Imperial College London

Department of Computing

Geometric Deep Learning with 3D Facial Motion

Author:
Jiali Zheng

Supervisor:
Dr Stefanos Zafeiriou

Submitted in partial fulfillment of the requirements for the MSc degree in of Imperial College London

September 4, 2019

Abstract

3D object representation learning has broad application prospects including 3D shape analysis and network analysis. The traditional machine learning approach attempt to manually define structural feature extraction about information of the 3D object, while a recently arising field named *geometric deep earning* can achieve function defined on 3D object learning automatically. In this thesis, we review the latest developed geometric deep learning technique and proposed a multi-output approach utilizing such technique to address the challenge of dynamic 3D facial expression recognition. The proposed model utilize graph convolutional operation with Chebyshev filtering can perform multi-label classification for both expressions and emotional states of a given dynamic 3D facial expression.

Acknowledgments

I would like to thank my supervisor Dr Stefanos Zafeiriou, for all his support and guidance throughout the thesis.

Additionally I'd like to thank Evangelos Ververas (PhD student at iBug group Imperial College London), for providing me valuable 4DFAB database[59] (2017) to help my experiment on 3D facial expressions. Also I want to thank my friends Yijia Lu, Bingyuan Jiang, Yixin Zhang for assisting the 4DFAB database annotation tasks.

Finally, I'd like to thank the Computing Support Group at Imperial College London for providing me with a NVIDIA GTX 1080 and GPU clusters used in this project.

Contents

1 Introduction

1.1 Overview

1.1.1 3-D Object Representations

Nowadays, *deep learning* is not an unfamiliar concept. The promise of deep learning is to discover rich, hierarchical models that represent probability distributions over the kinds of data encountered in artificial intelligence applications, such as natural images[50], audio waveforms[69], and symbols in natural language[14]. The deep architectures have the capacity to learn more complex models, which allow for learning powerful object representations without the need to hand design features. The term *object representation* is defined as *"Some form of visual data that possess some internal representation of the task and of the data."* People can actually see or capture the characteristic of the object from the representation.

Figure 1: Examples of object representation

Among different types of representation, 2D images have been considered as vital objects for machine learning tasks for decades. By utilizing the convolutional neural network, image processing becomes faster and more accurate. The development of science and technology results in better information representations can be achieved, while such representations are called 3D geometric data. This type of data models the target object more precisely since they are the closest to the object's realistic existence form. To work with the 3D geometric data, various deep learning techniques have been attempted that have produced considerable results on 2D images to work with 3D geometric data. One approach is to represent the 3D object as an assemblage of range images captured from diverse views of the object, and features of the 3D object are extracted from range images by a convolutional neural network architecture for description and recognition tasks[56, 70]. Another approach is using the probability distribution of binary variable on 3D voxel grid to represent the geometric 3D shape[71]. Alternatively, the depth images of 2D views projected from 3D objects can assist the retrieval of 3D object by training with a stacked local convolutional autoencoder[36].

Though considerable results have been obtained by these approaches, treating the geometric data as Euclidean structure is the main drawback. Since fine details of the 3D object are tend to be lost by the Euclidean structure such as voxel and depth image, also the topological structure might be broken. A common requirement in many computer

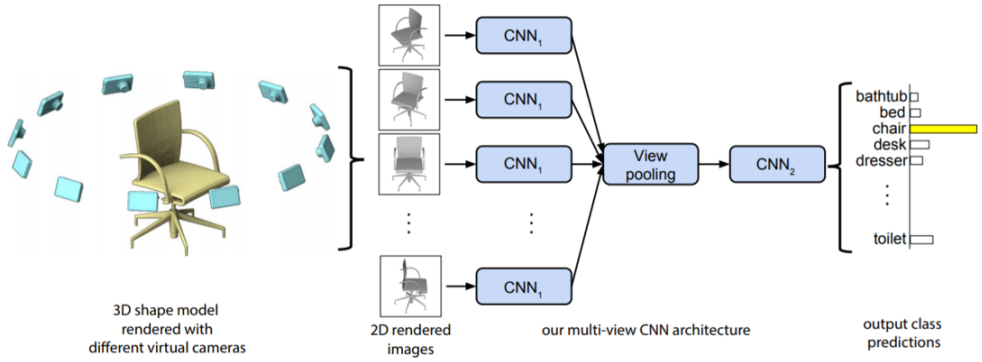

Figure 2: Multi-view CNN for 3D shape recognition. At test time a 3D shape is rendered from 12 different views and are passed thorough CNN_1 to extract view based features. These are then pooled across views and passed through CNN_2 to obtain a compact shape descriptor.[56]

vision tasks is to accomplish shape deformation invariance, while in Euclidean representation the variation of representation will occur on account of object pose changing or deformation[43]. Hence, the generalization of 3D object representation learning with deep models to non-Euclidean domain is crucial and will have great benefit. Such generalization of deep neural model on non-Euclidean domain is so-called *geometric deep learning*.

1.1.2 Geometric Deep Learning

Geometric Deep Learning is a field of emerging techniques that attempts to generalize deep learning architectures as well as the fundamental mathematical principles to non-Euclidean domains. Two commonly seen non-Euclidean structures are graph and manifold, which can represent data arising in numerous applications. In neuroscience, the anatomical and functional structure of the brain can be represented by graph models. In social networks, the signals on the vertices of the social graph can model the characteristics of the users[16]. In computer vision tasks, the 3D object can be represented by Riemannian manifolds or simplified as graphs, and the other visual properties can also be given by the color texture[70].

To work with the obtained non-Euclidean structures, analyzing functions defined on the non-Euclidean domain is important. In 2005, the first formulation of neural network on graphs was presented, which was called graph neural network(GNN).It extended the recursive neural network due to the ability to process a larger collection of graphs and being able to apply on node focus problems[44]. In 2009, a graph neural network model based on information diffusion and relaxation mechanism was presented, which the ability to work with large scale data was demonstrated[20]. Recently some reformulation of these approaches was carried out, which utilizing emerging techniques such as neural message passing[30] and gated recurrent units[75]. The convolution-like operation on non-Euclidean domain was formulated by Bruna et al.[31], which performed the convolution in spectral domain by constructing the eigenvectors of the graph Laplacian. The main shortage of this approach is the high computational complexity arising in explicitly compute the eigendecomposition of the graph Laplacian. By expressing the spectral filter function

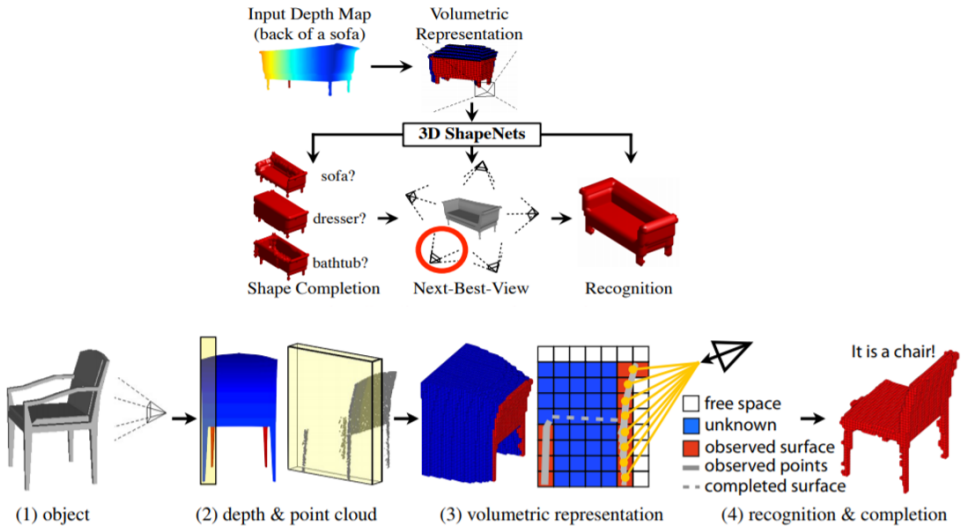

Figure 3: View-based 2.5D Object Recognition. (1) Illustrates that a depth map is taken from a physical object in the 3D world. (2) Shows the depth image captured from the back of the chair. A slice is used for visualization. (3) Shows the profile of the slice and different types of voxels. The surface voxels of the chair x_o are in red, and the occluded voxels x_u are in blue. (4) Shows the recognition and shape completion result, conditioned on the observed free space and surface. [71]

in terms of simple operations such as matrix multiplications and additions, the filter can be applied to the graph Laplacian directly without computing the eigendecomposition. Considerable approaches includes ChebNets[45] which utilizing polynomial functions to approximate filter and CayleyNets[54] using rational functions for filter parametrization. Another aspect of graph CNN are spatial methods, which apply on local neighborhoods on the non-Euclidean domain. Such approaches include making use of locally geodesically polar chart[32] or learnable Gaussian kernels[43], as well as using B-spline kernel to replace the Gaussians[68].

In this thesis, we proposed a problem lied in 3D computer vision with geometric deep learning.

1.2 Background

1.2.1 Facial Expression Classification

Human-computer interaction has been an important topic in artificial intelligence research. Scientists working hard on developing the machine for communicating tasks. In human communication, facial expression plays a significant role since people inferring the emotional status of others base on their facial expressions. According to previous surveys, one-third of human communication is conveyed by the verbal component while two-third is conveyed by nonverbal components. And in nonverbal components, facial expressions are one of the main information channels. *Facial Expression Recognition* is an approach to recognize expressions on human faces, which basically classifies the expression in certain categories.

The classification of expression often consists of two procedure: feature extraction

and decision. In the feature extraction procedure, pixel data were converted into higher level representation of the properties of the face such as shape, texture, and spatial configuration[58]. The dimensionality of the input space is reduced by the feature extraction procedure but the essential information was retained. In the decision procedure, various machine learning techniques were applied for pattern recognition, such as Neural Network, Support Vector Machine and Adaboost. There are six basic facial expressions: Happy, Surprise, Disgust, Sad, Fear and Angry[51]. Previous work of facial expression recognition makes use of facial data in 2D images[73], while the development of 3D sensor allows us collecting 3D data of human faces, which describes the facial shape and deformation more accurately and avoid the affection of illumination variation[59]. However, the lack of high resolution 3D facial expression database constrains the improvement in this research area.

In 2017, a high resolution 3D facial database *4DFAB* was published by the iBug group in Imperial College London, which contains 4D video of posed six basic expressions as well as spontaneous expressions from 180 participants on 4 sessions spanning over 5 year period[59]. The classification task performed on 4DFAB database achieves recognition rate 70.27% for session-1, which make use of PCA and LDA to extract features and multiclass SVM to classify expression from individual static 3D face mesh.

1.3 Thesis Statement

This thesis seeks to tackle the challenges arising in dynamic 3D facial expression recognition. With proper algorithm design and architecture construction, geometric deep learning techniques are adequate for tackling non-Euclidean-structured data for various application purpose. We expect that the approach proposed in this thesis will benefit more deep learning user when solving problems with underlying geometric data, that the geometric deep learning can work as a nice alternative and might provide better results.

1.4 Contributions

We summarise the major contributions we have made below:

- We introduce some essential knowledge in Euclidean deep learning as preliminaries for better understanding of the review of geometric deep learning techniques and the thesis approach later presented.
- We provide a review of the latest developed geometric deep learning approaches from spectral aspect and spatial aspect as well as introduce their design intuition and discuss their pros and cons.
- We proposed a multi-output model based on CoMA[55] and LSTM[25] on the problem of dynamic 3D facial expression recognition, which provides both expression classification result and emotional state classification result. To our best knowledge, we are the first multi-output dynamic 3D facial expression recognition architecture that utilizing geometric deep learning technique.

2 Deep Learning on Euclidean Domain

In this section, reviews of some essential background knowledge of deep learning on Euclidean domain is constructed for better understanding of the model developed in this thesis on non-Eulidean domain. A brief review of Convolutional Neural Network including convolution operations, pooling operations, activation function, and fully connected operation will be delivered first. Then the basic review of Recurrent Neural Network will be conducted which introduce different approaches to process temporal information in data. The autoencoder will be reviewed as it is a nice tool for low-dimensional feature extraction. The generative adversarial network will be discussed also.

2.1 Convolutional Neural Network

A convolutional neural network is a list of connected layers that transform the input object into an output volume for specific tasks(e.g. segmentation label), which employs a mathematical operation called convolution in place of general matrix multiplication. The common hidden layers and functions including convolution layer, pooling layer, fully connected layer and activation functions utilized in CNN architecture are discussed further in the following subsection.

2.1.1 Convolution layer

Intuition The convolution layer is the major building blocks used in CNN, whose parameters are formed by learnable filters. Each filter is a small spatial kernel that extend partially the width and height of the input volume as well as the full depth. The filter slides across every position of the input over the width and height and produce activation map that contains the responses of the filter at each position. One of the main target in CNN training is to train filters in each convolution layer to extract features, which produce set of activation maps to be stacked along the depth and forms the input of the next layer.

In a convolutional layer, applying a bank of filters $\Omega = (\omega_{l,l'}), l = 1, \ldots, q, l' = 1, \ldots, p$ on a p-dimensional input $h(x) = (h_1(x), \ldots, h_p(x))$ with a point-wise non-linearity σ can be denote as

$$z_l(x) = \sigma\Big(\sum_{l'=1}^{p} (h_{l'} \star \omega_{l,l'})(x) \Big) \tag{1}$$

where the

$$(h \star \omega)(x) = \int_{\Gamma} h(x - x')\omega(x')dx' \tag{2}$$

is the convolution operation.

Weight sharing In a convolutional layer, each parameter(weight) of the filter is used at every position of the input by sliding over the input(the boundary pixels might be skipped according to the design decision). Thus instead of learning distinct set of parameters for every position, only one set of parameters is learnt, which maintain the runtime

of forward propagation but dramatically reduce the number of parameters. Due to this parameter sharing, the convolutional layer is equivariance to translation and the number of parameter is independent to the input size.

Local connectivity and spatial arrangement For each neuron, instead of getting input from every part of the input volume, it only receives input from a small local group of the pixels in the input volume. Thus the inputs that go into a given neuron are actually close to each other(see figure 4). The output volume of the layer depends on the number of filter being used, the stride and the padding design. For a size W input, applying a size F filter with stride S and padding P would results in a size $(W - F + 2P)/S + 1$ output.

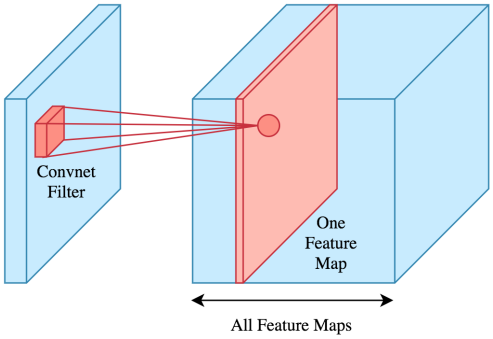

Figure 4: Graphical illustration of convolution operation

2.1.2 Pooling layer

The pooling operation use a summary statistic of the nearby input to replace the input at a certain position, which helps achieving representation invariance to translation by aggregating multiple low-level features in the neighborhood. Also, the dimension of the input can be reduced by pooling operation thus allowing multi-scale analysis of the input and the computation is less intensive. Common pooling method includes **max-pooling** and **average-pooling**, which the former takes the maximum value in the sliding window and the latter avarages the value in the window. Examples can be seen in figure 5.

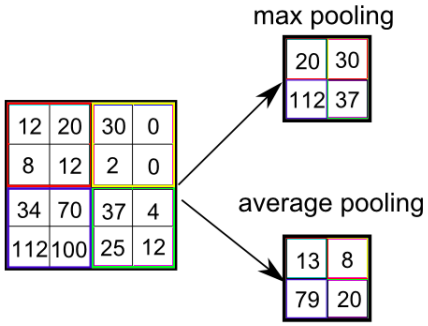

Figure 5: Examples of pooling operations.

In 2010, Boureau *et al* proposed a theoretical work that provides guidance for choosing pooling operation in different situations[4], while in 2011 he presented the idea that pooling features together dynamically is also a feasible method[7]. The Spatial pyramid pooling(SPP-net) porposed by He *et al* is a network-structure pooling strategy which is robust to object deformations.

2.1.3 Activation Function

In order to increase the power of the neural network to learn complex functional mapping from data, activation functions are utilized. The activation functions are non-linear functions that have degree more than one and are differentiable which enabling back-propogation optimization strategy during training. Several activation function that being utilized in this project would be introduced here, which includes Sigmoid, Softmax, ReLU, Leaky ReLU, Tanh activation function.

The sigmoid function is a function of S-shape curve whose range is between 0 and 1. It usually be applied in a binary classification problem or the task required an output between 0 and 1. The form of sigmoid function is

$$\sigma(x) = \frac{1}{1 + exp(-x)}. \tag{3}$$

The softmax function can convert an arbitrary N-dimensional real value vector into an N-dimensional vector with real values between 0 and 1, and the sum of these real values results in 1. It preserves the property of the input vector but present in a probabilistic sense, which makes it usually be applied in multi-class classification tasks as each element in the vector represents the probability of the input data belongs to certain class. The form of the softmax function is

$$\sigma(\mathbf{x})_j = \frac{e^{z_j}}{\sum_{k-1}^{K} e^{z_k}}, \quad for \quad j = 1, \ldots, K. \tag{4}$$

The ReLU refers to the Rectified Linear Units, which has become very popular in recent years computer vision applications. It returns the value of the input if it is positive, otherwise returns 0. The non-linearity of the overall network could be increase by adding ReLU between hidden layers.

$$ReLU(x) = max(0, x) \tag{5}$$

The Leaky ReLU is a form of ReLU with the same value for positive input but different for negative inputs. The ReLU function output 0 for negative input while Leaky ReLU returns the value itself times 0.01

$$LeakyReLU(x) = \begin{cases} x & x > 0 \\ 0.01x & otherwise \end{cases} \tag{6}$$

which correct the dying ReLU problem by avoiding no gradient flows.

The Tanh function is also an S-shape curve function which returns values between -1 to 1 for arbitrary input values. It is a scaled version sigmoid function with stronger gradient.

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{7}$$

2.1.4 Fully connected layer

The fully connected layer is usually being utilized when useful low-dimensional features have been extracted by the convolution operation from the original input volume, where each neuron in the layer is affected by each of the input element. Multiple fully connected layer could be stacked together with activation function in between, and the final output dimension depends on the task requirement(e.g. number of class label).

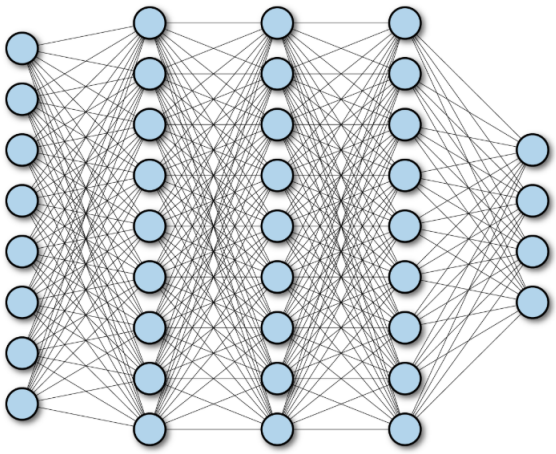

Figure 6: A multi-layer deep fully connected network.

2.2 Recurrent Neural Network

The recurrent neural network(RNN) is a type of neural network architecture that builds connections along the input temporal sequence. It has the ability to remember information learnt from prior inputs and the decision made by the network is affected by those information learnt, which allowing temporal dynamic behavior being exhibited. Successful applications of recurrent neural network includes text classification[37, 39], speech recognition[21] and image generation[22]. Three types of recurrent neural network will be discussed as well as their pros and cons, which are vanilla RNN, Long Short Term Memory(LSTM)[25], and Gated Recurrent Unit(GRU)[13].

2.2.1 Vanilla RNN

The term *valilla RNN* refers to the basic RNN structure that process sequences of information. Compared to the *Multi Layer Perceptron*(MLP) who map from input to ourput vectors, the vanilla RNN can map from the entire history information to output vectors in

principle. There is an equivalent result to MLP universal approximation theory for RNN, that is an RNN can approximate any sequence-to-sequence mapping that are measurable to arbitrary accuracy[27].

Structure A typical RNN can be described in figure 7 on the left, where the graph on the right shows an unfold-in-time version of RNN. The $\mathbf{x} = (x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots)$ denotes sequence of input to the network where t indicates the time step. The s_t denotes the hidden state of input $\mathbf{x}$ at time t, which is the 'memory' of the RNN that flows to the next time step. To compute s_t, the input x at time t and the state output s at time $t-1$ are required

$$s_t = f(Ux_t + Ws_{t-1}) \tag{8}$$

where the function$f$ is a nonlinear activation function for instance ReLU or tanh. The first hidden state s_{-1} is initialized to all zeros typically. For each time step t, an output o_t is computed by

$$o_t = softmax(Vs_t) \tag{9}$$

Since the state s_t captures the previous time steps information, the output o_t is computed based on the memory generated before time t. The weight (U, W, V) are the same in each time steps, which reduce the number of parameters to be learnt.

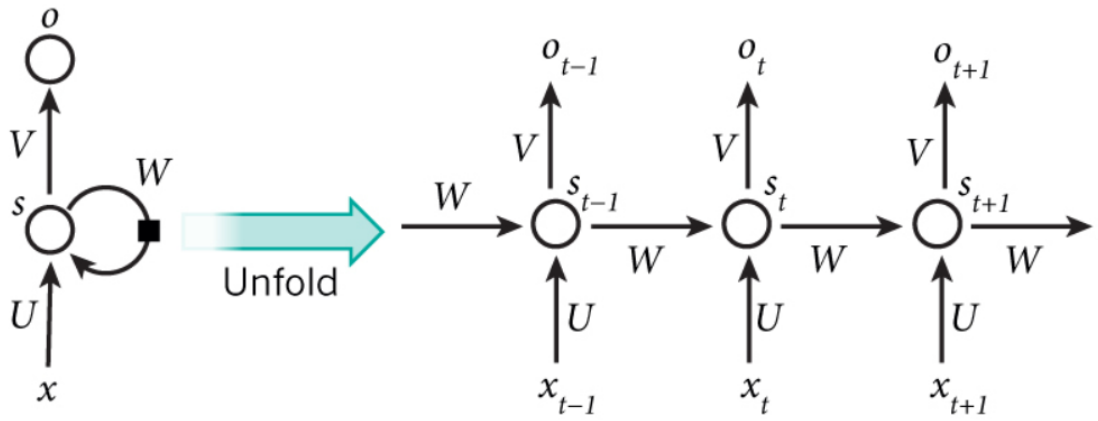

Figure 7: A recurrent neural network and the unfolding in time of the computation.

Backpropagation through time While training, backpropagation through time(BPTT)[72] algorithm is used to update the weight in the network, which computes the gradient for each time step by backpropagete the steps before that time step and sum them up. Denote the input of hidden state s_t as net_t, we have

$$net_t = Ux_t + Ws_{t-1} \tag{10}$$

$$s_{t-1} = f(net_{t-1}) \tag{11}$$

by chain rule we have

$$\frac{\partial net_t}{\partial net_{t-1}} = \frac{\partial net_t}{\partial s_{t-1}} \cdot \frac{\partial s_{t-1}}{\partial net_{t-1}} \tag{12}$$

$$= W diag[f'(net_{t-1})] \tag{13}$$

with equation (12), the error δ_k at any time step k can be computed by

$$\nabla_k^T = \frac{\partial E}{\partial net_k} \tag{14}$$

$$= \frac{\partial E}{\partial net_t}\frac{\partial net_t}{\partial net_k} \tag{15}$$

$$= \frac{\partial E}{\partial net_t}\frac{\partial net_t}{\partial net_{t-1}}\frac{\partial net_{t-1}}{\partial net_{t-2}} \cdots \frac{\partial net_{k+1}}{\partial net_k} \tag{16}$$

$$= \nabla_t^T W diag[f'(net_{t-1})] W diag[f'(net_{t-2})] \dots W diag[f'(net_k)] \tag{17}$$

$$= \nabla_t^T \prod_{i=k}^{t-1} W diag[f'(net_i)] \tag{18}$$

To backpropagate through time for weight W, the gradient of error E with respect to W is computed as

$$net_t = Ux_t + W(net_{t-1}) \tag{19}$$

$$\frac{\partial net_t}{\partial W} = \frac{\partial W}{\partial W} f(net_{t-1}) + W\frac{\partial f(net_{t-1})}{\partial W} \tag{20}$$

$$\Delta_W E = \frac{\partial E}{\partial W} \tag{21}$$

$$= \frac{\partial E}{\partial net_t}\frac{\partial net_t}{\partial W} \tag{22}$$

$$= \nabla_t^T \frac{\partial W}{\partial W} f(net_{t-1}) + \nabla_t^T W \frac{\partial f(net_{t-1})}{\partial W} \tag{23}$$

$$\tag{24}$$

where

$$\nabla_t^T \frac{\partial W}{\partial W} f(net_{t-1}) = \nabla_t^T \frac{\partial W}{\partial W} s_{t-1} \tag{25}$$

$$= \Delta_{W_t} E \tag{26}$$

$$\tag{27}$$

$$\nabla_t^T W \frac{\partial f(net_{t-1})}{\partial W} = \nabla_t^T W \frac{\partial f(net_{t-1})}{\partial net_{t-1}} \frac{\partial net_{t-1}}{\partial W} \tag{28}$$

$$= \nabla_t^T W f'(net_{t-1}) \frac{\partial net_{t-1}}{\partial W} \tag{29}$$

$$= \nabla_t^T \frac{\partial net_t}{\partial net_{t-1}} \frac{\partial net_{t-1}}{\partial W} \tag{30}$$

$$= \nabla_{t-1}^T \frac{\partial net_{t-1}}{\partial W} \tag{31}$$

Thus the gradient for updating W can be obtained by

$$\nabla_W E = \nabla_{W_t} E + \nabla_{t-1}^{T} \frac{\partial net_{t-1}}{\partial W} \tag{32}$$

$$= \nabla_{W_t} E + \nabla_{W_{t-1}} E + \nabla_{t-2}^{T} \frac{\partial net_{t-2}}{\partial W} \tag{33}$$

$$= \nabla_{W_t} E + \nabla_{W_{t-1}} E + \ldots + \nabla_{W_1} E \tag{34}$$

$$= \sum_{k-1}^{t} \nabla_{W_k} E \tag{35}$$

which is the sum of the gradient at every time step. Similarly for U the gradient is computed by

$$\nabla_U E = \sum_{i=1}^{t} \nabla_{U_i} E \tag{36}$$

Exploding and vanishing gradients When unfold the RNN overtime, the network is usually very deep. Thus the exploding and vanishing gradient problem for general deep neural network could also affects the training of RNN, that the gradient is not able to backpropagate through a relatively far time steps, which makes the RNN have difficulties learning long-term dependencies[52]. Solution of this problem have been proposed, which is the RNN structure being discussed in the following subsection, the Long Short Term Memory(LSTM).

2.2.2 Long Short Term Memory

The term *Long Short Term Memory* was first introduced by Hochreiter & Schmidhuber in 1997[25], which is designed to solve the long-term dependency problem explicitly. Similar to vanilla RNN, the LSTM has the form of a chain of repeating modules of neural network, but with a different structure in each module.

Structure In stead of having only the hidden state in the RNN hidden layer, the LSTM has two states: hidden state h_t and cell state c_t(see figure 8). Thus at time step t the cell takes three value as input including network input $\mathbf{x}_t$, hidden state $\mathbf{h}_{t-1}$ from last time step and cell state $\mathbf{c}_{t-1}$ from last time step. Then cell state $\mathbf{c}_t$ and hidden state $\mathbf{h}_t$ at time step t will be the output. The core idea of LSTM is to control what information should be store in 'memory' to pass to the next state, and what should be 'forget'.

