as the kernel only samples pixels close to each other, the network has the property of local connectivity [5]. This is useful as closely located features are often related.

Bronstein *et al.* [5] and Wu *et al.* [18] describes another important property of convolutional layers when used on images, the property of compositionality. This property is a product of the structure of the data and is helped by the property of translation equivalence. Compositionality means that low-level features have the ability to be combined to create higher-level features. This stacking property is an important aspect of convolutional neural networks as more and more data can be aggregated into a high-abstraction feature.

These properties allow convolution layers to be efficient in deep neural networks, with low computational complexity, low memory usage, and high usability.

2.2.3 Pooling

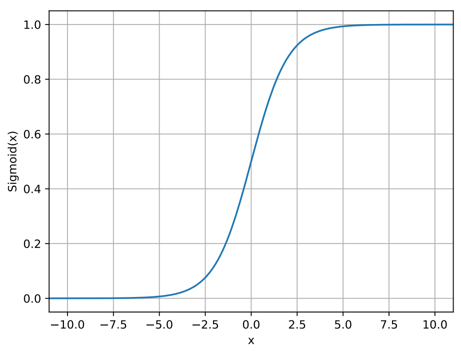
A convolutional layer consists of three stages [17, p. 335-341]. These are the convolution, nonlinear activation, and pooling stage. Pooling is the act of reducing the size of the domain. For 2D networks, the image resolution is sampled down, summarizing the information of neighbors. The pooling stage is strictly not necessary but helps the network by summarizing information from the previous layer, reducing parameters, the statistical burden, and the computational complexity of the next layer [17].

The usage of stride helps the network become translation-invariant. Stride is the amount the kernel in a convolutional or pooling layer moves. If the stride is more than 1 pixel, small variations in the translation will produce the same output, making the absolute position of pixels and features less relevant, helping the statistical efficiency of the network.

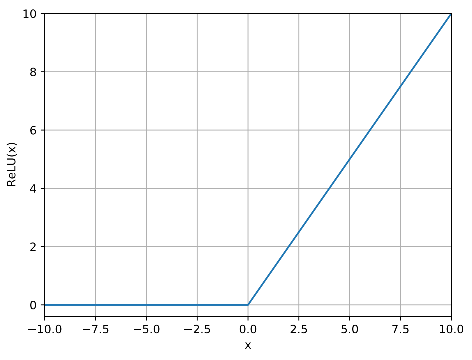
For 3D, there exist multiple pooling techniques. They all work by reducing the size or complexity of the 3D space. The most common way to do it is to combine vertices and or reduce edges [19]. Another way of doing 3D pooling is by reducing the amount of channels per vertex via some operator.

Global pooling is often used in models that deal with 3D data to get a predictable output [20]. Global pooling works on the graph-level-outputs by using a symmetric function on the channel-wise data across all nodes.

Two examples of global pooling, using the symmetric function average and max, are global-max-pooling and global-mean-pooling. They are defined in Equation 2.5 and 2.6, where $\mathbf{r}$ is the output matrix, M is the total amount of vertices, and $\mathbf{X}_n$ is the matrix containing the feature for node n (in other words, $\mathbf{X} \in \mathbb{R}^{(N_1+\ldots+N_M)\times F}$, where F is the amount of features). Global pooling reduces the 3D space from $\mathbb{R}^{(N_1+\ldots+N_M)\times F}$ to $\mathbb{R}^F$, where the output size F will be constant for any input size M. This allows the network to be connected to other machine learning layers that require defined input sizes like fully connected layers.



(a) Sigmoid activation function.



(b) ReLU activation function.

Figure 2.2: Figures of the most common activation functions showing the output (y-axis) based on the input (x-axis).

$$\mathbf{r} = \mathrm{Max}_{n=1}^{M} \mathbf{X}_n \tag{2.5}$$

$$\mathbf{r} = \frac{1}{M} \sum_{n=1}^{M} \mathbf{X}_n \tag{2.6}$$

The use of a symmetric function is important as it makes the permutation of vertices irrelevant and hinders a reliance on the order of the vertices [15, 21].

2.2.4 Fully Connected Layers

Fully connected layers are one of the simpler machine learning mechanisms. In a fully connected layer, all the input nodes are linked to every output node, changing the domain from $\mathbb{R}^{in}$ to $\mathbb{R}^{out}$. Each layer consist of $n_{in} \cdot n_{out} + n_{out}$ parameters including biases [17]. This method is generally computationally expensive as it needs $O(n^2)$ weights and operations.

Fully connected layers enable complex mappings between the input and output [17]. This property is often used in the last layers of classification networks to map features to an output.

2.2.5 Activation functions

Activation functions are an important part of neural networks. To allow complex mappings between features, non-linear functionality is needed.

One possible activation function is the Sigmoid activation function, defined in Equation 2.7 [17, p. 65-67] and shown in Figure 2.2a. The Sigmoid function maps input to the range $(0, 1)$. The problem with Sigmoid is that the function saturates when the input is either very positive or very negative. In other words the gradient approaches 0, which is unideal for machine-learning as it becomes insensitive to small changes in the input [17, p. 66].

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.7}$$

A more commonly used activation function is the Rectified Linear Unit (ReLU) [22]. It is defined as $f(x) = max(0, x)$ [23] and is shown in Figure 2.2b. As the ReLU function is only a max operation, it is computationally inexpensive, making it ideal for larger networks. It has also been shown that ReLU generally outperforms the sigmoid activation function in neural networks [22]. A problem with ReLU is that the derivative is 0 for negative numbers, creating what's called *dead units*, which may never be able to be updated again.

There exist multiple versions of ReLU which solve the dead unit problem, but they may, at best, only provide a small improvement [22].

2.2.6 Geometric Deep Learning

Bronstein *et al.* [5] defines Geometric Deep Learning as a term for methods that generalize neural network techniques to work on non-euclidean structures such as graphs and manifolds.

The main challenge of Geometric Deep Learning is defining and creating analogous methods to the convolution and pooling used on euclidean data. The main goal is to be able to work directly with data and structures like meshes, graphs, and point clouds to both create models which accurately predicts when presented new problems, but also to get a better formal understanding of what the models are doing, which opens for simpler and more effective methods.

Geometric Deep Learning works on the assumption that non-euclidean data manifest hierarchical structures similar to that of euclidean data [5]. An example of this on euclidean data is images, where high-level features are created from combining low-abstraction shapes like lines and dots, which convolutions exploit via the property of compositionality.

2.2.6.1 Graph convolution

There exist multiple different approaches to Graph Convolutions. The two main categories are spectral graph convolutions and spatial graph convolutions [5, 18].

In signal processing, the *convolution theorem* states that a convolution in the spatial domain is the same as a multiplication in the spectral domain [24, p. 527]. This theorem is described in Equation 2.8 where $\mathcal{F}$ is the Fourier transform operator. While not used much by 2D convolutional layers, the theorem is important for graph convolutions, as it means that the convolution can be done in both the spatial and spectral domain.

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\} \tag{2.8}$$

The spatial convolution is based on information propagation via the spatial relationships of a node. These are usually the neighbors of a node. A simple spatial graph convolution is further discussed in Section 2.2.6.2.

Spectral convolutions are based on the Laplacian of the graph. The normalized Laplacian matrix is used, which is defined as $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{A}$ is the adjacent matrix, $\mathbf{D}$ is the degree matrix of $\mathbf{A}$, and $\mathbf{I}$ is the identity matrix [25].

The Laplacian matrix has the property of being expressible as the eigendecomposition $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^{\mathrm{T}}$, where $\mathbf{U}$ is the eigenvector matrix, and Λ is the diagonal matrix of eigenvalues.

The graph Fourier transform is defined as $\mathcal{F}(\mathbf{x}) = \mathbf{U}^{\mathrm{T}}\mathbf{x}$, and the inverse graph Fourier transform is defined as $\mathcal{F}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}}$, where $\hat{\mathbf{x}}$ is the result from the graph Fourier transform.

This is used in the graph convolution, where $\mathbf{x}$ is the input signal, and $\mathbf{g}$ is the filter. The convolution is then defined as

$$\begin{aligned} \mathbf{x} * \mathbf{g} =& \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{g})) \\ =& \mathbf{U}(\mathbf{U}^{\mathrm{T}}\mathbf{x} \odot \mathbf{U}^{\mathrm{T}}\mathbf{g}) \end{aligned} \tag{2.9}$$

where $\odot$ is the element wise product. By denoting $\mathbf{g}_\theta = diag(\mathbf{U}^T\mathbf{g})$, the expression becomes $\mathbf{x} * \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^{\mathrm{T}}\mathbf{x}$.

Spectral convolutions without any approximations or simplifications have some limitations [5]. They assume graphs to be undirected, which is not problematic for meshes but is for many graphs. They also rely on eigendecomposition, which is an $\mathcal{O}(n^3)$ operation, making them expensive to use. There exist multiple methods like ChebNet [26] and GCN [27] that reduce the computation complexity to $\mathcal{O}(m)$ [5].

2.2.6.2 Graph Convolution Network (GCN)

The *Graph Convolution Network* or GCN is the currently most cited graph convolutions. The convolution is proposed by Kipf and Welling [27] as a fast approximate approach to the spectral graph convolutions.

From a spatial-based perspective, the graph convolution propagates information along the mesh edges. Given vertex features f_i, the updated features are calculated as shown in Equation 2.10 [28].

$$\hat{f}_i = \sigma(W_0 f_i + \Sigma_{j\in\mathcal{N}(i)} W_1 f_j) \tag{2.10}$$

Here, $\hat{f}_i$ is the updated features for the i-th vertex, σ is a non-linear activation function, $\mathcal{N}(i)$ are all vertex neighbors to node i in the graph, and W_0 and W_1 are the learned weight matrices. This convolution only utilizes the first-order neighbors of the node. Multiple applications of the filter allow convolution of the k^{th}-order neighbors.

The method proposed approximates the spectral graph convolution and can be both done in the spectral and spatial domain. This makes it so that no expensive eigendecomposition or eigenvalue-multiplication operations are needed [27].

This graph convolution has been successfully used in recent general mesh-based approaches [29, 30] and is one of the graph convolution operators implemented in the Pytorch Geometric framework [20] written about in Section 3.2.2.

2.2.7 Loss

In facial recognition, the goal is to identify the same identities while distinguishing between separate identities correctly. Multiple loss functions have been proposed and are actively used in the 2D face recognition field [2]. As there exist over 27 different high-performing loss functions used for face recognition [2], it is hard to choose a single one. Generally, they all work by separating dissimilar samples while clustering the similar samples.

This Thesis focuses on two different methods for training a face recognition network: the distance based triplet loss [31] and the binary cross-entropy trained siamese network [32].

2.2.7.1 Pairwise ranking loss

One of the simpler loss groups used in face identification and validation is based on the ranked loss principle [33]. These loss functions work with the distance between embeddings generated from a single neural network.

The simplest ranked loss function is the *pairwise ranking loss* [32]. The loss function takes two different embeddings and compare the euclidean distance $||f(x_i) - f(x_j)||_2$ between the inputs.

The distance is minimized if the two inputs are in the same group, like the same identity. In comparison, if the two inputs are dissimilar, like two different identities, the distance is maximized. This means that the loss function clusters the inputs of the same group and separates inputs of different groups.

The loss function can be modified to use a margin, restricting the function when negative pairs are distant enough [34], restricting training to more difficult pairs. The formula is described in Equation 2.11, where δ is the loss, $d(\cdot,\cdot)$ is the distance, x_a is the anchor sample, x_p is a positive sample, x_n is a negative sample, and m is a pre-defined margin.

$$\delta = \begin{cases} d(x_a, x_p), & \text{if positive pair.} \\ max(0, m - d(x_a, x_n)), & \text{if negative pair.} \end{cases} \tag{2.11}$$

As shown in Equation 2.11, the loss will be 0 if the input pair is negative and the pair distance is over the margin. On the other hand, positive pairs will always have their distance be minimized.

2.2.7.2 Triplet loss

Another popular ranked loss function is the triplet loss proposed by Weinberger and Saul [35]. The idea is to compare three samples, an anchor x_a, a positive sample x_p, and a negative sample x_n. The goal is to reduce the distance between the anchor and positive sample while simultaneously increasing the distance between the anchor and negative sample. Compared to the Pairwise ranking loss, triplet loss is less restrictive and learns faster [2]. The loss function is described in Equation 2.12 and illustrated in Figure 2.3.

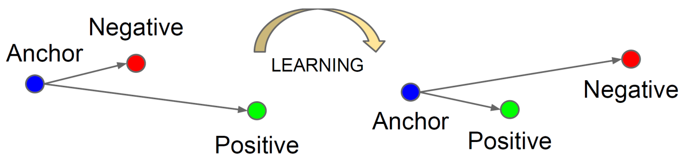


Figure 2.3: Figure from Schroff *et al.* [8] showing Triplet Loss minimize the distance between the *anchor* and the *positive* while maximizing the distance between the *anchor* and the *negative*.

$$\delta(x_a, x_p, x_n) = max(0, m + d(x_a, x_p) - d(x_a, x_n) \tag{2.12}$$

There are three different situations possible when using the loss function. They are different combinations of samples, called triplets, which give the loss function different properties.

The first triplets are called the easy triplets, where $d(x_a, x_n) > d(x_a, x_p) + m$. Here, the negative sample is distant enough, and the loss is 0, so the network parameters are not updated.

Another type of triplets are the hard triplets, where $d(x_a, x_n) < d(x_a, x_p)$. This means that the negative sample is closer to the anchor than the positive, and the loss will greater than m. These are the optimal triplets to train on. A technique called hard mining is used either *offline* before each epoch or *online* during each batch to find these triplets.

The last triplets are the semi-hard triplets. They happen when $d(x_a, x_p) < d(x_a, x_n) < d(x_a, x_p) + m$, meaning that the positive is closer to the anchor than the negative, but the distance is not greater than the margin. The loss will still be positive, but less than m.

A problem with triplet loss is that the model may collapse [8]. This is a scenario where the model finds a bad local minima where every model is mapped to the same point. Here, the distance between any two faces will be 0, and the loss will be equal to the margin. This is especially an issue if hard mining is used early on. One possibility to reduce the issue is by using L_2 norm on the descriptors, limiting the descriptor distance to be 1, placing every identity on a D-dimensional hypersphere [8].

2.2.7.3 Binary Cross Entropy loss

Cross entry loss is one of the most commonly used loss functions for training deep neural network models, most notably classification problems [36]. When training with cross entropy loss, the optimizer will update the weights so that the predictions get closer to the correct label. The binary cross-entropy loss is a special case that only has two classes.

The input to binary cross entropy is a label and a prediction. The label represents the two classes and uses either the value 0 or 1. Binary cross entropy is described in Equation 2.13, where $\ell(x, y)$ is the loss function, x is the prediction score between 0 and 1, and y is either the label 0 or 1. For face validation, the two different classes are genuine-pair and imposter-pair. The label will be 1 if the pair of data is of the same identity and 0 if they are not.

$$\ell(x, y) = -y \cdot \log x - (1-y) \cdot \log(1-x) \tag{2.13}$$

2.2.7.4 Siamese Network

A siamese network architecture is a network style that contains two identical sub-networks that is joined at their output [37]. The sub-network use the same weight in both pass-throughs to extract features called the descriptors. The features are then passed to another part that is used to determine the similarity score between the pairs.

