THE UNIVERSITY OF WARWICK

Unsupervised Face Clustering via Adversarial Variational Graph Auto-Encoder and Similarity Density

by

Kyungsu Choi

Thesis

Submitted to The University of Warwick

for the degree of

MSc Data Analytics

Department of Computer Science

Sep 2021

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisor Professor Tanaya Guha for her invaluable advice and support with this project

Abstract

In machine learning, clustering is an unsupervised learning technique, which helps understand data patterns and analysis. In clustering, face clustering plays an important role in developing applications related to face images. Since the advent of graph convolution networks, much research in recent years has focused on graph-based face clustering, and achieved noticeable performances.

However, these graph-based methods use label data for training models although face clustering is an unsupervised learning problem. Because of this, there still remains a need for an unsupervised learning method that can solve face clustering problems. In this regard, the aim of this study is unsupervised face clustering via adversarial variational graph auto-encoder(VGAE), which can be trained without label data. The adversarial VGAE makes use of latent information by using graph-structured data and can regularise latent information by a discriminator. Its decoder can achieve competitive performance on a link prediction task. By utilising the link prediction, the clusters of face images can be found.

Furthermore, recent research using graph-structure has created sub-graphs based on all nearest neighbouring vertices. These methods can restrict their accuracy and efficiency. To solve this problem, this study adopted a method that prunes sub-graphs through similarity density. Therefore, the proposed work is adversarial VGAE face clustering using similarity density. As a result, although the overall performance was not improved, the proposed work has better results in evaluation(recall score) based on identity(label). The proposed work was also compared with various graph auto-encoder techniques and baseline for deep analysis.

Chapter 1

Introduction

Face recognition has witnessed great progress in recent years, and its performance has achieved considerable improvements. With this rapid growth of face recognition, it is crucial to process facial images used in many applications. Among these facial image processing techniques, face clustering plays an important role in the development of applications using face photos. For instance, face clustering is utilised for creating digital photo albums[5], and video and images taken by surveillance cameras automatically[33] detect humans. Like these cases, face clustering is necessary to automatically manage a lot of face images generated through the web, social media, and smartphones.

Clustering is a technique that automatically grouping data points according to the characteristics of each data point. This technique basically proceeds with unsupervised learning, which performs without labels. Among the various clustering, face clustering is grouping face images. Ideally, the result of clustering is creating several groups,, and each group represents one person's face images. Then, the face clustering provides information about how many different people there are in many face photos and videos.

For clustering, traditional clustering methods(such as K-means[16] and DBSCAN[6]) has been proposed. These studies suggested an effective way for face clustering, but these algorithms have the performance of the model depended on pre-determined hyperparameters such as the number of clusters, epsilon distance, and minimum samples. Especially, in the case of Kmeans, random initialisation can make a huge impact on the performance. In other words, these methods have unstable results and low performances according to the hyperparameters.

One way to achieve stable results and high performance is to utilise a deep neural network. Unlike conventional methods, this way proceed learning datasets without predefined hyperparameters. Therefore, many studies for face clustering has focused on utilising a neural network. Among the diverse neural network structures, much research in recent years has been proposed methods using graph convolution network(GCN)[9][14], which uses a graph structure with advantages of convolution neural network.

One of the representative recent research is linkage based face clustering using graph convolution network[27]. To be specific, after extracting features from face images, these facial images can be allocated in vector space(feature space), and be represented as a vector. In this vector space, distances between the face images are used to create sub-graphs and then these sub-graphs be gone through GCNs. Finally merging sub-graphs enable to obtain clusters(groups).

However, although these methods using GCNs considerably improve the model's performance, much research was far from unsupervised learning because the GCNs layers need to be trained by data with labels. In addition, the research make use of sub-graphs using all neighbouring vertices. This subgraph can include unnecessary edges for face clustering. In this point of view, there kinds of studies have two main problems with previous clustering models. Firstly, previous methods are semi-supervised or supervised learning even though clustering is an unsupervised learning problem. Secondly, all edges of nearest neighbour data are generated for the construction of sub-graphs. This is far from efficiency and accuracy.

Project objectives: To clarify aims of project, the main project objectives are established as follows.

- To build an unsupervised learning model by using GCN.
- To remove unnecessary edges of sub-graphs for improving model.

Therefore, this paper focused on utilising adversarial variational graph auto-encoders for unsupervised learning clustering because the auto-encoder model theoretically uses no label data. In addition, sub-graphs were pruned by the similarity density, so unnecessary edges of sub-graphs are removed for accurate clustering. The related results was provided in the result section.

Chapter 2

Related Work and Literature Review

This chapter will introduce background and research related to the proposed work. The Related work section(Sec. 2.1) will overview graph convolution network, face clustering, and various auto-encoders. Then, the literature review section(Sec. 2.2) will explain important studies, which motivate the proposed method.

2.1 Related Work

2.1.1 Graph Convolution Network

Of the many approaches, graph convolution network(GCN) has shown a lot of progress on various tasks including face clustering. This GCN structure keeps characteristics of convolution neural network, which are shared weights(using same weights for each node) and utilisation of local features. The GCN can be categorized into spectral GCN[14] and spatial GCN[9]. Spectral GCN is based on the graph Fourier Transform, while spatial GCN makes use of the convolution of nodes and their neighbours. However, this basic GCN has a limitation of reflecting the degrees of each edge's importance.

To solve the problem mention above, a graph attention network(GAT)[25] is proposed. GAT contain contributions of neighbouring nodes to the central node. These contributions are also learned by including weights of edges.

This GAT outperformed previous graph-based approaches. However, this GAT needs more parameters, thus training run-time significantly increase compared with simple GCN. In addition, these GCN techniques including GAT have vanishing problems and are over-fitting when the GCN is deeper(using many GCN layers).

In addition, in terms of GCN's setting, there are two kinds of setting graphs which are transductive setting[14] and inductive setting[9]. transductive setting use all nodes in the graph, on the other hand, inductive setting utilizes local neighbourhood from a node.

In recent years, to improve training efficiency, there are several advanced GCN structures proposed. FastGCN[2] is proposed and decrease computational cost by sampling vertices according to a probability measure. Cluster-GCN[3] improve computational cost and memory usage by sampling a subgraph and limit the search range of the graph. In an aspect of reducing the size of the parameter, self-attention pooling[15] are proposed. Like max-pooling on CNN, these methods are essential to reduce the computation complexity of graph convolution.