Forward Pass In each cell of LSTM, there are four neural network layers which learn to control the memory, where each layer is equipped with an activation function(sigmoid or tanh). To decide what information to be thrown away, the cell look at the hidden state h_{t-1} from previous step and the network input x_t at current step, and compute the value for the 'forget gate' as

$$f_t = sigmoid(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{37}$$

which returns a value between 0 and 1 to indicate 'throw' or 'keep'. Similarly, to decide what information to be memorized, another set of weight is utilized to compute the value

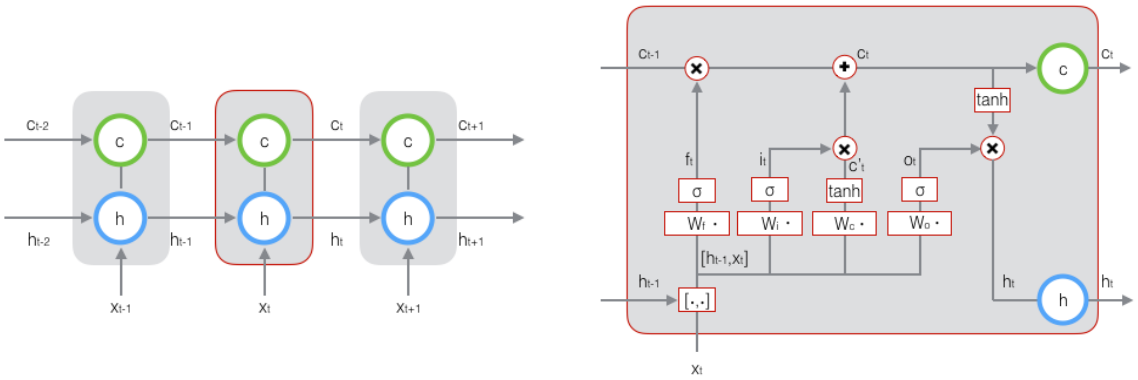

Figure 8: LSTM hidden structures. Left: The unfold LSTM. Right: the neural network in each cell.

for 'input gate' as

$$i_t = sigmoid(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{38}$$

Then, to create a vector $\widetilde{C}_t$ for new candidate values that will be added to the state, the tanh function is utilized, where

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{39}$$

And the new cell state is updated by

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t \tag{40}$$

Where the $\odot$ denotes the Hadamard(element-wise) product. Finally, to decide what to be output as the hidden state of current cell, an 'ouput gate' is computed and apply on the new cell state

$$o_t = sigmoid(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{41}$$

$$h_t = o_t \odot tanh(C_t) \tag{42}$$

The training of the LSTM is also done by BPTT.

2.2.3 Gated Recurrent Unit

In 2014, Cho *et al* proposed the Gated Recurrent Unit(GRU) which also solve the gradient vainishing problem and can be considered as a variation of the LSTM. Without using the memory unit, the structure of GRU is simpler which results in computationally more efficient. And the results produced by GRU is equally nice as the LSTM.

Structure In GRU, only two gates are utilized: update gate and reset gate, which decide the information be passed to the output. They are able to be trained to memorize information from long ago without removing information that are irrelevant to the tasks.

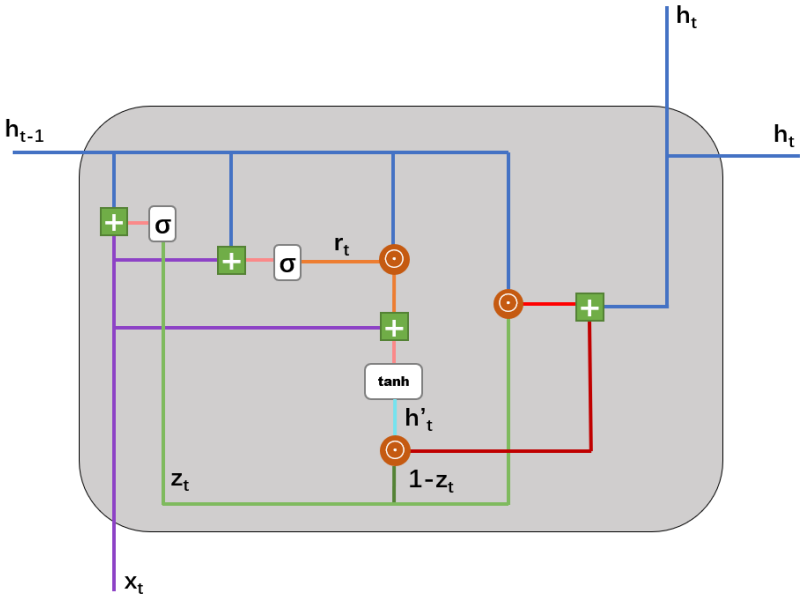

Figure 9: Detailed vision of a single GRU.

Forward Pass As receiving network input $\mathbf{x}_t$ and previous hidden state $\mathbf{h}_t$, the update gate is computed by

$$z_t = sigmoid(W_z x_t + U_z h_{t-1}) \tag{43}$$

which decide how much of the information from previous steps should be passed to the future. The reset gate has the same formula as update gate but with different parameters, computed by

$$r_t = sigmoid(W_r x_t + U_r h_{t-1}) \tag{44}$$

To store the relevant information from the previous step into a memory content h'_t, the reset gate is utilized as

$$h'_t = tanh(W x_t + r_t \odot U h_{t-1}) \tag{45}$$

And the output hiddent state $\mathbf{h}_t$ will be computed by

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \tag{46}$$

2.3 Autoencoder

The autoencoder is an artificial neural network that learns to compress(encode) data to a low-dimensional representation, then reconstruct(decode) the data from this representation to make it as close to the originial input as possible efficiently, in an unsupervised way. It reduces the dimension of the input data by learning to ignore the noise in the data.

2.3.1 Structure

An autoencoder consists of four parts: encoder, bottleneck, decoder, and reconstruction loss. The encoder and decoder works as transitions which can be defined as

$$Encoder \quad \gamma : \mathcal{X} \to \mathcal{F} \tag{47}$$

$$Decoder \quad \eta : \mathcal{F} \to \mathcal{X} \tag{48}$$

where $\mathcal{X}$ denotes the input space and $\mathcal{F}$ denotes the latent space that lies in the bottleneck. The reconstruction loss is usually computed by the L2 loss between the original input $\mathbf{X}$ and the reconstruction representation:

$$Loss = ||X - (\eta \circ \gamma)X||^2 \tag{49}$$

In a simplest case of autoencoder with one hidden layer, the encoder ecodes the input $\mathbf{x} \in \mathcal{X}$ to the latent representation $\mathbf{z} \in \mathcal{F}$ by

$$\mathbf{z} = f(\mathbf{W}\mathbf{x}+\mathbf{b}) \tag{50}$$

where the f is an element-wise activation function, and $\mathbf{W}$ and $\mathbf{b}$ are the weight matrix and bias vector respectively. In the decoder stage, the latent representation $\mathbf{z}$ is decoded to the reconstruction $\mathbf{x}'$ by

$$\mathbf{x}' = f'(\mathbf{W}'\mathbf{z}+\mathbf{b}') \tag{51}$$

the output $\mathbf{x}'$ has the same shape as input $\mathbf{x}$ and the $\mathbf{W}$', $\mathbf{b}$' and f' could be different from the encoder, depending on the design of the autoencoder.

The autoencoder is trained to minimized the reconstruction loss

$$\mathcal{L}(\mathbf{x},\mathbf{x}') = ||\mathbf{x}\text{-}\mathbf{x}'||^2 \tag{52}$$

$$= ||\mathbf{x} - f'(\mathbf{W}'(f(\mathbf{W}\mathbf{x}+\mathbf{b})) + \mathbf{b}')||^2 \tag{53}$$

$$\gamma, \eta = arg \quad min\mathcal{L}(\mathbf{x},\mathbf{x}') \tag{54}$$

If the latent space has lower dimension than the input space, the latent vector $\mathbf{z}$ can be regarded as a compressed representation of the input.

2.4 Generative Adversarial Network

The Generative Adversarial Network(GAN)[28] is a promising unsupervised machine learning methodology proposed by Ian Goodfellow in 2014, which consists of two deep neural network participating in a zero-sum game against each other. One of them is called *Generator Network* $G(\mathbf{z})$, which attempt to fool the discriminator by generating real-looking images from a noise variable $\mathbf{z}$ sampled from a prior distribution $P_{\mathbf{z}}(\mathbf{z})$, where the other one is the *Discriminator Network* $D(\cdot)$, that tries to distinguish between real images $\mathbf{x}$ and fake images $G(\mathbf{z})$. The value function $V(G, D)$ of the minimax game is

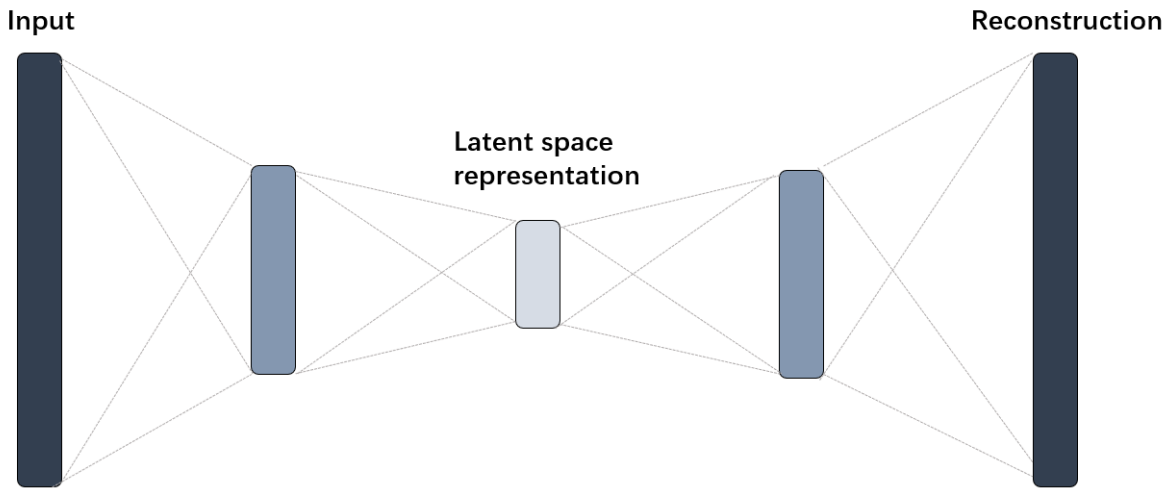

Figure 10: The structure of a standard autoencoder.

formulated as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim \mathrm{p_{data}}(\mathbf{x})}[\log \mathrm{D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim \mathrm{p_z}(\mathbf{z})}[\log(1-\mathrm{D}(\mathrm{G}(\mathbf{z})))]. \tag{55}$$

where the $P_{data}(\mathbf{x})$ is the data generating distribution. Such construction of architecture and objective function were criticized due to its prone to mode collapse, thus more variations of GAN has been proposed later on to address this problem,e.g. WGAN[49] and BEGAN[9].

2.5 Attention Mechanism

While processing sequences of information with recurrent neural network, most of the proposed model are designed to encode the input sequence into a fixed length vector[8], then generate output from this encoded vector. Since all the important information of the input sequence has been compressed to a fixed-length vector, the model is tend to lose useful information when dealing with longer input sequence, which has been proved by Cho *et al*[11] that the performance of RNN models using fix-length representation depreciate rapidly as the increment of the input sequence length. To address this problem, the *Attention Mechanism* has been proposed.

Intuition The attention mechanism can be regarded as a memory access mechanism, that it removes the information bottleneck and allows the model accessing not only the final step output of the RNN. The hidden states of RNN are utilized to make decision about which information is important at the moment, and the attention weights will be provided which indicates the importance of each input information regarding to specific tasks(e.g. how relative does a word in a sentence to a partition of the image).

Formation Denote the input of RNN at time step t as v_i, the hidden state of RNN from previous step as q. The attention weight of input v_i is computed by the softmax

value of a similarity function

$$weight(v_i, q) = softmax(similarity(v_i, q)) \tag{56}$$

While the similarity function can be designed depends on the task, two common ways is introduced here. The first one is *Additive Similarity*, which is basically using a neural network with single layer:

$$similarity(v, q) = tanh(W_v v + W_q q + bias) \tag{57}$$

where the activation function can be chosen to use alternatives. Another kind of similarity funcion is *Multiplication Similarity*, which is basically dot product between inputs and hidden states:

$$similarity(v, q) = v \cdot q \tag{58}$$

In the case that v and q are both normalized, this is equivalent to cosine similarity. The advantage of the multiplicative similarity is the speed of calculation, where the dot product is faster. The output of the attention mechanism is the weighted sum of the input v_i:

$$attention(v, q) = \sum(weight(v_i, q) \cdot v_i) \tag{59}$$

With the equipment of these background knowledge in Euclidean domain, generalization of deep learning on non-Euclidean domain will be delivered in next section.

3 Deep Learning on non-Euclidean Domain

In this section, we are going to discuss the generalization of deep learning on non-Eulidean domain. We will first introduce some notions and terminologies of the two prototytpical non-Euclidean objects: graph and manifold. Then we summarize different approaches to perform convolution on non-Euclidean domain in order to build neural network for representation learning tasks. Last, we review some state-of-the-art deep learning approaches on face analysis while the face object is defined on non-Euclidean domain.

3.1 Graph Theory

3.1.1 Definitions

The term *Graph* in mathematics refers to a collection of lines and points, where the points are denoted as *vertices* and the lines connecting the vertices are denoted as *edges*. Various types of graphs are available for different purpose including multigraph, pseudograph, oriented graph etc., while this thesis will focus on weighted undirected graph. Under this assumption, the order of the paired edges does not affect the representation meaning, i.e. edge $\mathcal{E}_{ij}$ is equivalent to $\mathcal{E}_{ji}$. The weight value on the edge of the graph is assumed to be non-negative, which usually represent the similarity or relation between vertices.

Define a weighted undirected graph $\mathcal{G}$ as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which consist of vertices $\mathcal{V} = \{v_1, \ldots, v_n\}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. For each connected pair of vertices, the weight of the edge between them is denoted by $w_{ij} \geq 0$, where $(i, j) \subseteq \mathcal{E}$. And the vertices that are joined by an edge are *adjacent*.

In graph analysis tasks, signals or vertex functions $f : \mathcal{V} \to \mathbb{R}$ that are defined on the graph have real value on each vertex, which can be represented by vectors $\mathbf{g} = (g_1, \ldots, g_n)$, where g_i is the scalar value(example of signals on graph see figure 11). In the case that taking two vertices and add them directly is not applicable due to the graph is not a vector space, researchers treating functions on the vertices as a vector space. Under this setting, adding two function on the graph is applicable. Define Hilbert spaces $L^2(\mathcal{V})$ and $L^2(\mathcal{E})$, where the standard inner product of two functions on vertices and edges of the graph can be conduct by

$$< f, g >_{L^2(\mathcal{V})} = \sum_{i \in \mathcal{V}} a_i f_i g_i \tag{60}$$

$$< F, G >_{L^2(\mathcal{E})} = \sum_{i \in \mathcal{E}} w_{ij} F_{ij} G_{ij} \tag{61}$$

With which performing operations on functions on vertices and edges results in performing operations on graph. The weight a_i and w_{ij} are positive values which invariant to permutation of indices. When these weights taking constant value 1, the inner product is the standard l^2-inner product.

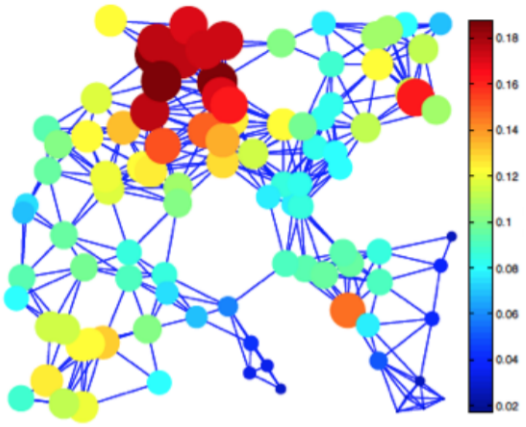

Figure 11: Heat diffusion represented by signals on graph. The size and the color of each vertex indicating the value of signal(temperature) on vertex.[66]

3.1.2 Graph Laplacian and Eigendecomposition

The *graph Laplacian* is a key construction in the field of spectral graph analysis. To derive this term, the *graph gradient* and *graph divergence* needed to be introduced[40].

Definition 3.1.2.1 The *graph gradient* is a linear operator denoted as $\nabla : L^2(\mathcal{V}) \to L^2(\mathcal{E})$, which maps vertex functions to edge functions.

$$(\nabla f)_{ij} = f_i - f_j \tag{62}$$

which satisfies $(\nabla f)_{ij} = -(\nabla f)_{ji}$ automatically.

Definition 3.1.2.2 The *graph divergence* is a linear operator that does the converse of gradient: $\nabla : L^2(\mathcal{E}) \to L^2(\mathcal{V})$, which maps from functions defined on edges to those defined on vertices,

$$(divF)_i = \frac{1}{a_i} \sum_{j:(i,j)\in\mathcal{E}} w_{ij} F_{ij}. \tag{63}$$

Definition 3.1.2.3 The *graph Laplacian* $\Delta : L^2(\mathcal{V}) \to L^2(\mathcal{V})$ can be constructed by these two operators, defined as $\Delta = -div\nabla$, can be expressed by a combination of (62)(63)

$$(\Delta f)_i = \frac{1}{a_i} \sum_{(i,j)\in\mathcal{E}} w_{ij}(f_i - f_j) \tag{64}$$

The intuitive geometric interpretation of the graph Laplacian is captured by this formula, which is taking differences between function on the vertex and its local average around it. There is an alternative to construct the graph Laplacian(unnormalized), which is

$$\Delta = \mathbf{D} - \mathbf{W} \tag{65}$$

The matrix $\mathbf{D} = diag(\sum_{j:j\neq i} w_{ij})$ is the degree matrix which contains the vertex degree $deg(v_i)$ at the $d_{i,j,i=j}$ position. The matrix $\mathbf{W} = (w_{ij})$ is an $n \times n$ matrix which contains

the edge weights. For those unconnected vertices $(i,j) \notin \mathcal{E}$, $w_{ij} = 0$. Applying the inverse of the diagonal matrix $\mathbf{A} = diag(a_1, \ldots, a_n)$ of vertex weights, the normalized graph Laplacian can be obtained

$$\Delta = \mathbf{A}^{-1}(\mathbf{D} - \mathbf{W}) \tag{66}$$

If choose $\mathbf{A}$=$\mathbf{D}$, the random walk laplacian can be obtained[67].

Eigendecomposition of Graph Laplacian The graph Laplacian is self-adjoint positive-semidefinite, which allows performing eigendecomposition that results in a discrete set of orthonormal eigenfunctions $\phi_0, \phi_1, \ldots$. The orthonormality guarantees that for any ϕ_i, ϕ_j, $< \phi_i, \phi_j >_{L^2(\mathcal{X})} = \delta_{ij}$. For each eigenfunction, there is a corresponding non-negative real eigenvalues $0 = \lambda_0 \leq \lambda_1 \leq \ldots$ where

$$\Delta\phi_i = \lambda_i\phi_i, \quad i = 0, 1, \ldots \tag{67}$$

The eigenfunctions are the smoothest functions when considering the Dirichlet energy.

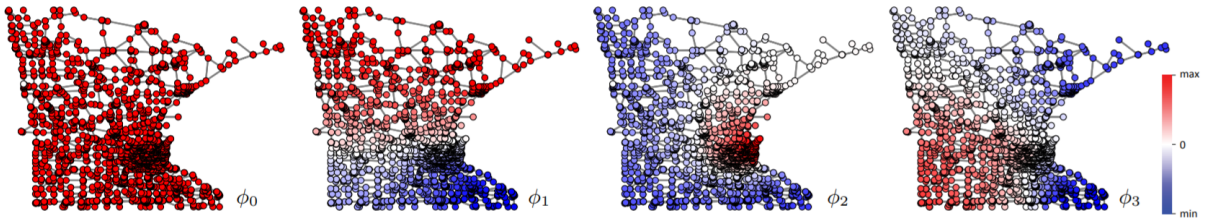

Figure 12: Visualization of the eigenfunction on non-Eulidean domain. The figure show the first four Laplacian eigenfunctions on Minnesota road graph[5]

The Dirichlet energy is a term that measures how smooth a funcion is. For a function f on domain $\mathcal{F}$, the Dirchlet energy is calculated by

$$\mathcal{E}_{Dir}(f) = \int_{\mathcal{F}} ||\nabla f(x)||^2_{T_x\mathcal{F}} dx \tag{68}$$

$$= \int_{\mathcal{F}} f(x)\nabla f(x) dx \tag{69}$$

where the smoothness is assigned according to the variation of the value of the function. For a function varies significantly, the value of the Dirichlet energy is large, for those that varies slightly, the value of the Dirichlet energy is small. If achieves zero energy, then the function is constant. To obtain the k smoothest possible functions(eigenfunctions), we solve the following optimization problem

$$\min_{\phi_0} \mathcal{E}_{Dir}(\phi_0) \quad s.t. \quad ||\phi_0|| = 1 \tag{70}$$

$$\min_{\phi_i} \mathcal{E}_{Dir}(\phi_i) \quad s.t. \quad ||\phi_i|| = 1, \quad i = 1, 2, \ldots, k-1 \tag{71}$$

$$\phi_i \perp span\{\phi_0, \ldots, \phi_{i-1}\} \tag{72}$$

In practise, the domain is usually sampled at n points, which converts the problem into formulation under the discrete setting

$$\min_{\mathbf{\Phi}_k \in \mathbb{R}^{n\times k}} trace(\mathbf{\Phi}_k^\top \Delta \mathbf{\Phi}_k) \quad s.t. \quad \mathbf{\Phi}_k^\top \mathbf{\Phi}_k = \mathbf{I} \tag{73}$$

where the solution $\mathbf{\Phi}_k$ is the collection of the first k eigenfunctions of the graph Laplacian, $\mathbf{\Phi}_k = (\phi_0, \ldots, \phi_{k-1})$ satisfying

$$\mathbf{\Delta}\mathbf{\Phi}_k = \mathbf{\Phi}_k \mathbf{\Lambda}_k \tag{74}$$

with $\mathbf{\Lambda} = diag(\lambda_0, \ldots, \lambda_{k-1})$ is a diagonal matrix containing the corresponding eigenvalues of $\mathbf{\Delta}$. The eigendecomposition of the Laplacian can be performed in two ways. One way is to rewrite the (74) as $(\mathbf{D} - \mathbf{W})\mathbf{\Phi}_k = \mathbf{A}\mathbf{\Phi}_k\mathbf{\Lambda}_k$, which is a generalized eigenproblem that results in $\mathbf{A}$-orthogonal eigenvectors satisfying $\mathbf{\Phi}_k^\top \mathbf{A} \mathbf{\Phi}_k = \mathbf{I}$. An alternative is utilizing the change of variable method with $\mathbf{\Psi}_k = \mathbf{A}^{\frac{1}{2}}\mathbf{\Phi}_k$, to transfer the problem to a standard eigendecomposition probelm $\mathbf{A}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{A}^{-\frac{1}{2}}\mathbf{\Psi}_k = \mathbf{\Psi}_k\mathbf{\Lambda}_k$ where the eigenvectors are orthogonal, $\mathbf{\Psi}_k^\top \mathbf{\Psi}_k = \mathbf{I}$.

3.1.3 Graph Fourier analysis

Graph Fourier Transform The eigenfunctions of the graph Laplacian can be explained as generalizing standard Fourier basis to the non-Euclidean domain. Because of the intrinsic construction of the Laplacian, the Laplacian eigenbasis is intrinsic. In Euclidean case, the discrete Fourier transform is performed by representing a function by a linear combination of orthogonal vectors, where the coefficient of the linear combination projects the function on the spaces formed by the orthogonal basis[65, 24]. With exponential basis, the Fourier decomposition of function $s(x)$ is

$$s(x) = \sum c_n \cdot e^{i\frac{2\pi n x}{P}} \tag{75}$$

where c_n is the coefficient and P is the period of the Fourier series. Replacing the exponential basis by the eigenvectors ϕ_k of Laplacian, the Fourier decomposition of a square-integrable function f on graph $\mathcal{X}$ can be obtained

$$f = \sum_{k=1}^{n} \underbrace{< f, \phi_k >_{L^2(\mathcal{X})}}_{\hat{f}_k} \phi_k \tag{76}$$

where the $\hat{f}_k$ denotes the discrete set of Fourier coefficients(example see figure 13) and the inner product is calculated by (60). In the sense of classical signal processing, formula (76) generalize the forward transform stage on non-Euclidean domain, while the inverse transform stage can be done by summing up the basis function ϕ_k with these coefficients $\hat{f}_k$. In matrix-vector notation, the orthogonal eigenfunction $\mathbf{\Phi} = \{\phi_1, \ldots, \phi_n\}$ can be obtained by eigendecomposition of the graph Laplacian

$$\Delta = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^\top \tag{77}$$

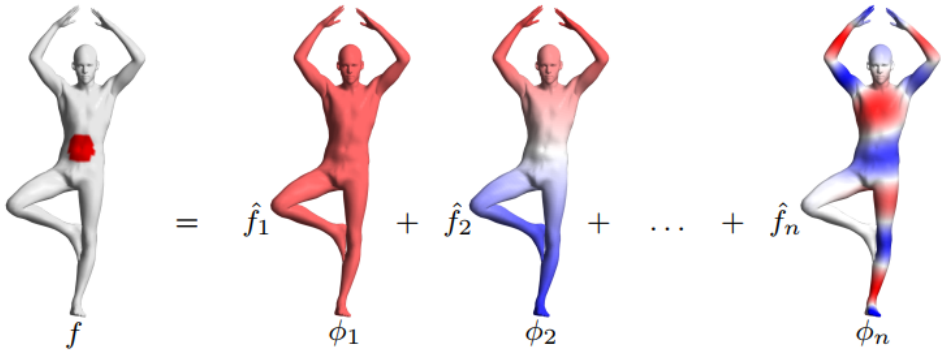

Figure 13: Example of function represented by linear combination of Laplacian eigenfunction on non-Euclidean domain.[6]

where $\mathbf{\Lambda} = diag([\lambda_1, \ldots, \lambda_n]) \in \mathbb{R}^{n \times n}$ is the matrix containing ordered real non-negative eigenvalues. The graph Fourier transform of a function $\mathbf{f}$ is performed by $\hat{\mathbf{f}} = \mathbf{\Phi}^\top \mathbf{f}$ while the inverse transform is performed by $\mathbf{f} = \mathbf{\Phi}\hat{\mathbf{f}}$[57].

Graph Convolution In classical Euclidean signal processing, the convolution operator can be diagonalized by Fourier transform, which allows expressing the convolution of two functions f and g in spectral domain by an element-wise product

$$\widehat{(f \star g)}(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx \int_{-\infty}^{\infty} g(x) e^{-i\omega x} dx \tag{78}$$

where the $\int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx$ and $\int_{-\infty}^{\infty} g(x) e^{-i\omega x} dx$ is the Fourier transform of f and g respectively. Due to the structure of non-Euclidean domain, the operation $x - x'$ cannot be defined. Thus the generalization of convolution on non-Euclidean domain utilize the *Convolution Theorem* as its definition, which is formulated by

$$(f \star g)(x) = \sum_{k \geq 0} < f, \phi_k >_{L^2(\mathcal{X})} < g, \phi_k >_{L^2(\mathcal{X})} \phi_k(x) \tag{79}$$

where the inner product is obtained by (60). Such construction is lack of shift-invariance, which can be explained as position-dependent filter in the sense of signal processing. Thus the spatial representation of the filter at different position of the object can vary dramatically when parametrized in the frequency domain by fixed number of coefficients. Since the graph Laplacian has n eigenvectors, the sum over i becomes finite. The convolution $f \star g$ can be expressed in matrix-vector notation

$$\mathbf{f} \star \mathbf{g} = \mathbf{\Phi}((\mathbf{\Phi}^\top \mathbf{f}) \odot (\mathbf{\Phi}^\top \mathbf{g})) \tag{80}$$

where the $\odot$ is the Hadamard product. The spectral graph filtering is defined as applying a filter $g_\theta(\Delta)$ on a function $\mathbf{f}$

$$\begin{aligned}
\mathbf{y} &= g_\theta(\Delta)\mathbf{f} && (81)\\
&= g_\theta(\mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^\top)\mathbf{f} && (82)\\
&= \mathbf{\Phi} g_\theta(\mathbf{\Lambda})\mathbf{\Phi}^\top\mathbf{f} && (83)\\
&= \mathbf{\Phi}\begin{bmatrix} g_\theta(\lambda_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & g_\theta(\lambda_n) \end{bmatrix}\mathbf{\Phi}^\top\mathbf{f} && (84)
\end{aligned}$$

3.1.4 Graph Coarsening

In classical CNN, pooling operation allows gaining computational performance with less spatial information, as well as less parameters which avoid overfitting. The global view provided by pooling operation enable obtaining rotation and position invariance. Thus for graph objects, a nice pooling method that preserve the properties of graph as in classical CNN would results in a nice hierarchical multi-scale representation, which allow global context be captured by deeper layers and local context be captured by shallow layers.