There are multiple ways to construct a siamese network. One possibility is to determine the similarity by purely relying on the distance between the descriptors without any neural network. Here, pairwise ranking loss or triplet loss can be used to learn the network to discriminate between faces.

An alternative is to use a neural-based siamese network. Here, a Sigmoid can be used to limit the output between 0 and 1, allowing the usage of binary cross entropy (2.2.7.3) as the loss function. Such a neural siamese network will be able to figure out the complex mappings itself. This will help the network as different parts of the embeddings-space might be more important than others.

2.2.8 Training

Training a neural network can be generalized into a few steps. The training data is passed through the network leading to an output. A loss function is then applied to the output, which generates an error. This error is used to optimize the model parameters via a backward-propagation algorithm to accelerate the process [17, p. 200].

The learning and back-propagation process can be performed automatically and hardware-accelerated by frameworks like PyTorch [38] and TensorFlow [39].

During training, the network is validated on a validation set to see how the model performs unseen data. Lastly, the model is used on a separate set called the test-set to get the final metric. Information like the error on the validation-set and test-set can indicate how the model is performing.

2.2.9 Generalization, Overfitting, and Underfitting

As stated by Goodfellow *et al.* [17, p. 110-116], the goal of a machine learning model is to perform well on new previously unused data. This ability is called generalization. Two factors determine the performance of the machine learning algorithm. First, the algorithm should make the training error small while keeping the gap between the training and test error small.

Two challenges in machine learning are underfitting and overfitting. Goodfellow *et al.* [17] defined underfitting as when the model is not able to gain a sufficiently low error on the training set and overfitting as when the gap between training and the test error is too large. Both underfitting and overfitting are a product of the model's complexity, or in other words, the model's capacity.

Generally, overfitting means that the model has a low error on the training data while it cannot predict well on unseen data. This means that the model has memorized the training data. Overfitting is often monitored by seeing the accuracy and error on the validation data. When the model starts overfitting, the validation error will go up. Overfitting may be caused by too much training or a model with too high capacity.

On the other side, underfitting means that the model is not able to generalize. For example, if a model is too simple, the model's capacity will not be able to represent the problem and will perform poorly. Underfitting can be seen as the training loss flattening at a high error rate.

2.2.10 Optimizer

An optimizer is an algorithm that minimizes the loss [17, p. 151-153]. The simplest optimizer is the gradient descent [17, p. 294] which finds the best gradient for the entire dataset. There exist multiple optimizations of this algorithm. One way is the inclusion of mini-batching. Mini-batching helps by calculating the average gradient on a sub-set of the dataset instead of the entire dataset to approximate the gradient. By using fewer samples, the weights are updated more frequently, which speeds up the learning process compared to having to go through the entire dataset each time.

Another technique to speed up training is the usage of momentum [17, p. 296-300]. Momentum uses previous gradients in an exponentially decaying average in addition to the regular gradient. This means that previous gradients can help in the optimizing process.

A popular implementation of these principles is the Adaptive Momentum Estimation (Adam) optimizer [40]. Adam uses both the first and second-order momentum in conjunction with individual adaptive learning rates. The adaptive learning rates set a parameter-individual upper learning rate limit and allow parts of the network that are already almost optimized not to change too much.

For all optimizers, the most important parameter is the learning rate. The learning rate dictates how much the weights are updated based on the gradient. Larger values allow faster learning but can be unstable as the weights change too much [17, p. 238]. On the other hand, smaller values will make the network take excess time to learn and may overfit the network [41].

2.2.11 Hyperparameters

Hyperparameters are the variables that defined a neural network before training begins. They contain information on the model structure and parameters such as learning rate and batch size [17, p. 422-431]. Hyperparameters-optimization of the network architecture and structure is one of the most important to tweak. A non-trivial problem regarding machine learning is the selection of optimal hyperparameters. There are multiple approaches for optimizing the hyperparameters. One way is manual search, where manual experimentation and reasoning is used to optimize the variables. Another alternative is to use automatic searching algorithms like grid search to try out multiple configurations and select the best one.

2.2.11.1 Network Architecture

The network architecture and structure decide if the model will be able to generalize. Before training, specifications like input and output size, number of hidden layers, and filter need to be specified.

Generally, a bigger network has the ability to perform better but will also be harder to optimize and train and increases the chance that the model memorizes the dataset resulting in overfitting. On the opposite side, a smaller network will be easier to train but may not generalize for a given problem [17, p. 110-115].

Regarding 2D convolutional networks, new networks are often based on existing architectures that have been tested. This helps the new models by having a baseline on which to build and improve upon. On the other hand, for 3D convolutional algorithms, not many models exist, which makes finding the correct balance more time-consuming.

In combination with selecting all these parameters. Larger networks often need more data to be able to generalize, which makes it harder to determine if the problem is with the network or data [17, p. 426]. Generally, implementing some sort of data augmentation is recommended whenever creating a shallow or deep model, as it will help the model for unseen data.

To optimize the network, one way is to monitor metrics like loss, validation loss, and accuracy. A model which is not able to generalize properly will have an unstable loss or low accuracy.

2.2.11.2 Regularization

Regularization is any modification to a learning algorithm intended to reduce its generalization error but not its training error [17, p 120]. Regularization aims to reduce the test error and increase generalization, possibly at the expense of the training error.

The best way to make a model generalize better is to acquire more training data [17]. A simple way to do this is via dataset augmentation. Dataset augmentation is the act of modifying or creating new synthetic data based on existing data. The new data will allow the network to train on more cases, increasing generalization.

For 3D data, there exist multiple data augmentations. The simplest is the affine transformations, namely translation, rotation, scale, and share. These techniques slightly change the 3D structure while keeping the data similar to the original. For meshes, it is also possible to rebuild all edges on the same vertices as a regularization technique.

A technique called batch normalization can also be implemented to make training more stable and faster [42]. Batch normalization works by normalizing the data, followed by re-centering and re-scaling. The normalization help training while the re-centering and re-scaling allow the network to represent the same functions, making sure the expressive power of the network is maintained [17, p. 320]. While improving network optimization, batch normalization has also been shown to have a regularization effect on networks [17, p. 268].

Chapter 3

Related Work

This chapter discusses academic publications related to 3D facial recognition and the tools used in the Thesis.

3.1 Academic publications

Most state-of-the-art 3D face recognition consists of traditional data-driven approaches. Moreover, most related deep-learning-based state-of-the-art actually performs a projection from 3D space to 2D space to address working with 3D data in a deep-learning model. Thus, they are not using the actual properties of the 3D manifold. This is the main difference between the proposed approach in this Thesis and some of the publicized work discussed in this section.

A summary of the academic publications gone through in this section is available in Table 3.1.

Table 3.1: Summary of the previous academic deep-learning-based 3D facial recognition models. 2.5D references to the use of a projection from 3D to 2D + inferred 3D information like depth.

Publication		Type	Processing time
Kim *et al.* [43]	3.1.2	2.5D	3.25s[1]
Zulqarnain Gilani and Mian [44]	3.1.3	2.5D	- [2]
Cai *et al.* [10]	3.1.4	2.5D	0.84s[1]
Bhople *et al.* [45]	3.1.5	Point Cloud	0.020s[3]

[1] Time to identify a probe from a gallery of 466 faces.
[2] No interference time given in the publication.
[3] Interference time per sample.

3.1.1 Traditional 3D Face Recognition

3D facial recognition can be performed by either using traditional methods or deep learning-based methods. The traditional methods can be divided into three main categories: holistic, local, and hybrid [46, 47].

The holistic approaches describe the face via global features and use the global similarity between faces for recognition. The most used techniques are principal component analysis (PCA) and deformation models [48]. The local methods, also called feature-based algorithms, use geometric features of the face to extract information about the identity [4]. These often use key point detection to find and create features from smaller known regions like the eyes and nose.

The hybrid methods utilize both holistic and local methods to do face recognition. However, after Soltanpour *et al.* [48], local methods generally better identify 3D faces than holistic methods, especially in noisy environments.

3.1.2 Deep 3D Face Identification

The first 3D facial recognition based on deep learning was proposed by Kim *et al.* [43]. They created their model from the pre-trained model Deep Face Recognition [49], a network created for 2D facial recognition, and fine-tuned the model with 3D scans.

To use the 3D data in a 2D-trained CNN, the 3D point clouds were projected with an orthographic projection onto a 2D plane. This method creates a depth map of the face which is further resized to match the pre-trained model input resolution of $224 \times 244 \times 3$ where the last dimensions are 3 duplicates of the same depth map.

To combat performance drop due to variations in expressions, they proposed a method for augmenting 3D face datasets by synthesizing new expressions with multi-linear 3D morphable models.

The model was later criticized by Cai *et al.* [10] as the architecture and structure were intrinsically designed for 2D facial recognition and that no specific optimization for 3D facial recognition was added. Zulqarnain Gilani and Mian [44] concluded that approaches similar to this are sub-optimal as "3D data has its own peculiarities defined by the underlying shape and geometry", which networks designed for 2D facial recognition would not have seen before.

Their proposed model uses 3.25 seconds, including time spent on pre-processing, to identify one identity in a gallery of 466 images. The majority of the time is used for the pre-processing step, which takes 3.16 seconds. This model achieved comparable performances to the state-of-the-art traditional methods.

3.1.3 Learning from Millions of 3D Scans for Large-scale 3D Face Recognition

The second attempt on a 3D facial recognition network based on deep learning was made by Zulqarnain Gilani and Mian [44]. They created and trained a model on a large-scale synthetic 3D facial dataset consisting of 3.1 million 3D scans and 100 000 unique identities. Originally, they started with 1785 3D scans and combined them to create the dataset. The synthetic dataset was created in response to the lack of any large-scale 3D facial datasets.

For the model, they projected the 3D model into 2D space. The resolution of the projected image is $160 \times 160 \times 3$, containing depth, azimuth, and elevation angles of the normal vector. During training, they discovered that larger kernel sizes in the 2D convolution layers gave better results because 3D facial surfaces are generally smooth, making the larger kernel size generated better abstractions than smaller filters.

The result of training on the large generated dataset in combination with a new proposed network designed for 3D facial recognition made the model outperform both the traditional deep learning-based state-of-the-art 3D facial recognition algorithms.

Their proposed model outperforms the model created by Kim *et al.* [43] discussed in Section 3.1.2. This shows that training a network designed for 3D facial recognition and with a large-scale dataset yields better results than trying to fine-tune a network designed for 2D facial recognition.

3.1.4 A fast and robust 3D face recognition approach based on deeply learned face representation

Cai *et al.* [10] proposed another method for 3D facial recognition. Their approach is another projection-based model, where a depth map is projected from raw 3D data.

Their technique creates four depth images with varying zoom. The first image contains the entire face, while the last image contains only the nose. Each image is fed into a separate network followed by combining the descriptors. The model uses both triplet loss and softmax as the loss function under training, followed by using both joint Bayesian and Euclidean distance for matching.

The proposed architecture performs as well as the state-of-the-art methods. It is the fastest model compared to all previously reported results, using 0.84s for matching a probe scan against a gallery of 466 faces. They do not directly compare themself to the model proposed by Zulqarnain Gilani and Mian [44], discussed in Section 3.1.3, as their publication does not provide any time analysis.

3.1.5 Point cloud-based deep convolutional neural network for 3D face recognition

Bhople *et al.* [45] proposed PointNet-CNN, a PointNet [21] based architecture that directly extracts features from point cloud data for face recognition. Their model then takes the features of two scans into a siamese network to predict their similarity.

They train and test the model on two datasets, the Bosphorus [50] dataset and an in-house dataset called IIT Indore. Compared to the other approaches tested and trained on different datasets, Bhople *et al.* trained and tested on the same dataset. They also limited the number of scans per identity in Bosphorus to 31 samples, where originally it has between 31 and 54 scans.

They created an equal amount of genuine and imposter pairs to have both balanced training and evaluation results. As input, the data were normalized, restricted to 2048 points, followed by augmentation by random rotation, translation, and permutation.

The proposed model ended up with an AUC of 99.4% and a recognition rate of 98.91% on Bosphorus. However, they do not specify the split exact used in the metric, the selected samples, nor the false acceptance rate used in the recognition rate. This makes it harder to compare the results compared to other methods.

In their publication, they do not compare the performance against any of the deep-learning-based methods. However, based on the reported result, the model performed better than the proposed method by Cai *et al.* [10], and worse than Zulqarnain Gilani and Mian [44][2].

3.2 Tools

For this project, multiple tools and frameworks were used. These are briefly described in the following subsections.

3.2.1 PyTorch

PyTorch [38] is Facebook's open-source framework for machine learning. The framework allows custom models and networks to run on accelerated hardware (CUDA).

3.2.2 PyTorch Geometric

PyTorch Geometric is a framework created by Fey and Lenssen [20]. This framework is an extension to Pytorch that enables deep learning on graphs, point clouds, and manifolds and implements multiple recent publicized Geometric Deep Learning methods and functions.

Some alternatives to this framework are Nvidia's kaolin for PyTorch, Google's TensorFlow, and Facebook's Pytorch3D. PyTorch Geometric was chosen as it was the most mature framework.

3.2.3 OnlineMiningTripletLoss Pytorch

OnlineMiningTripletLoss is an open-source implementation for PyTorch of batchwise triplet mining created by Rishaug [33]. The project is available under the

[2]We could not find Zulqarnain Gilani and Mian [44] identification rate on Bosphorus, but it is stated by Bhople *et al.* [45] that "There is one technique proposed in [44], which has shown perfect performance on Bosphorus 3D face data ...".

MIT License.

Chapter 4

Methodology

This section describes the implementation of the proposed 3D facial recognition model.

4.1 Overview

The model outlined in this Thesis was created due to the limited amount of models that use Geometric Deep Learning techniques for 3D facial recognition. To the best of our knowledge, there is only one promising model that that does that, proposed by Bhople *et al.* [45], which is based on the PointNet [21] architecture.

The proposed architecture of this Thesis implements 3D facial recognition by directly working on meshes combined with Geometric Deep Learning techniques for generating features. By directly using a mesh, expensive pre-processing operations as 3D to 2D projection used by other deep learning approaches [10, 43, 44] would be reduced or entirely removed.

The goal of this Thesis was to propose a new mesh-based network that would perform 3D facial recognition comparable to that of the state-of-the-art methods in terms of accuracy and time.

4.2 Model Architecture

The proposed network has two parts: the convolutional feature extraction network and the siamese network. Similar FaceNet [8], a convolution network is used to extract features from faces in the form of descriptors for face identification and validation. A new network had to be designed as the input, output, and filter size works differently between 2D convolutions and mesh convolutions so the network could not be based on an existing model.

To get the input from a variable amount of vertices down to a predictable set of features, the network uses the global max pooling used by Pointnet [21] and Pointnet++ [15], further explained in Section 2.2.3.