2.1.2 Face clustering

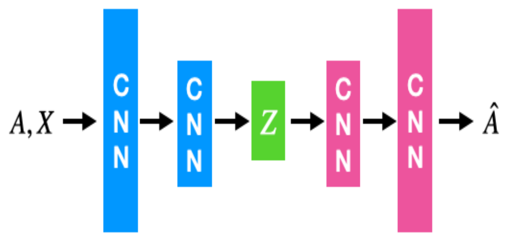
For clustering, conventional clustering models such as K-mean[16] and DBSCAN[6] all rely on rigid assumptions on the data distribution. K-means implicitly assumes that the samples in each group are located around a single centre and also is not robust. DBSCAN has an assumption in which the data is evenly spread(even density) in feature space in each cluster. A rank order based face clustering algorithm[34] is suggested which using the rank order of face images for each image according to similarity distance among face images. For improving the rank order based method, approximate rank order based clustering[19] is proposed which calculate the rank scores of instances within its k nearest neighbours(kNN). This approach gains the advantages of reducing computational complexity compared to previous rank order approach. A deep density-based image clustering(DDC)[22] process images based on a density of similarity instead of simple cosine similarity. However, these traditional approaches have limits of performance compared to deep neural network methods.

There is also a graph structure based face clustering method named consensus-driven propagation(CDP)[32] which exploits pairwise similarity scores. However, this approach requires separation of data for the learning model before clustering face images and The CDP also need a small portion of labelled data. This means that this technique is not unsupervised learning.

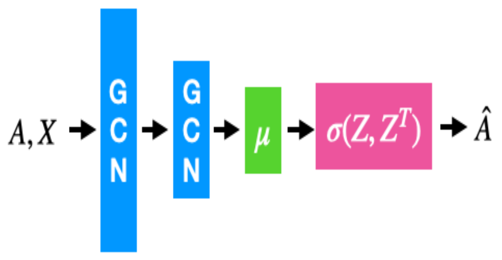
2.1.3 Auto-Encoder

Auto-encoder basically has a neural network structure(Fig. 2.1a) with unsupervised learning that contains a bottleneck hidden layer. The auto-encoder use reconstruction error by gradually minimising the difference between the input image and output image. The output image is predicted by latent information, which is extracted by CNN layers of the encoder. After learning, the encoder can be used for dimension reduction or feature extraction, and the decoder can be used for generating new images. However, this approach has a limitation of depending on data distribution assumption, in which latent distribution follow Gaussian or Bernoulli distribution. In addition, variational auto-encoder model[12] is suggested for probabilistic approximation.

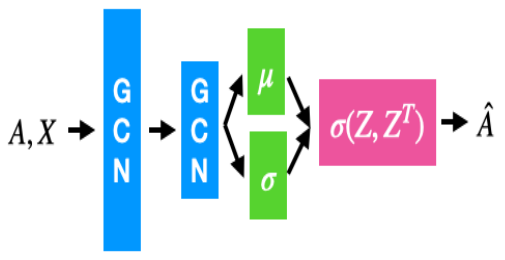
In this concept's graph version, graph auto-encoder and variational graph auto-encoder(VGAE)[13] is proposed for a link prediction as shown in Fig. 2.1b and 2.1c. Face clustering method[26] with this link prediction is also proposed. This method simply makes use of the VGAE. In addition, like GAT, graph attention auto-encoder(GATE)[23] is a graph auto-encoder applied with self-attention so as to utilise graph structure. Moreover, adversarial variational auto-encoder(AAE)[17] is proposed for better performance as seen in Fig. 2.1d. The AAE is applied with a discriminator, which distinguish whether the input is true or fake distribution. The proposed method will make use of adversarial variational graph auto-encoder concept for an unsupervised model.



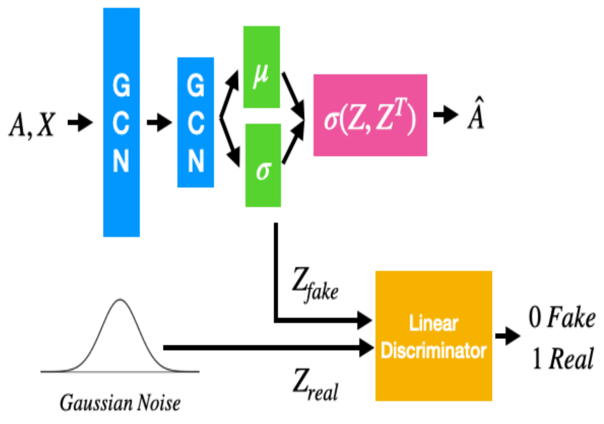
(a) Standard Auto-Encoder



(b) Graph Auto-Encoder



(c) Variational Graph Auto-Encoder



(d) Advarsarial Variational Graph Auto-Encoder

Figure 2.1: Various auto-encoder architectures.

2.2 Literature Review

In this section, each explanation of research will be presented in order of proposed approaches, an overview of the methodology, and an analysis of advantages and disadvantages, moreover each figure of architecture overview will be shown for easily understanding the model.

2.2.1 Learning to Cluster Faces via Confidence and Connectivity Estimation

The face clustering via confidence and connectivity estimation[30] creates a limited number of sub-graphs by confidence values instead of generating numerous sub-graphs. Pruning clusters generated by each sub-graph is proceeded by connectivity values.

There are two sequential graph convolution networks which are for calculating confidence and connectivity values. The first part of graph convolution layers is for a calculation of confidence which represents how close vertices are with neighbours in the same class. weights of this graph convolution network are trained by loss function such as the mean square error between true confidence and predicted confidence. The second part of graph convolution layers is used for the generation of sub-graphs based on vertices with high confidence, and then these sub-graphs are put into GCNs trained by the vertex-wise mean squared loss function. From the outputs of this GCN, connectivity can be calculated. As a result, clusters are obtained via the deduction of connections between nodes.

There are advantages and disadvantages to this model. One of the main advantages is selected nodes for generating sub-graphs based on confidence scores instead of entire sub-graphs. This leads to a reduction of computational complexity in comparison to other approaches. However, this model requires labelled data for training graph convolution layers to compute confidence and connectivity. In other words, this method is not a fully unsupervised learning model because supervise learning is needed for calculating confidence and connectivity, on the other hand, clustering problems contains using unsupervised learning because there are unlabelled data. This paper adopted similarity density concept without labels.

2.2.2 Density-Aware Feature Embedding for Face Clustering

Density-aware feature embedding method(DA-Net)[7] utilize both local and non-local information by similarity densities of chain graph, which approximates the probability of the person and also are used as inputs of GCN and a transformer. This is because data points with a higher density are more likely to have similar feature vectors and correspond to the same person.

Overview of the DA-Net will be presented in order of model procedures. The DA-Net generates several chain graphs based on similarity density, and then clusters are found by transformer including Long short term memory(LSTM).

At first, the density of each node is calculated by the inner product of the neighbours, which computes cosine similarity. Then, a chain graph is generated in a way to maximize the density. The chain graph expands in some steps, which sequentially takes one node to the chain graph for each step. Each node in a chain graph and its neighbours is used for the local clique network based on GCN. Through this GCN, local information is gained. Long-range information is computed by the transformer using the chain graph. As a result, face clustering is completed by the combined graph exploiting local node information from GCN and long-range information from a transformer.