The process of reducing number of vertices and edges while preserving intrinsic geometric structure is referred as *graph coarsening*, which transform a given graph $G = \{\mathcal{V}, \mathcal{E}\}$ into a coersed version of graph $G' = \{\mathcal{V}', \mathcal{E}'\}$[38]. Since the main graph object we study in this thesis is 3D face object, which is discretized as polygonal surface models, that is a form of relatively smoother graph. We introduced a graph coarsening approach utilized in [55] which focus on surface simplification that produce high quality lower dimensional approximation to the target graph model.

The pooling of the graph is performed by applying a transform matrix $Q_d \in \{0,1\}^{n\times m}$, which contains binary indication of whether to keep or discard a certain vertex. For discarding the q-th vertex, $Q_d(p,q) = 0, \forall p$, while for keeping the q-th vertex, $Q_d(p,q) = 1, \forall p$. The discipline for choosing vertex to be kept or discarded is minimizing the quadric error when contracting vertex pairs iteratively[23]. Denote the contraction of vertex pairs by $(\mathbf{v}_1, \mathbf{v}_2) \to \hat{v}$, which moves both vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ to the new position $\hat{v}$ and connect all the incident edges of $\mathbf{v}_1$ and $\mathbf{v}_2$ to $\hat{v}$. In some cases, the new position can simply be chosen as one of the two paired vertices and the other one will be discarded. The selection of vertices to be contracted assumes that points should not move long distance from it original position in a nice approximation, thus an appropriate pair of vertices $(\mathbf{v}_1, \mathbf{v}_2)$ to be considered should satisfy one of the two following criterion:

1.$(\mathbf{v}_1, \mathbf{v}_2)$ is an edge

2.$||\mathbf{v}_1 - \mathbf{v}_2|| < c$

where c is a set threshold to allow non-connected vertices to be considered. The simplest case can set $c = 0$. For each contraction, a *cost* is calculated for aiding decision making of which vertex to be kept. The cost is measured by the error at each vertex, where a symmetric matrix $\mathbf{M} \in \mathbb{R}^{4\times4}$ is computed for each vertex $\mathbf{v} = [v_x, v_y, v_z, 1]^\top$. An error

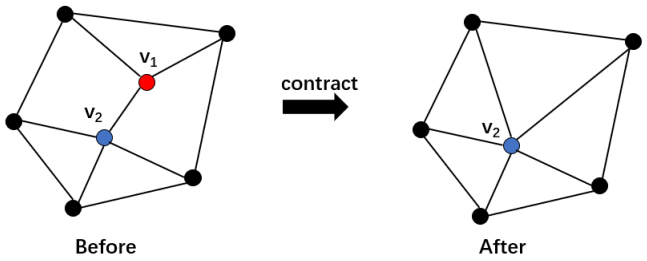

Figure 14: Contracting vertex pair $(\mathbf{v}_1, \mathbf{v}_2) \rightarrow \mathbf{v}_2$. Faces of the graph be removed during contracting.

metric Θ will be constructed based on the $\mathbf{M}$ matrix. Consider each vertex $\mathbf{v}$ as a solution of the intersection of the triangular faces around that position, the error $\Theta(\mathbf{v})$ of the vertex with respect to the set of planes containing the faces around it can be defined as the sum of squared distance to these planes. The plane $\mathbf{p}$ is denoted as $\mathbf{p} = [a, b, c, d]$ where $ax + by + cz + d = 0$ is the equation that defines a plane. To guarantee the uniqueness of the plane, a constrain $a^2 + b^2 + c^2 = 1$ requires to be satisfied.

$$\begin{aligned}
\Theta(\mathbf{v}) &= \Theta([v_x, v_y, v_z, 1]^\top) && (85)\\
&= \sum_{\mathbf{p} \in planes(\mathbf{v})} (\mathbf{p}^\top \mathbf{v})^2 && (86)\\
&= \sum_{\mathbf{p} \in planes(\mathbf{v})} (\mathbf{v}^\top \mathbf{p})(\mathbf{p}^\top \mathbf{v}) && (87)\\
&= \sum_{\mathbf{p} \in planes(\mathbf{v})} \mathbf{v}^\top \mathbf{p}\mathbf{p}^\top \mathbf{v} && (88)\\
&= \mathbf{v}^\top \Big(\sum_{\mathbf{p} \in planes(\mathbf{v})} K_p \Big) \mathbf{v} && (89)\\
& && (90)
\end{aligned}$$

where the K_p is the fundamental error metric that helps to obtain the squared distance of arbitrary point in space to the plane $\mathbf{p}$.

$$K_p = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \tag{91}$$

The $\mathbf{M}$ matrix mentioned before is computed by summing the K_p of each of the plane in the plane set, $\mathbf{M} = \sum_{\mathbf{p} \in planes(\mathbf{v})} K_p$. Hence, the error of each vertex can be rewritten as

$$\Theta(\mathbf{v}) = \mathbf{v}^\top \mathbf{M} \mathbf{v} \tag{92}$$

When considering the contraction, an updating rule $\widehat{\mathbf{M}} = \mathbf{M_1} + \mathbf{M_2}$ for $\mathbf{M}$ is computed for each vertex pair while the resulting value of error $\Theta(\mathbf{v})$ will be placed in a heap keyed on cost. The smaller the cost, the higher it will be placed in the heap. The least cost pair

$(\mathbf{v_i}, \mathbf{v_j})$ will be removed iteratively from the heap i.e. contracted. The cost of existing pairs is updated after every contraction. The final contracting decision will be filled into the transform matrix Q_d and the coarsening operation is done by sparse matrix multiplication $\mathbf{G}_d = Q_d\mathbf{G}$.

3.2 Manifolds

3.2.1 Definition

The *manifold* in geometric deep learning refers to the continuous topological spaces which is locally Euclidean. Due to lacking of global vector space structure, performing addition or subtraction with two points on a manifold is not appropriate, but within a small local region, the small curvature allows it to be considered as Euclidean space. For each point x on a r-dimensional manifold $\mathcal{X}$, the neighborhood around x is equivalent to a r-dimensional Euclidean space topologically, which is referred as tangent space $T_x\mathcal{X}$. The *tangent bundle* $T\mathcal{X} = [T_1\mathcal{X}, \ldots, T_n\mathcal{X}]$ is the collection of all tangent space in a manifold(see figure 15 left). To perform local measurements of geometric structures such as angles, distances and volumes, the *Riemannian metric* is defined, which is an inner product within a tangent space

$$< \cdot, \cdot >_{T_x\mathcal{X}} : T_x\mathcal{X} \times T_x\mathcal{X} \to \mathbb{R} \tag{93}$$

With the equipment of this metric, a manifold is called *Riemannian manifold*. By the *Nash Embedding Theorem*, any Riemannian manifold that is smooth sufficiently can be realized in a sufficiently high dimensional Euclidean space[34], where multiple realizations might be available for one Riemannian metric, called *isometries*.

In computer vision tasks, the two-dimensional manifolds embedded into $\mathbb{R}^3$ are utilized to model boundary surfaces of 3D objects, where the "3D" refers to the embedding space dimensionality. The fact that isometries do not affect the metric structure of the manifold when deformation happened results in any quantities expressed in terms of Riemannian metric being preserved. Such nice property is called *intrinsic*.

3.2.2 Functions on manifold

To represent the information on manifold as well as perform operations, functions on manifold need to be defined, in particular: *scalar field* and *vector field*. The scalar field is a smooth real function $f : \mathcal{X} \to \mathbb{R}$ that assigns real value to points on the manifold, where the vector field is a mapping $F : \mathcal{X} \to T\mathcal{X}$ that take each point x on the manifold and assigns a tangent vector $F(x) \in T_x\mathcal{X}$. Define a standard Hilbert space of functions on manifolds, denoted by $L^2(\mathcal{X})$ for for scalar field and $L^2(T\mathcal{X})$ for vector field, the inner

product of these functions expressed as

$$< f, g >_{L^2(\mathcal{X})} = \int_{\mathcal{X}} f(x)g(x)dx \tag{94}$$

$$< F, G >_{L^2(T\mathcal{X})} = \int_{\mathcal{X}} < F(x), G(x) >_{T_x\mathcal{X}} dx \tag{95}$$

where the d-dimensional volume element induced by the Riemannian metric is denoted by dx.

When analysing functions, the derivative measures the sensitivity of the function change with an infinitesimal change of the input argument, which is used extensively when analyzing function behavior. However, since the manifold does not have the vector space structure, the generalization of derivative on functions on manifold need to work within the tangent space locally. Define the derivative of f as $df : T\mathcal{X} \to \mathbb{R}$, which defines an operator at every point x as $df(x) =< \nabla f(x), \cdot >_{T_x\mathcal{X}}$ to model the small displacement around x. When apply on a tangent vector $F(x) \in T_x\mathcal{X}$, the value change of the function due to the displacement is given by $df(x)F(x) =< \nabla f(x), F(x) >_{T_x\mathcal{X}}$. This operator ∇f is called the *intrinsic gradient*, which is similar to the classical notion of the gradient except for the steepest change direction is a tangent vector.

Similarly, the divergence operator is defined as $L^2(T\mathcal{X}) \to L^2(\mathcal{X})$ which is adjoint to the gradient[62]

$$< F, \nabla f >_{L^2(T\mathcal{X})} =< \nabla^* F, f >_{L^2(\mathcal{X})} =< -divF, \nabla f >_{L^2(\mathcal{X})} \tag{96}$$

Finally the Laplacian $\Delta : L^2(\mathcal{X}) \to L^2(\mathcal{X})$ of the manifold is defined as an operator acting on scalar field as

$$\Delta f = -div(\nabla F) \tag{97}$$

which is symmetric.

3.2.3 Discrete Manifold

In practise situation, the manifold is discretized for shape modeling tasks. To guarantee the geometrically consistency, a valid approach is to consider not only the sampled points $\mathcal{V} = \{v_i\}_{i=1,\dots,n}$ and connected edges $\mathcal{E} = \{(i,j)\}_{i,j=1,\dots,n}$, but also the faces $\mathcal{F} \in \mathcal{V} \times \mathcal{V} \times \mathcal{V}$, where the continuous manifold is modeled as a polyhedral surface. The resulting manifold dicretization is referred to as *triangular mesh* $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. The mesh Laplacian is defined as

$$(\Delta f)_i = \frac{1}{\frac{1}{3}\sum_{jk:(i,j,k)\in\mathcal{F}} a_{ijk}} \sum_{(i,j)\in\mathcal{E}} \left(\frac{-l_{ij}^2 + l_{jk}^2 + l_{ik}^2}{8a_{ijk}} + \frac{-l_{ij}^2 + l_{jh}^2 + l_{ih}^2}{8a_{ijh}}\right)(f_i - f_j) \tag{98}$$

where the $a_{ijk} = \sqrt{s_{ijk}(s_{ijk} - l_{ij})(s_{ijk} - l_{jk})(s_{ijk} - l_{ik})}$ is the area of the triangle (i,j,k) and $s_{ijk} = \frac{1}{2}(l_{ij} + l_{jk} + l_{ki})$ is the semi-perimeter. The weights in formula (98) are intrinsic due to solely expressed in terms of the discrete metric. If infinitely refined the mesh under certain technical condition, the resulting construction can converge to the continuous

Laplacian of the manifold[47].

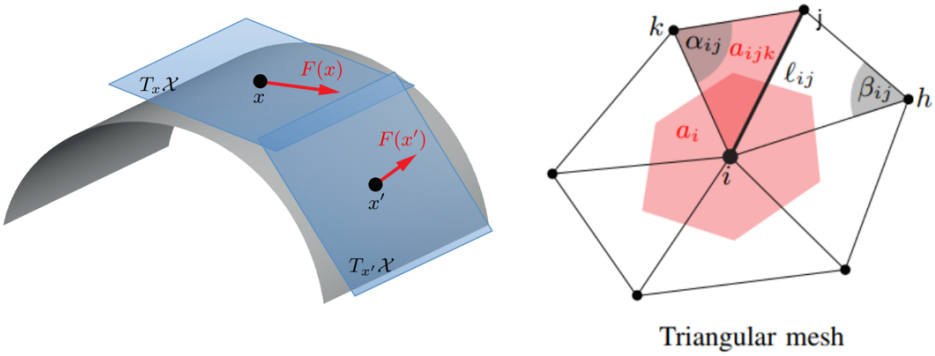

Figure 15: Left: Two-dimensional manifold with tangent space and tangent vectors. Right: A triangular mesh discretization of a two-dimensional manifold, where the $a_i = \frac{1}{3}\sum_{jk:(i,j,k)\in\mathcal{F}} a_{ijk}$ is the local area element[5].

3.3 Spectral Convolution Operations

In this section, different approaches to generalize convolutional architecture on non-Euclidean domain are discussed, which, in particular, operating on the spectrum of the graph weights. The domain being study here are assumed to be fixed.

3.3.1 Spectral CNN

Similar to the classical CNN architecture in Euclidean domain, the spectral CNN consists of functional layers stacking together(see figure 16)[18]. Due to the special structure of the input graph, the convolutional operation and pooling operation is different from classical CNN.

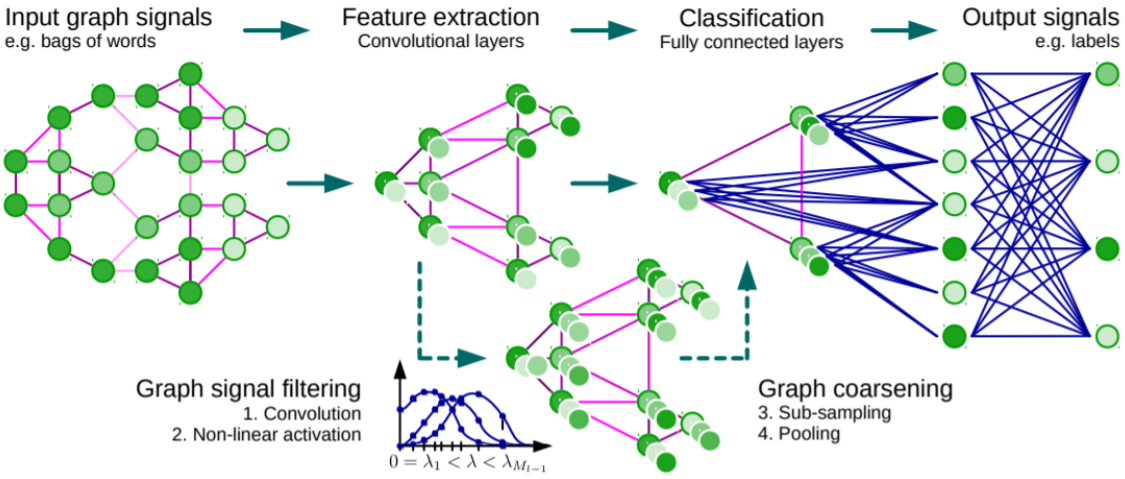

Figure 16: Typical architecture of spectral CNN[18].

Convolutional Layer The convolutional layer expressed in spectral domain is

$$\mathbf{g}_l = \xi\left(\sum_{l'=1}^{p} \mathbf{\Phi}\mathbf{W}_{l,l'}\mathbf{\Phi}^\top \mathbf{f}_{l'}\right) \tag{99}$$

where $l = 1, \ldots, q$ and $l' = 1, \ldots, p$ indicating the dimension of the input and output respectively. The input function $\mathbf{f}$ of the layer are stored in an $n \times q$ matrix $\mathbf{F} = (\mathbf{f}_1, \ldots, \mathbf{f}_q)$ while the output $\mathbf{g}$ are stored in an $n \times p$ matrix $\mathbf{G} = (\mathbf{g}_1, \ldots, \mathbf{g}_p)$, $n = |\mathcal{V}|$ indicating the number of vertices of the graph. The $\mathbf{W}_{l,l'} \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ is a diagonal matrix containing spectral multipliers that express the convolutional filter in frequency domain. The $\mathbf{\Phi}$ is the matrix of eigenvectors of graph Laplacian computed by eigendecomposition $\mathbf{\Delta} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^\top$. And the ξ represents a nonlinear function applied to each function value, which is the same as activation function in classical CNN case. According to the intrinsic regularity of the graph as well as the sample size, a cutoff frequency will be set by using only the first k eigenvectors, which describes the smooth structure of the graph, usually $k << n$.

Pooling Layer The pooling operation is done by graph coarsening, which results in a subset of the vertices of the graph is retained. Denotes the fraction of the left vertices as α, the relation of the eigenvectors of graph Laplacian $\mathbf{\Phi} \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ of original graph and the eigenvectors of graph Laplacian $\tilde{\mathbf{\Phi}} \in \mathbb{R}^{\alpha\mathrm{n}\times\alpha\mathrm{n}}$ of the coarsed graph can be expressed by

$$\tilde{\mathbf{\Phi}} \approx \mathbf{P}\mathbf{\Phi}\begin{pmatrix}\mathbf{I}_{\alpha n} \\ \mathbf{0}\end{pmatrix} \tag{100}$$

where the binary matrix $\mathbf{P} \in \mathbb{R}^{\alpha\mathrm{n}\times\mathrm{n}}$ encodes the position of the i-th vertex of the coarse graph with respect to the original graph in the i-th row. After each pooling step, the graph Laplacian eigenvectors need to be recomputed.

Discussion Due to the basis dependency of the spectral filter coefficients, the filter learned with respect to basis $\mathbf{\Phi}_k$ on one domain could have very different result when applied to another domain with different basis $\mathbf{\Psi}_k$. And the construction of compatible orthogonal basis across distinct domain requires the correspondence between domains, which is a hard problem. Hence, the filter is not able to generalize across different domain. The number of parameter on each layer is $\mathcal{O}(n)$ at least when all the eigenvoectors of the Laplacian are used. The cost of computing the filter is also high, which is $\mathcal{O}(n^2)$ due to multiplication with the dense matrix $\mathbf{\Phi}$ and $\mathbf{\Phi}^\top$ when performing forward and invese Fourier transform. Also, there is no guarantee of the spatial localization of the filters since the filter is defined in spectral domain. The reason of attempting to achieve spatial localization is that filter with this property does not require the input graph to be *homogeneous*, i.e. graphs does not need to have same number of vertices and edge connections, and can process *heterogeneous* graphs that vary in the number of vertices and the distribution of edge connections(see figure 17)[63], which has more benefit in real world applications.

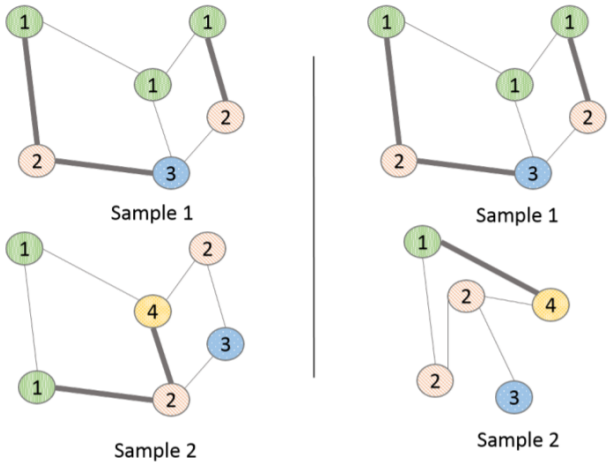

Figure 17: Left: Homogeneous graphs which only difference between graphs is the vertex values. Right: Heterogeneous graphs which difference can be structure of edge connection, number of vertices and vertex values[63].

3.3.2 Smooth spectral filter Spectral CNN

The smooth spectral filter spectral CNN has the same architecture and pooling operation as spectral CNN, while the only difference is the convolutional filter being utilized. To introduce the filter construction, the concept of localization and smoothness of the filter is required to be discussed first.

Localization and Smoothness In the Euclidean setting, to express the spatial locality in the frequency domain, the moment of a function is related to the derivative of its Fourier transform, which is the consequence of the Parseval's identity.

$$\int_{-\infty}^{+\infty} |x|^{2k} |f(x)|^2 dx = \int_{-\infty}^{+\infty} \left| \frac{\partial^k \hat{f}(\omega)}{\partial \omega^k} \right|^2 dw \tag{101}$$

To obtain a spatially localized filter, the higher order moments of the function are required to decay fast, which means that by Parseval's identity, the higher degrees of derivative of the Fourier transform should also be small, which essentially means the Fourier transform for localized signal is smooth. The notion of smoothness assumes some geometric structure in the spectral domain. In 1-dimensional Euclidean case, the structure comes in the form of distance between two basis functions, thus the measurement can be simply taking the absolute value of the difference between two frequencies

$$d(e^{i\omega x}, e^{i\omega' x}) = |\omega - \omega'| \tag{102}$$

If work in n-dimensional Euclidean space, taking the norm of the frequency vectors will be the measurement

$$d(e^{i\omega^\top x}, e^{i\omega'^\top x}) = ||\omega - \omega'|| \tag{103}$$

Though in non-Euclidean domain defining the geometric of the spectral domain, the eigenvectors of the Laplacian can be interpreted as the frquencies, thus the absolute value of the difference of the eigenvectors can be the measurement

$$d(\phi_i, \phi_j) = |\lambda_i - \lambda_j| \tag{104}$$

where the eigenvalue λ is ordered in increasing order.

Convolutional Filter Parametrize the filter as a smooth transfer function $\tau(\lambda)$, the application of the filter will amount to applying this transfer function to the Laplacian $\mathbf{\Delta}$.

$$\tau(\mathbf{\Delta})\mathbf{f} = \mathbf{\Phi}\tau(\mathbf{\Lambda})\mathbf{\Phi}^\top\mathbf{f} \tag{105}$$

The smooth transfer function can be parametrized by small number of parameters, denoted by α, where the application of the filter becomes

$$\tau(\mathbf{\Delta})\mathbf{f} = \mathbf{\Phi}\begin{pmatrix}\tau(\lambda_1) & & \\ & \ddots & \\ & & \tau(\lambda_n)\end{pmatrix}\mathbf{\Phi}^\top\mathbf{f} \tag{106}$$

$$= \mathbf{\Phi}\begin{pmatrix}\tau_\alpha(\lambda_1) & & \\ & \ddots & \\ & & \tau_\alpha(\lambda_n)\end{pmatrix}\mathbf{\Phi}^\top\mathbf{f} \tag{107}$$

A simple way of parametrizing the transfer function is to represent it as a linear combination of smooth basis functions $\beta_1(\lambda), \ldots, \beta_r(\lambda)$.(e.g. B-Spline basis[68])

$$\tau_\alpha(\lambda) = \sum_{j=1}^{r} \alpha_j\beta_j(\lambda) \tag{108}$$

where the $\alpha = (\alpha_1, \ldots, \alpha_r)^\top$ is the vector of the parameters of the filter. Written in matrix-vector form, for eigenvalue λ_k, $\tau_\alpha(\lambda) = \sum_{j=1}^{r} \alpha_j\beta_j(\lambda) = \mathbf{B}\alpha$ where $\mathbf{B} = (b_{kj}) = (\beta_j(\lambda_k))$. The overall filter is a diagonal matrix $\mathbf{W} = Diag(\mathbf{B}\alpha)$.

Discussion Utilizing the parametrized filter, the spatial localization has been guarantee by virtue of the Parseval's identity. The number of parameters describing the filter is small, ideally could be $\mathcal{O}(1)$. However, explicit computation of the Fourier transform is still required, thus the cost of computation stays $\mathcal{O}(n^2)$.

3.3.3 ChebyNets

To get rid of the cost $\mathcal{O}(n^2)$, approaches that avoid explicit computation of the eigenvectors has been proposed, which is by representing the spectral transfer function as a polynomial of order r.

$$\tau_\alpha(\lambda) = \sum_{j=0}^{r} \alpha_j\lambda^j \tag{109}$$

For guarantee the stability under coefficients perturbation, orthogonal basis of polynomials is utilized, in particular, Chebyshev polynomials. The Chebyshev polynomials are defined recursively as

$$T_j(\tilde{\lambda}) = 2\tilde{\lambda}T_{j-1}(\tilde{\lambda}) - T_{j-2}(\tilde{\lambda}) \qquad T_0(\tilde{\lambda}) = 1, T_1(\tilde{\lambda}) = \tilde{\lambda} \tag{110}$$

The $\tilde{\lambda}$ denotes the scaled frequency between $[-1, +1]$ due to this basis is orthonormal on $L^2([-1, +1])$ with respect to $< f, g >= \int_{-1}^{+1} f(\tilde{\lambda})g(\tilde{\lambda})\frac{d\tilde{\lambda}}{\sqrt{1-\tilde{\lambda}^2}}$. The reason to do so is that $[-1, 1]$ is the domain that the Chebyshev polynomial can form an orthonomal basis.

Convolutional Layer The filter parametrized by Chebyshev polynomial is defined as

$$\tau_\alpha(\mathbf{\Delta}) = \sum_{j=0}^{r} \alpha_j T_j(\tilde{\mathbf{\Delta}}) \tag{111}$$

where the $\tilde{\mathbf{\Delta}} = 2\lambda_n^{-1}\mathbf{\Delta} - \mathbf{I}$ is the rescaled Laplacian that map the eigenvalue from $[0, \lambda_n]$ to $[-1, 1]$[45]. Applying filter to the function $\mathbf{f}$ on the graph is denoted as $\bar{\mathbf{f}}^{(j)} = T_j(\tilde{\mathbf{\Delta}})\mathbf{f}$ and by recurrent relation (110) we have

$$\bar{\mathbf{f}}^{(j)} = 2\tilde{\mathbf{\Delta}}\mathbf{f}^{(j-1)} - \mathbf{f}^{(j-2)} \tag{112}$$

$$\bar{\mathbf{f}}^{(0)} = \mathbf{f} \tag{113}$$

$$\bar{\mathbf{f}}^{(1)} = \tilde{\mathbf{\Delta}}\mathbf{f} \tag{114}$$

Pooling Layer The pooling operation of the ChebyNet is the same as spectral CNN which making use of the graph coarsening technique.

Simplified ChebyNet In 2016, Kips and Welling proposed a simplification construction[35] to the ChebyNet algorithm, which assumes that the largest value of the eigenvalue $\lambda_{[}n] = 2$ and the highest degree of the polynomial is $r = 2$. In this case, applying convolutional filter to function $\mathbf{f}$ is of the form

$$\tau_\alpha(\mathbf{f}) = \alpha_0\mathbf{f} + \alpha_1(\mathbf{\Delta} - \mathbf{I})\mathbf{f} \tag{115}$$

$$= \alpha_0\mathbf{f} - \alpha_1\mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}\mathbf{f}. \tag{116}$$

By adding a further constrains $\alpha = \alpha_0 = -\alpha_1$, a single parameter filter can be obtained as

$$\tau_\alpha(\mathbf{f}) = \alpha(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}})\mathbf{f} \tag{117}$$

Since the eigenvalues of the matrix $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$ ranges from 0 and 2, applying the filter repeatedly can result in numerical instability. To solve this numerical instability, a renormalization has been applied to the matrix for mapping the eigenvalues into $[0, 1]$, as

$$\tau_\alpha(\mathbf{f}) = \alpha\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{W}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{f} \tag{118}$$

with renormalization $\tilde{\mathbf{W}} = \mathbf{W} + \mathbf{I}$ and $\tilde{\mathbf{D}} = diag(\sum_{j \neq i} \tilde{\omega}_{ij})$. Such construction has been proved to have nice result on simple graphs such as the CORA citation network[53].

Discussion Under this setting, the number of the parameter of the filter per layer is $\mathcal{O}(1)$ which is fixed. The filter is not only spatially decaying but also guarantee compact support, specifically, r-hops support. The Laplacian is an local operator that acts on nearest neighbor around a vertex, thus the r-th power of the Laplacian will affect the r-neighbors which results in the filter localized to r-neighbors in space. Explicit computation of the eigenvectors can be avoid and the resulting complexity is $\mathcal{O}(|\mathcal{E}|r)$ under the assumption that the graph is sparsely connected. Since the Chebyshev polynomial basis is orthonormal, the stability under perturbation is guarantee due to the projection on this basis are stable. For smooth functions, the coefficient decay fast.

A main drawback of the Chebyshev polynomial approximation of the filter is that when approximate a filter having features in a given scale, the number of Chebyshev coefficients required is proportional to the scale given. If there is a cluster of the eigenvalues around one frequency, the filter used to separates the eigenvalues need to have enough features in scale proportional to the radius of the eigenvalue cluster, which might results in very high degrees of polynomial[41].