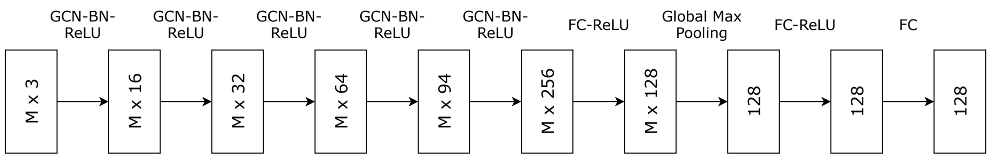


Figure 4.1: Figure of the proposed feature extraction architecture. In the figure, M denotes to number of vertices in the input, GCN refers to Graph Convolution Network [27] (2.2.6.2), BN to Batch Normalization, FC to fully connected (2.2.4), and ReLu to Rectified Linear Unit [23] (2.2.5).

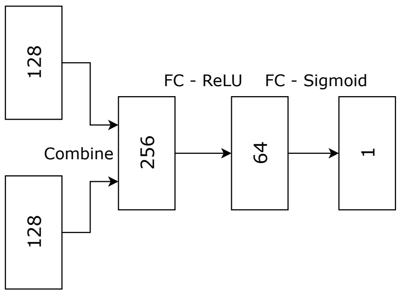


Figure 4.2: Figure of the siamese network architecture. FC refers to fully connected (2.2.4), and ReLu to Rectified Linear Unit [23] (2.2.5).

The proposed feature extraction structure can be summed up in Figure 4.1. The figure shows that the network begins with 5 blocks containing graph convolution, batch normalization, and ReLU that increase the per vertex filter size from 3 (the x,y,z coordinates) up to 256 features. This is followed by a fully connected layer that reduces the features down to 128 features per-vertex before the network is pooled down to 128 features. A fully connected layer with ReLU maps the features to descriptors before it is returned. The input for the model is a variable size mesh, and the output is either a descriptor vector of size 128.

Lastly, the descriptors were compared to do facial validation between pairs. Initially, the siamese network seen in Figure 4.2 was used. Pairs of extracted descriptors are inputted, combined, and passed through two fully connected layers. The last fully-connected layer uses Sigmoid instead of ReLU to return a number between 0 and 1. This number indicates the similarity of the input pair. Later, a triplet loss variation was also created that performed face validation via the euclidean distance between the descriptors.

The entire pipeline can be seen in Figure 4.3. Firstly, the two meshes are pre-processed before they are each passed through the feature extraction. The descriptors are then compared with the siamese network. The descriptors of a gallery-set can be run through the feature extraction network once and saved for later use. This allows the gallery-descriptors to be directly inserted into the siamese network together with the descriptors from a probe speeding up face identification. For ranking, a probe is matched against all descriptors in the gallery and the scan with the highest score is selected as the most likely identity.

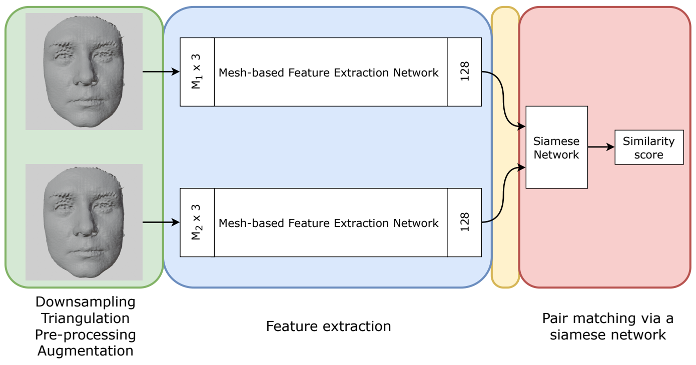


Figure 4.3: Figure of the proposed pipeline. Two meshes are pre-processed before inputted into a feature extraction network. The feature extraction networks are the same network in both instances.

4.2.1 Pre-processing

Currently, the proposed network requires pre-processing in order to get the data in a form suited for the model.

Firstly, if the data is in a point-cloud form, some sort of triangulation is required. Based on the method used by the datasets Bosphorus [50] and FRGCv2 [51], Delaunay triangulation is applied[3] to get a mesh input for the model.

If the input data has too many vertices, mesh sampling is applied to reduce the complexity of the 3D model. This step was either done using the quadratic-error-based polygon simplification method Theoharis *et al.* [53, p. 187-189] or via uniformly sampled vertices. By reducing the mesh quality, more scans fit into memory, allowing a larger batch size. This creates a trade-off between batch size and sampling. More sampling reduces the mesh quality, but a larger batch size is important for the model to learn features common for multiple faces.

Centering and scale normalization was used to standardize the input. Every model had its data normalized between -1 and 1. This generally helps the network learn quicker as no value dominates the gradient [17]. Every dataset had the 3D object located in a different place in the 3D space. For example, BU-3DFE has the models placed around the origin while the models in FRGCv2 are located -2500 in the Z-axis.

4.2.2 Hyperparameters

For the proposed model, multiple hyperparameters had to be chosen or experimented with. In general, all hyperparameters were chosen and optimized via manual experimentation.

Some parameters were chosen based on the work of others. The number of descriptors used is 128, as proposed by Schroff *et al.* [8], which showed that this

[3]This step is done via SciPy's [52] implementation of Delaunay triangulations. (doc)

allowed separating over 200 million facial images. The inspiration for experimenting with triplet-loss also came from the same source. The usage of Adam as the optimizer with a learning rate of $1 \cdot 10^{-3}$ was chosen from experimentation done by Wardeberg [1] in combination with manual tweaking. The Adam coefficients were the default recommendation from the original paper [40].

As described in Section 2.2.11.1, the number of convolutions and filter sizes is important for a model to perform optimally. The proposed models' convolutional hyperparameters, like the number of convolutions and convolution filter, were firstly based on Wardeberg [1], followed by manual experimentation. The usage of GCN, described in Section 2.2.6.2, was inspired from Wardeberg [1].

When trained with triplet loss, the network had L_2 normalization used on the last layer. The training was performed via online triplet loss, with a margin of 0.2.

4.2.3 Training

During training, pairs of scans are created and labeled either as a genuine pair or an imposter pair. The problem is that there will be more imposter pairs than genuine pairs, making the training unbalanced.

Take a dataset where each identity has 25 scans, and the maximum number of identities that fit in the memory is 10. This will create $(\frac{25 \cdot 24}{2} + 25) \cdot 10 = 3250$ total unique pairs including pairs of the same identity, and $\frac{(25 \cdot 10) \cdot (25 \cdot 9)}{2} = 28125$ unique imposter pairs. In big O notation, where n is the number of scans per person, and m is the number of identities, the number of imposter pairs grows with $\mathcal{O}(n^2 m^2)$ and the genuine pairs with $\mathcal{O}(n^2 m)$.

A limitation was added to the imposter pairs to combat the unbalance. For both training, validation, and testing accuracy, the amount of negative pairs is sampled to be equal to the number of positive pairs. For training, pairs were selected randomly, while for the validation and testing accuracy, the same random seed was used to have reproducibility. This reduced the efficiency of training on each epoch as many possible pairs were thrown away, making the network need more epochs to see all possible pairs.

For all experiments, training was run at enough epochs for the validation loss to become minimal. As specified in Section 4.2.2, the optimizer Adam [40] was used with a learning rate of $1 \cdot 10^{-3}$. We trained on an Nvidia Titan X (pascal) with 12 GB of memory. The batch size was tweaked to fill 90% of the available memory, giving room for some minor mesh variance. Experimentation on the sampling amount was done to find a balance between the maximum amount of meshes that would fit into the memory while not reducing the quality too much. Bhople *et al.* [45] sampled their data to 2048 points while achieving good results, showing that input sizes of that resolution still allow for high accuracy. In the end, the original meshes were downsampled to around 2048 vertices +- 64 after experimentation. This allowed around 250 meshes to be loaded into the memory, using around 11GB.

Before the final evaluation, multiple augmentation techniques were tested on the BU-3DFE dataset [54] to see how different 3D augmentations affected the results. The feature extraction network and siamese network used for the augmentation test were the structures shown in Figure 4.1 and Figure 4.2.

For evaluation, the model was trained in multiple ways. The model was first trained and tested on the same dataset for each dataset, similar to that of Bhople *et al.* [45]. This style of evaluation tests how the model predicts on new identities taken with the same 3D scanner and environment. A split of 80% training data and 20% testing data was used[4]. The problem with this approach is that the datasets used in this Thesis are relatively small, and with experiments (4.3.3) that divided up the data into smaller sets, the testing data would contain very few scans and identities. To help against the small probe-sets, every evaluation had an *All vs. All* type experiment added to get more accurate curves in combination with the other dataset-specific experiments. This type of experiment includes some never seen harder pairs compared to the other experiments.

The second way of training was inspired by Kim *et al.* [43], Zulqarnain Gilani and Mian [44], and Cai *et al.* [10], where they would train and test on different datasets. This helps the network by having more data to train and test on. The style of training and testing also indicates how the model predicts on new datasets taken with different scanners and environments. Unfortunately, every publication trains on different combinations of datasets and augmentations, so the comparison is not standardized and may differ based on the datasets trained on.

4.3 Performance Evaluation

We performed multiple experiments on the three 3D face datasets, that is, BU-3DFE [54], Bosphorus [50], and FRGCv2 [51]. This section describes the background information for the evaluations and analyses, which evaluation metrics were recruited, and how the different experiments were conducted. For evaluation, the experiments generally follow the same procedure implemented in Cai *et al.* [10] and Kim *et al.* [43] but with exceptions.

4.3.1 Sets

For face recognition evaluations, experiments are carried out and distinguished between the gallery-set and the probe-set.

The gallery-set contains all known scans and identities. These are the faces we want to recognize and match against. An example of an image in the gallery-set is a passport photo. This photo is later used for identification or verification of a person's identity. The other set is the probe-set, containing all unknown scans we want to identify among the gallery set. Each scan can either be identified among the gallery-set or as an unknown identity. A threshold is used to limit the descriptor distance or matching score when identifying someone as unknown.

[4]This split was not similar to Bhople *et al.* [45], as they used a different ratio and also minimized the maximum scans per individual.

Sets can either be a closed set or an open set [3]. If a probe-set is closed, it means that every identity is known in the gallery-set. If the set is open, the probe-set may contain unknown faces which do not have a correct sample in the gallery-set. For all experiments done in the Thesis, closed sets were used.

4.3.2 Metrics

For each experiment, multiple metrics were taken. The following sections contain the concepts, intermediate metrics, and final metrics used.

4.3.2.1 Rank1

Rank-1 in the context of facial identification means selecting the sample closest to the probe. An example would be to compare the distance between descriptors or a matching score. Rank-1 means that if multiple identities match, only the best one is used, even if multiple or no identities would be classified as positive. Rank-1 can also be extended to Rank-N, where the N closest matches are used to see if any of them are correct.

Rank is an important metric in closed-set experiments to validate identification results [3]. When testing, a higher Rank-N metric means a better model.

4.3.2.2 True positive, True negative, False positive, False negative

In face verification, four outcomes are possible. These are based on the actual label of a pair, either genuine or imposter, and the predicted label output from the model. The model uses a threshold when predicting to determine the result. This will be a max-length for descriptors, and for a siamese neural network, it will be a score threshold. This threshold will be determined by what characteristics the network will have.

Pairs can either be genuine, also called positive, or an imposter pair, also called a negative pair. The four outcomes based on the actual label and the predicted label can be seen in Table 4.1.

A higher threshold means that the model only accepts high confidence positives, and more of the samples will be false negatives. A lower threshold will have more false positives. This threshold value is important to achieve a specific false acceptance rate, further elaborated on in Section 4.3.2.4.

Table 4.1: Possible outcomes based on actual label and predicted label.

		Actual	
		Positive	Negative
Predicted	Positive	**True Positive**	False Positive
	Negative	False Negative	**True Negative**

A true positive or true negative means that the model correctly predicted the actual label, while a false positive or false negative means that the model misclassified the pair.

4.3.2.3 Accuracy, Precision, and Recall

Multiple metrics are defined by comparing the four base classes. The simplest one is the unbalanced accuracy, as shown in Equation 4.1. This predicts the percentage of the correct predictions. There will both be an accuracy for the identification and the verification as they test different things. The downside is that it may incorrectly show good results if the input is unbalanced. In a scenario where the ground truth is negative in most cases, a model that always predicts false will gain an artificially high accuracy score.

One way to solve the unbalanced accuracy is to measure precision and recall [17]. Precision is defined as the fraction of positive predictions by the model that was correct. On the other side, recall, also called true positive rate, is the fraction of the positive pairs that were correctly predicted [17, p 423-424]. They are both described mathematically in Equation 4.2 and Equation 4.3. Optimally they should be as high as possible, but there is often a trade-off between them.

When a model has no predictive, the verification and identification rate will be around two different values. The verification rate will be equal to 50% if the input is balanced. The identification rate will be around the expected value of the binomial distribution $X \sim Bin(n, p)$, where n is the amount of scans in the probe-set and p is the probability that a random gallery-scan will have the same identity as a randomly selected probe-scan.

$$Accuracy^{5} = \frac{tp + tn}{tp + tn + fp + fn} \tag{4.1}$$

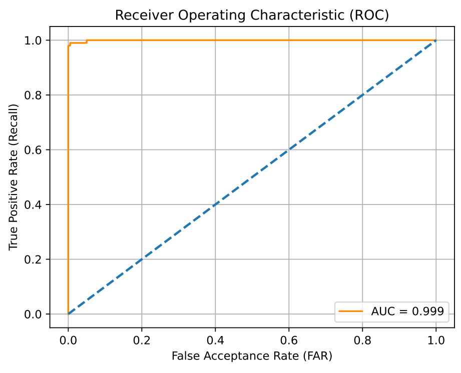
$$Precision = \frac{tp}{tp + fp} \tag{4.2}$$

$$Recall = \frac{tp}{tp + fn} \tag{4.3}$$

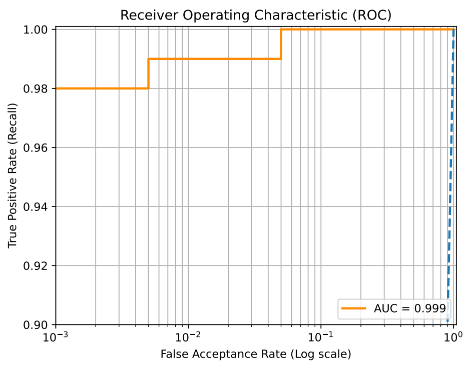
4.3.2.4 False Acceptance Rate, False Reject Rate, and Receiver Operating Characteristic

In facial verification, an important characteristic is the False Acceptance Rate (FAR) (Equation 4.4), also called False Positive Rate. This metric is balanced against the False Reject Rate (Equation 4.5), also called False Negative Rate. As the name implies, they give the probability that a face is falsely accepted or rejected. The metric is used when presenting the verification rate (VR) of a system. Here, the threshold value is selected such that the specified false acceptance rate is reached. This threshold is then used when calculating the verification rate.

[5] unbalanced



(a) Linear ROC curve.



(b) logarithmic ROC curve.

Figure 4.4: An artificially created almost ideal Receiver Operating Characteristic curve with an AUC of 0.999 shown in a linear (4.4a) and logarithmic scale (4.4b). The orange line shows the ideal ROC, while the blue line shows a model with no predictive power, and that is randomly guessing.