This DA-Net has a distinct advantage to make use of local and non-local information. In addition, the time complexity is low compared with other clustering algorithms since the only small graph is computed, however, compared with L-GCN, this algorithm is expected to use memory capacity because each chain and local nodes are stored in memory storage; moreover creating a chain graph could be unstable if the density doesn't be evenly spread.

2.2.3 Linkage Based Face Clustering via Graph Convolution Network

This linkage based face clustering using GCN(L-GCN)[27] is an excellent model because this approach starts to use graph convolution layers with the inductive setting for face clustering.

There are three steps, which are creating Instance Pivot Sub-graph(IPS), graph convolution layers, and combine all link predictions. Firstly, all nodes(using pivot) with feature vectors extracted from original images are used for creating IPSs. IPS is generated by taking the node's nearest neighbours(kNN) and first hop. Secondly, each IPS is used as inputs of graph convolution layers to update the centre node's feature vector. Finally, the output IPSs are merged and pruned based on a threshold for cutting edges.

This GCNs method should require labelled image data for training weights of graph convolution layers. In other words, there is a limitation of unsupervised learning; In addition, only local information is used because nodes are selected based on K-nearest neighbours, h-hops, and the number of active connections. Lastly, In this model, a sallow graph convolution network(four layers) is used because the deeper layers make a vanishing problem.

To solve this vanishing problem, the graph convolution layers are used with connections between the input and output of graph convolution layers for direct mapping. This model is called the residual GCN(RGCN)[21]. This RGCN enables GCN to use more graph convolution layers, but this leads to increasing the size of parameters. This causes the rising computational cost.

2.2.4 Variational Graph Auto-Encoder

Variational graph auto-encoder(VGAE)[13] uses probabilistic model which including inference model(encoder) and generative model(decoder). In the encoder, after using GCN, the mean vector μ and log standard deviation ρ are inferred from the feature matrix and adjacency matrix. These parameters(μ and ρ) are for making Gaussian noise based on Gaussian distribution. After encoding, the decoder generates a predicted adjacency matrix. The optimal model can be found by minimising variational lower bound loss function with the Kullback-Leibler divergence between an assumed distribution and actual distribution.

This VGAE model has weaknesses that are speed of computation and assumption of Gaussian distribution regarding data points. However, this auto-encoder model is suitable for unsupervised learning. Therefore, this research of face clustering utilises this graph auto-encoder for the unsupervised solution.

2.2.5 Adversarial Attention-Based Variational Graph Auto-Encoder

Adversarial Attention-Based Variational Graph Auto-Encoder(AAVGA)[20] considers both basic auto-encoder structure and data distribution from graphs. A separated deep neural network is designed for applying data distribution to loss functions.

There are two main parts of the AAVGA method . The first part is the attention auto-encoder. In the first step of the part, the adjacency matrix and feature matrix are used as inputs of the graph attention encoder. The output of the encoder is a latent matrix. Then, the predicted adjacency matrix is reconstructed by the latent matrix through the decoder. lastly, the difference between the predicted adjacency matrix(decoder output) and the original adjacency matrix(encoder input) is used as a loss function. The second part is the discriminator. Inputs are random noise matrix and latent matrix. Outputs of the discriminator predict results such as "Real" and "Fake". This discriminator has the function of regularising auto-encoder learning.

This AAVGA architecture has two advantages which are from attention layers and the adversarial mechanism. Attention layers allow the model to use the information of the graph structure. The adversarial mechanism brings the benefit of regularising the loss of the auto-encoder. On the other hand, the number of parameters increases due to attention layers. In addition, if training of the adversarial part is wrong, it is a bit difficult to learn data to minimise loss of auto-encoder.

Chapter 3

Proposed Approach

This chapter will explain reasons why the methods were adopted for the proposed model with referred studies in section 3.1, then the next section will overview the proposed model in section 3.2. Lastly, each step of the proposed model will be explained in detail.

3.1 Motivation

This section fully explains sufficient reasons why the proposed method was chosen. Here is the order of this chapter: defining problems, the unsupervised learning solution, and pruning edges of sub-graphs.

The reason why GCN was used: To handle the face images, the face images should be transformed to feature vectors, which have values of characteristics of face images, such as a shape of a mouth and a nose. The feature vectors are extracted by a face recognition deep learning model in order to obtain distinguishable features. Therefore, the face images are allocated in feature space as feature vectors. In the feature space, a face image can be represented as a vertex. if face images are considered vertices, a cluster of face images is a graph containing vertices and edges. In this point of view, much research in recent years has used graph convolution network(GCN) for solving the clustering problem such as linkage base face clustering[27] and learning to cluster faces on an affinity graph[31].

Definition problems: The proposed model also used GCN. However, as mentioned in the introduction(Chapter 1), these recent methods using GCN have two limitations as follows:

1. [**Problem 1**] Using supervised learning for training GCN model by using labels.

2. [**Problem 2**] Utilising all edges of sub-graphs to find final clusters.

The rest of this section will focus on how to approach these two problems and to find optimal methods. The following solutions were applied to the proposed model.

1. [**Solution 1**] Graph Auto-encoder was adopted for unsupervised learning method.

2. [**Solution 2**] To remove unnecessary edges of sub-graphs by similarity density.

The detailed explanation regarding these solutions is presented as follows:

[**Solution 1**] **Unsupervised learning:** For unsupervised learning clustering, one of the reasonable solutions is the auto-encoder whose architecture does not technically need labels(classes) of images. This is because training losses are calculated by comparison between inputs and reconstructed inputs, which are computed by the decoder without labels. One of the recent methods has attempted to take advantage of both auto-encoders and GCN so that variational graph auto-encoders(VGAE)[13] are proposed. Especially, the decoder of the VGAE acts as calculating similarity scores among vertices with latent feature vectors. These similarity scores can be used for link prediction to find final clusters. In this regard, this paper uses the VGAE architecture with some alterations to take an advantage of the unsupervised learning method.

The variational graph auto-encoder is chosen as base model architecture. However, the performance of the auto-encoder is likely to be unstable and slightly decrease because of training the model without labels, so much research in recent years have proposed various auto-encoder techniques to improve performance. The techniques are attention-based VGAE and adversarial VGAE. The proposed model also adopted the following architectures as options.

- Attention-based VGAE

- Adversarial VGAE

- Adversarial Attention-based VGAE

Firstly, the attention-based VGAE is applied GAT[25](instead of GCN) to the auto-encoder. The attention graph makes use of both graph structure and the relevance of neighbouring vertices to a vertex in a graph. The research proposes this attention-based variational graph auto-encoder[23], however this model is applied only to link prediction of network connection datasets. Therefore, unlike previous works, this study implemented the model for face image data.