3.3.4 Cayley Net

An alternative of using polynomials to represent filter is the Cayley polynomials[41]. The Cayley polynomials of order r is defined as

$$g_{\mathbf{c}}(\lambda) = c_0 + 2Re\left(\sum_{j=1}^{r} c_j(\lambda - i)^j(\lambda + i)^{-j}\right) \tag{119}$$

which is a real-valued function with complex coefficients. The $\mathbf{c} = (c_0, \ldots, c_r)$ is a vector containing one real coefficient and r complex coefficients, and the i is the complex unit. This polynomial makes use of the Cayley transform $C(\lambda) = \frac{\lambda - i}{\lambda + i}$ which is a smooth bijection from real number domain $\mathbb{R}$ to the complex unit circle with 1 removed $e^{i\mathbb{R}} \backslash \{1\}$. Since this polynomial works on the unit circle, we have $z^{-1} = \bar{z}$ for $z \in e^{i\mathbb{R}}$ and $2Re\{z\} = z + \bar{z}$. The polynomial can be rewritten as a conjugate-even Laurent polynomial with respect to $C(\lambda)$

$$g_{\mathbf{c}}(\lambda) = c_0 + \sum_{j=1}^{r} c_j C^j(\lambda) + \bar{c_j} C^{-j}(\lambda). \tag{120}$$

Each $C(\lambda)$ is a number on complex unit half circle(due to only non-negative frequencies), which basically is an element of the classical Fourier basis, expressed as $C(\lambda) = e^{ij\omega_\lambda}$ with basis frequency ω_λ, and the integer multiple of the frequency j. Hence the polynomial

(120) is essentially a trigonometric polynomial expressed in $sin(\cdot)$ and $cos(\cdot)$

$$g_{\mathbf{c}}(\lambda) = c_0 + \sum_{j=1}^{r} c_j e^{ij\omega_\lambda} + \bar{c_j} e^{-ij\omega_\lambda} \tag{121}$$

$$= c_0 + 2\sum_{j=1}^{r} Re\{c_j\} cos(j\omega_\lambda) - Im\{c_j\} sin(j\omega_\lambda) \tag{122}$$

where the trigonometric polynomial is generally well behave in function representation. To apply the transfer function to the Laplacian, a scaled factor h is utilized for scaling Laplacian $h\mathbf{\Delta}$, which usually called spectral zoom and could achieve non-linear transformation of the eigenvalues. The spectrum of $\mathbf{\Delta}$ is dilated by the multiplication between $\mathbf{\Delta}$ and h. By applying the Cayley transform

$$C(h\mathbf{\Delta}) = (h\mathbf{\Delta} - i\mathbf{I})(h\mathbf{\Delta} + i\mathbf{I})^{-1} \tag{123}$$

the non-negative spectrum is mapped to the complex half-circle. With larger value of h, more the spectrum of $h\mathbf{\Delta}$ is spread apart in $\mathbb{R}_+$, which achieves better spacing of the smaller eigenvalues of $C(h\mathbf{\Delta})$, while smaller value of h results in better spread apart the high frequencies(see figure 18). Thus h is also a learnt parameter to find a suitable value

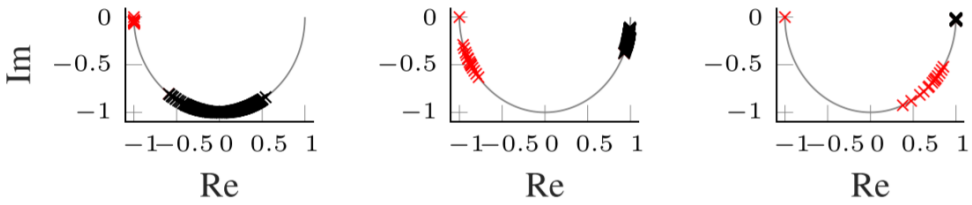

Figure 18: Cayley transform $C(h\mathbf{\Delta})$ for h=0.1,1,10(from left to right) of the 15-communities graph Laplacian spectrum[41].

to 'zoom' in to specific parts of the spectrum as required.

Convolutional Layer Applying the Cayley filter on a function $\mathbf{f}$ is denoted as

$$g_{\mathbf{c},h}(\mathbf{\Delta})\mathbf{f} = c_0\mathbf{f} + 2Re\left(\sum_{j=1}^{r} c_j (h\mathbf{\Delta} - i\mathbf{I})^j (h\mathbf{\Delta} + i\mathbf{I})^{-j}\mathbf{f}\right) \tag{124}$$

which performed in a sequential manner. Denotes the solution of the recurrsive system

$$\mathbf{y}_0 = \mathbf{f} \quad (h\mathbf{\Delta} + i\mathbf{I})\mathbf{y}_j = (h\mathbf{\Delta} - i\mathbf{I})\mathbf{y}_{j-1} \quad j = 1, \ldots, r \tag{125}$$

as $\mathbf{y}_o, \ldots, \mathbf{y}_r$, to avoid the matrix inversion that required $\mathcal{O}(n^3)$ computational cost, the Jacobi method is utilized for approximating the solution $\tilde{\mathbf{y}}_j \approx \mathbf{y}_j$. The Jacobi iteration matrix associated with equation (125) is denote as $\mathbf{J} = -(Diag(h\mathbf{\Delta} + i\mathbf{I}))^{-1} off(h\mathbf{\Delta} + i\mathbf{I})$ while for unnormalized Laplacian $\mathbf{J} = (h\mathbf{D} + i\mathbf{I})^{-1} h\mathbf{W}$. For a given j, the Jacobi iteration

approximation is of the form

$$\tilde{\mathbf{y}}_j^{(k+1)} = \mathbf{J}\tilde{\mathbf{y}}_j^{(k)} + \mathbf{b}_j \tag{126}$$

$$\mathbf{b}_j = (Diag(h\boldsymbol{\Delta} + i\mathbf{I}))^{-1}(h\boldsymbol{\Delta} - i\mathbf{I})\tilde{\mathbf{y}}_{j-1} \tag{127}$$

with $\tilde{\mathbf{y}}_j^{(0)}$ initialized as $\mathbf{b}_j$, the iteration terminated after K iteration and obtain $\tilde{\mathbf{y}}_j = \tilde{\mathbf{y}}_j^{(K)}$. Denotes $\tilde{\mathbf{y}}_0 = \mathbf{y}_0$, applying the approximate Cayley filter is of the form $\widetilde{\mathbf{Gf}} = c_0\tilde{\mathbf{y}}_0 + 2Re\sum_{j=1}^{r} c_j\tilde{\mathbf{y}}_j \approx \mathbf{Gf}$.

Pooling Layer The pooling layer of the Cayley Net still utilize the graph coarsening technique for multiscale graphz transfermation.

Discussion The Cayley filter has the same advantages as in Chebyshev case, which includes $\mathcal{O}(1)$ parameters per layer and exponential spatial decay of the filter. The computational complexity is $\mathcal{O}(nr)$ owe to the Jacobi approximation. The big advantage is that the zoom property allows the filter to better localized in frequency. And because of the over completeness of the trigonometric polynomial form, richer family of the filters can be obtained compared to Chebyshev polynomial with the same order.

3.4 Spatial Convolution Operations

An alternative to perform convolution on non-Euclidean domain is the spatial convolution, which is a class of approaches that does not rely on the spectral information of the given geometric data. In Euclidean case, when performing the spatial convolution for an image, a patch operator is utilized to extract local feature for a small partition of the image each time. Due to shift-invariance, the patch can be passed at each point of the image and record the correlation between the patch and the subset of the image. To generalize this concept to non-Euclidean domain, notions of patch is needed. Define a local system of coordinates on non-Euclidean domain, specifically 2-dimensional manifold, as a local bijective map $\varsigma_x : B_{\rho_0}(x) \to [0,1]^2$ from a metric ball around a point x to a unit square. Such construction of local coordinate is position dependent. The patch operator $\mathcal{D} : L^2(\mathcal{X}) \to L^2([0,1]^2)$ is a map from functions lives on the entire manifold to the system of coordinates, denoted by

$$(\mathcal{D}(x)f)(\mathbf{u}) = (f \circ \varsigma_x^{-1})(\mathbf{u}) \tag{128}$$

where $\mathbf{u}$ is the vector of local system of coordinates. Due to the change of the local coordinates, the patch operator is also position dependent, thus have different mapping at different position(see figure 19).

The spatial convolution on manifold of function $f \in L^2(\mathcal{X})$ with continuous filter $g(\mathbf{u}) \in L^2([0,1]^2)$ is defined as

$$(f \star g)(x) = \int_{[0,1]^2} g(\mathbf{u})(\mathcal{D}(x)f)(\mathbf{u})d\mathbf{u} \tag{129}$$

which apply the patch operator $\mathcal{D}(x)$ to extract local patch from f and multiplied with

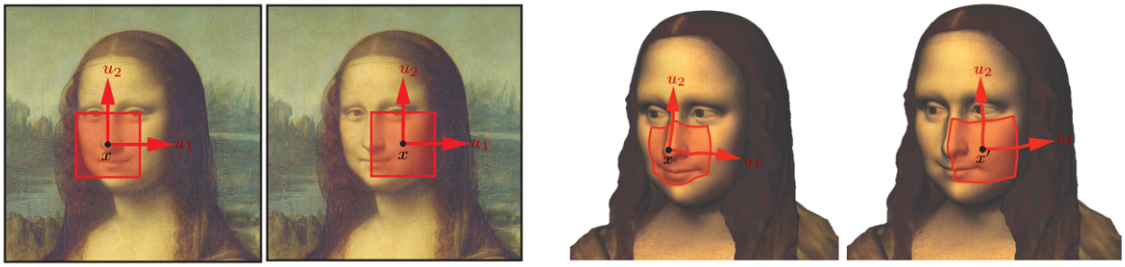

Figure 19: Patch operator moving on image and manifold.

the filter g, depends on the point x. In the discrete case, the convolution is denoted as

$$(f \star g)(x) = \sum_{j=1}^{J} g_j (\mathcal{D}(x) f)_j \tag{130}$$

where $g = (g_1, \ldots, g_J)$ is the discrete filter. In particular, when functions are defined on graph(i.e. under finite setting), the spatial convolution can be written in matrix-vector notation as

$$\mathbf{f} \star \mathbf{g} = \mathbf{g}^\top (\mathcal{D}\mathbf{f}) \tag{131}$$

where $\mathcal{D}\mathbf{f}$ is an $n \times J$ matrix that collects patches at each point in each rows. Several approaches to construct the patch operator will be introduced in the following subsections, including their pros and cons.

3.4.1 Geodesic CNN

The Geodesic CNN is proposed by Masci *et al*[32] in 2015, which making use of the low-dimensional tangent spaces of the manifold, associated with each point. The geodesic polar coordinate is constructed within a small radius area around a point x on the manifold, denoted by

$$\varsigma_x : B_{\rho_0}(x) \to [0, \rho_0] \times [0, 2\pi) \tag{132}$$

where ρ_o is the radius of the geodesic disc. The geodesic distances is measured by $\rho(x, x') = d_{\mathcal{X}}(x, x')$ between the point x and points x' around it, which forms the radial coordinate. And the angular coordinate is constructed by shooting geodesics $\Gamma(x, \theta)$ from x in direction θ. The radius should be sufficiently small to guarantee the local geodesic disc structure, empirically $\rho_0 \approx 0.01$.

Patch Operator The construction of the patch operator in Geodesic CNN can be regarded as a local weighting

$$(\mathcal{D}(x) f)(\rho, \theta) = \int_{\mathcal{X}} \omega_\rho(x, x') \omega_\theta(x, x') f(x') dx' \tag{133}$$

where $\omega_\rho(x, x') \propto e^{-(d_{\mathcal{X}(x,x')} - \rho)^2 / \rho_p^2}$ are weights localized in certain radius and $\omega_\theta(x, x') \propto e^{-d^2_{\mathcal{X}}(\Gamma_\theta(x), x')/\rho_\theta^2}$ are weights localized around certain directions. Example of the Geodesic

patch operator is shown in figure 21.

Convolutional Layer The geodesic convolution is denoted as

$$(f \star g)(x) = \int_0^{\rho_0} \int_0^{2\pi} (\mathcal{D}(x)f)(\rho, \theta) g(\theta + \Delta\theta, \rho) d\rho d\theta \tag{134}$$

where $g(\theta, \rho)$ is the filter applied on the patch. To remedy the angular ambiguity, a term $\Delta\theta$ is added to represent an arbitrary angle that the filter can be rotated. Assume the input of the convolutional layer is a p-dimentional function $\mathbf{f} = (f_1(x), \ldots, f_p(x))$ applies on point x, the output of the convolutional layer is obtained by

$$g_{\Delta\theta,l}(x) = \xi\left(\sum_{l'=1}^{p} (f_{l'} \star \omega_{\Delta\theta,l,l'})(x) \right) \tag{135}$$

$$= \xi\left(\sum_{l'=1}^{p} \int_0^{\rho_0} \int_0^{2\pi} w_{l,l'}(\rho, \theta + \Delta\theta)(\mathcal{D}(x) f_{l'})(\rho, \theta) d\rho d\theta \right) \tag{136}$$

where $l = 1, \ldots, q$ and $l' = 1, \ldots, p$. The $\Delta\theta = \frac{2\pi}{N_\theta}, \ldots, 2\pi$ indicates the number of times the patch is rotated and the $\mathbf{w}_{l,l'} = (w_{l,l',1}, \ldots, w_{l,l',J})$ is the collection of spatial filter coefficients.

Pooling Layer When stacking many of the convolutional layers together, the complexity will explode exponentially. To avoid this situation, the angular max pooling operation is utilized

$$g_l(x) = \max_{\Delta\theta} g_{\Delta\theta,l}(x) \tag{137}$$

which takes the maximum value among the patch rotated output.

Discussion The directional filters can be explicitly represented in such constructions, and they are spatially localized. The number of parameter per layer is fixed as $\mathcal{O}(1)$ that depends on the number of the weighting functions and independent on the input size. Since all the operations are local, the computational complexity is $\mathcal{O}(n)$. But the angular max pooling may reduced the discriminativity of the patches i.e. richness of the filters.

3.4.2 Anisotropic CNN

The Anisotropic CNN[10] was proposed in 2016 by Boscaini *et al.* which construct the patch operator using the anisotropic heat kernels. The anisotropic heat kernels comes from the anisotropic diffusion

$$f_t(x) = -div(\mathbf{D}(x)\nabla f(x)) \tag{138}$$

that the heat conductivity properties are not only position dependent but also direction dependent(figure **??** left). The tensor $\mathbf{D}(x)$ describes such conduction properties that scales locally the gradient vector(tangent vector) $\nabla f(x)$. By applying anisotropic tensor

of the form $\mathbf{D}_{\alpha\theta}(x) = \mathbf{R}_\theta \begin{pmatrix} \alpha & \\ & 1 \end{pmatrix} \mathbf{R}_\theta^\top$, the gradient vector is rotated by angle θ with respect to certain reference direction, scaled by α, and rotate back(figure **??** right). Such construction defines the anisotropic Laplacian

$$\Delta_{\alpha\theta} f(x) = -div(\mathbf{D}_{\alpha\theta}(x)\nabla f(x)) \tag{139}$$

where parameter α controls how much the elogation of the scaling and parameter θ controls the orientation with respect to the max curvature direction. The eigenvectors and eigenvalues of the anisotropic Laplacian are denoted as $\{\phi_{\alpha\theta i}, \lambda_{\alpha\theta i}\}_{i\geq 0}$, while the anisotropic heat kernel[17] is expressed as

$$h_{\alpha\theta t}(x, x') = \sum_{k\geq 0} e^{-t\lambda_{\alpha\theta k}} \phi_{\alpha\theta k}(x)\phi_{\alpha\theta k}(x'). \tag{140}$$

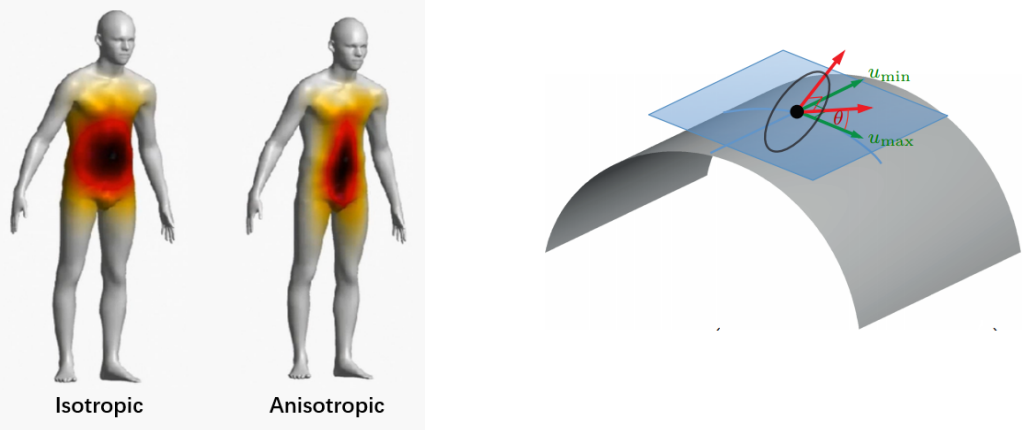

Figure 20: Left: Anisotropic diffusion compared to isotropic diffusion which the diffusion depends on the curvature of the manifold. Right: Local tangent space on a 2D manifold rotated by reference angle θ to compute the anisotropic Laplacian[6]

Convolutional Layer Construct the patch operator by using the anisotropic heat kernel as weighting function and apply such patch operator on functions on manifold is expressed as

$$(\mathcal{D}(x)f)(\theta, t) =< f, h_{\alpha\theta t}(x, \cdot) >_{L^2(\mathcal{X})} \tag{141}$$

For a discrete set of angle constant θ_j, scales constant α_j and anisotropic constant t_j, $j = 1, \ldots, J$.

$$\mathcal{D}(x)f = (< f, h_{\alpha_1,\theta_1,t_1}(x, \cdot) >_{L^2(\mathcal{X})}, \ldots, < f, h_{\alpha_J,\theta_J,t_J}(x, \cdot) >_{L^2(\mathcal{X})})^\top \tag{142}$$

Example of the anisotropic patch operator is shown in figure 21. The convolutional layer expressed in the spatial domain is of the form

$$g_l(x) = \xi\left(\sum_{l'=1}^{p} \mathbf{w}_{l,l'}^{\top} \mathcal{D}(x) f\right) \tag{143}$$

with $l = 1, \ldots, q$, $l' = 1, \ldots, p$, and $\mathbf{w}_{l,l'} = (w_{l,l',1}, \ldots, w_{l,l',J})$ is the collection of spatial filter coefficients.

Discussion Same as the geodesic convolution, such construction has obtained directional filters which is spatially localized. And the number of parameter is $\mathcal{O}(1)$ per layer. The drawback of such approach is the explicit computation of eigendecomposition of the Laplacian for obtaining the isotropic heat kernel, which is quite expensive since the $\mathcal{O}(n^2)$ complexity has to multiplied by the number of orientations.

3.4.3 Mixture Model Network

The Mixture model network[43] proposed by Monti et al. utilize learnable patch operator to construct the convolution operator. Compared to the previous approaches using fixed patches, the patch operator is constructed by studying a family of functions represented as a mixture of Gaussian kernels.

Learnable Patch operator Given a local system of coordinates $\mathbf{u}(x, x')$ around point x, denote the parametric weighting functions express in this system of coordinates as $\mathbf{w}_{\boldsymbol{\Theta}}(\mathbf{u})$, the parametric patch operator is constructed by applying J such weighting functions

$$(\mathcal{D}_{\boldsymbol{\Theta}_1,\ldots,\boldsymbol{\Theta}_J}(x) f)_j = < f, w_{\boldsymbol{\Theta}_1}(\mathbf{u}(x,\cdot)) >_{L^2(\mathcal{X})} \quad j = 1, \ldots, J \tag{144}$$

Utilizing the Gaussian functions $\mathbf{w}_{\mu,\boldsymbol{\Sigma}}(\mathbf{u}) = e^{-\frac{1}{2}(\mathbf{u}-\mu)^{\top}\boldsymbol{\Sigma}^{-1}(\mathbf{u}-\mu)}$ as the parametric functions, the parametric patch operator will become

$$(\mathcal{D}_{\mu_1,\boldsymbol{\Sigma}_1,\ldots,\mu_J,\boldsymbol{\Sigma}_J}(x) f)_j = < f, w_{\mu_j,\boldsymbol{\Sigma}_j}(\mathbf{u}(x,\cdot)) >_{L^2(\mathcal{X})} \quad j = 1, \ldots, J \tag{145}$$

where $\mu, \boldsymbol{\Sigma}$ are the parameters of the Gaussian. Example of the learnable patch operator is shown in figure 21.

Convolutional Layer The spatial convolution on a point x is denoted as

$$(f \star g)(x) = \sum_{j=1}^{J} g_j \int_{\mathcal{X}} w_{\mu_j,\boldsymbol{\Sigma}_j}(\mathbf{u}(x,x')) f(x') dx' \tag{146}$$

by exchanging the order of integration and summation, it can be seen from the formula that the function f is averaged with some local Gaussian mixture function and then

integrated on the spatial domain.

$$(f \star g)(x) = \int_{\mathcal{X}} \underbrace{\sum_{j=1}^{J} g_j w_{\mu_\mathbf{j}, \mathbf{\Sigma_j}}(\mathbf{u}(x, x'))}_{Gaussian \quad mixture \quad g(\mathbf{u}(\mathbf{x},\mathbf{x}'))} f(x')dx' \tag{147}$$

Hence the convolutional layer expressed in the spatial domain utilizing the parametric patch operator $\mathcal{D}_{\mathbf{\Theta}}(x)f$ and the non-linearity ξ is of the form

$$g_l(x) = \xi\left(\sum_{l'=1}^{p} \mathbf{w}_{l,l'}^{\top} \mathcal{D}_{\mathbf{\Theta}}(x) f\right) \tag{148}$$

In such construction both the local patch operator parameter $\mathbf{\Gamma} = (\mu_\mathbf{1}, \mathbf{\Sigma_1}, \ldots, \mu_\mathbf{J}, \mathbf{\Sigma_J})$ and the spatial coefficients $\mathbf{w}_{l,l'} = (w_{l,l',1}, \ldots, w_{l,l',J})$ are learnable.

Discussion In addition to the spatially-localized properties, the learnable patch operator of the mixture model network gives additional degrees of freedom for processing sufficiently complex tasks. The weighting function could have more complex choices including additional non-linear transformation of the pseudo-coordinates $\mathbf{u}$ or some network-in-a-network architectures[48]. The number of parameter per layer is $\mathcal{O}(1)$.

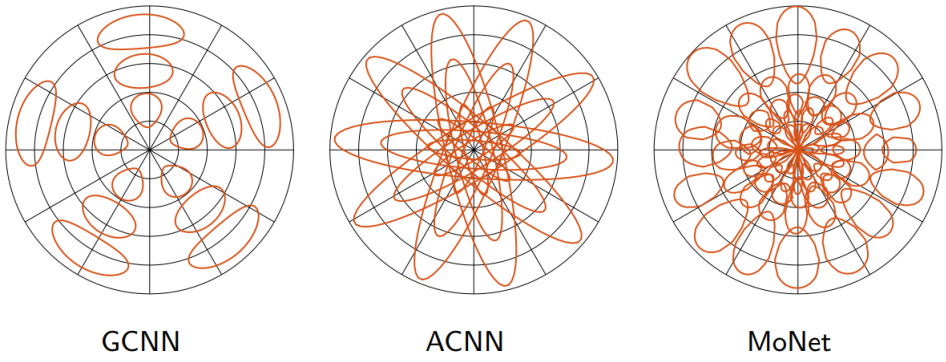

Figure 21: Representation of the local weighting functions for Geodesic CNN(left), Anisotropic CNN(middle), and Mixture model network(right)[43].

3.5 Geometric Deep Learning with 3D face analysis

Affected by factors such as sex, age, ethnicity, human faces are highly distinct in shape. The research area of capturing, reconstruction and tracking 3D faces have raised huge interest in the past decade due to its various applications[61]. Without utilizing geometric deep learning technique, the existing state-of-the-art 3D face analysis model mostly rely on linear transformation or higher-order tensor generalization[55], which rarely capture the non-linear deformation caused by exptreme facial expressions. In the following subsections, three state-of-the-art approaches for 3D face analysis making use of geometric deep learning technique has been introduced, which produced considerable result and being part of the baseline of the model develop in this thesis.

3.5.1 Convolutional Mesh Autoencoder

The Convolutional Mesh Autoencoder(CoMA)[55] is proposed by Ranjan *et al* in 2018 which learns 3D representation of human faces. The model is constructed by an autoencoder structure consisting of fast localized convolutional filters and mesh downsampling and upsampling layers(see figure 22). 3D faces are represented by meshes with 5023 vertices and each vertex has 3-dimensional information.

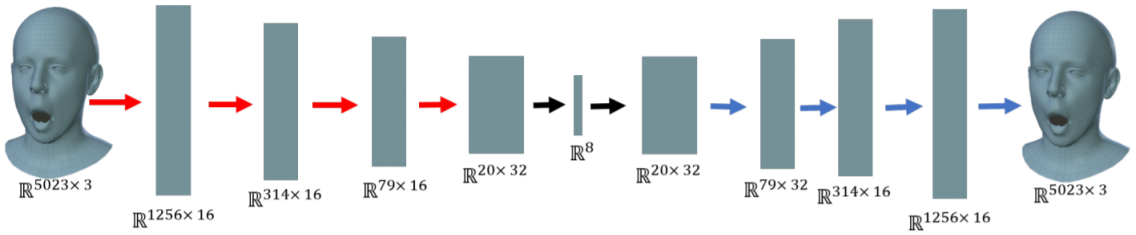

Figure 22: Structure of the Convolutional Mesh Autoencoder. The red arrow represents the down-sampling operation and the blue arrow represents the up-sampling operation of the given mesh[55].

Fast Localized Convolutional Filter Define a 3D face mesh as a collection of edges $\mathcal{E}$ and vertices $\mathcal{V}$, $\mathcal{F} = (\mathcal{V}, \mathcal{E})$, where $|\mathcal{V}| = n$ denotes the number of vertices. The edge connection is represented by a sparse adjacency matrix $A \in \{0,1\}^{n\times n}$ with $A_{ij} = 1$ denotes connected vertices (i, j) and $A_{ij} = 0$ denotes no connection. Compute the non-normalized graph Laplacian by $\mathbf{L} = \mathbf{D} - \mathbf{A}$ where $\mathbf{D}$ is a diagonal matrix containing the degree of each vertex in the diagonal entries, $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. To avoid explicit computation of eigendecomposition of the graph Laplacian, CoMA construct the convolution kernel g_θ by Chebyshev polynomial of order K

$$g_\theta(\mathbf{L}) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}}) \tag{149}$$

where $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I}_n$ denotes the scaled Laplacian, for mapping the function domain to $[-1, 1]$ to guarantee the orthogonality of the Chebyshev polynomial basis. The θ_k is the learnable Chebyshev coefficients and T_k is the Chebyshev polynomial of order k. With $T_0 = 1, T_1 = \tilde{\mathbf{L}}$, the Chebyshev polynomials can be computed recursively by $T_k(\tilde{\mathbf{L}}) = 2xT_{k-1}(\tilde{\mathbf{L}}) - T_{k-2}(\tilde{\mathbf{L}})$. Under such setting, the convolution apply on input $x_i \in \mathbb{R}^{\mathrm{n}\times \mathrm{F_{in}}}$ is defined as

$$y_j = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(\mathbf{L}) x_i \in \mathbb{R}^{\mathrm{n}} \tag{150}$$

where $F_{in} = 3$ is the dimension of the input feature.

Down-sampling and Up-sampling Layer The down-sampling of the face mesh is performed by the graph coarsening technique introduced in section 3.1.4, which construct a down-sampling matrix Q_d by contracting vertex pairs iteratively. An upsampling matrix

Q_u is constructed at the same time, which the point (p, q) has value $Q_d(p, q) = 1$ in down-sampling matrix would also have $Q_u(q, p) = 1$ in up-sampling matrix. For the point be removed during down-sampling computation, the weights (w_i, w_j, w_k) that project this point to the closest triangle (i, j, k) by $\tilde{v_p} = w_i v_i + w_j v_j + w_k v_k$ are recorded in the up-sampling matrix by $Q_u(q, i) = w_i$, $Q_u(q, j) = w_j$ and $Q_u(q, k) = w_k$. The up-sampling applying on a give graph is formulate as $\mathcal{V}_u = Q_u \mathcal{V}_d$.

Training The training of the network is conduct by using stochastic gradient descent to minimize the L1 loss between input mesh vertices and the reconstruction mesh vertices. The chebyshev filtering is used with order $K = 6$.