In terms of security, the false acceptance rate is often set to be at most 0.1% or 0.01% depending on the application. Both metrics are both important as wrongfully identifying a person means that the system is not secure while failing to verify a person means that the system is unusable.

$$False\ Acceptance\ Rate = \frac{fp}{fp + tn} \tag{4.4}$$

$$False\ Reject\ Rate = \frac{fn}{fn + tn} \tag{4.5}$$

Another important way of validating verification results is via a Receiver Operating Characteristic (ROC) curve and the Area Under Curve (AUC) score. The ROC curve shows the relationship between recall and the false acceptance rate at different thresholds. In other terms, it shows the relationship between genuine and imposter pairs and how well the system distinguishes between faces [55]. In an ideal system, the true positive rate (recall) will be 1.0 for all false acceptance rates. An artificially created almost ideal ROC curve is presented in Figure 4.4.

The Area Under Curve (AUC) shows a summarized performance of the ROC curve at all different thresholds. The score is equal to the probability that the classifier will rank a randomly drawn positive sample higher than a randomly drawn negative sample, i.e., $P(score(x_p) > score(x_n))$ [56]. The ideal system will have an AUC of 1.

4.3.2.5 Cumulative Match Curve

For close-set face identification, the Cumulative Match Curve (CMC) is an important rank-based metric. In contrast to the ROC curve, which measures verifica-

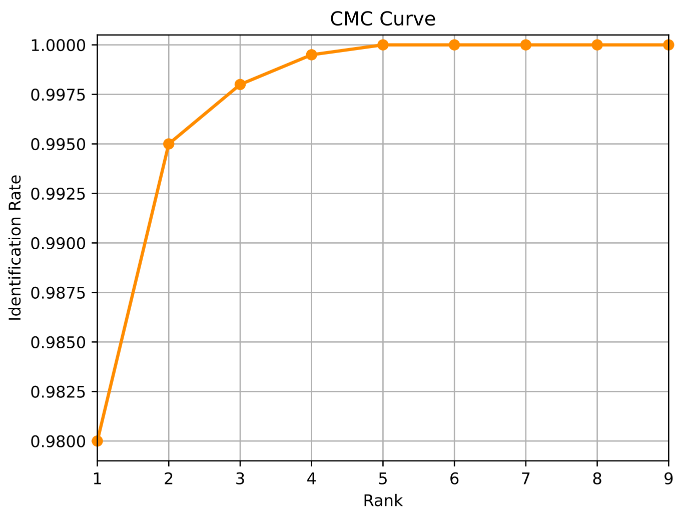


Figure 4.5: An artificially created almost ideal Cumulative Match Curve (CMC) with limited y-axis.

tion performance, the CMC curve measures identification performance at different rank-N's [55].

The CMC curve is generated by matching each probe against the gallery and evaluating whether a true positive is within the N ranks. This creates a graph with rank-N on the x-axis and true positive identification rate (IR) within N ranks on the y-axis. Ideally, the true positive identification rate should be as high as possible. Figure 4.5 illustrates an artificially created, almost ideal, CMC, where the Rank-1 equals 98%. In an ideal curve, this would be 100%.

4.3.3 Datasets

The datasets used for training 3D facial recognition are rather limited compared to the 2D counterpart. For example, most datasets consist of only a few hundred identities and a small number of poses for each identity [7]. A summary of the most popular datasets is listed in Table 4.2, showing the number of unique identities and total scans available in each dataset.

A combination dataset called *LS3DFace*, created by Zulqarnain Gilani and Mian [44] from multiple 3D datasets, including those shown in Table 4.2, contains only 1853 IDs and 31 860 scans, which shows the sparsity of 3D face datasets.

For this Thesis, three datasets were selected and used for experiments similar to [10, 43–45]. These are BU-3DFE [54], Bosphorus [50], and FRGCv2 [51]. A more detailed summary of the datasets is presented in Table 4.2. In addition, example images and 3D scans are presented in Figure 4.6.

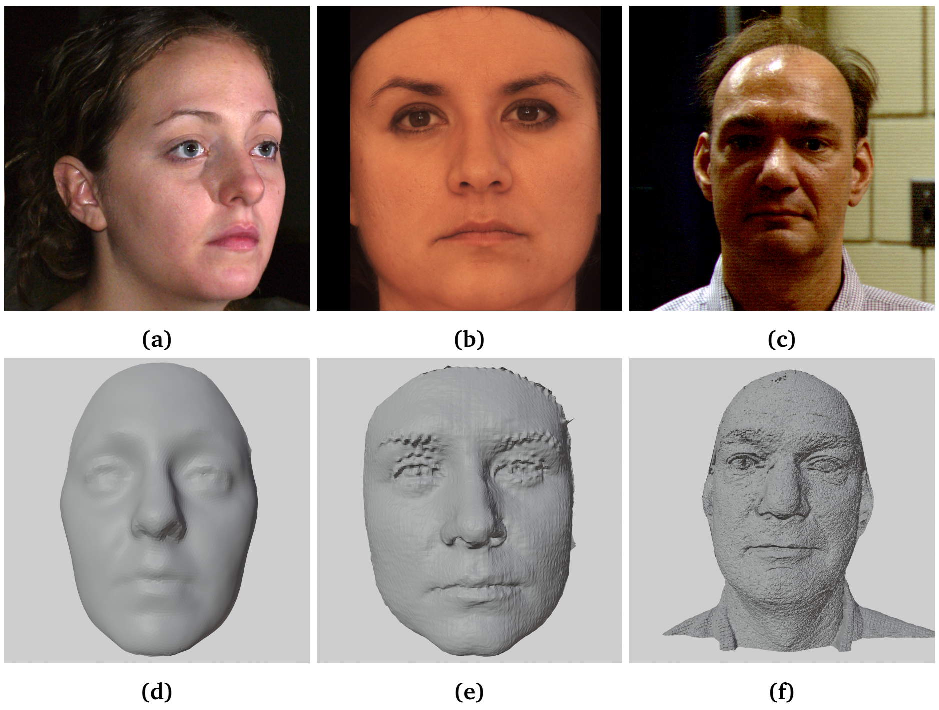


Figure 4.6: Example data from the datasets used. Figure 4.6a, 4.6b, and 4.6c are cropped versions of the 2D images while 4.6d, 4.6e, and 4.6f are the corresponding 3D models shown as triangulated 3D meshes projected via a perspective projection. Figure 4.6a and 4.6d are from BU-3DFE, 4.6b and 4.6e are from Bosphorus, and 4.6c and 4.6f are from FRGCv2. Both the 3D image from Bosphorus and FRGCv2 (4.6e, 4.6f) are originally a point cloud, but for visualization purposes, triangulation is applied.

Table 4.2: Information of the contents, resolution, and scan-by-scan variance of the most popular 3D datasets used in face recognition.

Dataset	Identities	Scans	Resolution	Cropping	Variance
BU-3DFE [54]	100	2500	≈ 8k Vertices, 23k Edges, 15k Faces	Face Only	Neutral + 6 Expressions
Bosphorus [50]	105	4666	≈ 135k Vertices	Face Only	Neutral + 6 Expressions + Rotation, Occlusion, & Various face actions
FRGCv2 [51]	466	4007	≈ 2.5M Vertices	Face & Shoulder	Between 1 and 22 scans Neutral + 3 Expressions Controlled & Uncontrolled environment
3D-TEC [57]	214	428	70k to 195k Vertices (avg 135k)	Face Only	Neutral + Smiling
ND-2006[1] [9]	888	13450	≈ 112k Vertices (frontal)	Face & Shoulder Face crop	Neutral + 5 Expressions

[1] ND-2006 is a superset of FRGCv2

Table 4.3: Distribution of the relevant face scans in the Bosphorus dataset.

Scan type	Number
Neutral	299
Expression	453
Lower Face Action Unit (LFAU)	1549
Upper Face Action Unit (UFAU)	432
Combined Action Unit (CAU)	169
Total	2902

4.3.3.1 BU-3DFE

The BU-3DFE dataset [54] contains 2500 scans of 100 individuals. Each individual has six different expressions recorded at different intensities and one neutral scan. These are angry, disgust, fear, happiness, sadness, and surprise, where each of these has four different intensities ranging from 1 to 4. An intensity of 1 or 2 is considered low intensity, while 3 and 4 are considered high intensity [10].

The BU-3DFE dataset has a wide range of diversity in terms of age, gender, and ethnicity while also having a lower resolution than the other datasets, as shown in Table 4.2. Both the diversity and lower resolution make the dataset more challenging than other datasets [10].

The experiment on BU-3DFE follows the procedure done by Cai *et al.* [10]. The neutral scans are used as the gallery-set, while compared against low-intensity expressions, high-intensity expressions, and lastly, both the low and high-intensity expressions combined.

4.3.3.2 Bosphorus

The Bosphorus dataset [50] contains 4666 scans of 105 individuals ranging from age 25 to 35. For each individual, multiple scans were taken with different types of variations. The dataset contains at least one neutral face and six expressions, together with various rotations, occlusions, upper- and lower-face actions.

This Thesis follows the same procedure as Kim *et al.* [43] and Cai *et al.* [10], where scans marked as occlusion, ignored, and rotation is not used. The total number of scans used in the experiment then becomes 2902, as shown in Table 4.3.

For the experiments, the first neutral scan from each 105 identities is selected to be in the gallery-set. The rest of the data is used in three different experiments. The first experiment is *neutral vs. neutral*, where the probe-set is the remaining 194 neutral scans. The second experiment is *neutral vs. non-neutral*. Here, the probe-set contains 2603 scans from the expression, lower face action unit, upper face action unit, and combined action unit sub-sets. The third experiment is *neutral vs. all*, making the probe-set contain 2797 scans.

4.3.3.3 FRGCv2

The third dataset is the FRGCv2 dataset [51]. The experiment setup followed a modified procedure to that of Kim *et al.* [43], Zulqarnain Gilani and Mian [44], and Cai *et al.* [10]. Due to the modified procedure, the results are not directly comparable but will help the reader get an idea of the proposed methodology.

As proposed by the creators of the dataset [51], the dataset is divided into three partitions based on the acquisition of the data. Namely, the Spring2003-set, consisting of 275 individuals and 943 scans, the Fall2003-set with 370 individuals and 1893 scans, and the Spring 2004-set with 345 individuals and 2114 scans. Combined, the total for all three sets is 557 unique identities with a total of 4950 scans.

As done by Cai *et al.* [10], the Spring2003 dataset is used for training while the two other sets are used for validation. The total size of the validation set is 4007 scans of 466 unique individuals.

Normally the dataset is split into three experiments. Namely, *neutral vs. neutral*, *neutral vs. non-neutral*, and *neutral vs. all*. These experiments could not be done as the FRGCv2 dataset used in this Thesis does not contain the expression data. An additional experiment done by Cai *et al.* [10] could also not be implemented as the web page containing the probe list no longer exists. As a replacement for the experiments, this Thesis uses a new split. The first scan of each subject is used as the gallery, while the rest of the scans are used as probes.

Chapter 5

Results

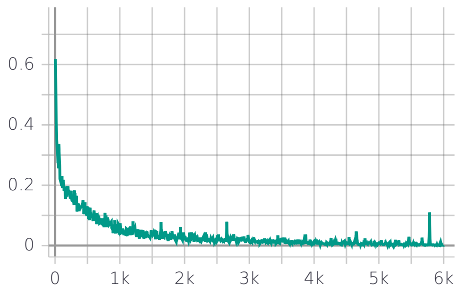
This chapter contains the results from the training and experimentation for this Thesis. Firstly, it describes the loss during training in Section 5.1. This chapter then contains various experiments. These include augmentation experiments (5.2.1), cross-data testing (5.2.2), and a siamese network test (5.2.3). This is followed by the final results for the triplet net (Section 5.3.1) on BU-3DFE, Bosphorus, and FRGCv2, and the final results for the siamese network (Section 5.3.2). A time analysis of the model is presented in Section 5.3.3, and lastly, an overview of the final results and comparisons against other approaches is presented in Section 5.4. Every table and figure is further explained in the appropriate section.

5.1 Training

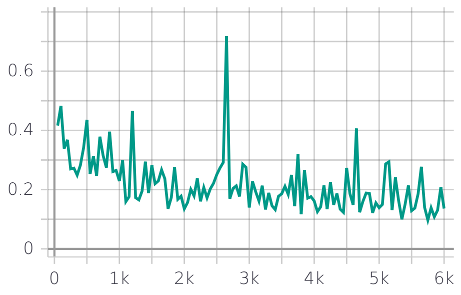
The training and testing loss on the BU-3DFE dataset when using the siamese network can be seen in Figure 5.1. Near the end of the training, the training loss was in the range $[0.005, 0.003]$, while the testing loss was in the range $[0.14, 0.15]$. The lowest point for the test data was at epoch 5750 with a loss of 0.09.

As laid out in Section 2.2.9, underfitting and overfitting can be determined by the training and test loss. As the training loss is smaller than the testing loss, a gap of around 0.14 is created. This shows overfitting indicating an issue with the model's capacity or a problem with the data.

These results were similar but less extreme on the Bosphorus dataset, where there was a gap of around 0.09. For FRGCv2, the gap was 0.01, meaning that little overfitting was seen.



(a) Training loss



(b) Training loss loss

Figure 5.1: Plot of training and training loss on the BU-3DFE dataset with the siamese network. The y-axis is the loss from binary cross entropy while the x-axis is epochs.

5.2 Experiments

5.2.1 Augmentation results

The augmentation experiments were performed by gradually changing the amount of augmentation. Every experiment used some sort of pre-sampling for the scans to fit into CPU and GPU memory, specified in the *mesh resolution* column. The results, shown in Table 5.1, are divided up into two parts: one trained where the input mesh topology stayed consistent, and one where the data was resampled before each time they were used.

Table 5.1 show that the accuracy went down when applying normalization but slowly increased as more augmentation was applied. When re-sampling, more points did not necessarily mean higher accuracy.

The best results, outlined in bold, show that *Center + Normalize Scale + Translation 0.01 + Rotate 5 degrees 3 axis* generally performed the best except in rank-1. The model *Center+Normalize Scale+Resample 1024 points* managed to beat the augmentation combination by 0.83 percentage points.

The augmentation results are discussed in Section 6.1

Table 5.1: Various augmentation experiments and the corresponding results. Rank-1 IR, and AUC is the *neutral vs. all* experiment while VR were from *all vs. all* experiment.