Secondly, the adversarial auto-encoder[17] is suggested for regularising the latent feature vectors. A variational graph version[20] of the adversarial auto-encoder is also proposed. In addition, adversarial attention-based VGAE[28] adopts both the adversarial concept(applying a discriminator) and self-attention graph. Like these previous studies, numerous models have already utilised these concepts, but these concepts have not been applied to face the clustering problem. Therefore, the proposed work attempted to utilise this adversarial VGAE and attention-based VGAE to improve the model performance for face clustering.

[Solution 2] Pruning edges of sub-graphs: As mentioned previously, there is a problem that is using all edges of sub-graphs to obtain final clusters because this enables to restrict the model's accuracy and efficiency. In this regard, by utilising similarity density, some research has overcome this kind of difficulty. Let the similarity density denote a mean of cosine similarity score between a vertex and its neighbours. Intuitively, a vertex with high similarity density is likely to be allocated in the place where the vertices are densely distributed and belong to the same cluster, whereas the vertices with low similarity density usually are positioned on the boundary among several clusters.

Some research has used this density concept. For example, in face clustering using confidence and connectivity estimation [30], utilise mean value(called "confidence" value) of cosine similarity score between a centre node and its k nearest neighbours (kNN) by using labelled images, the edges of sub-graphs are selected in the basis of the mean value. In addition, in face clustering using the density-aware feature embedding[7], like the confidence value, the research utilises sum(called data density) of cosine similarity between a centre node and its kNN. However, these studies calculate similarity density of nodes selected by labels. For example, neighbour nodes are selected if the center node label is equal to the neighbour nodes label. Therefore, the proposed model computed similarity density without using labels, then the similarity density was used for pruning the edges of sub-graphs.

3.2 Overview

3.2.1 Problem Statement

This section provides the mathematical notations used in this paper, and the problem definition of face clustering are expressed by the notations. The lower-case letters are used for vectors (e.g. x, e, and v). The upper-case letters are used for matrices (e.g. X, A, and H). The transpose of a matrix A is represented as A^T. The notation x_i is represented as i^{th} element of a vector x, and the notation X_{ij} is denoted the element of matrix X at the i^{th} row and the j^{th} column. In addition, *din* is an input dimension and *dout* is an output dimension of GCN layers in the encoder.

For representation of face clustering problem, x_i means a i^{th} feature vector extracted from a i^{th} face image through face recognition. A feature matrix is represented as $X = [x_1, x_2, \ldots, x_N]$, where N is the number of face images(the number of data). The x_i's dimensionality is denoted as $x_i \in \mathbb{R}^{d^{(k)}}$, which means a x_i vector with d dimensions in the k^{th} layer. If there is N data(feature vectors), a feature matrix can be indicated by $X \in \mathbb{R}^{N\times d^{(k)}}$. an adjacency matrix $A \in \mathbb{R}^{N\times N}$ are given because the adjacency matrix means connections of combination for N instances. If there is an edge between x_i and x_j, $A_{ij} = 1$, and if there is no edge between them, $A_{ij} = 0$. As mentioned above, a input feature vector and matrix of GCN are represented as $x \in \mathbb{R}^{1\times d^{(k)}}$ and $X \in \mathbb{R}^{N\times d^{(k)}}$, and a output feature vector and matrix of each GCN layer are indicated as $h \in \mathbb{R}^{1\times d^{(k+1)}}$ and $H \in \mathbb{R}^{N\times d^{(k+1)}}$. In addition, H^k means output feature matrix by the k th layer. Like the feature vector and matrix, a latent feature vector and matrix are represented as $z \in \mathbb{R}^{1\times dout}$ and $Z \in \mathbb{R}^{N\times dout}$ respectively. These notations are written in Table 3.1.

3.2.2 Pipeline

This study goals basically are to apply an unsupervised learning model to face clustering and to reduce the number of sub-graphs' edges to improve model performance. To achieve the goals, this work uses a graph structure including nodes and edges. Graphs are generated based on each instance(face image), which is called a "c_i center node" ($c_i : i = 1, 2, 3, \ldots, N$). A graph made from a c_i center node is called a S_{c_i} "sub-graph" of the center node. The sub-graphs are created by every instance, which means that every instance can be a center node. For example, if the number of face images is 1,000, 1,000 sub-graphs are generated from all feature vectors of face images. In a sub-graph, nodes(feature vectors) are selected on the basis of k-nearest neighbours (kNN), therefore a matrix of kNN should be generated from feature vectors in advance for simply selecting graph nodes. The kNN matrix is created based on the distance of cosine similarity. This preprocessing kNN will be explained in the preprocessing section 4.2.3. A procedure of creating kNN is presented as shown in Algorithm 1:

Algorithm 1 Generating kNN matrix

```
    Input: X feature matrix
    Output: kNN matrix
    Require: $N$: the number of instances

Require: k = the number of nodes which are searched
 1: for iteration = $[1, 2, 3, \ldots, N]$ do
 2:     Select $i^{th}$ node as a center node
 3:     Calculate distances of cosine similarity between $i^{th}$ node and the others
 4:     Sort the node number except $i^{th}$ node out in order of closest distance
 5: end for
```

Therefore, sub-graphs made from each center node c_i. Vertices of the sub-graphs contain the kNNs of center node and the high-order neighbouring nodes up to h-hops of c as illustrated by Fig. 3.2 (d). To calculate similarity density for each center node, adversarial VGAE(called similarity density

Notation	**Definition**
N	The number of vertices(nodes) in entire dataset
c_i	Center node i
N_i	The neighbour nodes from node i
$d^{(k)}$	d dimensions by k^{th} layer
V_i	The unique vertices in a sub-graph if the center node is i vertex, including node i
$\|V_i\|$	The number of the unique vertices in a sub-graph if the center node is i vertex, including node i
$\mathcal{S}_i$	The sub-graph from center node i
$\mathcal{S}_i^{'}$	The pruned sub-graph from center node i
$A \in \mathbb{R}^{N \times N}$	The adjacent matrix for all nodes
$A \in \mathbb{R}^{\|V_i\| \times \|V_i\|}$	The adjacent matrix for all nodes in a sub-graph
$\hat{A} \in \mathbb{R}^{N \times N}$	The reconstructed adjacent matrix for all nodes
$\hat{A} \in \mathbb{R}^{\|V_i\| \times \|V_i\|}$	The reconstructed adjacent matrix for all nodes in a sub-graph
$X \in \mathbb{R}^{N \times d^{(k)}}$	The feature matrix for all nodes by k^{th} layer
$X \in \mathbb{R}^{\|V_i\| \times d^{(k)}}$	The feature matrix for all nodes in a sub-graph by k^{th} layer
$x \in \mathbb{R}^{1 \times d^{(k)}}$	The feature vector for a node by k^{th} layer
$H \in \mathbb{R}^{\|V_i\| \times d^{(k+1)}}$	The feature matrix for all nodes in a sub-graph by $k+1^{th}$ layer
$h \in \mathbb{R}^{1 \times d^{(k+1)}}$	The feature vector for a node by $k+1^{th}$ layer
$Z \in \mathbb{R}^{\|V_i\| \times dout}$	The latent feature matrix for all node in a sub-graph with output dimensions of Encoder
$z \in \mathbb{R}^{1 \times dout}$	The latent feature vector for a node with output dimensions of Encoder
$\alpha_{ij}^{(k)}$	The attention coefficient indicating the relative relevance of neighboring node j to node i in the k^{th} layer.