Discussion experiments showed the performance of COMA is better than traditional PCA method in reconstruction tasks. But the requirement of input of CoMA is that the meshes should all share identical topology, thus all face meshes should be aligned before being used. Also, the latent spaces learnt by CoMA mixing the identity and the expression, which can only generate faces for certain identities. Such draw back has been resolved by the later proposed meshGAN[15], which will be introduced in 3.5.2.

3.5.2 MeshGAN

Face generation have various applications including face recognition, emotion prediction and entertainment. In entertainment aspect, the characters in films and digital games can be more realistic with the help of face generation. In face recognition aspect, better quality of synthetic face raise the challenge for the recognition tasks, which would also promote the development of face recognition techniques. Previously the face generation model making use of GAN[28] architecture and produced considerable results with face images, but the generation of 3D faces is still challenging and have difficulties in producing satisfactory result since the previous approach applied the GAN as 3D convolutional architectures to discrete volumetric representations of 3D objects. Also for 3D faces, synthesizing expressions on real faces for different identity still got a lot work to do[60].

Recently, a MeshGAN[15] was proposed which is the first intrinsic GANs architecture operating on 3D meshed directly. The model is trained with 4DFAB database which can generate static 3D face mesh with different expression for different identities. The model ability of generating 3D faces is stronger than the state-of-the-art autoencoder[55] due to more reasonable details are presented and better modeling of the distribution of the faces.

Approach and Architecture The architecture of meshGAN is desiged based on BEGAN[9] where the objective is as follows:

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - k_t \mathcal{L}(G(z_D)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(G(z_G)) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k(\gamma \mathcal{L}(x) - \mathcal{L}(G(z_G))) & \text{for each training step t} \end{cases} \tag{151}$$

The hyper-parameter $k_t \in [0, 1]$ can maintain the balance between loss expectation of generator G and discriminator D, which can control the impact of the fake loss $\mathcal{L}(G(z))$

on discriminator. When updating k_t a learning rate λ_k is utilized. The term γ is a hyper-parameter that can decide the diversity of generated face meshes, which is set to be 0.7 empirically to allow more variations. Other terms are the same definition as traditional GAN network such as $\mathcal{L}(\cdot)$ denotes the loss of discriminator,z_d represents the latent vector of generator, and θ_D, θ_G are the parameters of the discriminator and generator respectively.

The generator and discriminator network are built according to the decoder and encoder network of CoMA(see figure 23), where the chebyshev filtering is also utilized here to perform graph convolution. The latent space size is enlarged to 64 for allowing for more representation powers. And skip connections are applied in the discriminator for encouraging more facial details. One of the difference in data pre-processing for mesh-GAN is that the facial identity is decoupled from every expression mesh, which is done by subtracting the neutral face from every expression to obtain the facial deformation.

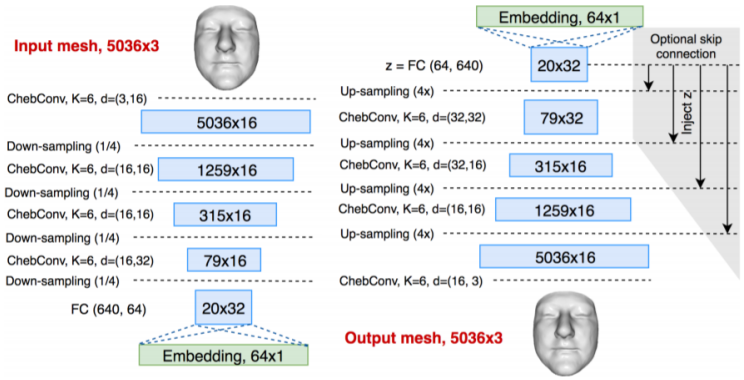

Figure 23: MeshGAN architecture[15].

Discussion Experiments have shown that though the reconstruction performance of meshGAN is a bit lower than CoMA, the synthesised faces produced by meshGAN is more realistic then those produced by CoMA, with lower quantitative errors in both identity and expression aspects. Also the meshGAN can generate more diversity faces than CoMA. The decoupling of facial identity and expression results in pure expression can be learnt by the generator instead of mix of expression and identity.

3.5.3 Joint Texture & Shape Convolutional Mesh Decoders

The above two models are both focus on the shape variation of human face meshes, while in 2019 a non-linear 3d Morphable Model(3DMM)[79] proposed by Zhou *et al* is able to learn not only shape variations but also facial texture. The convolutional operation of this model is also performed by Chebyshev filtering and the down-sampling up-sampling operation is also done by iteratively contracting vertex.

Approach and Architecture The architecture of the non-linear 3DMM is shown in figure 24, where a coloured mesh auto-encoder and a non-linear 3DMM in-the-wild are trained jointly. The coloured mesh auto-encoder has the similar architecture as CoMA[55] with an embedding size 256×1, which is trained with controlled face mesh data. The

non-linear 3DMM shares the same mesh decoder with the coloured mesh auto-encoder and is trained with large scale face images with landmarks. The embedding vector $\mathbf{f_{SA}}$ of the auto-encoder represents the 3D shape and texture regression produced by the image encoder $E_I(I;\theta_I)$ in the non-linear 3DMM model, where the I is the original image. The mesh decoder $\mathcal{D}(\mathbf{f_{SA}};\theta_D)$ will decode this embedding representation and the resulting coloured mesh will be projected onto the image plane $\hat{\mathbf{I}}$ by a differentiable renderer $\hat{\mathbf{I}} = \mathcal{P}(\mathcal{D}(\mathbf{f_{SA}});\mathbf{m})$ with rendering parameter $\mathbf{m} = [\mathbf{c}^\top, \mathbf{l}^\top]$, where $\mathbf{c}^\top$ denotes the perspective transformation parameters and $\mathbf{l}^\top$ denotes the illumination parameters. The loss of the jointly training is defined as the sum of loss between reconstruction meshes $\mathcal{D}(\mathbf{f_{SA}};\theta_D)$ and original meshes as well as loss between produced renderered images $\hat{\mathbf{I}}$ and the original images $\mathbf{I}$.

$$\text{Loss}= L_{\text{reconstruction}} + \lambda L_{\text{render}} \tag{152}$$

Where λ is a hyperparameter being increased gradually during training.

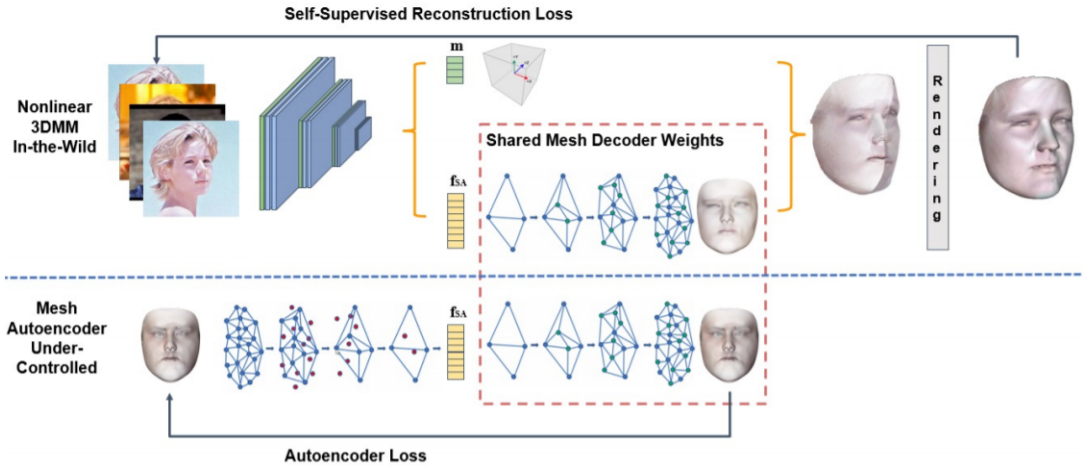

Figure 24: Architecture of the non-linear 3DMM. [79].

Discussion Experiments have shown that the proposed non-linear 3DMM has better performance in 3D face alignment tasks compared to the state-of-the-art methods 3DDFA[80] and N-3DMM[42], with more compact model size and more efficient computational time, though due to the complexity of model the performance is slightly worse than PRNet[76]. For 3D face reconstruction tasks, the non-linear 3DMM is able to produce high-quality reconstruction even the texture information is hairy, and the output results are more smooth due to directly modeling the shape and texture of vertices. It is the first non-linear 3DMM using directly mesh convolutions.

In next section, we will present the model developed in this thesis which also tackling 3D face analysis task utilizing geometric deep learning techniques introduced in this section.

4 3D Facial Motion Classification

Over the past few years, various works has been conducted to study face recognition. Most of the work involves 2D images or 2D videos which are typically affected by the illumination variation as well as the limited vision of the camera that can only capture part of the face deformation. To address this problem, 3D face recognition has raise heavy interest, which use 3-dimensional geometric of the human face to avoid ambiguity caused by lighting, make up, and face orientaion.

In this thesis, we seek to apply geometric deep learning techniques to facial expression recognition tasks, which not only study the static 3D facial deformation but also the dynamic motion of the whole expression. The reason to study the whole dynamic motion is that watching the whole process of expression should capture emotional information better than just looking at the momentary expression, as this is the case when people communicate in real world. The study is based on the 4DFAB database[59]. We start by defining the problem to show the tasks we performed and the dataset we used. Then we introduce the approach we use to deal with this problem. Finally, we present the experiment conducted to indicate the progress being made until a good result has been achieved. To our best knowledge, we are the first to build a multi-output 3D dynamic facial expression recognition system.

4.1 Problem Definition

Given sequences of 3D face meshes from 4DFAB database with each sequence represent the whole facial motion of a person performing one of six facial expressions. Our tasks is to classify which expression category that the sequence from person not shown in the training set belongs to, as well as classifying what is the emotional state of each frame. The emotional state contains 4 categories: neutral, onset apex, offset, which represent the evolution of expression from the start to the end(detailed definition for emotional state will be provided in section 4.2.2). We use data from Session1 of 4DFAB dataset(the 4DFAB dataset contains 4 sessions recording facial motion performed by same participants spanning a period of 5 years),which consists of facial expressions performance of 175 participants. Each participant provides 4 to 6 basic expressions, including anger, disgust, fear, happiness, sadness, and surprise(example see figure 25), resulting in a total of 1018 distinct sequences of expressions. The number of expression sequences in each class is roughly balance in both training set and testing set. The model has been divided to two part: In the first part we trained a convolutional mesh autoencoder for extracting low dimensional representation of the face mesh; In the second part we use the produced low dimensional representation of face meshes to perform classification tasks.

4DFAB Database The 4DFAB database formulate the face meshes by using a Python mesh manipulating and visualizing package `menpo`[82] as `menpo.TriMesh`. The meshes were aligned to avoid intra-class variability such as pose and perspective transformation[26]. The main attributes we use include:

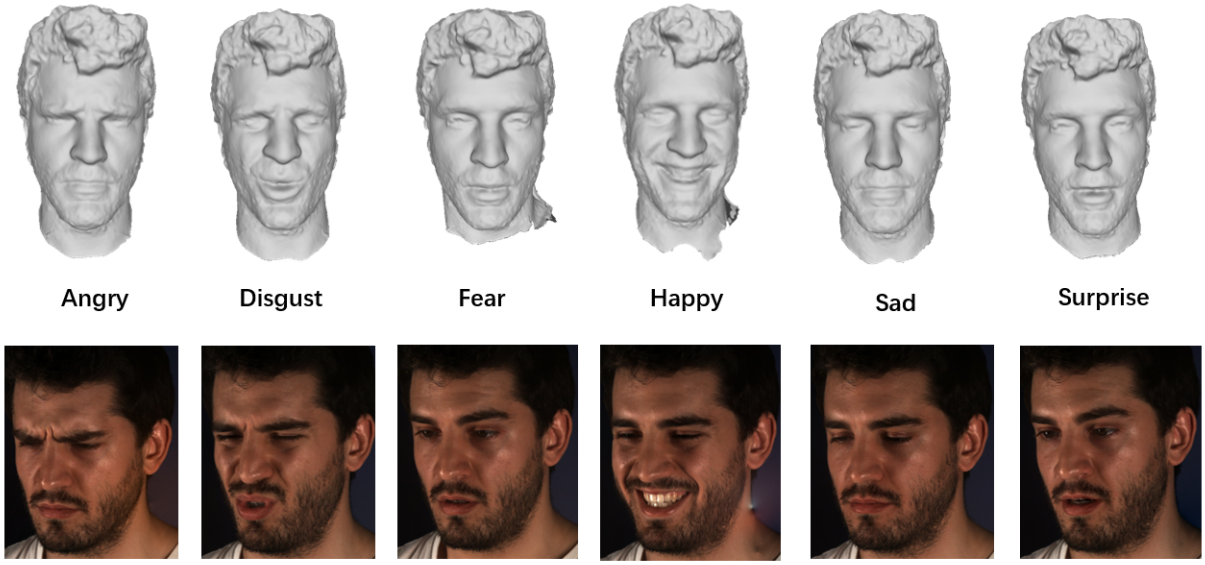

Figure 25: Examples of 6 basic expressions of one participant. The upper row are the visualization of 3D data in 4DFAB database while the data we use are the cropped version which only the face area are included, the hair and neck segments are ignored. The lower row are the image showing the participant's performance during capturing these 3D object.

- `points`. For each mesh this attribute contains 3D Cartesian coordinates with shape $[28431, 3]$, with 28431 denotes the number of vertices.
- `label`. For each mesh a label is set to represent which expression categories it belongs to, the label is range from $[1, 6]$.

For easy manipulation of the mesh, the vertices information were stored in numpy arrays, and the class label were converted to one-hot representation.

4.2 Data Preprocessing

4.2.1 Data Clean

To guarantee the quality of the training data as the corrupted data would affect the model's performance, we went through the whole dataset and visualize each of the mesh by `matplotlib qt` toolkit. We excluded these corrupted meshes(example see figure 26) from training data and results in 24,147 face meshes left in total. For sequences of face mesh, we also excluded the sequence that contains corrupted meshes or misses some frames of the face motion as this may affect the integrity of the temporal information of facial expression, which results in 956 sequences of facial expressions remained in total. All the experiment in this thesis are performed with cleaned dataset unless explicit statement.

4.2.2 Data Annotation

As a multi-output supervised learning tasks, two types of label are required to train the model. For the expression category classification output, labels that indicating which expression the sequence represents is needed, which is already provided by 4DFAB database.

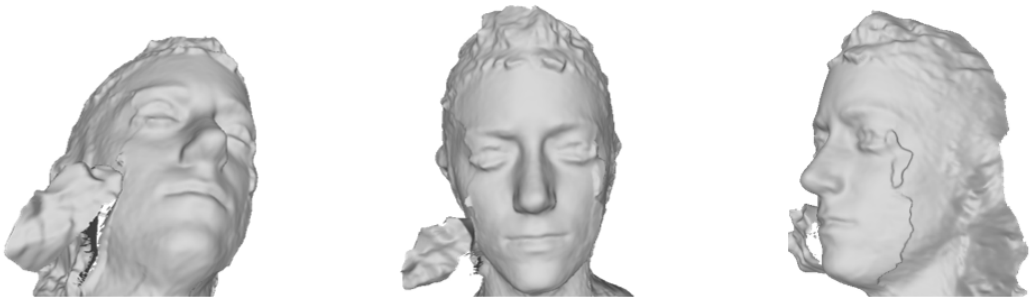

Figure 26: An example of corrupt mesh in 4DFAB database(viewed from three different orientation).

For the emotional state classification output, labels that indicating the state of each frame is needed, which not yet provided by 4DFAB database. Thus manually annotation of each frame of the sequence for emotional state is required to be performed. We define the emotional state as below:

- **Neutral** Neutral is defined as the face status that the person has no expression at all, only blink motion is allowed.
- **Onset** The occurrence of the first visible face deformation action unit will be the start of onset, and the evolution of facial motion before achieving apex are all belongs to onset.
- **Apex** Apex is when facial motion reaches the peak i.e. the highest intensity.
- **Offset** The occurrence of the weaken of the peak facial motion will be the start of offset, and the evolution of facial motion before return to neutral status are all belongs to offset.

Examples of emotional states in sequences are shown in figure 27.

The annotation strategy is as follows: we arrange two people to go through the whole cleaned dataset individually, during which they looked at each frame of the sequence and marked them as one of the 4 states. The result of their annotations will be compared to see whether they agree with each other for the same frame. For those frames receiving different decisions, a third person will have a look at the frame and a decision will be made by the majority vote among the three people. The final decision of the annotation are stored in an Excel file and later be converted to `.npy` file with one-hot representation. A snapshot of the annotation result for several sequences is shown in figure 28.

Figure 27: Examples of emotional states in sequences of 3D face. The occur timing and length of each state may vary.

Identity	Neutral	onset	apex	offset	Neutral
P001_S1_T1_FE	000-011	012-026	027-072	073-089	090-125
P002_S1_T1_FE	000-014	015-032	033-081	082-096	097-151
P003_S1_T1_FE	000-011	012-027	028-053	054-095	096-115
P004_S1_T1_FE	000-004	005-020	021-057	058-082	083-176
P005_S1_T1_FE	000-013	014-029	030-137	138-157	158-166
P006_S1_T1_FE	000-022	023-045	046-050	051-077	078-141
P007_S1_T1_FE	000-024	025-036	037-093	094-103	104-132
P008_S1_T1_FE	000-005	006-023	024-077	078-098	099-154
P009_S1_T1_FE	000-009	010-031	032-133	134-152	153-161
P010_S1_T1_FE	000-012	013-030	031-133	134-149	150-167
P011_S1_T1_FE	000-015	016-037	038-106	107-139	140-188
P012_S1_T1_FE	000-002	003-014	015-070	070-084	085-086
P013_S1_T1_FE	000-018	019-057	058-114	115-136	137-166
P014_S1_T1_FE	000-000	001-015	016-085	086-151	152-192
P015_S1_T1_FE	000-008	009-021	022-118	119-177	178-219
P016_S1_T1_FE	000-005	006-022	023-094	095-109	110-162
P017_S1_T1_FE	000-014	015-044	045-054	055-137	138-177
P018_S1_T1_FE	000-018	019-041	042-112	113-127	128-157

Figure 28: Examples of annotation result of the 4DFAB database session1 data for several sequences. The *identity* contains the participant number(P001), the session(S1), and the expression category(FE indicates "fear"). The three-digit number in the states column indicate the frames number that belong to certain state.

4.3 Approach

In this section, we present the detail of our approach as well as the final architecture(see figure 29) that produce the current state-of-the-art result on this problem based on our experiments. We report the exploration to reach the final structure step by step. The model is constructed by two part: Feature extraction model and classification model. The feature extraction model making use of the graph convolution and graph coarsening to obtain low-dimensional representation of the face mesh, while the classification model utilize the sequence of low-dimensional representation as input to perform classification tasks.

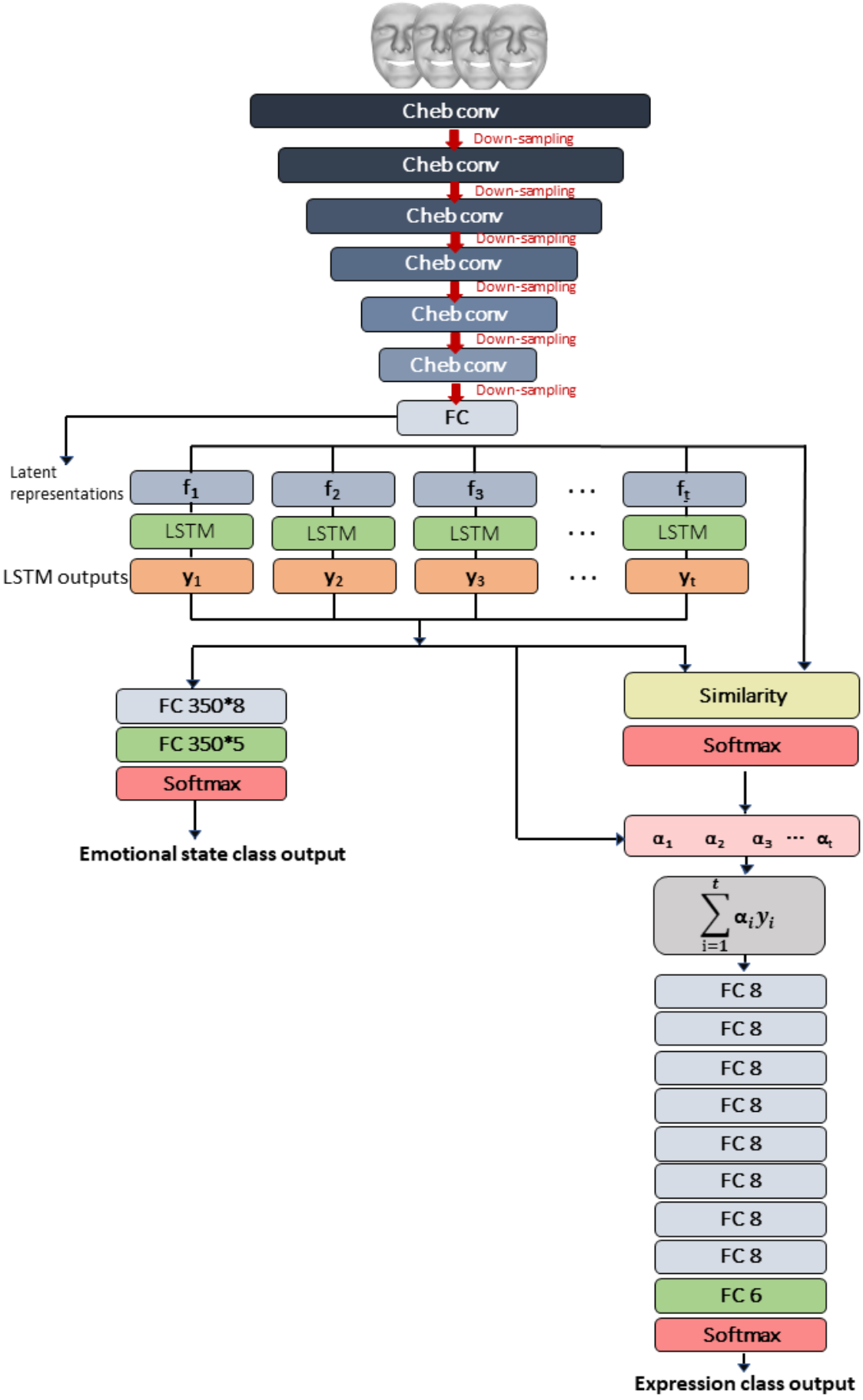

Figure 29: Final architecture.

4.3.1 Notations

The face mesh data in this thesis are formulated as signals defined on undirected connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the finite set of vertices is represented by $\mathcal{V}$ with $|\mathcal{V}| = n$. The $\mathcal{E}$ denotes the collection of edges. Functions on graphs are defined as $\mathbf{f}$, and the graph Laplacian computed by the graph is denoted as $\mathbf{\Delta} = \mathbf{D} - \mathbf{W}$ with degree matrix $\mathbf{D}$ and adjacency matrix $\mathbf{W}$.

4.3.2 Weight Initialization

We initialize the weight matrix and the bias with random number sampled from truncated normal distribution, that values more than two standard deviations from the mean are discarded and re-drawn. The chosen value of mean is 0 and standard deviation is 0.1.

4.3.3 Chebyshev Convolution

We approximate the convolutional filter by Chebyshev polynomials and construct it by

$$\tau_\alpha(\mathbf{\Delta}) = \sum_{j=0}^{r} \alpha_j T_j(\tilde{\mathbf{\Delta}}) \tag{153}$$

with $\tilde{\mathbf{\Delta}} = 2\lambda_n^{-1}\mathbf{\Delta} - \mathbf{I}$ the scaled Laplacian. The polynomials are constructed recursively by $T_j(\tilde{\mathbf{\Delta}}) = 2\tilde{\mathbf{\Delta}}T_{j-1}(\tilde{\mathbf{\Delta}}) - T_{j-2}(\tilde{\mathbf{\Delta}})$ where $T_0(\tilde{\mathbf{\Delta}}) = \mathbf{I}, T_1(\tilde{\mathbf{\Delta}}) = \tilde{\mathbf{\Delta}}$. The convolutional operation is performed by multiplying the chebyshev polynomials with the function, denoted by $\tau_\alpha(\mathbf{\Delta})\mathbf{f}$.

4.3.4 Graph Coarsening

The graph coarsening is the technique that utilized in the pooling layer. The technique is introduced in section 3.1.4 and how it work in this case has been introduced in section 3.5.1 in the **Down-sampling and Up-sampling layer** part.

4.3.5 Long Short Term Memory

The long-short-term memory is used to extract temporal information from give sequence, in this case the given input to the LSTM network would be the low-dimensional feature extracted by the convolution operation. Denote the sequence of low-dimensional feature as $\mathbf{x} = (x_0, \ldots, x_t)$, where x_t denotes the input at time step t. The new candidate vector $\tilde{C}_t$ for the current time step is computed by

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{154}$$

with weight matrix W_C and bias b_C. The h_{t-1} is the hidden state from previous time step. The forget gate z_f, input gate z_i, output gate z_o are computed by

$$z_f = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{155}$$
$$z_i = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{156}$$
$$z_o = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{157}$$

to control the output cell state C_t and hidden state h_t of the current cell

$$C_t = z_f \circ C_{t-1} + z_i + \widetilde{C}_t \tag{158}$$
$$h_t = z_o \circ tanh(C_t) \tag{159}$$
$$\tag{160}$$

where $\mathbf{h} = (h_0, \ldots, h_t)$ is the matrix collecting hidden states for each time step of the LSTM network.

4.3.6 Batch Normalization

To accelerate the training of deep network by reducing internal covariate shift, the batch normalization is used each time before the activation function be applied[29]. For each batch, denotes the input of the batch normalization layer by $\mathbf{g}$, the mean μ_B and variance σ_B^2 of the batch is calculated by

$$\boldsymbol{\mu}_B = \frac{1}{m}\sum_{i=1}^{m} \mathbf{g}_i \tag{161}$$
$$\boldsymbol{\sigma}_B^2 = \frac{1}{m}\sum_{i=1}^{m} (\mathbf{x}_i - \mu_B)^2 \tag{162}$$

where m is the number of sample in each batch. The normalization of the input is computed by

$$\bar{\mathbf{g}}_i = \frac{\mathbf{g}_i - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}} \cdot \gamma + \beta \tag{163}$$

The parameter γ and β are learned during the training of the network.

4.3.7 Attention Mechanism

It can be seen from figure 28 that the length of the input sequence are vary, and the rate of expression change are highly variable. For arbitrary length input, even the LSTM network might not be able to capture long terms dependencies. In 2016 Bahdanau *et al* found that the performance of the LSTM become worse when the sequence length increases from about 30[8]. To address this problem, the attention mechanism is used, which is able to focus on certain input time step for an input sequence of arbitrary length.

In out model we chose additive similarity of input v and hidden state q to compute

the weight for each time step

$$weight(v, q) = softmax(tanh(W_v v + W_q q + b)) \tag{164}$$

where the matrix W_v, W_q, and vector b are learnable parameters. The attension output is computed by the weighted sum of the input v

$$attention(v, q) = \sum(weight(v, q) \cdot v) \tag{165}$$

4.4 Training

In this section we present the training detail of our models. The training strategy for the feature extraction model and the classification model are distinct and we will discuss it separately.

4.4.1 Loss function

Feature Extraction Since the baseline for the feature extraction model is an autoencoder, the target of such model is to reconstruct from the low-dimensional representation to make it as close as possible to the original input. The loss function in this case is defined as the L1 loss between functions $\mathbf{f}$ on the original input graph $\mathcal{G}$ and functions $\hat{\mathbf{f}}$ on the reconstruction graph $\hat{\mathcal{G}}$

$$loss = |\mathbf{f} - \hat{\mathbf{f}}| \tag{166}$$

Classification For multi-class classification tasks, the cross entropy loss is the general choice. Since the label we use are not sparse, there is no necessity to consider specifying the weight. For predicted value $\tilde{\mathbf{Y}}$ and label $\mathbf{Y}$, the cross entropy is calculated by

$$loss(\tilde{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{N} \sum_{i=1}^{N} l_i(\tilde{y}, y) \tag{167}$$

$$l_i(\tilde{y}, y) = -ylog(\tilde{y}) \tag{168}$$

4.4.2 Optimizer

Feature Extraction The autoencoder uses the stochastic gradient descent with a momentum of 0.9, where the initial learning rate is 8e-3 and the learning rate decay is set to be multiplying with 0.99 every epoch.