Augmentation style	Mesh resolution (vertices)	Rank-1 IR	AUC	VR at 1% FAR	VR at 0.1% FAR	Epoch
Center	2048	80.00%	99.00%	87.79%	67.46%	4800
Center + Translation $0.01 \cdot max(scan)$ + Rotate 5 degrees 3 axis	2048	86.04%	99.32%	90.42%	71.56%	2400
Center + Normalize Scale	2048	76.25%	97.75%	82.35%	62.68%	4050
Center + Normalize scale + translation 0.01 + Rotate 5 degrees 1 axis	2048	83.96%	98.24%	84.52%	62.79%	5100
Center + Normalize Scale + Translation 0.01 + Rotate 5 degrees 3 axis	2048	90.21%	**99.40%**	**91.34%**	**75.92%**	5950
Center + Normalize Scale + Resample 512 points	4096	70.42%	97.40%	79.05%	68.29%	4450
Center + Normalize Scale + Resample 1024 points	4096	**91.04%**	99.27%	88.55%	73.42%	3900
Center + Normalize Scale + Translation 0.01 + Rotate 5 degrees 3 axis Resample 1024 points	4096	90.63%	98.94%	87.77%	73.35%	4750
Center + Normalize Scale + Resample 2048 points	4096	88.53%	99.32%	87.64%	70.03%	3500

5.2.2 Cross-dataset testing

For the cross-dataset testing, the model was trained with a single or multiple datasets, followed by using the last to test. The best results, and results relevant for the discussion are then presented in Table 5.2.

When trained on multiple datasets, it always performed worse than when training on a single one. Also, the model only managed to get good performance when it was trained and tested on the BU-3DFE and Bosphorus dataset. Little accuracy improvements were achieved when training or testing on the FRGCv2 dataset.

When the model was trained on BU-3DFE, it managed to get a 72.7% rank-1 identification rate and a 77.2% verification rate at 0.1% false acceptance rate at Bosphorus. The results on FRGCv2 were 3.5 rank-1 IR and 51.3% VR, close to a model that would have no predictive power.

When the model was trained on Bosphorus, it managed to get a 63.3% rank-1 IR and 69.2% VR at BU-3DFE,

Table 5.2: Results when training and testing on different datasets. Only the best-resulting results are shown.

		BU-3DFE		Bosphorus		FRGCv2	
Augmentation technique	Trained on	Rank-1 IR	VR at 0.1% FAR	Rank-1 IR	VR at 0.1% FAR	Rank-1 IR	VR at 0.1% FAR
Center + Norm + Resample 1024 Siamese	BU-3DFE	–	–	39.8%	56.9%	1.3%	50.0%
Center + Scale norm + Translate 0.01 + Rotate 5 degrees, 3 axis Triplet loss	BU-3DFE	–	–	**72.7%**	**77.2%**	3.5%	51.3%
Center + Scale norm + Translate 0.01 + Rotate 5 degress, 3 axis Triplett loss	Bosphorus	**63.3%**	**69.2%**	–	–	4.0%	51.4%
Center + Scale Norm + Translate 0.01 + Rotate 5 degrees, 3 axis Triplet loss	FRGCv2	21.2%	55.1%	27.8%	53.4%	–	–

5.2.3 Siamese experiment

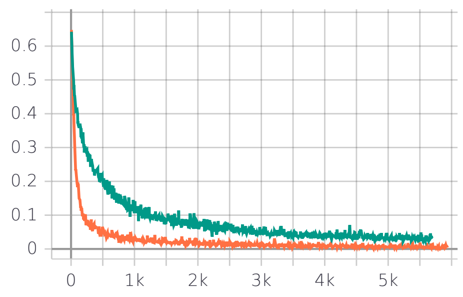
As outlined in Section 5.1 and will be discussed in Section 6.2, there is a problem with the siamese network. This section contains the result of using a different neural siamese network. This new network does not impact the triplet network and has not been used in any other results than Table 5.3 and Figure 5.2.

Table 5.3 shows the result from the siamese network used in this Thesis, the improved one, and the triplet net on the FRGCv2 dataset. The result shows that the improved siamese network outperforms the one in this Thesis. The table also shows that there is a similar performance between the triplet net and the more improved siamese network. Due to time limitations and that the improved siamese net performed similarly to the triplet net, the improved siamese net was not used.

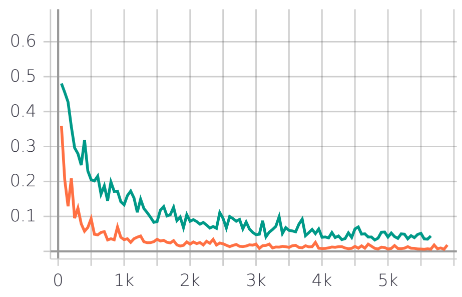
Section 5.3 shows the training and testing loss on the FRGCv2. Here, the orange line is the improved and green the original. The orange line converges quicker, and there is a smaller gap between the training and testing loss, meaning that it does not experience the same overfitting.

Table 5.3: Comparison between the used siamese network, a slightly improved siamese network, and the triplet net. The results are from training and testing on FRGCv2.

Experiment	Rank-1 IR	AUC	VR at 1% FAR	VR at 0.1% FAR
Original Siamese network	90.12%	99.95%	99.40%	96.26%
Improved Siamese network	98.90%	99.99%	99.85%	98.28%
Triplet net	98.65%	99.99%	99.55%	99.55%



(a) Training loss



(b) Testing loss

Figure 5.2: Plot of training and testing loss. The green line is the used siamese network, and the orange is the improved (and not used in the results) network discussed in Section 6.6.

5.3 Final results

This section contains the final results for the dataset.

Triplet loss results

When trained with triplet loss, the results look promising. On BU-3DFE [54] (Figure 5.3), the results look promising. It reached a 99.6%, 95.0%, and 92.5% rank-1 identification rate in the *neutral vs. low-intensity*, *neutral vs. high intensity*, and *neutral vs. all* experiments, respectively. In addition, the model achieved a 92%, 88%, and 77% verification rate at 0.1% false acceptance rate on the same experiments. The AUC was between 99.2% and 99.7% for all experiments. Based on the results, the model struggled the most with *neutral vs. high-intensity*.

The results on Bosphorus [50] (Figure 5.4d) are a bit better than the results on BU-3DFE. The model achieved a 98.2%, 95.1%, 94.0% rank-1 IR, and a 100%, 86.0%, and 86.0% VR at 0.1% FAR on the *neutral vs. neutral*, *neutral vs non-neutral*, and *neutral vs. all* experiments, respectively. The model managed to get a 100% AUC on the *neutral vs. neutral*, meaning that it verified all pairs correctly on the reduced balanced set. However, due to a rounding error, explained in Appendix A, the graph shows only a VR of 50%. This should be 100% for every FPR less than 100%.

The results on FRGCv2 [51] (Figure 5.4d) are almost perfect. The *first vs. rest* experiment reached a 98.65% rank-1 identification rate and a 99.55% verification rate at 0.1% false acceptance rate. The model managed to get a 100% identification rate at rank-2.

Siamese loss results

The proposed approach performs worse when using the siamese network outlined in Figure 4.2 than when using triplet loss.

On the BU-3DFE dataset [54], as seen in Figure 5.6, it obtained a 90.21% rank-1 IR, an 86.35% VR at 0.1% FAR at the *neutral vs. all* experiment. The AUC for the experiments was between 99.0% and 99.5%.

On Bosphorus [50], seen in Figure 5.7, the proposed approach achieved a 90.08% rank-1 IR, an AUC of 99.60%, 91.07% VR at 0.1% FAR on the *neutral vs. all* experiment. Like the triplet network trained Bosphorus result, the *neutral vs. neutral* AUC was 100%, inducing a rounding error and incorrectly creating the graph. This is further explained in Appendix A, and the correct graph should show 100% for every FPR less than 100%.

The siamese performed better on the FRGCv2 [51] dataset than the BU-3DFE and Bosphorus datasets. On FRGCv2 it reached a 90.12% rank-1 IR, an 99.95% AUC, and a 96.26% VR at 0.1% FAR.

5.3.1 Triplet loss results

5.3.1.1 Triplet loss results on BU-3DFE

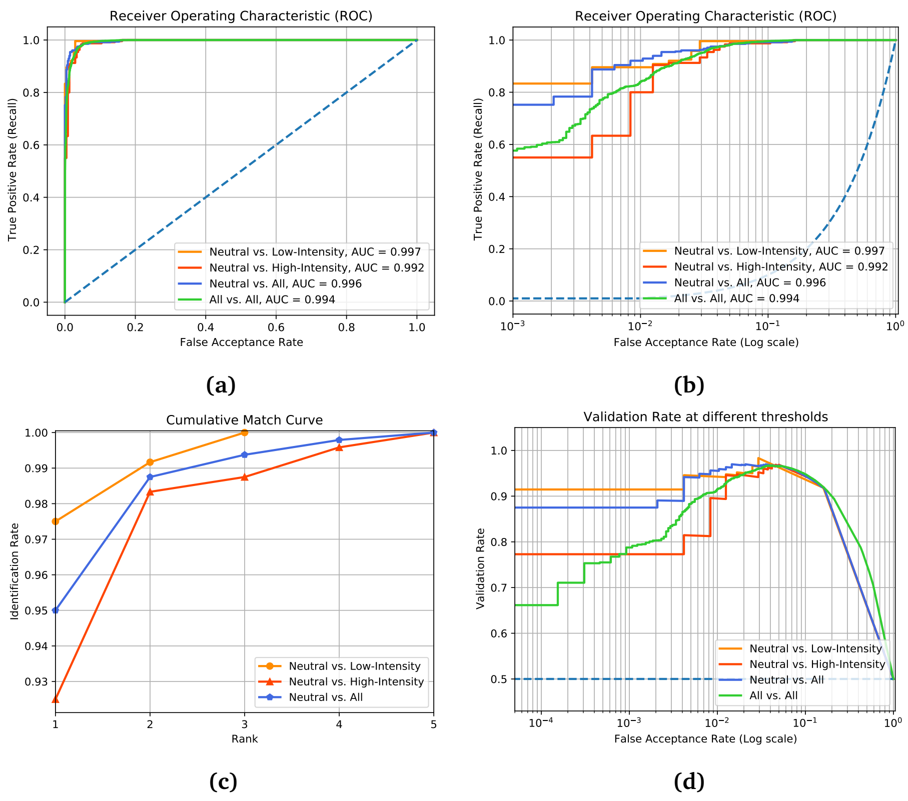
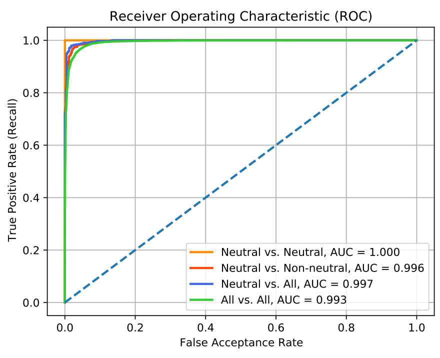
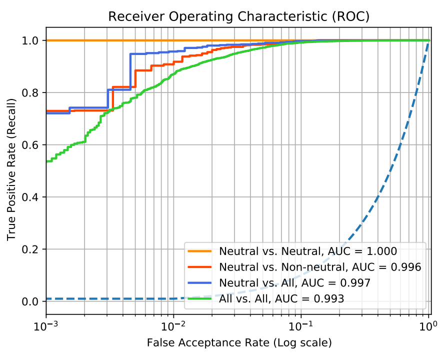


Figure 5.3: ROC, CMC, and FAR-VR curves on the **BU-3DFE** dataset. 5.6a shows the linear ROC curve. 5.3b shows the logarithmic ROC curve. 5.3c shows the CMC curve. 5.3d shows the Validation Rate plotted against False Acceptance Rate.

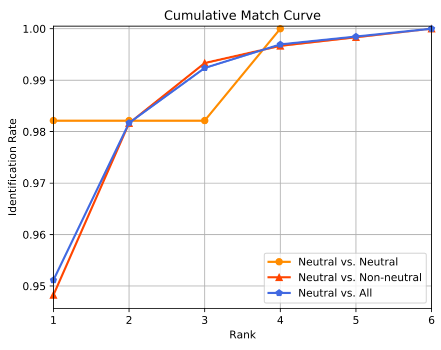
5.3.1.2 Triplet loss results on Bosphorus



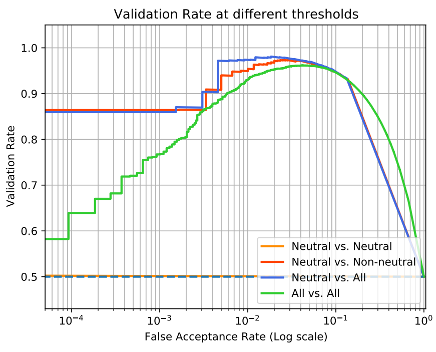
(a)



(b)



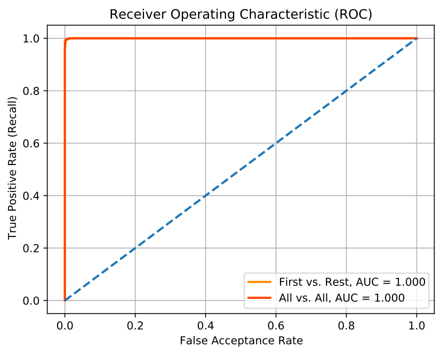
(c)



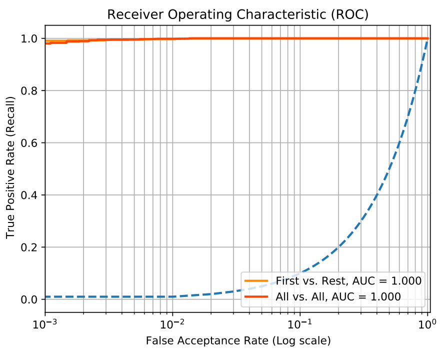
(d)

Figure 5.4: ROC, CMC, and FAR-VR curves on the **Bosphorus** dataset. 5.4a shows the linear ROC curve. 5.4b shows the logarithmic ROC curve. 5.4c shows the CMC curve. 5.4d shows the Validation Rate plotted against False Acceptance Rate.

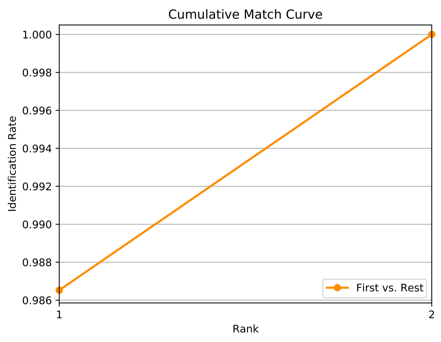
5.3.1.3 Triplet loss results on FRGCv2



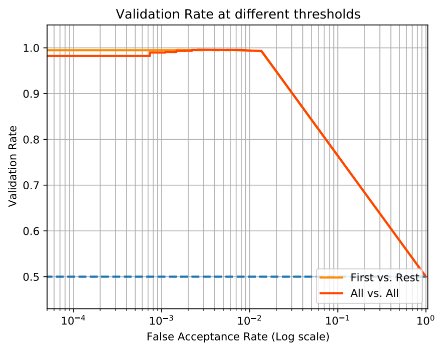
(a)



(b)



(c)

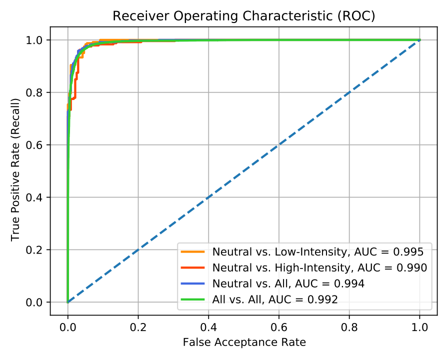


(d)

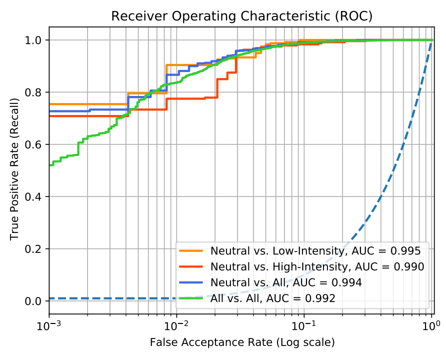
Figure 5.5: ROC, CMC, and FAR-VR curves on the **FRGCv2** dataset. 5.5a shows the linear ROC curve. 5.5b shows the logarithmic ROC curve. 5.5c shows the CMC curve. 5.5d shows the Validation Rate plotted against False Acceptance Rate.