Table 3.1: The main notations used in this paper

estimator) are used. The adversarial VGAE computes latent feature vectors, then the similarity densities of each center node are calculated by using latent feature vector z for generating 2^{th} sub-graphs(reduced edges compared to 1^{th} sub-graphs). Given the 2^{th} sub-graphs, linkage prediction between c and neighbours are conducted by one additional graph auto-encoder(called link prediction estimator). For better understanding, an overview of the proposed model can be seen in Fig. 3.1, and the procedure of the proposed model is presented as follows:

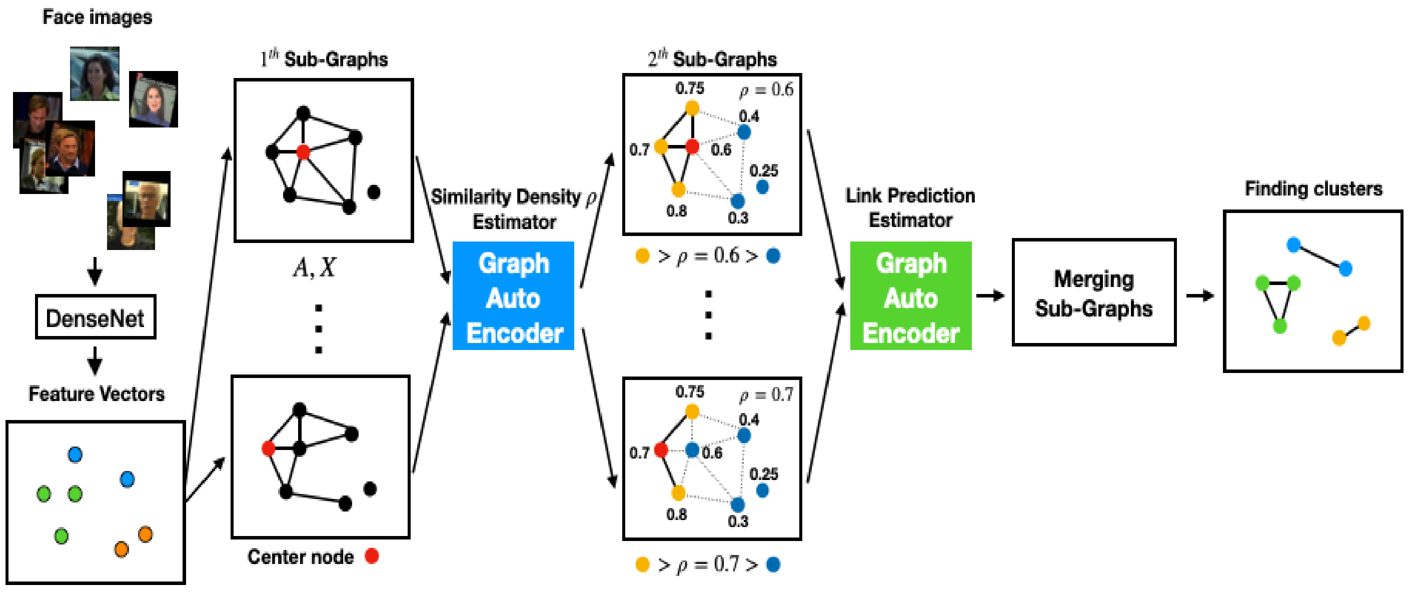


Figure 3.1: Overview of the proposed architecture

1. **Construction of 1^{th} sub-graphs:** All instances are used as a center node, and sub-graphs are generated based on the center nodes. The procedure of constructing sub-graphs is explained in detail in Section 3.3.

2. **Calculation of similarity density from 1^{th} sub-graphs:** Through variational auto-encoder with GCNs, latent feature vectors z are created. Then, similarity density(cosine similarity) of each center node is computed. This similarity density auto-encoder is described in section 3.4.

3. **Construction of 2^{th} sub-graphs:** Unlike 1^{th} sub-graphs, edges are taken on the basis of similarity density of center nodes and neighbors. Hence, the 2^{th} sub-graphs are generated. The 2^{th} sub-graphs have the fewer edges compared to 1^{th} sub-graphs. The procedure of constructing sub-graphs is explained in detail in Section 3.5.

4. **Link prediction through an auto-encoder:** Given the 2^{th} sub-graph, the latent feature matrix(Z) are computed. Then adjacency matrix($\hat{A}$) are calculated based on the latent feature matrix(Z). By using $\hat{A}$, link predictions is carried out. The link prediction auto-encoder is represented in section 3.6.

5. **Merging 2^{th} sub-graph:** By the link prediction, 2^{th} sub-graph are merged. clusters are found from merged sub-graphs. This merging process is described in section 3.7

3.3 Construction of 1^{th} Sub-graphs

Before the way to construct sub-graphs is explained, the reason why sub-graphs are generated should be clarified. As mentioned previously, a graph with nodes and edges is easily handled for extracting meaningful results. This is due to the fact that feature vectors are extracted from a face image, and the feature vectors and similarity distances are represented in the feature space as nodes and edges. This means that local information(data distribution) can be applied to the graph.

Construction of sub-graphs proceeds through two steps. Firstly, finding candidate nodes of sub-graphs is carried out based on a center node. Secondly, edges among nodes are added. The procedure of construction sub-graphs is illustrated by Fig. 3.2. These steps are adapted from the linkage face clustering via GCN [27] with one modification, which is not to normalise feature vectors of nodes by the feature vector of a center node on account of the fact that a feature vector of the center node has 0 values if all feature vectors are normalised by a center node. Because of 0 values, similarity scores between the center node and the other nodes are 0 values, hence the similarity density of the center node cannot be calculated.

Step 1 Finding nodes: First of all, every node can be selected as a center node(denoted as 'c_i'), then 1^{th} hop nearest neighbors(called 1^{th} hop nodes) from each c_i node are generated according to pre-determined h_1 hop value(1-hop). The 1^{th} hop nearest neighbors has cosine similarity in descend order. Like $1^{th}h$ hop nodes, 2^{th} hop nearest neighbors from 1^{th} hop nodes are generated under a condition, which the number of 2^{th} hop nodes is a pre-determined h_2 hop value(2-hop). However, the maximum number of edges from a node should be within a predetermined active connections(denoted by u).