Classification The classification model uses the Adam optimizer with initial learning rate 0.0001.

4.4.3 Regularization

To address overfitting problem, which is a common issue occur in deep neural network training, we add L2 regularization to create less complex (parsimonious) model.

$$\text{Cost function} = loss(\tilde{\mathbf{Y}}, \mathbf{Y}) + \lambda \sum_j \theta_j^2 \tag{169}$$

where $\lambda = 0.00001$ is the regularization coefficient and θ_j represents the parameter in the network.

4.5 Experiments

In this section, we will described the exploration and experiments conducted to obtain the final architecture step by step.

4.5.1 Experiment I: Feature Extraction

The first step of our experiment is to train an autoencoder for feature extraction, which obtained low-dimensional representation of the face meshes. We make use of the state-of-the-art convolutional mesh autoencoder(CoMA)[55] proposed by Ranjan *et al* as our base model and modify the structure of it to make it compatible with the mesh data in 4DFAB database, since the original CoMA can only process face meshes with 5023 vertices. We add a convolutional layer to each side of the CoMA, and a down-sampling layer to the encoder as well as an up-sampling layer to the decoder. We enlarge the size of the latent space to $\mathbb{R}^{18}$ in order to improve the capacity of the autoencoder by using more parameters for better representation of the face mesh, since the face mesh contains much more vertices which mean more information. The structure of the modified version of CoMA is shown in figure 30.

Layer	Input Size	Output size
Convolution	28431×3	28431×16
Down-sampling	28431×16	7108×16
Convolution	7108×16	7108×16
Down-sampling	7108×16	1777×16
Convolution	1777×16	1777×16
Down-sampling	1777×16	444×16
Convolution	445×16	445×32
Down-sampling	445×32	112×32
Convolution	112×32	112×32
Down-sampling	112×32	28×32
Fully Connected	28×32	18

Layer	Input Size	Output size
Fully Connected	18	28×32
Up-sampling	28×32	112×32
Convolution	112×32	112×32
Up-sampling	112×32	445×32
Convolution	445×32	445×16
Up-sampling	445×16	1777×16
Convolution	1777×16	1777×16
Up-sampling	1777×16	7108×16
Convolution	7108×16	7108×16
Up-sampling	7108×16	28431×16
Convolution	28431×16	28431×3

Figure 30: The architecture of the modified CoMA which is compatible with 4DFAB face mesh. The table on the left is the structure of the encoder while the table on the right is the structure of the decoder.

In order to train the autoencoder being able to produce valid low-dimensional representation for arbitrary expression, we use all frames in the clean dataset as our training

set, no matter what expression the frame belongs to and what identity it is. We shuffle the training set and split the training set as *train set* and *validation set* with ratio 9:1. When reproducing the original CoMA experiments, for a trained CoMA model that reconstructs 72.6% of the vertices within a Euclidean error of 1 mm, the average validation loss is 2.56e-01. Hence we try to train the modified CoMA to achieve a lower validation loss in order to guarantee the quality of the reconstruction. After training for 32 epochs with the 24,147 meshes, the validation loss achieves 1.93e-01. To check the quality of the autoencoder, we encoded all the meshes to the latent space as $\mathbf{z}$ and computed the mean $\boldsymbol{\mu}_z$ and covariance $\boldsymbol{\Sigma}_z$. We sampled 100 latent representations from the estimated multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$ then decoded and visualized them to see the performance of the autoencoder. Several examples of the sampled mesh has shown in figure 31, from which we can have a general feeling that they are from different identity and expressing distinct emotion, which shows that the autoencoder has achieved a considerable quality.

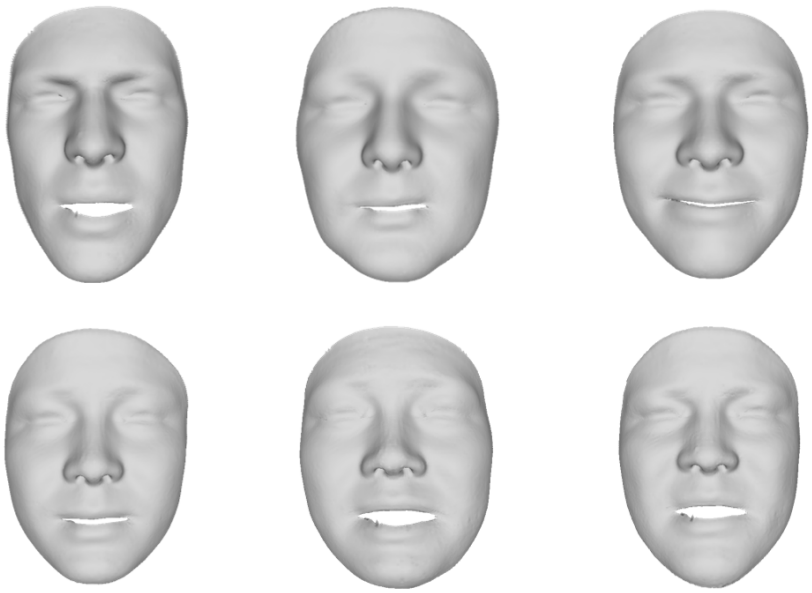

Figure 31: Examples of the face mesh produced by decoding the sampled latent vectors.

Therefore, we will use the encoder of this autoencoder as our feature extraction model to compute low-dimensional representation of the face mesh. To construct the training set for the classification tasks, the order of the face mesh has to be taken care of, which should follow its appearance order. Due to the length of sequences are vary, each sequence are encoded separately from $\mathbf{F} = (F_1, \ldots, F_t) \in \mathbb{R}^{t \times 28431 \times 3}$ into low-dimensional vectors $\mathbf{f} = (f_1, \ldots, f_t) \in \mathbb{R}^{t \times 18}$, where t denotes the number of frames in a sequence and F_i and f_i denote the individual original mesh and individual latent representation respectively, which results in 956 such sequence of latent vectors. The input of the classification is formed by concatenate all sequences of latent vectors together while each of them has been padded with zero vectors to a sequence of length=350 frames and reshaped as $\hat{f}_i \in \mathbb{R}^{1 \times 350 \times 18}$, and the input is of shape $(956, 350, 18)$. The choice of 350 is made by looking through all the lengths of sequence and observed that 91% of them are below 350

while the rest of them has got a length over 350 frames. For those longer sequences, we take the 350 very middle frames and discard $t-350$ frames from the start of the sequence and the end of the sequence.

The label for classification tasks are also constructed at this stage. For both expression classification purpose and emotional state classification purpose, labels are constructed in one-hot representation. Details of the label numbering is presented in table 1 and table 2.

Expression	Label number
Angry	0
Disgust	1
Fear	2
Happy	3
Sad	4
Surprise	5

Table 1: Expression classification label numbering

Emotional state	Label number
Neutral	0
Onset	1
Apex	2
Offset	3
None of them	4

Table 2: Emotional state classification label numbering

Noticing that the emotional state has a class "None of them" which indicates those zero padding frames formed due to concatenate purpose. The shape of the expression class label is $(956, 6)$ and shape of the emotional state label is $(956, 350, 5)$.

4.5.2 Experiment II: Classification Model Architecture Tuning(Expression Network Depth)

From experiment II we are going to present the exploration of the classification model. Inspired by [78], we proposed a model using LSTM working with attention mechanism to process length-varying sequences of data. The initial design of the architecture is shown in figure 32. The input sequence of low-dimensional representation $\mathbf{f} = (f_1, \ldots, f_t) \in \mathbb{R}^{t\times 18}$ will be processed by the one layer LSTM network with hidden size=8, which results in LSTM output $\mathbf{y} = (y_1, \ldots, y_t) \in \mathbb{R}^{t\times 8}$. In this thesis value of t is chosen to be 350. This LSTM output will goes to two different fully connected networks, where one is for emotional state classification purpose(State Network), another is for expression classification purpose(Expression Network). We fix the State network to be a fully connected neural network with 3 hidden layer followed by ReLU activation functions and a softmax function at the output layer, and tune the architecture of the Expression network first.

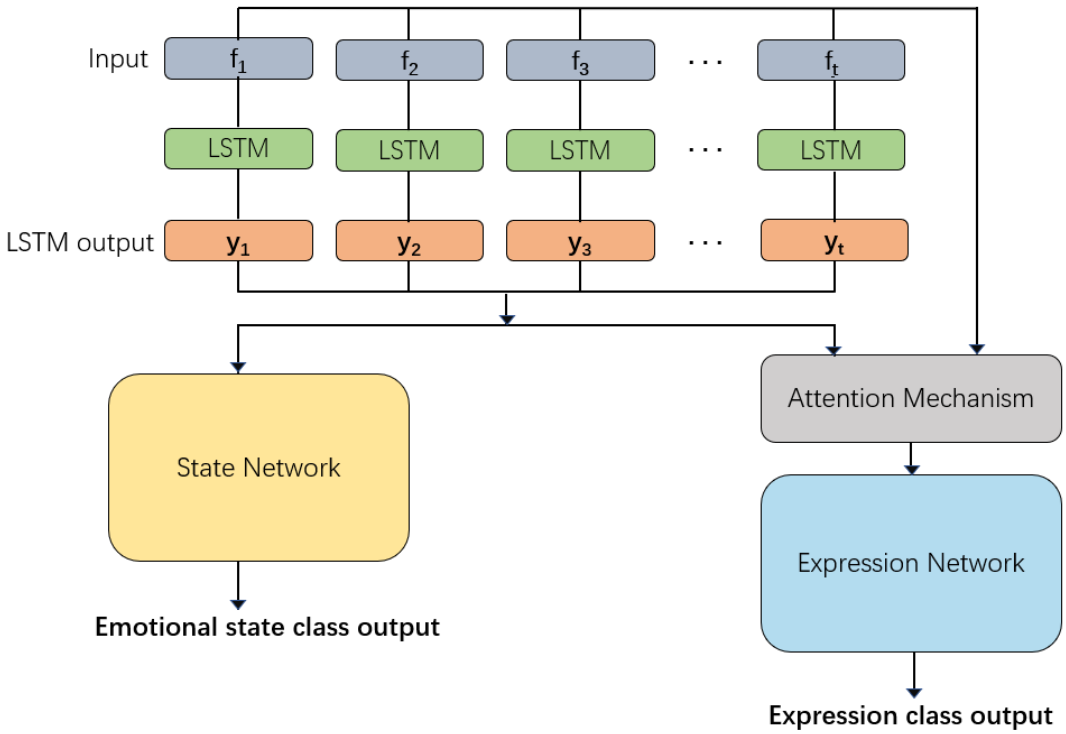

Figure 32: Initial architecture of the classification model.

The attention mechanism is applied before the Expression network, which is a two layer neural network computes weights $(\alpha_1, \ldots, \alpha_t)$ by calculate the similarity between input $\mathbf{f}$ and LSTM output $\mathbf{y}$, where the attention output will be the weighted sum $A = \sum_i \alpha_i f_i \in \mathbb{R}^{1\times \text{attentionsize}}$(see figure 33). The attention size is chosen to be 8 in this thesis, for consistency with the followed fully connected network. We start with a 5 layers fully connected network following the attention mechanism, where each layer is equipped with 8 neurons, and gradually add 2 layers each time to the network until 15 layers to see how they work. Due to equal number of neuron in each layer we call such network "Straight". The attempted fully connected architectures are shown in figure 34.

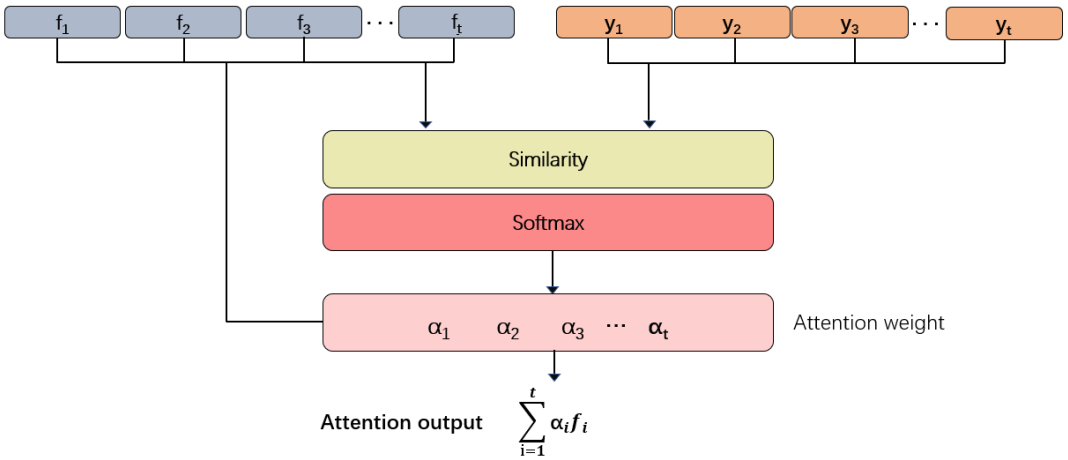

Figure 33: Illustration of the attention mechanism where the output is computed by weighted average of latent vectors.

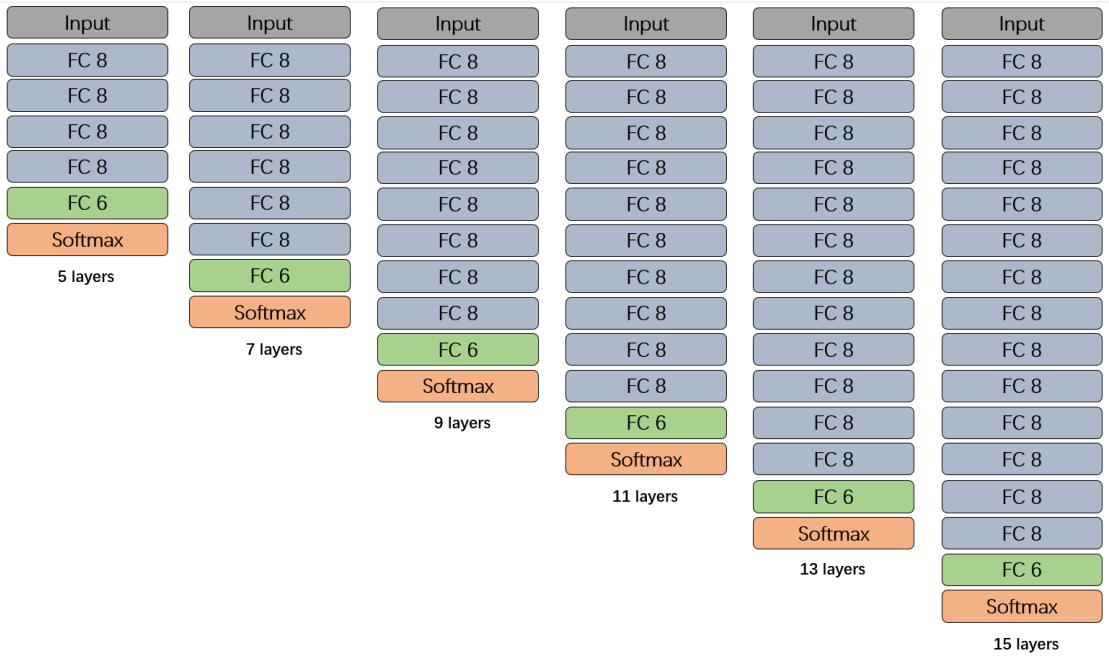

Figure 34: Sub-architecture tried for expression network architecture tuning(depth tuning). The leaky ReLU activation function is applied between layers.

To see the limit of the capacity of the model, we set the training epoch to be 15,000 and decided to stop when an obvious trend occur in the training log thus might not finish the whole 15,000 epochs. We split the dataset into training set and validation set with ratio 9:1 results in 856 sequences in the training set and 101 sequences in the validation set. We plot the loss and accuracy of both training and validation set for observing the behaviour of the network. Results of the plot are shown in figure 36.

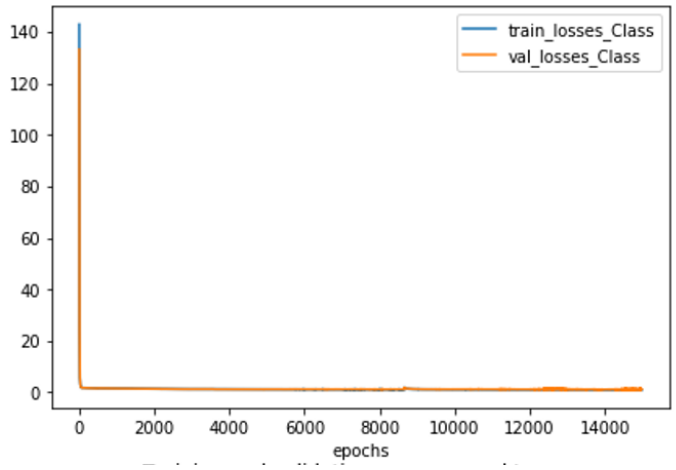

Figure 35: Plots of expression classification loss for training and validation set produced by fully connected networks with 5 layers.

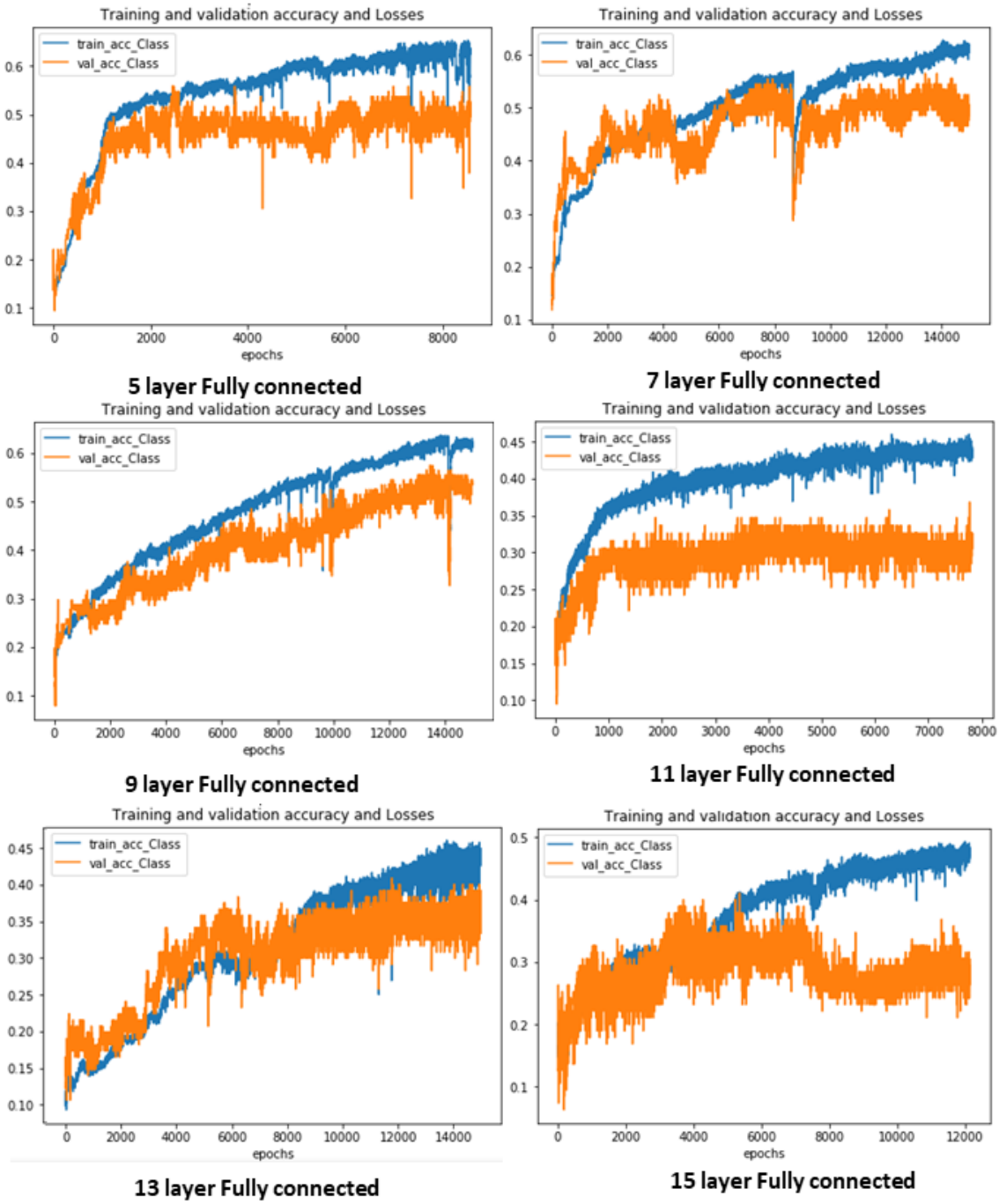

Figure 36: Plots of expression classification accuracy for training and validation set produced by fully connected networks with number of layers=5, 7, 9, 11, 13, 15.

Discussion From plots of the train loss and validation loss(figure 35), we can observe a steep drop of the value of loss at the first several epochs of training, which shows the benefit brought by Adam optimizer, that is robust and well-suited to a wide range of non-convex optimization problems. Due to the scale of the plotting graph, the further progress on loss is not obvious but can be observed through the plot of accuracy. The loss plot for other architecture looks in the same trend. It can be seen from figure 36 that

networks with 5-layers, 7-layers and 9-layers can achieve validation accuracy above 50% within 15000 epochs while the 5-layer can achieve such accuracy the earliest, at around 1300 epochs. It takes about 2000 epochs for 7-layer network achieves such accuracy and about 9800 epochs for 9-layer network to achieve, but the trend of the accuracy of the 9-layers network is still increasing. The accuracy plots of the validation set shows more obvious perturbation than the train set which is due to the size of the validation set is relatively small. But the plot for 7 layer network shows stronger perturbation than the other.

For network with number of layers=11,13,15, the average validation accuracy that stabilised within set epochs is around $30-35\%$ though the 15-layers network has achieved 39% accuracy at around 3900 epochs. The case that more layers results in lower accuracy is due to the optimization difficulty caused by deeper architecture since the train accuracy is increased slowly. The stabilisation of the accuracy is caused by adding regularization to avoid overfitting, which is the case that overfitting is happening due to the capacity of the model is larger than desired. The lack of number of training sequence of data is also the reason that deeper network is hard to train since there are more parameters thus requires more information, while out training set contains only 861 sequences. We present the test accuracy for each architecture in table 3, from which we can observe the accuracy of individual expression category. The expression with highest accuracy has been highlighted while the majority of architectures achieves highest accuracy for "happy" expression, which is the case that this expression is relatively unique and obvious due to its extreme face deformation and less ambiguity compared to other expressions. The lowest accuracy of expression for different architectures are vary.

Hence, we choose number of layers=5,7,9 to proceed our next step tuning experiments.

	5-layers	**7-layers**	**9-layers**	**11-layers**	**13-layers**	**15-layers**
Angry	35%	35%	45%	20%	45%	20%
Disgust	40%	27%	33%	47%	40%	33%
Fear	59%	24%	41%	29%	18%	29%
Happy	**67%**	**89%**	**94%**	**67%**	**78%**	33%
Sad	33%	42%	58%	33%	67	42%
Surprise	58%	79%	47%	16%	53%	**63%**
Overall	49.47%	51.48%	55.44%	29.47%	47.52%	36.84%

Table 3: Expression classification accuracy for each architecture(depth tuning).

4.5.3 Experiment III: Classification Model Architecture Tuning(Expression Network Shape)

In this experiment, we aim at tuning the network architecture by fixing the depth of network but varying the width of each layer. Empirically, the design of the network is tend to be a "cone" shape which the closer to the output layer, the narrower the width of the layer. Such design can naturally drop some useless information while desperately trying to keep as much relevant information as possible as the number of neuron is less

than previous layer. From previous experiments, we have found that depth of 5,7,9 is relatively suitable for this tasks, so we use these depth but increase the width of each layer gradually from the output layer to the input(See figure 37). We plot the accuracy of these architecture on performing classification tasks and compare to those "straight" networks that have the same depths but equal neurons in each layer. Results have been shown in figure 38.

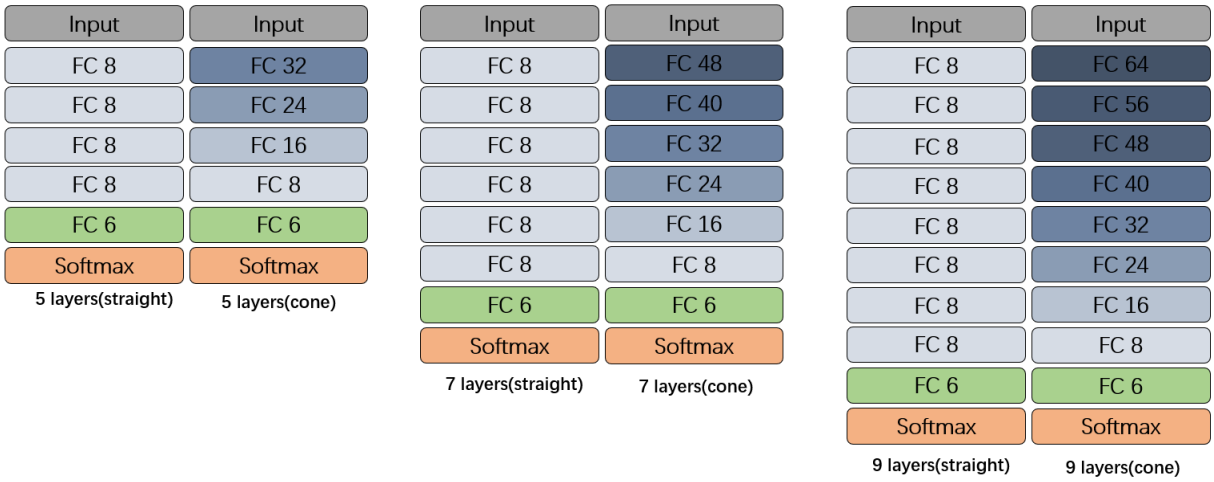

Figure 37: Sub-architecture tried for expression network architecture tuning(shape tuning).

Discussion From the plots we can see that the highest accuracy of both shapes architecture is roughly 50% while in the case when number of layers is 5 and 7 the straight network achieves such accuracy earlier than the cone network. But in the case when number of layers is 9 the cone network achieves 50% accuracy at around 1900 epochs and proceed to overfitting in the later epochs while the straight network reaches such level of accuracy at around 10000 epochs. The trend of the straight networks still shows an increasing behaviour after reaching 50% though not obvious but the trend of the cone networks is stabilized even some slight decreasing. The accuracy of architecture for individual expression is shown in table 4, in which the "happy" expression still earn the highest accuracy in all cases, while the cone shape architectures can also distinguish "sad" expression quite well.

In this case, increasing the width of the layer to form a cone shape does increase the capacity of the network, but the result in both shapes are roughly the same, while cone shape network has more parameters. The performance increase by using cone shape network is not worth the increased cost.

Hence we choose the straight shape network to proceed our next step tuning experiments.

	5-layers (straight)	**5-layers (cone)**	**7-layers (straight)**	**7-layers (cone)**	**9-layers (straight)**	**9-layers (cone)**
Angry	35%	55%	35%	50%	45%	55%
Disgust	40%	27%	27%	40%	33%	20%
Fear	59%	35%	24%	41%	41%	41%
Happy	**67%**	**83%**	**89%**	**67%**	**94%**	**67%**
Sad	33%	58%	42%	**67%**	58%	**67%**
Surprise	58%	47%	79%	37%	47%	37%
Overall	49.47%	51.48%	51.48%	49.50%	55.44%	47.52%

Table 4: Expression classification accuracy for each architecture(shape tuning).

	5-layers (straight)	**5-layers (cone)**	**7-layers (straight)**	**7-layers (cone)**	**9-layers (straight)**	**9-layers (cone)**
Number of parameters	304	1584	432	4912	560	11312

Table 5: Number of parameters of architectures in expression network shape tuning.