5.3.2 Siamese results

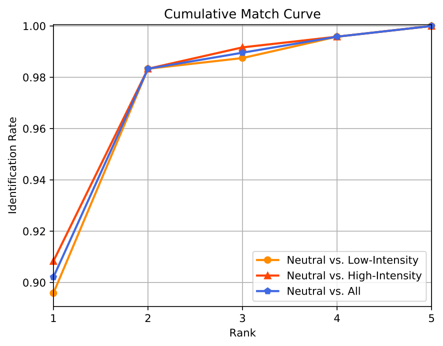
5.3.2.1 Siamese results on BU-3DFE



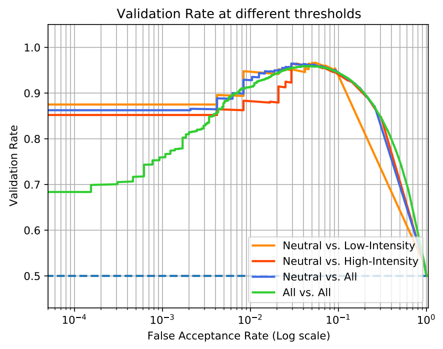
(a)



(b)



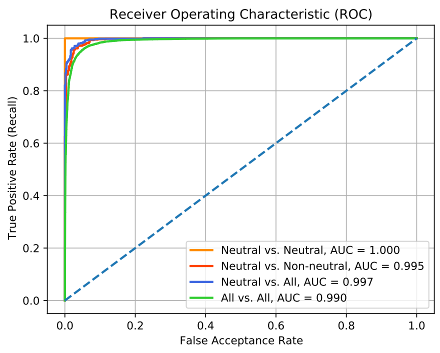
(c)



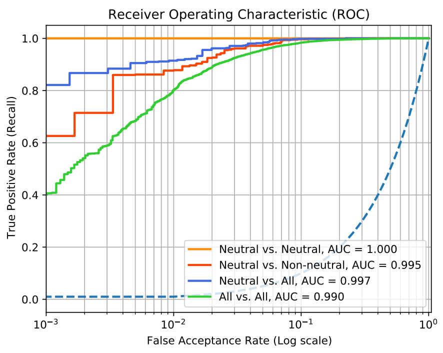
(d)

Figure 5.6: ROC, CMC, and FAR-VR curves on the **bu-3dfe** dataset. 5.6a shows the linear ROC curve. 5.6b shows the logarithmic ROC curve. 5.6c shows the CMC curve. 5.6d shows the Validation Rate plotted against False Acceptance Rate. The blue striped line in 5.6a and 5.6b shows the random-guessing line discussed in Section 4.3.2.4.

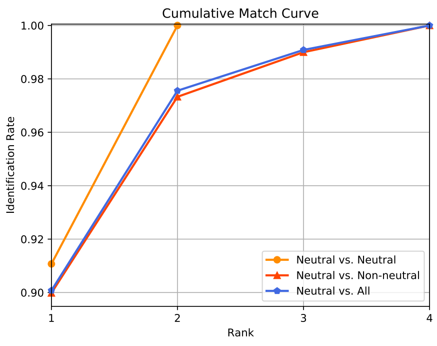
5.3.2.2 Siamese results on Bosphorus



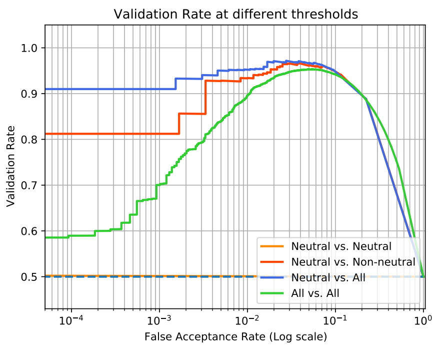
(a)



(b)



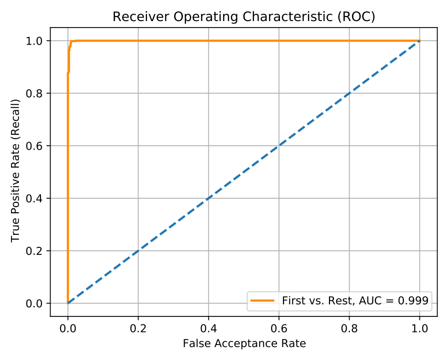
(c)



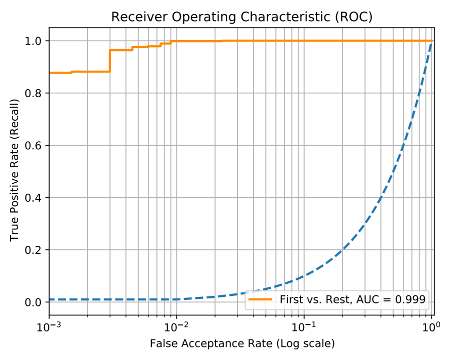
(d)

Figure 5.7: ROC, CMC, and FAR-VR curves on the **Bosphorus** dataset. 5.7a shows the linear ROC curve. 5.7b shows the logarithmic ROC curve. 5.7c shows the CMC curve. 5.7d shows the Validation Rate plotted against False Acceptance Rate. The blue striped line in 5.7a and 5.7b shows the random-guessing line discussed in Section 4.3.2.4.

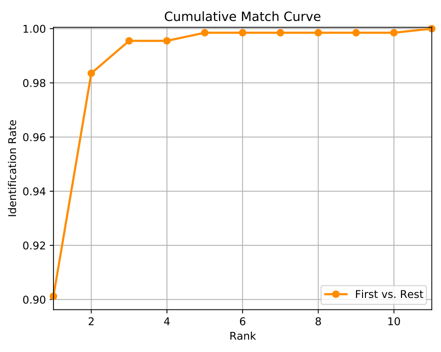
5.3.2.3 Siamese results on FRGCv2



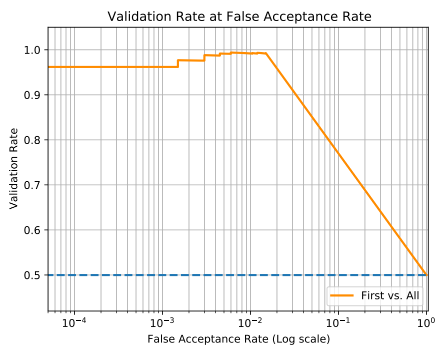
(a)



(b)



(c)



(d)

Figure 5.8: ROC, CMC, and FAR-VR curves on the **FRGCv2** dataset. 5.8a shows the linear ROC curve. 5.8b shows the logarithmic ROC curve. 5.8c shows the CMC curve. 5.8d shows the Validation Rate plotted against False Acceptance Rate. The blue striped line in 5.8a and 5.8b shows the random-guessing line discussed in Section 4.3.2.4.

5.3.3 Time analysis

The time analysis, shown in Table 5.4, was take on an Intel Core i7-8700K CPU and an NVIDIA TITAN X (pascal) GPU. The table is divided up into three parts, not necessarily required, required, and total. The total time used by the network to match one probe against a gallery of size 466 is 8.042 milliseconds with subsampling and triangulation and 5.353 milliseconds without subsampling and triangulation. The sampling, triangulation, preprocessing, and ranking depend on the CPU, while feature extraction and ranking depend on the GPU.

The preprocessing and feature extraction time assumes a mesh of 2048 vertices. The feature extraction time is increased to 45.586ms when simultaneously extracting features for 100 identities, showing that simultaneous feature extraction grows less in time complexity than n for small numbers.

Table 5.4: Time analysis to match one probe in a gallery of 466 images.

Type	Time
Sampling (per sample, not required)	496µs
Triangulating (per sample, may be required)	2.193ms
Preprocessing (per sample)	46µs
Feature extraction (single)	5.225ms
Mid-processing	1µs
Siamese Network	74µs
Ranking	7µs
Total with subsampling and triangulation	8.042ms
Total without subsampling and triangulation	5.353ms

5.4 Overview

A summary of the main results presented in this Thesis is shown in Table 5.5. Our proposed methods' identification and verification rate are compared against other deep-learning methods in Table 5.6 and Table 5.7. Lastly, an interference time comparison is presented in Table 5.8.

The comparisons show that our proposed model performs a bit under the other state-of-the-art methods in respect to identification rate and verification rate. Based on Table 5.8, our method is the fastest when compared against the state-of-the-art methods, using around 8 milliseconds to identify a singly identity in a gallery of 466 faces.

Table 5.5: Summary of the results in this Thesis on the *neutral vs. all* experiment for BU-3DFE and Bosphorus, and *first vs. rest* for FRGCv2.

	Siamese neural network			
Dataset	Rank-1 IR	AUC	VR at 1% FAR	VR at 0.1% FAR
BU-3DFE [54]	90.21%	99.40%	92.92%	86.35%
Bosphorus [50]	90.08%	99.69%	95.27%	91.07%
FRGCv2 [51]	90.12%	99.95%	99.40%	96.26%
	Distance based Siamese network trained with tripplet loss			
Dataset	Rank-1 IR	AUC	VR at 1% FAR	VR at 0.1% FAR
BU-3DFE [54]	95.00%	99.60%	95.63%	87.60%
Bosphorus [50]	95.11%	99.79%	97.40%	86.03%
FRGCv2 [51]	98.65%	99.99%	99.55%	99.55%

Table 5.6: Comparison of Rank1 Identification Rates between different deep-learning-based methods in the *neutral vs. all* experiment on BU-3DFE, Bosphorus, and FRGCv2.

Method	BU-3DFE	Bosphorus	FRGCv2
Kim *et al.* [43]	95.0%	99.2%	–
Zulqarnain Gilani and Mian [44]	**99.9%**	**100.0%**	99.9%
Cai *et al.* [10]	99.88%	99.75	**100.0%**
Ours (Same-set-test)	95.00%	95.11%	98.65%[1]

[1] Used a non-standard experiment as discussed in Section 4.3.3.3.
– Not reported

Table 5.7: Comparison of deep-learning-based methods on Verification Rate at 0.1% False Acceptance Rate on the *neutral vs. all* experiments on the datasets.

Method	BU-3DFE	Bosphorus	FRGCv2
Cai *et al.* [10]	**98.92%**	98.39%	**100.0%**
Bhople *et al.* [45]	–	**98.91%**[1,2,3]	–
Ours[1]	87.60%	86.03%	99.55%[2]

[1] Used a non-standard split.
[2] Used a non-standard experiment.
[3] Does not specify FAR threshold.
– Not reported

Table 5.8: Comparison of processing time for one feature extraction and matching a probe against a gallery of 466 faces.

Approaches	Processing	Matching	Total	Relative factor
Kim *et al.* [43]	3160ms	90ms	3250ms	401.1x
Zulqarnain Gilani and Mian [44] [1]	–	–	–	–
Cai *et al.* [10]	760ms	80ms	840ms	104.5x
Bhople *et al.* [45] [2]	20ms	–	–	2.5x[(3)]
Ours$^{\text{with-sub}}$	7.960ms	0.082ms	8.042ms	1x
Ours$^{\text{no-sub}}$	5.271ms	0.082ms	5.353ms	0.67x

[1] No interference time given in the publication.
[2] Interference time per sample.
[3] Estimated.
$^{\text{with-sub}}$ Processing time with subsampling and triangulation.
$^{\text{no-sub}}$ Processing time without subsampling and triangulation.

Chapter 6

Discussion

This chapter contains the discussion and findings of this Thesis.

6.1 Augmentation results

In this Thesis, we compared the performance of different augmentation methods on the BU-3DFE dataset. The summary of the metrics is displayed in Table 5.1.

The augmentation results show some observations. Firstly, there is a loss of accuracy when applying scale normalization. This is logical as size information is lost during this transformation, and the network has to rely on ratios instead of absolute distances. The problem is that this transformation may be a necessary evil. For example, if the network relies too much on absolute distances, variations like distance from the scanner may impact the performance. Interestingly, it does not perform much differently when applying augmentation to the non-scale-normalized data than the scale-normalized data. This means that scale must be important, but the network uses some other feature of the scan to separate scans.

Secondly, small changes in rotation and translation help the network learn better. The simple augmentation methods make the network more robust to small variations in the rotation that can naturally occur from scanning. Zulqarnain Gilani and Mian [44] stated that larger kernels worked better on 2D depth images compared to smaller kernels as 3D faces are generally smooth. Therefore, individual point translations have little effect on the overall geometric structure while still helping the network become less sensitive to absolute positions. This helps the network learn multiple possible variations of the same identity.

Based on the results, two augmentation styles distinguish themselves. Namely, resampling with no augmentation and using the original mesh with maximum augmentation. Generally, the model trained on the original mesh performed just a bit better. Based on the experiments, when re-sampling the mesh, the optimal number of vertices is around 1024 points. Augmentation by resampling gained no extra performance when adding more augmentations techniques. This means that a uniform sampling of the mesh worked efficiently as a regularization technique and that changing the underlying mesh before sampling had little effect. Minor point translation will be of little significance when the entire mesh is resampled from a larger mesh each time.

For the experiment that did not resample the mesh at each epoch, there was a slight benefit when using 2048 vertices instead of 1024. This is in contrast to when using re-sampling, where 1024 points seemed to work better. One possibility is that the network is overfitting, and therefore more restrictive data helps the model create more optimal features at the expense of a lower theoretical maximum accuracy. While the model is overfitting, it is not able to hit the theoretical accuracy, and therefore performs better as the data is simpler.

6.2 Underperforming feature extraction

The proposed network displays a lower than expected identification rate and verification rate when compared to other approaches. The network must therefore be performing suboptimally. As seen in the earlier 3D face recognition methods [10, 43, 44], it is possible to achieve perfect performance on most datasets when using a projection-based approach. Bhople *et al.* [45] had similar results when using their geometric deep learning approach. This means that it is possible to perform well on the datasets.

Based on the training and testing loss, the model is able to learn the training data but not generalize enough to perform as well on the test data. This means that the feature extraction network or the siamese network is overfitting and is an indication that the capacity of the network is too high.

The performance became better when using the triplet network, indicating a problem with the siamese network. This problem is further discussed in Section 6.6.

Even with the triplet network, the model is performing slightly under that of the other proposed methods. A possible explanation for this is that the model cannot extract optimal features and that the training finds a set of suboptimal features that can separate the training data, but that is not prevalent in the test data.