Step 2 Adding edges: unique vertices(denoted by V_c) of a sub-graph include c_i node, 1^{th} hop nodes, and 2^{th} hop nodes. A set of sub-graph's edges(E_c) contains edges between the c_i nodes and 1^{th} hop nodes and edges between 1^{th} hop nodes and 2^{th} hop nodes within the active connections(u). For example, if h_1=7, h_2=2, and u=3 are determined, its nodes and edges are found as shown in Fig. 3.2 (d). Theses edges are transformed to an adjacency matrix $A_c \in \mathbb{R}^{|V_c|\times|V_c|}$ for computation. The nodes feature matrix is $X_c \in \mathbb{R}^{|V_c|\times d}$.

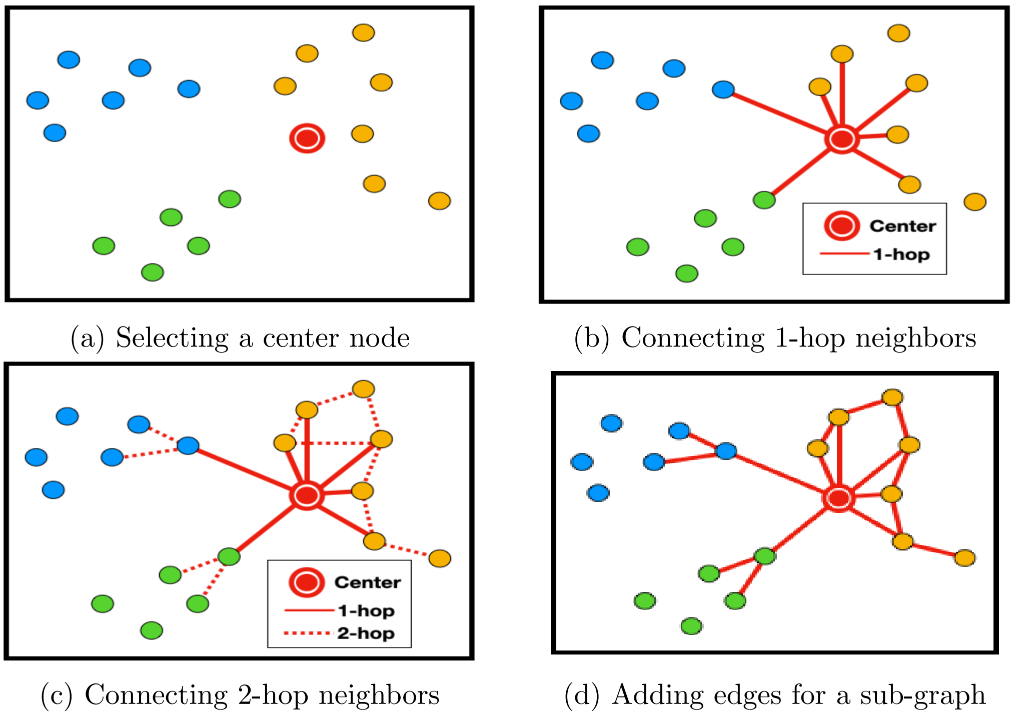
3.4 Similarity Density Estimator

To prune sub-graphs, the proposed model should calculate each center node's similarity density, which is the average cosine similarity score between each center node and center node's k nearest neighbours. For computing, meaningful similarity density, an adversarial variational graph auto-encoder(Fig. 3.3) is used for generating a latent feature matrix Z, which has more distinguishable and lower-dimensional features rather than the original feature matrix X. There are three parts of the similarity density estimator: Adversarial VGAE(3.4.1), Discriminator(3.4.2), and similarity density calculation(3.4.3).

3.4.1 Adversarial Variational Graph Autoencoder

There are three sections of this adversarial VGAE: (1) GCN layer, (2) Encoder, and (3) Decoder.

(1) GCN layer in Encoder: In the layers of graph auto-encoder,



(a) Selecting a center node (b) Connecting 1-hop neighbors (c) Connecting 2-hop neighbors (d) Adding edges for a sub-graph

Figure 3.2: There are h_1=7, h_2=2, and u=3. 7(h_1) 1-hop nodes are connected to the center node. 2(h_2) nodes are connected to each 1-hop node. Each node has only 3 edges(maximum). (a) selecting a center node (b) selecting seven 1-hop nodes (c) choosing two 2-hop nodes from each 1-hop node (d) because of u=2, each node has only two edges.

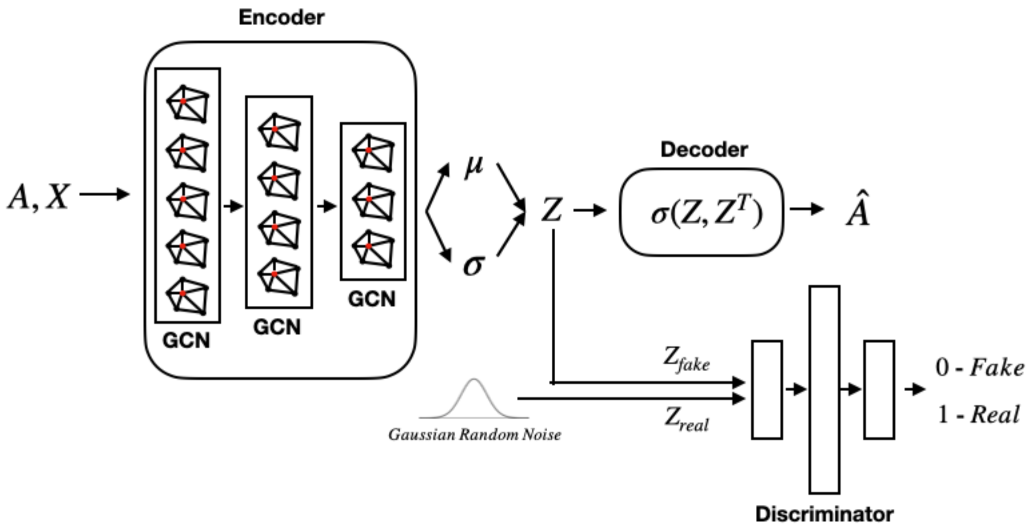


Figure 3.3: Architecture of the adverarial variational auto-encoder.

graph convolution networks(GCNs) are used in each layer because local(neighbours) information in the feature space can be used. The GCN layer is used according to concatenated GCN[9]. Each graph convolution layer take an adjacency matrix $A \in \mathbb{R}^{|V_i| \times |V_i|}$ and a feature matrix $X \in \mathbb{R}^{|V_i| \times din}$ as inputs. The feature matrix is X in the first layer, and H^k represent a feature matrix in the k^{th} layer. The computation of each concatenated GCN layer($GCN(X, A)$) can be formulated as follows:

$$Agg(A, H^k) = AH^k \tag{3.1}$$

$$H^{k+1} = \sigma([H^k || Agg(A, H^k)])W, \tag{3.2}$$

where σ is a rectified linear activation function. The $H^k \in \mathbb{R}^{|V_i| \times d^{(k)}}$ is an input feature matrix of the k^{th} layer. $\tilde{A_{ij}} \in \mathbb{R}^{|V_i| \times |V_i|}$ is normalised by $D = \sum_j (A+I)$, which is a diagonal degree matrix. I is a matrix with ones on the diagonal and zeros elsewhere for self-connection. $Agg(A, H^k)$ is an aggregation function. This is generating new feature matrix which applied with local information(feature vectors and edges of neighbour node). The weight matrix of the concatenated GCN is $W \in \mathbb{R}^{2d^{(k)} \times d^{(k+1)}}$. The [||] is concatenation of two matrices in the row axis(concatenation output $\mathbb{R}^{N \times 2d^{(k)}}$).