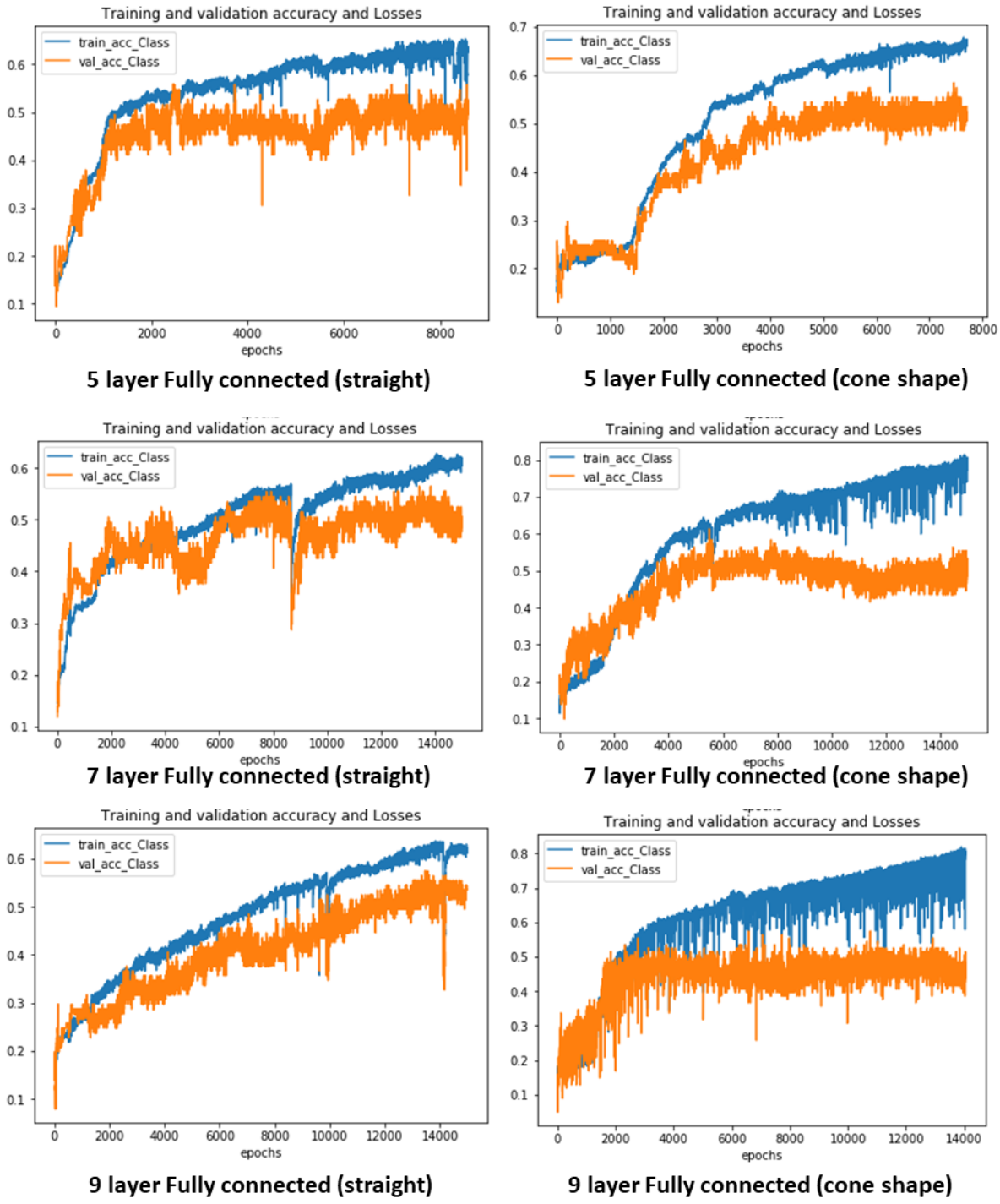

Figure 38: Plots of expression classification accuracy and loss for training and validation set produced by fully connected networks with number of layers=5, 7, 9 but different shapes. The left figure show the plot of the straight shape network while the right show the plot of the cone shape network.

4.5.4 Experiment IV: Classification Model Architecture Tuning(Attention output)

In this experiment, we focus on the computation of the attention output. There are two ways of computing the attention output, where one is to calculate the weighted sum of the input latent vectors by $\sum_{i=1}^{t} \boldsymbol{\alpha}_i \mathbf{f}_i$, another is to calculate the weighted sum of the LSTM output by $\sum_{i=1}^{t} \boldsymbol{\alpha}_i \mathbf{y}_i$(see figure 39). For more convincing measure of performance, we still tried straight network with depth of 5, 7, and 9 with different computation of attention output. Results of the accuracy plot is shown in figure 40 .

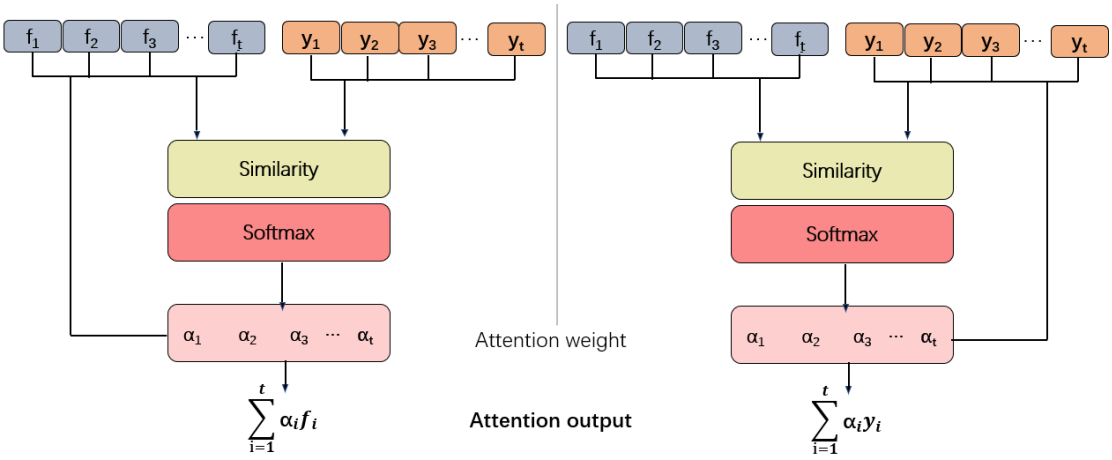

Figure 39: Different choice of attention mechanism output computation.

Discussion For 5 layers straight network, there is no obvious advantage on accuracy when using LSTM output to compute the attention output, while the training takes more epochs to reach the same accuracy as network using latent vectors to compute the attention output. But for 7 and 9 layers straight network, using the LSTM output to compute the attention output has raise the accuracy to 60% level, especially for 9 layers network, that the accuracy is stabilized as around 60% while the 7 layers network's accuracy experienced some perturbation and decrease to 50% level. The time taken to reach such accuracy level is much less than using the latent vector for attention computation, where the 7 layer network takes roughly 3000 epochs and 9 layer network takes about 4000 epochs. The reasion of such situation might caused by the richness in filtered information of the LSTM output, that by passing through the LSTM network all the hidden state has contain useful information from all previous states and discard those useless features, which results in the important feature has been emphasize more times. The table of accuracy shown in table 6 indicates that the architecture using LSTM output to compute attention can achieve accuracy for individual expression at least as high as architectures using input f with equal depth generally.
Hence, we decide to construct the expression network by 9 layers straight network with LSTM output as attention computation.

	5-layers (input f)	**5-layers (LSTM output)**	**7-layers (input f)**	**7-layers (LSTM output)**	**9-layers (input f)**	**9-layers (LSTM output)**
Angry	35%	50%	35%	40%	45%	65%
Disgust	40%	47%	27%	40%	33%	47%
Fear	59%	35%	24%	41%	41%	53%
Happy	**67%**	**78%**	**89%**	**89%**	**94%**	**94%**
Sad	33%	50%	42%	67%	58%	50%
Surprise	58%	74%	79%	74%	47%	58%
Overall	49.47%	56.43%	51.48%	57.42%	55.44%	62.37%

Table 6: Expression classification accuracy for each architecture(Attention output).

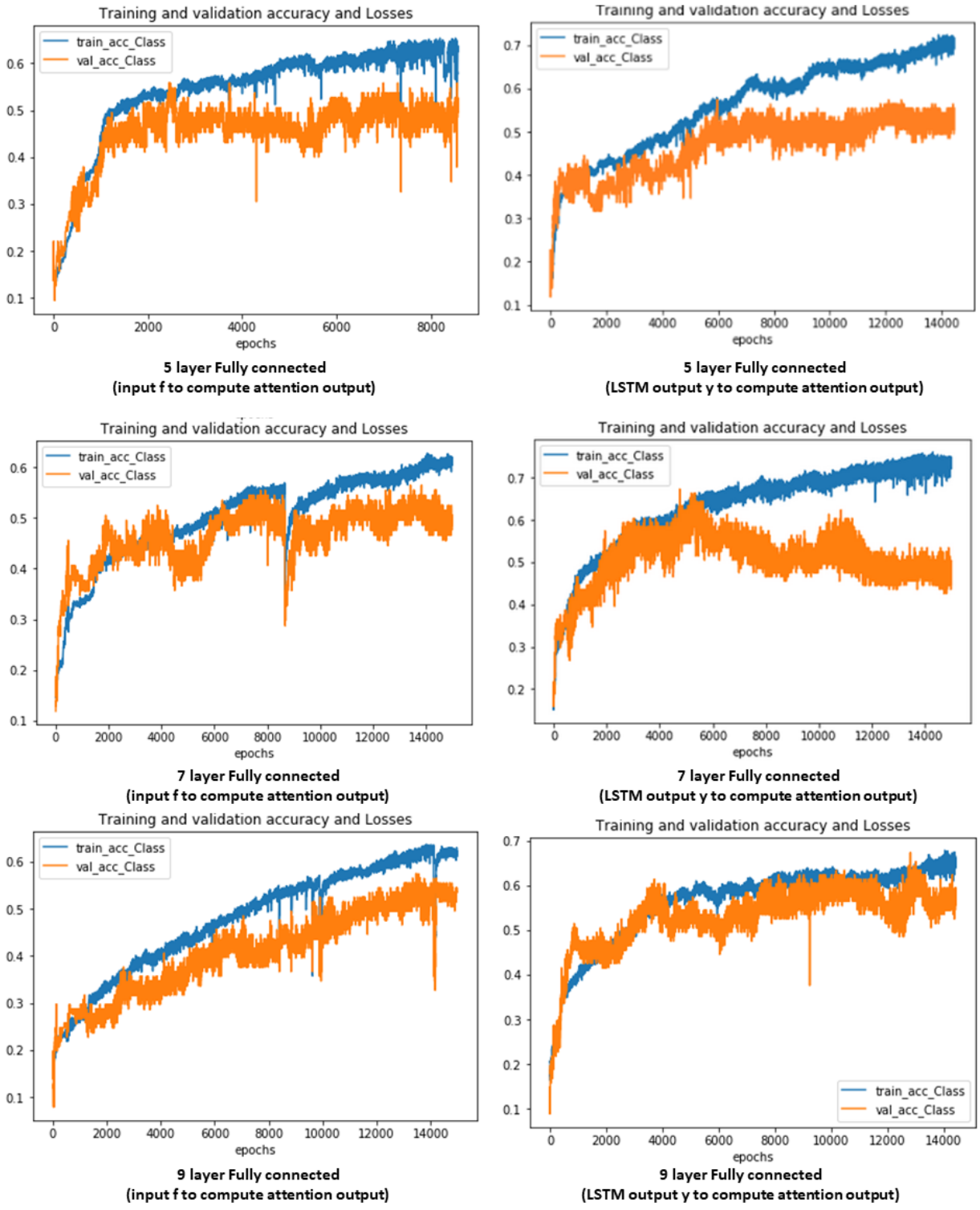

Figure 40: Plots of expression classification accuracy and loss for training and validation set produced by fully connected networks with number of layers=5, 7, 9 but different attension output computation. The left figure show the plot of attention network using input low-dimensional representation $\mathbf{f}$ to compute the attention output. while the right show the plot of attention network using LSTM output $\mathbf{y}$ to compute the attention output.

4.5.5 Experiment V: Classification Model Architecture Tuning(State Network)

In experiment V, we aim at tuning the architecture for the state network, which classify the emotional state for each frame. We use the same tuning strategy as expression network tuning which tune the depth first then the shape of the network. The input of this sub-network is also the LSTM output $\mathbf{y} = (y_1, \ldots, y_t) \in \mathbb{R}^{t \times 8}$, which is reshaped as a tensor with size $(t * 8,)$ where $t = 350$ in this thesis. Then the reshaped input will be pass through a stack of fully connected layer with different depth to see how it work. Before applying the softmax function, the tensor is reshaped to size $(350, 5)$ and thus the softmax function is apply on the last dimension of the tensor to decide which of the 5 classes it belongs to. We start with one layer network which convert the LSTM output to the shape of predicted label straightly, and gradually add one layer with number of neuron $=350 * 8$ each time until five layers. The attempt architecture is shown in figure 41 where ReLU activation function is applied between layers. The resulting accuracy of state classification is shown in figure 43.

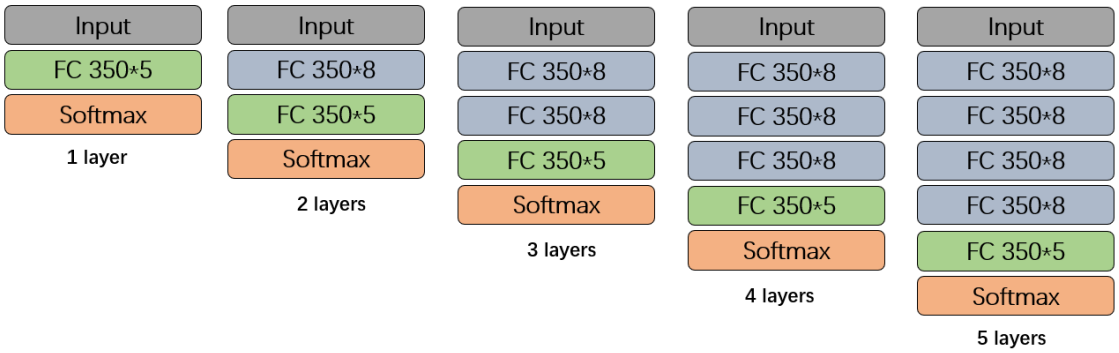

Figure 41: Sub-architecture tried for state network architecture tuning(depth tuning). The ReLU activation function is applied between layers.

Also, to tune the shape of the architecture, we tried cone shape networks(see figure 42) with the same depth as straight networks as in the case of expression classification architecture tuning, and compared the result to the straight network. The accuracy comparison of these networks is shown in figure 44.

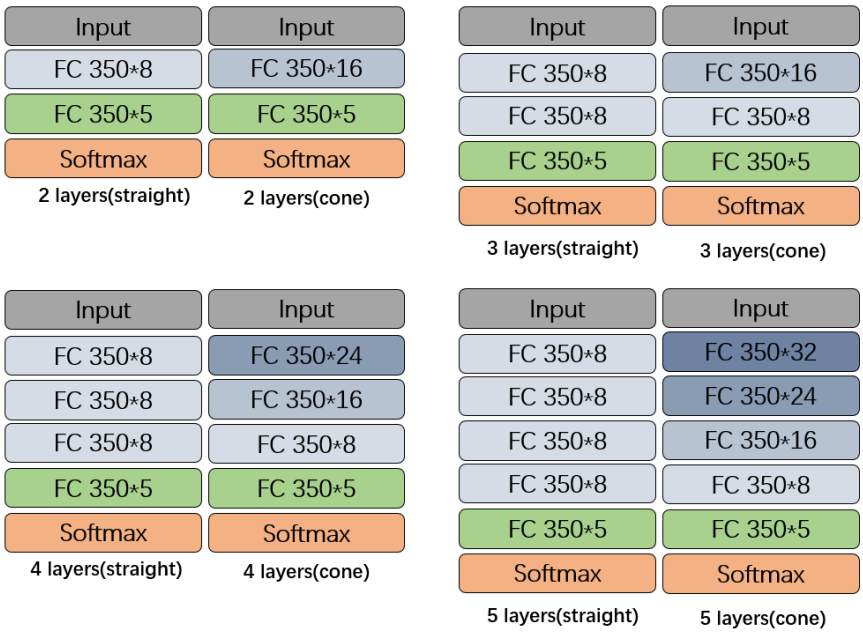

Figure 42: Sub-architecture tried for state network architecture tuning(shape tuning). The ReLU activation function is applied between layers.

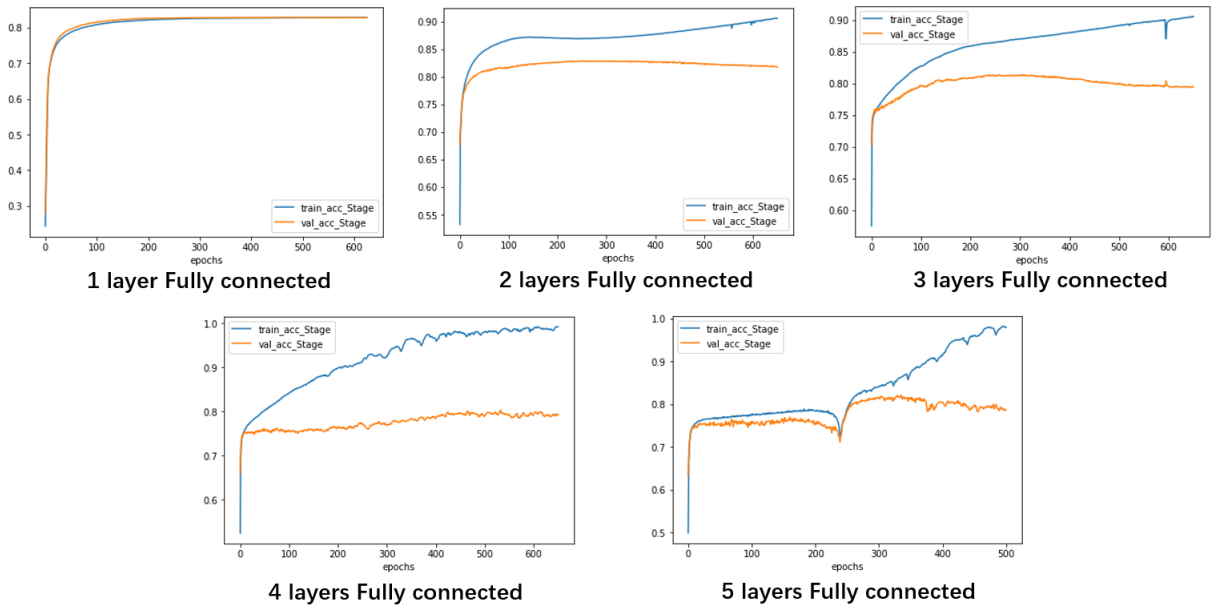

Figure 43: Plots of emotional state classification accuracy for training and validation set produced by fully connected networks with number of layers=1, 2, 3, 4, 5.

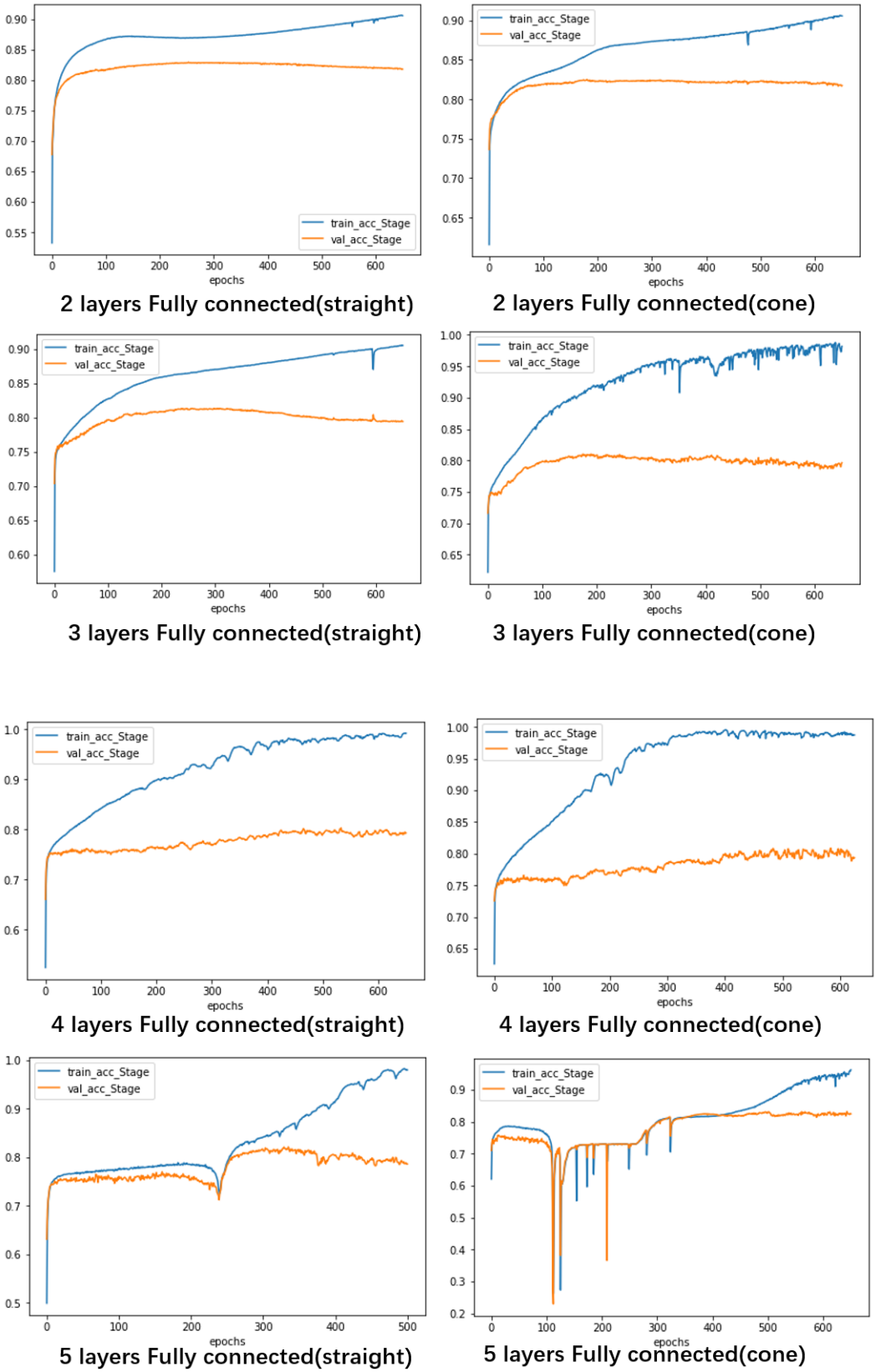

Figure 44: Plots of emotional state classification accuracy for training and validation set produced by fully connected networks with number of layers=2, 3, 4, 5 but different shape.

	1-layer	**2-layers**	**3-layers**	**4-layers**	**5-layers**
Neutral	50%	52%	44%	46%	51%
Onset	65%	62%	58%	62%	59%
Apex	**88%**	**84%**	**80%**	**79%**	**78%**
Offset	39%	41%	45%	45%	50%
None	**100%**	**100%**	**99%**%	**97%**	**98%**
Overall	82.97%	82.02%	79.73%	79.34%	80.17%
Exclude padding	67.77%	66.2%	63.01%	63.49%	64.58%

Table 7: State classification accuracy for each architecture(depth tuning).

	2-layers	**2-layers (cone)**	**3-layers**	**3-layers (cone)**	**4-layers**	**4-layers (cone)**	**5-layers**	**5-layers (cone)**
Neutral	52%	60%	44%	49%	46%	50%	51%	52%
Onset	62%	63%	58%	61%	62%	62%	59%	61%
Apex	**84%**	**76%**	**80%**	**74%**	**79%**	**78%**	**78%**	**83%**
Offset	41%	47%	45%	49%	45%	44%	50%	46%
None	**100%**	**99%**	**99%**%	**99%**	**97%**	**99%**	**98%**	**98%**
Overall	82.02%	81.24%	79.73%	79.59%	79.34%	80.44%	80.17%	81.71%
Exclude padding	66.20%	65.40%	63.01%	62.38%	63.49%	63.63%	64.58%	66.63%

Table 8: State classification accuracy for each architecture(shape tuning).

Discussion In figure 43 a basic trend of the network accuracy is that the majority of them has achieved 80% except for the 4 layers network who remains at 78% level. The one layer network achieves such accuracy fast but the train accuracy also stayed at this level without increasing, which is the case that the network is too shallow thus the capacity of the model is not large enough, i.e. under-fitting. By adding layers to the network, we observed a further increment to the train accuracy but the validation accuracy stabilized at around 80% with slightly oscillation. Also, the training of the state network takes much less epochs to reach a considerable classification quality than the expression network, for number of layers=1,2 the network can achieve 80% within 50 epochs while for 3 layers network it takes about 200 epochs. In table 10 the state classification accuracy of each state for these attempted architectures is presented. Since we define a class named "None" which indicates the padding frames, the overall accuracy also counts this part of the information while it is actually not the most relevant classification detail we need. So we also calculate the accuracy of classification excluding the padding frames and presented in the last row in the table, with heading "Exclude Padding". It can be seen from the table that the state network is good at distinguishing padding frames, which achieved accuracy over 97% in all case we tried, especially for one and two layers network that achieves 100%. Except for the "None" class, the network achieves highest accuracy with the class "Apex", which gain an average accuracy of 81.1% among the five attempted architecture. The classification performance of the "Onset" frames are usually better then the "Offset" frames though they have similar face deformation at individual frame.

When comparing the shape of the network, similar case happened that the validation accuracy does not have obvious increment when using cone shapes architecture. In the case of 2 to 4 layers architectures the cone shape network gained relatively lower accuracy of the classification of "Apex" class but slightly higher accuracy in "Neutral", "Onset", "Offset" classes, while in the 5 layers architecture case the cone shape network "wins" in "Neutral", "Onset", "Apex" classes and gained slightly less accuracy in "Offset" class. Overall, the best three performance is obtained by 1 and 2 layers straight network as well as the 5 layers cone network. We calculate the number of parameter for these three architectures and present in table 9, from which we can see that with a slightly higher accuracy, the 5 layers network required a lot more parameters to be train, which results in more expensive computational cost. Hence, we choose the 2 layers network as out state network architectures, since the 1 layer network shows an underfitting behaviour that we want to equip a bit more capacity to the network.

	1-layer	**2-layers**	**5-layers(cone)**
Number of parameters	4,900,000	12,740,000	193,060,000

Table 9: Number of parameter for state network architectures.

4.5.6 Experiment VII: Final Architecture Testing

In this experiment we aim at training the chosen architectures all together and observe the performance of the final model on both clean and corrupt data set. We take the corrupt sequences of face mesh and labeled them using the same strategy as in 4.2 to construct a test set with corrupt sequences i.e. frames missing sequence or corrupt mesh in some individual frame. We use the same dataset split setting as before, and we decide to train the final architecture with clean dataset and see the performance on both clean test set and corrupt test set, in order to see the ability of the model when processing corrupt data, as in practical application the new coming data required to be processed is usually less ideal than the training data. It can be seen from previous experiments, the quality of the trained model can be determined within 8000 epochs. Thus we decided to train the final architecture for 8000 epochs.

Discussion We plot the expression classification accuracy for training set, clean validation set and corrupt validation set in figure 45, from which we can found that the value of the clean validation accuracy and corrupt validation accuracy is around 50% after 7000 epochs, while the clean validation accuracy is usually slightly higher than the corrupt validation accuracy. The increment of the accuracy for clean dataset is slightly faster than the corrupt validation accuracy as the former achieves 50% accuracy at around 1000 epoch while the latter takes about 2000 epochs to reach the same level. The oscillation of the accuracies might caused by the size of the validation set is too small, since the accuracy of the training set is relatively more stable.

Figure 46 shows the accuracy of training set, clean validation set and corrupt validation set for emotional state classification. The accuracy of clean validation set gets stable at

around 81% while the accuracy of corrupt validation set gets stable at around 74%. The rate of increment of both validation sets are roughly the same as they both achieved their stabilized accuracies after around 800 epochs. Here the calculation of accuracy treat individual frame as a sample instead of the whole sequence as sample, so the size of the validation set is much bigger than the case in expression classification. Thus the resulting accuracy is more stable which further proves the explanation for oscillation in expression classification accuracies.

To explicitly see the performance for each class in both tasks for clean dataset and corrupt dataset, we compute the normalized confusion matrix for the above 4 cases and presented in figure 47-50. For expression classification with clean dataset, the highest accuracy is achieved by class "happy" with value 72% while the lowest accuracie occurred in class "angry" and "fear" with value 40% and 41% respectively. For expression classification with corrupt dataset, the highest accuracy is achieved by class "sad" with value 70% while the lowest occurred in class "fear" with value 15%. When observing the miss-classified situation for both case, the "fear" faces are mostly miss-classified as "angry" face and "surprise" face, which might due to the case that these expression it self is hard to distinguish since they all contain facial deformation such as eyes wide open and mouth open. And the "angry" faces are mostly miss-classified as "disgust" and "sad" faces which might because of they all contains frown and mouth pursed. Such similar trend of face deformation increase the difficulty of classification. For the emotional state classification, the lowest accuracy occurred in "offset" class and those "offset" frames are usually miss-classified as "neutral" and "apex". Interestingly, none of the "offset" frame is miss-classified as "onset" frame though they actually has same trend of deformation, which proves the effect of the LSTM does affect the decision making of the classification since the temporal element is also being considered, that the frame after "apex" frame cannot be the "onset" frame.