There is a possibility that the network is too shallow for the geometric information to traverse throughout the network. If this is the case, repeated application of the graph convolutions or implementation of vertex-pooling may help the issue. This issue is further discussion discussed in Section 6.3.

Another possibility with the feature extraction network is that it requires more data for finding optimal features. As the datasets are relatively small, the model might not see enough different samples to learn the optimal features. As shown in the augmentation results in Table 5.1, the models that had more augmentation generally performed better than those without. The experiments that used the combined datasets did not improve the accuracy. This issue is further discussed in Section 6.8.

The pre-subsampling may also cause some issues. It is possible that the pre-subsampling step removes some of the important geometrical information in the structures, making it harder for the network to perform on the test data. This is less likely as Bhople *et al.* [45] managed to get close to state-of-the-art results with 2048 points, equal to that of the pre-subsampling used in this Thesis. This assumes that a mesh of size 2048 will perform around equal to that of a point cloud of size 2048, which may be incorrect as the convolution techniques used on point clouds and meshes operate differently.

The last possibility is that mesh convolutions may have a harder time with inter-class separability. This Thesis provides no results that definitively prove or disprove this statement. Further investigation into this topic is presented in Section 6.3.

6.3 Graph convolution limitation

A possibility is that the graph convolution struggles on interclass separation problems. Using a graph convolution on a mesh has been done before [29, 30, 58], which shows that a graph convolution can be used on mesh structures. The fact that graph convolution works on meshes suggests that GCNs should work on interclass separation problems, but it does not prove it.

There is also a problem with multiple applications of graph convolutions. Chen *et al.* [59] states that most graph convolution networks such as GCN (Kipf and Welling [27]) and GAT (Chen *et al.* [60]) achieved their best performance with 2-layer modes. Shallow architectures limit a model's ability to extract information from higher-order neighbors. This is an issue because graph convolutions perform Laplacian smoothing that mixes features of its neighbors [61]. Repeated applications of the graph convolutions will create a phenomenon called *over-smoothing*, meaning that the vertices from different clusters become indistinguishable and hurts classification accuracy [61]. This mixing happens even faster when graph convolutions are used on small datasets [61]. These findings may explain why the model is underperforming, especially on the smallest dataset BU-3DFE.

6.4 The problem with the verification rate and false acceptance rate

Generally, the verification rate is lower than expected when compared to the other deep-learning-based methods. The way that the verification rate is calculated severely punishes models that are not performing close to perfect. This problem occurs due to the dataset size and the experiment style. Cai *et al.* [10] calculated the verification and identification rate on the same set. Similarly, this Thesis uses the same approach to have comparable results.

Take FRGCv2, where the dataset contains 466 identities. For the *first vs. all* experiment, the gallery-set contained 466 unique identities and one scan per identity. The verification rate is calculated on a balanced set of negative and positive pairs as described in Section 4.2.3. This means that the maximum amount allowed false positives becomes very low, as seen in Equation 6.1.

For a 0.1% FAR, the maximum allowed number of false positives is 0.93, less than a single fault. This is in the best-case scenario where every single positive pair must be correctly labeled. This means that the reported 0.1% FAR is equivalent to 0.0% FAR. The result of this can be seen in Figure 5.8d, where the graph plateaus around the $3 \cdot 10^{-3}$ FAR threshold. This means that the metric is rather strict and punishes models that cannot perform perfectly.

$$\begin{gathered} FAR = \frac{FP}{TP + TN} \\ if\, FAR \leq 0.1\% \;\wedge\; T = TP = 466 \;\wedge\; T + N = 466 \cdot 2 \\ \implies \quad FP \leq 0.93 \end{gathered} \tag{6.1}$$

For BU-3DFE with 100 identities, the maximum allowed false positives at best becomes 0.199 for a FAR of 0.1% and 1.98 for a FAR of 1%. This number becomes even less if the dataset is divided up or there are any false negatives. The triplet network BU-3DFE results, as presented in Table 5.5, goes from 87.60% to 95.63% when increasing the FAR from 0.1% to 1%.

The problem can be reduced by increasing the probe-set size, not balancing the pairs, or making the experiment less restrictive. By adding the *all vs. all* experiment, every scan is matched against every other scan. This increases the total amount of pairs available and creates a higher fidelity graph, as shown in Figure 5.6, Figure 5.7, and Figure 5.8.

Interestingly, the result from the *all vs. all* experiment is not much different from the *neutral vs. non-neutral* result on Bosphorus and the *neutral vs. high-intensity* result on BU-3DFE. This is logical as the *all vs. all* split contains the most difficult pairs, including pairs like *high-intensity vs. high-intensity*, which no other experiment contains, while also containing the easiest pairs of *neutral vs. neutral*.

The limitation outlined in this section is a problem every 3D face recognition algorithm has. To better characterize 3D face recognition algorithms, a larger dataset is required, as results on small datasets are more influenced by random noise.

6.5 The gap between Rank-1 identification rate and AUC

There are often large gaps between the rank-1 identification rate and AUC in the results. This can especially be seen in the augmentation table 5.1. Logically, these should be strongly connected. There are two reasons why they are slightly deviating.

The first reason is that the ROC is based on a balanced set. Because of the balancing, a lot of imposter pairs are thrown away. The CMC curve will contain these pairs, meaning that there will more likely be a reduction in the rank-N.

One reason why the rank-1 identification rate can be low while the AUC is high comes from the definition of the metrics. As described in Section 4.3.2.4, AUC represents the probability that a randomly selected positive pair has a higher score than a randomly selected negative pair. On the other side, as described in Section 4.3.2.1, rank-n measures that the correct identity is within the n highest-ranked gallery scans. For rank-1, it means that the highest-ranked gallery scan must be the correct identity. The reason why the scores are different is that they test different things. AUC is used for pairs of data and therefore face validation, while rank-1 uses a ranking score useful for face identification. Even though face identification can be built using face validation, the metric still does not translate over as AUC determines the probability of a binary output while rank-n relies on a discrete set of scores. DeCann and Ross [55] concludes that there exist multiple CMC curves that can be associated with a single ROC curve, meaning it is hard to create a direct translation between ROC and CMC.

6.6 Siamese vs. triplet loss

There are large differences between the neural siamese network and the triplet network. This means that the used siamese network is not able to help distinguish between the identities.

As the current siamese network is rather elementary, a changed version is more likely to perform better. Instead of only combining the vectors, functions like $f_1 - f_2$ and $f1 \cdot f2$ are more likely to help the network distinguish between identities. This is similar to DeepFace [62], where the siamese network does the operation $f_{out} = \sum_i \alpha_i |f_1[i] - f_2[i]|$, where α_i is the learnable parameter.

By changing the siamese network to use $f_1 - f_2$ and $(f_1 - f_2)^2$, the performance increases. This result is shown in Table 5.3. Here, the result becomes around equal in performance to the triplet net.

The original underperforming siamese network does not invalidate the relative difference when testing different augmentations (Table 5.1). As each test used the same network, the relative differences still show that the network learns better with more augmentations. This means that the results might not be directly translatable over to a high-performing mesh-based network but still indicate that more augmentation is important.

Due to time limitations, the augmentation results were not rerun with the improved siamese network. The final siamese results were also not rerun as the triplet net results show similar results to the improved siamese network.

6.7 Time Analysis

Table 5.8 displays the time analysis comparison against other deep-learning-based methods. Because our method requires little pre-processing and is lightweight, it performs a lot faster compared to other approaches. A direct comparison against Bhople *et al.* shows that our proposed method is around 2.5 times faster.

The processing time experiments were conducted on the Intel Core i7-8700K processor with a 3.70GHz core clock and an NVIDIA TITAN X (pascal). Compared to the other publications, Kim *et al.* [43] used a 2.6 GHz dual-processors and an NVIDIA K40 GPU, Cai *et al.* [10] used a 3.2GHZ Intel processor and an NVIDIA GTX750 GPU, and Bhople *et al.* [45] did not specify any processor but used an NVIDIA Tesla V100 GPU. Our CPU may is most likely the best used, and the GPU performs better than Kim *et al.* and Cai *et al.*, and similar, except with less memory than the GPU used by Bhople *et al.*

This means that the time performance of our proposed method is more advantageous when compared against Kim *et al.* and Cai *et al.* and similar when compared against Bhople *et al.*

6.8 Need of a consistent mesh structure

The cross-dataset testing results, presented in Table 5.2, show that the proposed model performs worse when trained and tested on different datasets than when it is trained and tested on the same dataset. This evaluation style, where multiple datasets are used for training and one unseen dataset is used for testing, is done in the publications that use 3D to 2D projection [10, 43, 44]. Their projection-based architecture has the advantage that different mesh topology structures look similar when projected.

Table 5.2 show the model is able to learn some features that translate between Bosphorus and BU-3DFE. When trained on BU-3DFE, the model manages to get a 72.7% rank-1 and a 77.2% VR at 0.1% FAR on Bosphorus. The results are not as good when trained on Bosphorus and tested on BU-3DFE. This means that the features the model found on BU-3DFE translates better over to Bosphorus than the other way. Interestingly, when trained or tested on FRGCv2, the results are close to random, seen as the verification rate is close to 50%.

These results are most likely from the different mesh structures and croppings used by the different datasets. As seen in Figure 4.6, the different datasets have different mesh topologies and structures. For example, the BU-3DFE 3D scan (4.6d) is smooth, the scans from Bosphorus (4.6e) have vertical stripes as an artifact from the scanner, and the scans from FRGCv2 (4.6f) are more rough and uneven compared to the other models and also contain parts of the neck, side of the head and shoulder.

The different mesh cropping is most likely the reason why the model struggles on the cross-dataset performance on the FRGCv2 dataset. The scans in FRGCv2 contain part of the neck and shoulders, something it has never seen when trained on BU-3DFE and Bosphorus. This means that noise in the mesh, like a new structure, will confuse the current feature extraction. More augmentation, like adding objects to the scanning environment, either real or synthetic, might reduce this problem.

But as cross-dataset performance between Bosphorus and BU-3DFE are not that good indicates that mesh structure is also important. FRGCv2 and Bosphorus use the same triangulation algorithm, while BU-3DFE used something different.

The cross-dataset performance is unexpectedly worsened when resampling is used. Logically, re-sampling resampling of the points would create a similar mesh structure for all datasets. This indicates that resampling is not usable when different datasets are used. As this experiment used the sub-optimal siamese network, this statement needs further research.

6.9 Mesh-based vs. Point Cloud-based

Due to the limited amount of available models, it's unsure if a mesh-based or point-cloud-based approach is optimal for 3D face recognition. A mesh is a processed form of the raw point-cloud data and may add unwanted information or restrictions to the features. Intuitively, a network that is free to extract features directly from raw data has greater freedom and fewer restrictions than a model that relies on pre-processing. A point cloud-based network may, in theory, perform better as it can extract the optimal features directly from the point cloud where projected data may lose important 3D information, and a mesh may ignore some larger abstract feature of the data. As stated by Zulqarnain Gilani and Mian [44], 3D data has its own peculiarities defined by the underlying shape and geometry. These peculiarities may be lost during triangulation.

On the other side, a mesh may help when the training data is limited. When there is less data available, the network will have a harder time learning the optimal features for generalization. By forcing some structure to the data via either projection to the 2D plane or via connections between the vertices, the statistical burden may be reduced in exchange for lower theoretical maximum accuracy.

6.10 Plausibility of mesh-based Face Recognition

As seen in the comparisons, Table 5.6 and 5.7, our model performs slightly under the other deep-learning-based methods. In the example of FRGCv2 [51], the model almost achieves a perfect verification and identification rate result, where the state-of-the-art method achieved 100%.

Based on the time analysis, a mesh-based or geometric-deep-learning-based architecture will be faster than the projection-based face recognition approaches. Working directly with the 3D data has the advantage that the model gains efficiency in terms of speed.

To the best of our knowledge, this Thesis was the first mesh-based geometric deep learning face recognition. With further research, better models, and larger datasets, mesh-based face recognition may become a reliable and efficient method performing equal to the state-of-the-art methods.

6.11 Potential Sources of Error

6.11.1 Non-deterministic behavior

Unfortunately, our implementation of the proposed model has a limitation where non-deterministic behavior cannot be eliminated which hinders exact reproducibility across multiple executions. This is the cause for both concurrent and single-threaded executions. Non-deterministic behavior creates uncertainty and adds randomness to experiments which may have impacted the final results.

6.11.2 Race, gender, and age bias

The datasets used has various distributions of ethnicities, gender, and ages. The most diverse is dataset is BU-3DFE [54], containing scans of 44 men and 56 women ranging from age 18 to 70, where 51 of them are white, 24 east Asian, 9 black, 8 Latino Hispanic, 6 Indian, and 2 middle-east Asian. The Bosphorus dataset [50] contains 60 men and 45 women aged between 25 and 35, where the majority of the subjects are Caucasian. FRGCv2 [51] contains 466 individuals where 57% of them are male and 43% female, 68% are white, 22% Asian, and 10% other. In addition, 65% of the participants are between the age of 18 and 22, 18% are age 23 to 27, and 17% are age 28 and over.

This means that the datasets are biased towards young white people, which may impact real-life performance and scenarios that use separate testing and training datasets. Every deep-learning-based 3D face recognition uses at least one of the datasets and will have some bias.

Chapter 7

Conclusion and future work

7.1 Conclusion

In this Thesis, we have proposed and evaluated a new geometric-deep-learning-based method for 3D face recognition. The method uses graph convolution to extract features directly from meshes, improving the speed compared to other models. The spatial and geometric information in the 3D data should allow geometric deep learning approaches to become more robust and outperform methods that rely on mappings that induce information loss. This was not seen in practice, and currently, geometric deep learning models perform slightly below or equal to the mapping method. To the best of our knowledge, our proposed method is the second face recognition algorithm that directly uses 3D data with deep learning and the first graph-convolution-based network used in face recognition, meaning that the field is open for more research and improvements.

One possible explanation for why the model performed worse than the state-of-the-art approaches may be that mesh-based models require more training data than other approaches for interclass separation problems. Experiments show that small augmentations of the data provide large improvements in the performance. The problem is that there exist very few datasets suitable for 3D face recognition.

Our proposed method achieves a 90% rank-1 identification rate on the BU-3DFE, Bosphorus, and FRGCv2 datasets, and a verification rate at 0.1% false acceptance rate of 87.6%, 86%, and 99.55%, respectively.

As seen in the results, even if the model performs slightly under the state-of-the-art deep-learning-based models, it still shows the potential of mesh-based face recognition. It is unsure if the underperforming is a product of the model structure, its capacity, or because of the graph convolution. With more experimentation, better models, larger datasets, and more advanced data augmentation, mesh-based face recognition may be an alternative to other geometric deep learning-based methods.

7.2 Future work

To improve the performance of our proposed method, we recommend several possible improvements, described below.

7.2.1 Dataset-specific

One general issue with 3D face recognition is the amount and size of datasets. Larger and more datasets would be beneficial for all future research. More augmentation and morphable models should be added to our proposed approach as the limited amount of data is a problem.

As discussed in Section 6.8, the current model struggles with different mesh typologies and mesh noise. By adding in synthetic objects, the network may be able to ignore more irrelevant parts of the mesh, allowing different face sections to be recognized.