(2) Encoder: The encoder is used for generating the latent matrix Z by output feature matrix from multiple GCN layers. The adversarial VGAE is defined by an inference model:

$$q(Z|X, A) = \prod_i^N q(z_i|X, A), \text{ with } q(z_i|X, A) = N(z_i|\mu_i, diag(\sigma^2)) \tag{3.3}$$

Where $\mu = GCN_\mu(X, A)$ is the matrix of mean vectors μ_i; similarly $log\ \sigma = GCN_\sigma(X, A)$ is the matrix of standard deviation vectors. The vector and matrix formulation of creating latent features can be represented as follows:

$$z_{ij} = \mu_{ij} + log\ \sigma_{ij} \times noise \tag{3.4}$$

$$Z = \mu \text{ matrix} + \log \sigma \text{ matrix} * noise \text{ matrix} \tag{3.5}$$

where the noise means Gaussian random noise.

(3) Decoder: The graph decoder is used for reconstructing an adjacency matrix $\hat{A}$ from the latent matrix Z by Eq. 3.7. The vector and matrix formulation of the decoder can be represented as follows:

$$p(A|Z) = \prod_{i}^{N}\prod_{j}^{N} p(A_{ij}|z_i, z_j), \text{ with } p(A_{ij} = 1|z_i, z_j) = \sigma(z_i^T z_j) \tag{3.6}$$

$$\hat{A} = \sigma(ZZ^T) \tag{3.7}$$

where there are $Z \in \mathbb{R}^{|V_i| \times dout}$ and $Z^T \in \mathbb{R}^{dout \times |V_i|}$. The reconstructed adjacency matrix $\hat{A}$ has similarity scores among vertices.

Optimization: After optimizing the discriminator, the encoder is updated by total loss function(Eq. 3.8). the total loss includes three kinds of losses: reconstruction loss, KL divergence loss, and regularization loss.

$$\mathcal{L}_{total} = \mathcal{L}_{recon} + \mathcal{L}_{kl} + \mathcal{L}_{reg} \tag{3.8}$$

Firstly, the reconstruction loss is different between input adjacency matrix A and output(reconstruct) adjacency matrix A. This model use mean square error(MSE) as the reconstruction loss. The reconstruction loss is formulated as follows:

$$\mathcal{L}_{recon} = ||A - \hat{A}||_2 \tag{3.9}$$

where $||A-\hat{A}||_2$ is MSE between input adjacency matrix A and output(predicted) adjacency matrix $\hat{A}$.

Secondly, the KL divergence loss is calculated by the Kullback-Leibler divergence as follows:

$$\mathcal{L}_{kl} = -KL[\ q(Z|X, A)\ ||\ p(Z)\ ] \tag{3.10}$$

where the $KL[\,q(\ )\,||\,p(\ )\,]$ is the Kullback-Leibler divergence between $q(\ )$ and $p(\ )$. If difference between $q(\ )$ and $p(\ )$ is small, the KL value is also big, therefore KL divergence means how difference the two value are. The detail mathematical proof of the loss function is presented in Appendix A.

Lastly, the regularisation loss is computed by updated discriminator, so optimization of the encoder should be carried out after updating discriminator. The regularisation loss is represented in Eq. 3.11.

$$\mathcal{L}_{reg} = -\frac{1}{2}\mathbb{E}_X[log(\mathcal{D}(\mathcal{G}(X, A)))] \tag{3.11}$$

3.4.2 Discriminator

The adversarial VGAE[20] contains a discriminator. This discriminator plays a pivotal role in regularising the encoder in order to create a robust latent feature matrix Z. This discriminator makes a decision of whether a latent vector is from Gaussian random noise(”True”) p_z or from graph encoder(”Fake”) $\mathcal{G}(X, A)$. The discriminator is built on fully connected layers. The discriminator can be seen in the bottom right corner of Fig. 3.3.

Discriminator loss($\mathcal{L}_{disc}$:) Before updating weights of the encoder, the discriminator’s weight is learned by discriminator loss. By minimizing discriminator loss(Eq 3.12), the discriminator can be trained, and the latent vectors z are regularised.

$$\mathcal{L}_{disc} = -\frac{1}{2}\mathbb{E}_{z\sim p_z}[log\mathcal{D}(Z)] + -\frac{1}{2}\mathbb{E}_X[log(1 - \mathcal{D}(\mathcal{G}(X, A)))] \tag{3.12}$$

where $\mathcal{D}(Z)$ and $\mathcal{G}(X, A)$ indicate the discriminator and generator(encoder).

Consequently, the main objective of the adversarial VGAE is to jointly optimise the encoder and the discriminator via a minimax game such that they update each other as follows:

$$\min_G \max_D \mathbb{E}_{z\sim p_z}[log\mathcal{D}(Z)] + \mathbb{E}_{x\sim p(x)}[log(1 - \mathcal{D}(\mathcal{G}(X, A)))] \tag{3.13}$$

3.4.3 Similarity Density Calculation

After completion of training the encoder, a new latent feature matrix Z is generated by the encoder. Then, similarity density from each center node

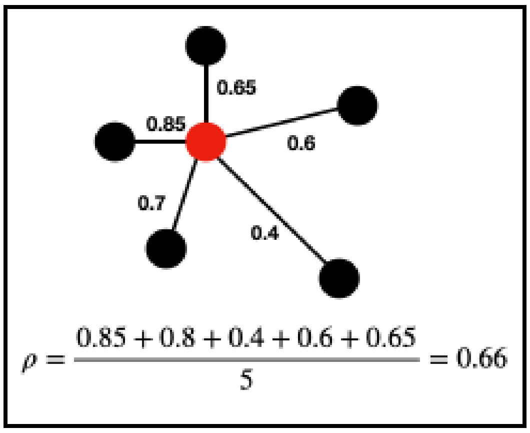


Figure 3.4: [Similarity density calculation] the red color circle is a center node of a sub-graph. Firstly, a center node is selected, then similarity score is calculated for edges between the center node and neighbouring nodes within the pre-determined maximum node. This example has maximum node = 5. Then, the mean of similarity scores is computed for a similarity density for a center node colored red

is calculated by using feature vectors of neighbours, which can take the new latent matrix Z. The example can be seen in Fig. 3.4. The following is the calculation of the similarity density for center node v_i.

$$\rho(v_i) = \frac{1}{N_{c_i}} \sum_{v_j \in N_{c_i}} < z(c_i), z(v_j) > \tag{3.14}$$

where N_{c_i} is neighbor nodes from center node c_i and $z(c_i)$ is the latent feature vector of center node c_i. The number of neighbor nodes is a predefined parameter. $< z(c_i), z(v_j) >$ is dot product between $z(c_i)$ and $z(v_j)$. This is a similarity score between them. Algorithm explanation of similarity density calculation is provided as shown in Alg. 2.