	Clean data	**Corrupt data**
Exclude padding	66.46%	58.02%

Table 10: State classification accuracy for final architecture.

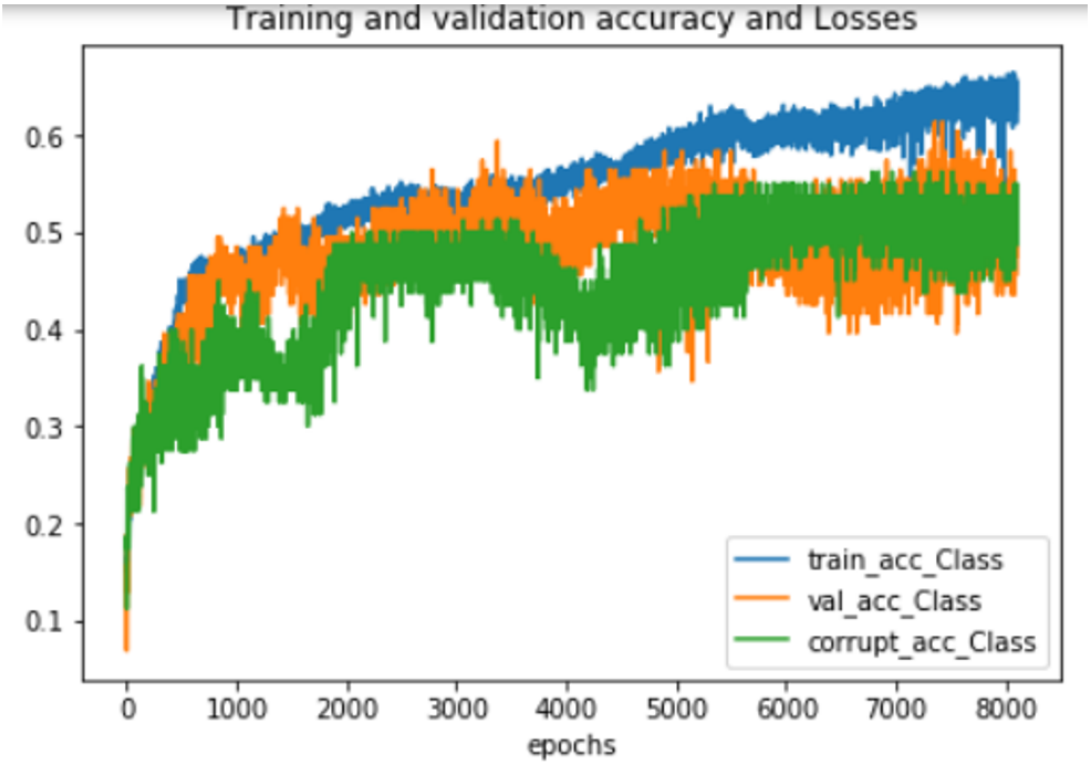

Figure 45: Plots of expression classification accuracy for training and validation set produced by final architecture.

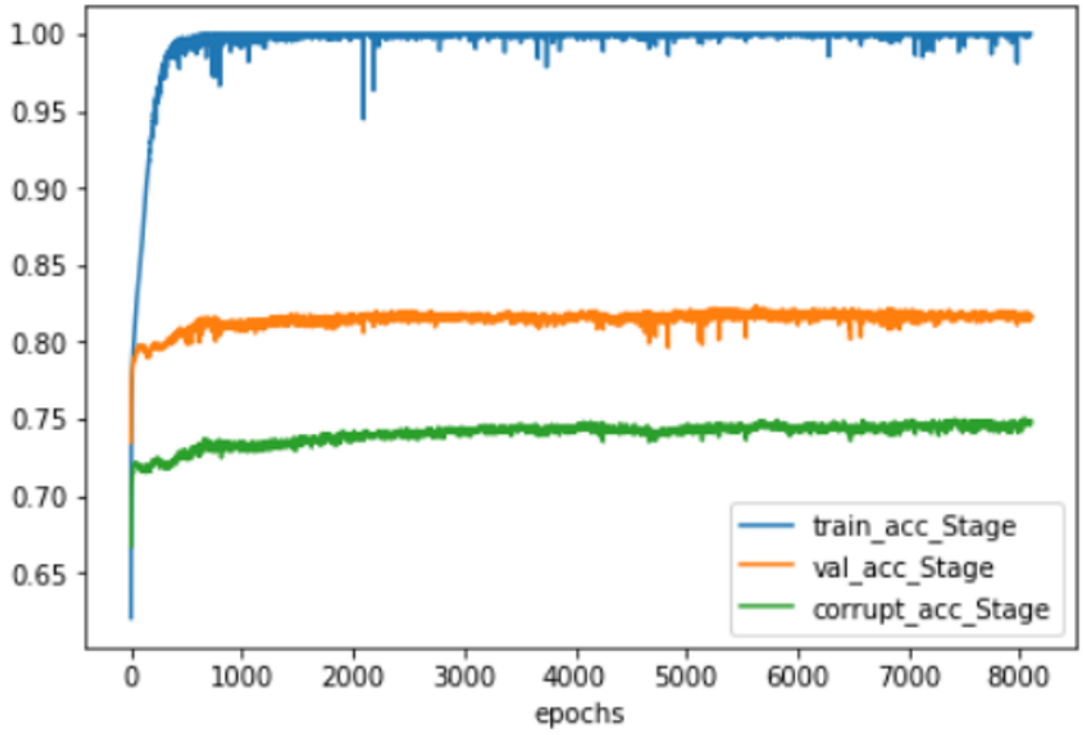

Figure 46: Plots of emotional state classification accuracy for training and validation set produced by final architecture.

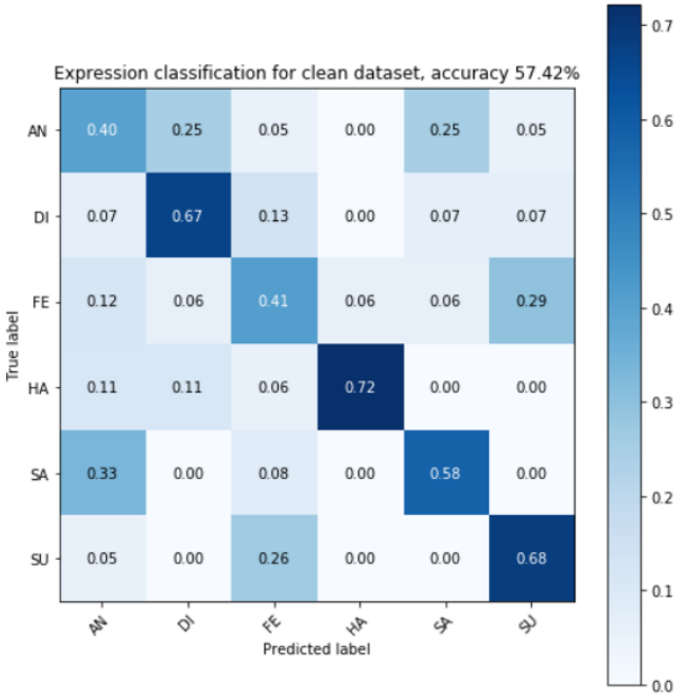

Figure 47: Confusion matrix of expression classification result for clean dataset produced by final architecture.

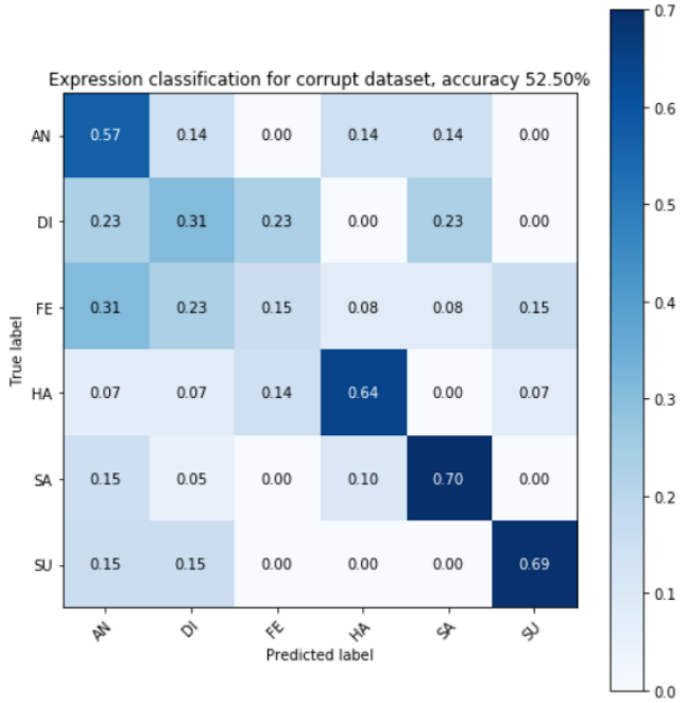

Figure 48: Confusion matrix of expression classification result for corrupt dataset produced by final architecture.

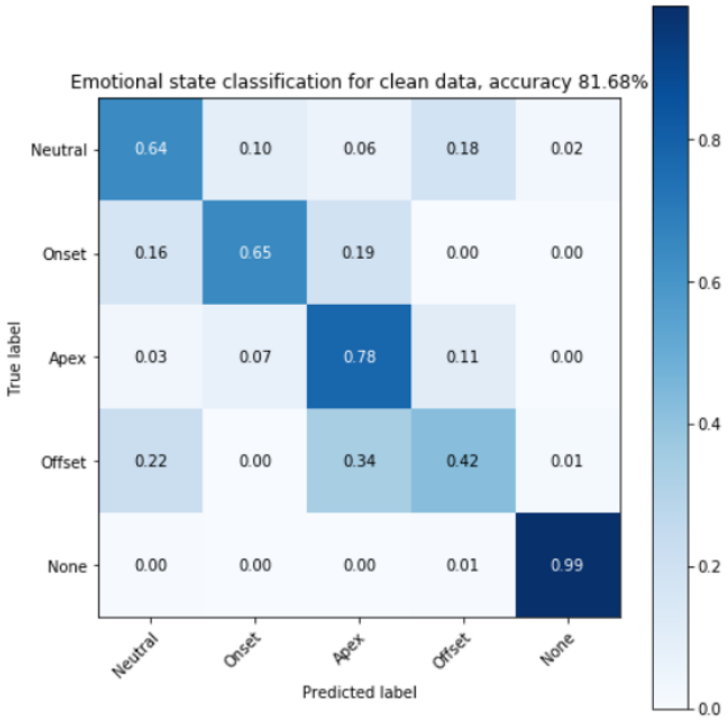

Figure 49: Confusion matrix of emotional state classification result for clean dataset produced by final architecture.

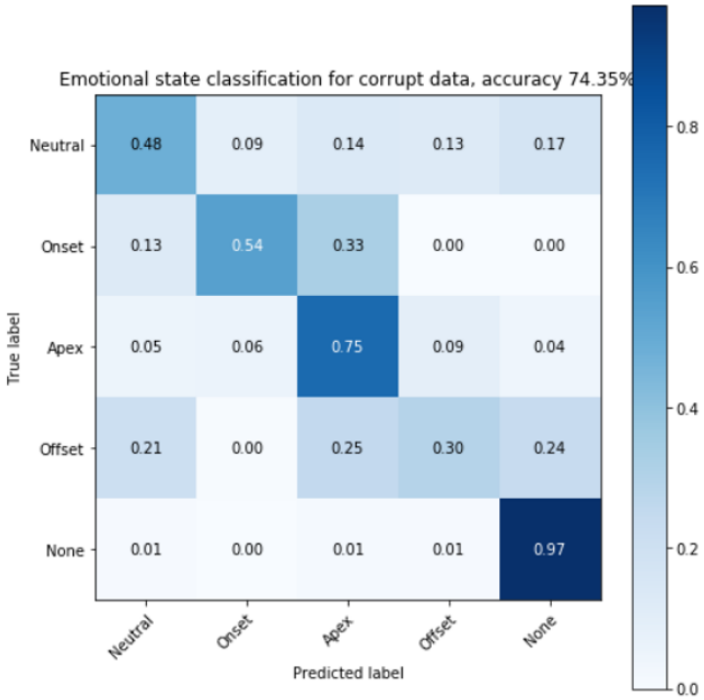

Figure 50: Confusion matrix of emotional state classification result for corrupt dataset produced by final architecture.

4.5.7 Conclusion

In this section, we present the architecture we developed and the exploration to achieve the final architecture. We have found that in practical tasks, for limited number of data, the too deep or too wide neural network architecture may have worse performance than the shallower and narrower design, since the limited amount of information is not enough to train an architecture with large capacity. And for the final architecture we achieved, the expression classification performance does not affected seriously by the corrupt data while the emotional state classification result is affected bit more obvious(57.42% and 52.50% for clean and corrupt data respectively in expression classification tasks, and 81.68% and 74.35% for clean and corrupt data respectively in emotional state classification tasks).

5 Conclusion and Future Direction

Geometric Deep Learning is a recently emerging field in machine learning that generalized deep learning to non-Euclidean domains, where the prototypical objects are graphs and manifolds. Various applications of this field include network analysis, particle physics, computer vision and computer graphics. Owing to the intrinsic property of geometric deep neural network, invariance to shape deformation can be achieved, which have great benefit in 3D object representation learning. As one of the most common 3D object, the human face has raised heavy interest due to its function of individual identification and social communication. The high complexity of deformation of the human face has brought challenges to extreme deformations capturing and non-linear expressions modeling of 3D faces. By utilizing geometric deep learning technique, the shortage caused by using modern deep learning approach can be resolved such as topological structure breaking and fine details losing. In this thesis, we attempt to address an arising challenge in computer vision for decades utilizing geometric deep learning, which is the dynamic 3D facial expression recognition. The dynamic 3D faces are modeled by sequences of manifolds and were discretized as graphs. We hope the methods proposed in this thesis could contribute to applications in other domains.

In this thesis, we started with providing basic deep learning definitions and methodologies on Euclidean domain in chapter 2, which are essential knowledge for understanding advanced designed intuition of the methodology on non-Euclidean domain reviewed in chapter 3, as well as the approach we proposed in chapter 4. In chapter 3, two prototypical types of non-Euclidean domain were introduced and the construction of function analysis on these domains was described. As the vital element in deep learning, convolution operations on non-Euclidean domain including spectral and spatial approaches were presented and their performance was discussed. Three geometric deep learning approaches tackling 3D face analysis tasks were reviewed at the end of chapter 3, which work as part of the inspiration of our proposed approach where the Chebyshev Convolution operation is utilized.

In chapter 4, we designed a multi-output approach utilizing graph convolutional network and LSTM to recognize dynamic 3D facial expressions, where the model not only produce the expression classification of the given dynamic face sequence but also the emotional state of each individual frame. To our best knowledge, we are the first to bring geometric deep learning in dynamic 3D facial expression recognition. The experiments details are provided in this chapter which described each step of our progress being made during the exploration.

Future Direction

- **Efficient graph pooling strategy.** The contracting-vertex-iteratively graph coarsening strategy utilized in this thesis is graph specific i.e. the transform matrices for up-sampling and down-sampling are required to be computed for each mesh individually in train dataset, and needed to be stored during the whole training process. For extreme large-scale data, the memory required and computational costs will be

high and is time-consuming. An efficient graph pooling strategy that can be generalized over different graphs would accelerate the training process as well as reduce the cost.

- **Synthesis dynamic 3D objects.** The arising MeshGAN has produced considerable results in 3D objects generation assisted by the geometric deep learning technique. Exploring 3D generative models that generates dynamic 3D objects would have great benefit in various real world applications involving time-varying domain.

List of Figures

List of Tables

References

[1] A. Makhzani et al, *Adversarial autoencoders,*arXiv Preprint arXiv:1511.05644, 2015.

[2] A. B. L. Larsen et al, *Autoencoding beyond pixels using a learned similarity metric,* arXiv Preprint arXiv:1512.09300, 2015.

[3] A. Ghodrati et al, *Towards automatic image editing: Learning to see another you,* arXiv Preprint arXiv:1511.08446, 2015.

[4] Boureau, Y. L., Ponce, J., LeCun, Y. (2010). *A theoretical analysis of feature pooling in visual recognition.* In Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 111-118).

[5] Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A. and Vandergheynst, P., 2017. *Geometric deep learning: going beyond euclidean data.* IEEE Signal Processing Magazine, 34(4), pp.18-42.

[6] Bronstein M. 2019 *CO460 Lecture: Geometric Deep Learning.*Imperial College London

[7] Boureau, Y. L., Le Roux, N., Bach, F., Ponce, J., LeCun, Y. *Ask the locals: multi-way local pooling for image recognition.*2011.

[8] Bahdanau, D., Cho, K., Bengio, Y. (2014). *Neural machine translation by jointly learning to align and translate.* arXiv preprint arXiv:1409.0473.

[9] Berthelot, D., Schumm, T. and Metz, L., 2017. *Began: Boundary equilibrium generative adversarial networks.* arXiv preprint arXiv:1703.10717.

[10] Boscaini, D., Masci, J., Rodolà, E. and Bronstein, M., 2016. *Learning shape correspondence with anisotropic convolutional neural networks.* In Advances in Neural Information Processing Systems (pp. 3189-3197).

[11] Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y. (2014). *On the properties of neural machine translation: Encoder-decoder approaches.* arXiv preprint arXiv:1409.1259.

[12] Clevert, D. A., Unterthiner, T., Hochreiter, S. (2015). *Fast and accurate deep network learning by exponential linear units (elus).* arXiv preprint arXiv:1511.07289.

[13] Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014). *Empirical evaluation of gated recurrent neural networks on sequence modeling.* arXiv preprint arXiv:1412.3555.

[14] Collobert, R., Weston, J. *A unified architecture for natural language processing: Deep neural networks with multitask learning.* In Proceedings of the 25th international conference on Machine learning (2008, July). (pp. 160-167). ACM.

[15] Cheng, S., Bronstein, M., Zhou, Y., Kotsia, I., Pantic, M. and Zafeiriou, S., 2019. *MeshGAN: Non-linear 3D Morphable Models of Faces.* arXiv preprint arXiv:1903.10384.

[16] D. Lazer et al., *Life in the network: the coming age of computational social science,* Science, vol. 323, no. 5915, p. 721, 2009.

[17] D. Boscaini, J. Masci, E. Rodola, M. M. Bronstein, and D. Cremers. *Anisotropic diffusion descriptors. '* Computer Graphics Forum, 35(2), 2016.

[18] Defferrard, M., Bresson, X. and Vandergheynst, P., 2016. *Convolutional neural networks on graphs with fast localized spectral filtering.* In Advances in neural information processing systems (pp. 3844-3852).

[19] Devanne, M., Wannous, H., Berretti, S., Pala, P., Daoudi, M., Del Bimbo, A. (2014). *3-d human action recognition by shape analysis of motion trajectories on riemannian manifold.* IEEE transactions on cybernetics, 45(7), 1340-1352.

[20] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. *The graph neural network model.* IEEE Trans. Neural Networks, 20(1):61–80, 2009.

[21] Graves, A., Mohamed, A. R., Hinton, G. (2013, May). *Speech recognition with deep recurrent neural networks.* In 2013 IEEE international conference on acoustics, speech and signal processing (pp. 6645-6649). IEEE.

[22] Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., Wierstra, D. (2015). *Draw: A recurrent neural network for image generation.* arXiv preprint arXiv:1502.04623.

[23] Garland, M. and Heckbert, P.S., 1997, August. *Surface simplification using quadric error metrics.* In Proceedings of the 24th annual conference on Computer graphics and interactive techniques (pp. 209-216). ACM Press/Addison-Wesley Publishing Co.

[24] Grafakos, L. (2008). *Classical fourier analysis* (Vol. 2). New York: Springer.

[25] Hochreiter, S., Schmidhuber, J. (1997). *Long short-term memory.* Neural computation, 9(8), 1735-1780.

[26] Huang, G., Mattar, M., Lee, H. and Learned-Miller, E.G., 2012. *Learning to align from scratch.* In Advances in neural information processing systems (pp. 764-772).

[27] Hammer, B. *Learning with Recurrent Neural Networks.* Assembly Automation 21.2 (2001): 178-83. Web.

[28] I. Goodfellow et al, *Generative adversarial nets,* in Advances in Neural Information Processing Systems, 2014, pp. 2672-2680

[29] Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[30] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. *Neural message passing for quantum chemistry.* arXiv:1704.01212, 2017

[31] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. *Spec- tral networks and locally connected networks on graphs.* arXiv:1312.6203, 2013.

[32] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. *Geodesic convolutional neural networks on riemannian manifolds.* In 3dRR, 2015.

[33] J. M. Susskind et al, *Generating facial expressions with deep belief nets,* in Affective Computing, 2008.

[34] J. Nash, *The imbedding problem for Riemannian manifolds,* Annals of Mathematics, vol. 63, no. 1, pp. 20–63, 1956.

[35] Kipf, T.N. and Welling, M., 2016. *Semi-supervised classification with graph convolutional networks.* arXiv preprint arXiv:1609.02907.

[36] Leng, B., Guo, S., Zhang, X., Xiong, Z. *3D object retrieval with stacked local convolutional autoencoder.* (2015).Signal Processing, 112, 119-128.

[37] Lai, S., Xu, L., Liu, K., Zhao, J. (2015, February). *Recurrent convolutional neural networks for text classification.* In Twenty-ninth AAAI conference on artificial intelligence.

[38] Lafon, S. and Lee, A.B., 2006. *Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization.* IEEE transactions on pattern analysis and machine intelligence, 28(9), pp.1393-1403.

[39] Liu, P., Qiu, X., Huang, X. (2016). *Recurrent neural network for text classification with multi-task learning.* arXiv preprint arXiv:1605.05101.

[40] L.-H. Lim, *Hodge Laplacians on graphs.* arXiv:1507.05379, 2015

[41] Levie, R., Monti, F., Bresson, X. and Bronstein, M.M., 2018. *Cayleynets: Graph convolutional neural networks with complex rational spectral filters.* IEEE Transactions on Signal Processing, 67(1), pp.97-109.

[42] Luan Tran and Xiaoming Liu. *On learning 3d face morphable model from in-the-wild images.* TPAMI, 2019.

[43] Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M. M. *Geometric deep learning on graphs and manifolds using mixture model CNNs.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2017).(pp. 5115-5124).

[44] M. Gori, G. Monfardini, and F. Scarselli. *A new model for learning in graph domains.* In IJCNN, 2005

[45] M. Defferrard, X. Bresson, and P. Vandergheynst. *Convolutional neural networks on graphs with fast localized spectral filtering.* In NIPS, 2016.

[46] M.Fey,J.E.Lenssen,F.Weichert,andH.Mu ller *.Splinecnn: Fast geometric deep learning with continuous B-spline kernels.* In CVPR, 2018.

[47] M. Wardetzky, *Convergence of the cotangent formula: An overview,* in Discrete Differential Geometry, 2008, pp. 275–286.

[48] M. Lin, Q. Chen, and S. Yan. *Network in network.* CoRR, abs/1312.4400, 2013.

[49] M. Arjovsky, S. Chintala, and L. Bottou. *Wasserstein gan.* arXiv preprint arXiv:1701.07875, 2017.

[50] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A. Y. *Reading digits in natural images with unsupervised feature learning.* (2011).

[51] Naveen Kumar H N, Jagadeesha S, Amith K Jain," *Human Facial Expression Recognition from Static Images using Shape and Appearance Feature"*, 978-1- 5090-2399-8/16/ 2016 IEEE

[52] Pascanu, R., Mikolov, T., Bengio, Y. (2013, February). *On the difficulty of training recurrent neural networks.* In International conference on machine learning (pp. 1310-1318).

[53] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. EliassiRad, *Collective classification in network data.*AI Magazine, vol. 29, no. 3, p. 93, 2008.

[54] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. *Cayleynets: Graph convolutional neural networks with complex rational spectral filters.* arXiv:1705.07664, 2017.

[55] Ranjan, A., Bolkart, T., Sanyal, S. and Black, M.J., 2018. *Generating 3D faces using convolutional mesh autoencoders.* In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 704-720).

[56] Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E. *Multi-view convolutional neural networks for 3d shape recognition.* In Proceedings of the IEEE international conference on computer vision (2015).(pp. 945-953).

[57] Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A. and Vandergheynst, P., 2012. *The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains.* arXiv preprint arXiv:1211.0053.

[58] S.Kumar, A.Gupta *Facial Expression Recognition: A Review*

[59] Shiyang Cheng, Irene Kotsia, Maja Pantic, and Stefanos Zafeiriou. *4dfab: a large scale 4d facial expression database for biometric applications.* arXiv preprint arXiv:1712.01443, 2017

[60] S. Zhang et al, *Facial expression synthesis based on emotion dimensions for affective talking avatar.* in Modeling Machine Emotions for Realizing Intelligence ,2010, pp. 109-132.

[61] Sandbach, G., Zafeiriou, S., Pantic, M. and Yin, L., 2012. *Static and dynamic 3D facial expression recognition: A comprehensive survey.* Image and Vision Computing, 30(10), pp.683-697.

[62] S. Rosenberg, *The Laplacian on a Riemannian manifold: an introduction to analysis on manifolds.* Cambridge University Press, 1997.

[63] Such, F.P., Sah, S., Dominguez, M.A., Pillai, S., Zhang, C., Michael, A., Cahill, N.D. and Ptucha, R., 2017. *Robust spatial filtering with graph convolutional neural networks.* IEEE Journal of Selected Topics in Signal Processing, 11(6), pp.884-896.

[64] Saha, S. (2018). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.* [online] Medium. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 [Accessed 4 Aug. 2019].

[65] Stein, E. M., Weiss, G. (2016). *Introduction to Fourier analysis on Euclidean spaces* (PMS-32) (Vol. 32). Princeton university press.

[66] Thanou, D., Dong, X., Kressner, D. and Frossard, P., 2017. *Learning heat diffusion graphs.* IEEE Transactions on Signal and Information Processing over Networks, 3(3), pp.484-499.

[67] U. Von Luxburg, *A tutorial on spectral clustering* Statistics and Computing, vol. 17, no. 4, pp. 395–416, 2007.

[68] Unser, M., Aldroubi, A., Eden, M. (1993). *B-spline signal processing. I. Theory.* IEEE transactions on signal processing, 41(2), 821-833.

[69] Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. *WaveNet: A generative model for raw audio.*(2016). SSW, 125.

[70] Wei, L., Huang, Q., Ceylan, D., Vouga, E., Li, H. *Dense human body correspondences using convolutional networks.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2016)(pp. 1544-1553).

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

[72] Werbos PJ. *Backpropagation through time: what it does and how to do it.* Proceedings of the IEEE. 1990 Oct 1;78(10):1550-60.

[73] Walecki, R., Rudovic, O., Pavlovic, V., Schuller, B., Pantic, M. (2017). *Deep structured learning for facial expression intensity estimation.* Image Vis. Comput, 259, 143-154.

[74] X. Yan et al, "Attribute2image: Conditional image generation from visual attributes," in European Conference on Computer Vision, 2016, pp. 776-791.

[75] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. *Gated graph sequence neural networks.* In ICLR, 2016.

[76] Yao Feng, Fan Wu, Xiaohu Shao, Yanfeng Wang, and Xi Zhou. *Joint 3d face reconstruction and dense alignment with position map regression network.* In ECCV, 2018.

[77] Z. Zhang, Y. Song and H. Qi, *Age Progression/Regression by Conditional Adversarial Autoencoder,* arXiv Preprint arXiv:1702.08423, 2017.

[78] Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H. and Xu, B., 2016, August. *Attention-based bidirectional long short-term memory networks for relation classification.* In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (pp. 207-212).

[79] Zhou, Y., Deng, J., Kotsia, I. and Zafeiriou, S., 2019. *Dense 3D Face Decoding over 2500FPS: Joint Texture Shape Convolutional Mesh Decoders.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1097-1106).

[80] Zhu, X., Lei, Z., Liu, X., Shi, H. and Li, S.Z., 2016. *Face alignment across large poses: A 3d solution.* In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 146-155).

[81] *Generating Faces with Deconvolution Networks.* Available: https://zo7.github.io/blog/2016/09/25/generating-faces.html.

[82] The Menpo Project, https://www.menpo.org/