7.2.2 Network-specific

As stated in the discussion, our feature extraction network is performing suboptimally. There are multiple possible routes to fix this issue. One way is to optimize the current network. Changes in the number of filters and depth may drastically improve performance. Fine-tuning was not utilized in the current approach and may also be beneficial for the model to improve the performance when the learning plateaus.

The implementation of more pooling algorithms may also improve the architecture. Vertex pooling help reduce the statistical burden as it will summarize spatial and geometric information of multiple vertices. This will help the network combine more spatial and geometrical features, which may be a limitation of the current model.

Multiple new graph convolutions and operators have also lately been proposed. These may help if the problem stems from the use of GCN [27]. A graph normalization operator [63] has also been proposed, which should help the graph-based networks converge quicker.

The use of different loss functions may also help. By using a softmax-like loss like arcface [64], or more experimentation with triplet loss [31] could improve the performance.

Adding more regularization techniques like dropout [17, p. 258] and L2 norm [17, p. 39] may also improve the network's performance.

Bibliography

[1] H. Wardeberg, '3d face recognition based on geometric deep learning,' Department of Computer, Information Science, NTNU – Norwegian University of Science and Technology, Project report in TDT4501, Dec. 2019 (pages v, 26).

[2] G. Guo and N. Zhang, 'A survey on deep learning based face recognition,' *Computer Vision and Image Understanding*, vol. 189, p. 102 805, 2019 (pages 1, 12).

[3] I. Adjabi, A. Ouahabi, A. Benzaoui and A. Taleb-Ahmed, 'Past, present, and future of face recognition: A review,' *Electronics*, vol. 9, no. 8, p. 1188, 2020 (pages 1, 2, 28).

[4] A. S. Mian, M. Bennamoun and R. Owens, 'Keypoint detection and local feature matching for textured 3d face recognition,' *International Journal of Computer Vision*, vol. 79, no. 1, pp. 1–12, 2008 (pages 1, 19).

[5] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam and P. Vandergheynst, 'Geometric deep learning: Going beyond euclidean data,' *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017 (pages 1, 2, 8, 10, 11).

[6] A. Savran, B. Sankur and M. T. Bilge, 'Comparative evaluation of 3d vs. 2d modality for automatic detection of facial action units,' *Pattern recognition*, vol. 45, no. 2, pp. 767–782, 2012 (pages 1, 2).

[7] H. Patil, A. Kothari and K. Bhurchandi, '3-d face recognition: Features, databases, algorithms and challenges,' *Artificial Intelligence Review*, vol. 44, no. 3, pp. 393–441, 2015 (pages 1, 31).

[8] F. Schroff, D. Kalenichenko and J. Philbin, 'Facenet: A unified embedding for face recognition and clustering,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823 (pages 2, 13, 23, 25).

[9] T. C. Faltemier, K. W. Bowyer and P. J. Flynn, 'Using a multi-instance enrollment representation to improve 3d face recognition,' in *2007 First IEEE International Conference on Biometrics: Theory, Applications, and Systems*, IEEE, 2007, pp. 1–6 (pages 2, 32).

[10] Y. Cai, Y. Lei, M. Yang, Z. You and S. Shan, 'A fast and robust 3d face recognition approach based on deeply learned face representation,' *Neurocomputing*, vol. 363, pp. 375–397, 2019 (pages 2, 18–21, 23, 27, 31, 33, 34, 47, 49, 50, 53).

[11] N. Verma, E. Boyer and J. Verbeek, 'Feastnet: Feature-steered graph convolutions for 3d shape analysis,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2598–2606 (page 4).

[12] M. S. Dizaji and D. K. Harris, '3d inspectionnet: A deep 3d convolutional neural networks based approach for 3d defect detection on concrete columns,' in *Nondestructive Characterization and Monitoring of Advanced Materials, Aerospace, Civil Infrastructure, and Transportation XIII*, International Society for Optics and Photonics, vol. 10971, 2019, 109710E (page 4).

[13] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao, '3d shapenets: A deep representation for volumetric shapes,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920 (page 5).

[14] A. L. Apolinário Jr, C. Esperança and L. Velho, 'A representation of implicit objects based on multiscale euclidean distance fields,' *Proceedings of SIACG*, pp. 119–129, 2002 (page 5).

[15] C. R. Qi, L. Yi, H. Su and L. J. Guibas, 'Pointnet++: Deep hierarchical feature learning on point sets in a metric space,' *arXiv preprint arXiv:1706.02413*, 2017 (pages 5, 9, 23).

[16] G. Bouritsas, S. Bokhnyak, S. Ploumpis, M. Bronstein and S. Zafeiriou, 'Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation,' in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 7213–7222 (page 6).

[17] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`, Accessed: 10.05.2020 (pages 6–9, 14–17, 25, 29, 57).

[18] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and S. Y. Philip, 'A comprehensive survey on graph neural networks,' *IEEE Transactions on Neural Networks and Learning Systems*, 2020 (pages 8, 10).

[19] C. Cangea, P. Veličković, N. Jovanović, T. Kipf and P. Liò, 'Towards sparse hierarchical graph classifiers,' *arXiv preprint arXiv:1811.01287*, 2018 (page 8).

[20] M. Fey and J. E. Lenssen, 'Fast graph representation learning with PyTorch Geometric,' in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019 (pages 8, 11, 21).

[21] C. R. Qi, H. Su, K. Mo and L. J. Guibas, 'Pointnet: Deep learning on point sets for 3d classification and segmentation,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660 (pages 9, 20, 23).

[22] B. Xu, N. Wang, T. Chen and M. Li, 'Empirical evaluation of rectified activations in convolutional network,' *arXiv preprint arXiv:1505.00853*, 2015 (page 10).

[23] V. Nair and G. E. Hinton, 'Rectified linear units improve restricted boltzmann machines,' in *Icml*, 2010 (pages 10, 24).

[24] E. Kreyszig, H. Kreyszig and E. J. Norminton, *Advanced engineering mathematics*, 10th ed. Hoboken, NJ: John Wiley, 2011, ISBN: 9780470458365 (page 10).

[25] E. W. Weisstein, *Laplacian matrix. From MathWorld—A Wolfram Web Resource*, Last visited on 10/12/2020. [Online]. Available: https://mathworld.wolfram.com/LaplacianMatrix.html (page 11).

[26] M. Defferrard, X. Bresson and P. Vandergheynst, 'Convolutional neural networks on graphs with fast localized spectral filtering,' *arXiv preprint arXiv:1606.09375*, 2016 (page 11).

[27] T. N. Kipf and M. Welling, 'Semi-supervised classification with graph convolutional networks,' *arXiv preprint arXiv:1609.02907*, 2016 (pages 11, 24, 50, 57).

[28] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson and G. Gkioxari, 'Accelerating 3d deep learning with pytorch3d,' *arXiv:2007.08501*, 2020 (page 11).

[29] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu and Y.-G. Jiang, 'Pixel2mesh: Generating 3d mesh models from single rgb images,' in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 52–67 (pages 11, 50).

[30] G. Gkioxari, J. Malik and J. Johnson, 'Mesh r-cnn,' in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9785–9795 (pages 11, 50).

[31] E. Hoffer and N. Ailon, 'Deep metric learning using triplet network,' in *International Workshop on Similarity-Based Pattern Recognition*, Springer, 2015, pp. 84–92 (pages 12, 57).

[32] R. Hadsell, S. Chopra and Y. LeCun, 'Dimensionality reduction by learning an invariant mapping,' in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, IEEE, vol. 2, 2006, pp. 1735–1742 (page 12).

[33] J. Rishaug, *Onlineminingtripletloss*, https://github.com/NegatioN/OnlineMiningTripletLoss, 2019 (pages 12, 21).

[34] R. Gómez, 'Understanding ranking loss, contrastive loss, margin loss, triplet loss, hinge loss and all those confusing names,' *Raúl Gómez blog*, 2019. [Online]. Available: https://gombru.github.io/2019/04/03/ranking_loss (page 12).

[35] K. Q. Weinberger and L. K. Saul, 'Distance metric learning for large margin nearest neighbor classification.,' *Journal of machine learning research*, vol. 10, no. 2, 2009 (page 12).

[36] A. Buja, W. Stuetzle and Y. Shen, 'Loss functions for binary class probability estimation and classification: Structure and applications,' *Working draft, November*, vol. 3, 2005 (page 13).

[37] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger and R. Shah, 'Signature verification using a" siamese" time delay neural network,' *Advances in neural information processing systems*, vol. 6, pp. 737–744, 1993 (page 14).

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, 'Pytorch: An imperative style, high-performance deep learning library,' in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (pages 14, 21).

[39] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/ (page 14).

[40] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization,' *arXiv preprint arXiv:1412.6980*, 2014 (pages 15, 26).

[41] L. N. Smith, 'A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay,' *arXiv preprint arXiv:1803.09820*, 2018 (page 15).

[42] S. Ioffe and C. Szegedy, 'Batch normalization: Accelerating deep network training by reducing internal covariate shift,' in *International conference on machine learning*, PMLR, 2015, pp. 448–456 (page 17).

[43] D. Kim, M. Hernandez, J. Choi and G. Medioni, 'Deep 3d face identification,' in *2017 IEEE international joint conference on biometrics (IJCB)*, IEEE, 2017, pp. 133–142 (pages 18–20, 23, 27, 31, 33, 34, 47, 49, 53).

[44] S. Zulqarnain Gilani and A. Mian, 'Learning from millions of 3d scans for large-scale 3d face recognition,' in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1896–1905 (pages 18–21, 23, 27, 31, 34, 47–49, 53, 54).

[45] A. R. Bhople, A. M. Shrivastava and S. Prakash, 'Point cloud based deep convolutional neural network for 3d face recognition,' *Multimedia Tools and Applications*, pp. 1–23, 2020 (pages 18, 20, 21, 23, 26, 27, 31, 47, 49, 50, 53).

[46] W. Zhao, R. Chellappa, P. J. Phillips and A. Rosenfeld, 'Face recognition: A literature survey,' *ACM computing surveys (CSUR)*, vol. 35, no. 4, pp. 399–458, 2003 (page 19).

[47] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min and W. Worek, 'Overview of the face recognition grand challenge,' in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, IEEE, vol. 1, 2005, pp. 947–954 (page 19).

[48] S. Soltanpour, B. Boufama and Q. J. Wu, 'A survey of local feature methods for 3d face recognition,' *Pattern Recognition*, vol. 72, pp. 391–406, 2017 (page 19).

[49] O. M. Parkhi, A. Vedaldi and A. Zisserman, 'Deep face recognition,' 2015 (page 19).

[50] A. Savran, N. Alyüz, H. Dibeklioğlu, O. Çeliktutan, B. Gökberk, B. Sankur and L. Akarun, 'Bosphorus database for 3d face analysis,' in *European workshop on biometrics and identity management*, Springer, 2008, pp. 47–56 (pages 20, 25, 27, 31–33, 39, 40, 46, 55).

[51] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min and W. Worek, 'Overview of the face recognition grand challenge,' in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, IEEE, vol. 1, 2005, pp. 947–954 (pages 25, 27, 31, 32, 34, 39, 40, 46, 54, 55).

[52] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt and SciPy 1.0 Contributors, 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,' *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2 (page 25).

[53] T. Theoharis, G. Papaioannou, N. Platis and N. M. Patrikalakis, *Graphics and visualization: principles & algorithms*. CrC Press, 2008 (page 25).

[54] L. Yin, X. Wei, Y. Sun, J. Wang and M. J. Rosato, 'A 3d facial expression database for facial behavior research,' in *7th international conference on automatic face and gesture recognition (FGR06)*, IEEE, 2006, pp. 211–216 (pages 27, 31–33, 39, 40, 46, 55).

[55] B. DeCann and A. Ross, 'Relating roc and cmc curves via the biometric menagerie,' in *2013 IEEE Sixth International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, IEEE, 2013, pp. 1–8 (pages 30, 31, 52).

[56] A. G. (https://stats.stackexchange.com/users/49130/alexey-grigorev), *What does auc stand for and what is it?* Cross Validated, version: 2015-12-30. eprint: https://stats.stackexchange.com/q/133435. [Online]. Available: https://stats.stackexchange.com/q/133435 (page 30).

[57] V. Vijayan, K. W. Bowyer, P. J. Flynn, D. Huang, L. Chen, M. Hansen, O. Ocegueda, S. K. Shah and I. A. Kakadiaris, 'Twins 3d face recognition challenge,' in *2011 International Joint Conference on Biometrics (IJCB)*, IEEE, 2011, pp. 1–7 (page 32).

[58] C. Wen, Y. Zhang, Z. Li and Y. Fu, 'Pixel2mesh++: Multi-view 3d mesh generation via deformation,' in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1042–1051 (page 50).

[59] M. Chen, Z. Wei, Z. Huang, B. Ding and Y. Li, 'Simple and deep graph convolutional networks,' in *International Conference on Machine Learning*, PMLR, 2020, pp. 1725–1735 (page 50).

[60] M. Chen, Z. Wei, Z. Huang, B. Ding and Y. Li, 'Simple and deep graph convolutional networks,' in *International Conference on Machine Learning*, PMLR, 2020, pp. 1725–1735 (page 50).

[61] Q. Li, Z. Han and X.-M. Wu, 'Deeper insights into graph convolutional networks for semi-supervised learning,' in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018 (page 50).

[62] Y. Taigman, M. Yang, M. Ranzato and L. Wolf, 'Deepface: Closing the gap to human-level performance in face verification,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708 (page 52).

[63] T. Cai, S. Luo, K. Xu, D. He, T.-y. Liu and L. Wang, 'Graphnorm: A principled approach to accelerating graph neural network training,' *arXiv preprint arXiv:2009.03294*, 2020 (page 57).

[64] J. Deng, J. Guo, N. Xue and S. Zafeiriou, 'Arcface: Additive angular margin loss for deep face recognition,' in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4690–4699 (page 57).

Appendix A

Explanation of VR-FPR graph bug

As seen in Figure 5.4d, the *neutral vs. neutral* experiment is laying at a VR of 50%. This would normally be indicating that the model has no predictive power. In reality, when the AUC is 100% the VR-FPR graph should be at 100% for every FPR less than 100%. When the FPR is 100%, there must exist no true negatives, and a VR of 50% is achieved. This bug is due to a rounding error in the code that calculates the graphs.

Take the example of a siamese network with Sigmoid. Due to the Sigmoid, the output of the network will be between, not including, 0 and 1. When the AUC is 100%, the code samples three points. These are at the threshold at the maximum seen value for a negative sample, the minimum seen value for a positive sample, and 2x the max value. For the siamese, this will be around 0.00001, 0.999..., and 2. Due to a rounding error, the values are rounded to 0, 1, and 2. This results in that every threshold value creates a VR of 50%.

At threshold 0, every pair is labeled as a positive, creating a VR of 50%. At threshold 1, due to the rounding, every pair will be labeled as negative creating a VR of 50%. At threshold 2, every pair will be labeled as negative, as it is impossible to get this score, and a VR of 50% is achieved.

When the AUC is less than 100%, more samples are generated, making this a non-issue.