3.5 Construction of 2^{th} Sub-graphs

The 2^{th} Sub-graphs is generated by pruning 1^{th} Sub-graphs based on the similarity density(Eq. 3.14) of center nodes. Let S_{c_i} denote candidate node set of 2^{th} Sub-graphs of center node c_i, and let N_{c_i} denote nearest neighbors from center node c_i. Given the similarity density, the candidate node set $\mathcal{S}_{c_i}$ of 2^{th}

Algorithm 2 Similarity density calculation via variational auto-encoder

```
    Input: $A$ adjacency matrix, $X$ feature matrix
    Output: Similarity density score for each center node
    Require: $N_e$: the number of epochs
             $N_{disc}$: the number of epochs for Discriminator
             $N_{den}$: the number of neighbors for density

 1: for $i = [1, 2, 3, \ldots, N_e]$ do
 2:     Generate $\mu$ and $\log \sigma$ by the encoder including GCN($A(\mathcal{S}_{c_i})$, $H(\mathcal{S}_{c_i})$ layers
 3:     Generate latent matrix $Z$ from $\mu$ and $log\sigma$
 4:     Reconstruct $\hat{A}$ by $\sigma(ZZ^T)$
 5:     for $i = [1, 2, 3, \ldots, N_{disc}]$ do
 6:         Sample m entities $\{z_f^1, z_f^2, z_f^3, \ldots, z_f^{|V|}\}$ from latent feature matrix $Z$
 7:         Sample m entities $\{z_r^1, z_r^2, z_r^3, \ldots, z_f^{|V|}\}$ from Gaussian Random Noise $p_z$
 8:         Update the adversarial model through Eq. 3.12
 9:     end for
10:     Update the encoder with its stochastic gradient by total loss (Eq. 3.8)
11: end for
12:
13: Generate new latent matrix $Z'$ from trained $\mu$ and log $\sigma$
14: for $i = [1, 2, 3, \ldots, N_{den}]$ do
15:     Calculate similarity density of each center nodes based on new $Z'$ by Eq. 3.8
16: end for
```

sub-graphs are selected based on following criteria.

$$\mathcal{S}_{c_i} = \{v_j |\ \rho_j > \rho_{c_i},\ v_j \in N_{c_i}\}, \tag{3.15}$$

where ρ_i and ρ_j are similarity density of center node c_i and neighbor node v_j respectively. If the similarity density of neighbor nodes are bigger than similarity density of the center node, edges between the center node and the neighbor nodes are added for generating 2^{th} sub-graphs. The 2^{th} sub-graphs is represented by the new adjacency matrix $A(\mathcal{S}_{c_i})$.

3.6 Link Prediction Estimator

Given 2^{th} Sub-graphs (denoted by $\mathcal{G}(\mathcal{S}_{c_i})$), variation graph auto-encoder(VGAE) can perform link prediction by using reconstruct the adjacency matrix $\hat{A}(\mathcal{S}_{c_i})$. This is due to the fact that the dot product($\hat{A} = \sigma(ZZ^T)$) gives edges and similarity scores between center nodes and neighbour nodes. Especially, the similarity scores are used for link prediction scores.

Each graph convolution layer of the VGAE takes an adjacency matrix $A(\mathcal{S}_{c_i}) \in \mathbb{R}^{|V_i|\times|V_i|}$ and the feature matrix $H(\mathcal{S}_{c_i}) \in \mathbb{R}^{|V_i|\times d}$ as inputs. $H(\mathcal{S}_{c_i})$ is subtracted from center node feature vector x_{c_i}. The subtracted feature matrix is denoted by $\bar{H}(\mathcal{S}_{c_i})$. The GCN of the link prediction estimator can be represented as follows:

$$\bar{H}^{k+1} = \sigma([\bar{H}^k(\mathcal{S}_{c_i})||Agg(A(\mathcal{S}_{c_i}), \bar{H}^k(\mathcal{S}_{c_i})])W', \tag{3.16}$$

where σ, *Agg(A,H)*, A, and W' are defined similar to those in Eq. 3.2.

This link prediction estimator adopts the same adversarial VGAE of the similarity density estimator with an additional option: Attention-based GCN. The model without attention and model with attention were also implemented, then these models were compared with various previous models in result section. In the rest of this section, the attention-based GCN will be explained. For clarifying the options, the itemized models are presented as follows:

- **[Option 1] Adversarial VGAE**
- **[Option 2] Attention-based VGAE**
- **[Option 3] Adversarial Attention-based VGAE**

These three cases are implemented and related results are provided in Section 6.3.1.

Attention based GCN: This concept is same as GAT[25]. Instead of GCN layer(Eq. 3.2, the attention based GCN is applied to VGAE(called attention-based VGAE). The attention based GCN can be formulated as follows:

$$e_{i,j} = a[Wh_i^k, Wh_j^k] \tag{3.17}$$

Where e_{ij} is the relevance score of a neighboring node j to node i.

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{exp(e_{ij})}{\sum\limits_{k \in N_i} exp(e_{ik})} \tag{3.18}$$

$$\alpha_{ij} = \frac{exp(LeakyReLU(a^T[Wh_i^k || Wh_j^k]))}{\sum\limits_{k \in N_i} exp(LeakyReLU(a^T[Wh_i^k || Wh_k^k]))} \tag{3.19}$$

Where $||$ is the concatenation operation. The $a \in \mathbb{R}^{2*dout}$ is a weight vector for the attention mechanism.

$$h_i^{k+1} = ||_{l=1}^{L} \sigma(\sum_{j \in N_i} \alpha_{ij}^l W^l h_j) \tag{3.20}$$

Where $||$ represents concatenation for multi-head attention. L is the number of multi-head attention in order to stabilise the learning process of self-attention. Except attention based GCN, this attention-based graph auto-encoder use the same encoder, decoder and loss function, which is adopted in the similarity density estimator(Section 3.4).

For better understanding, the overall procedure of the link prediction estimator can be represented in algorithm 3

Algorithm 3 Link prediction estimator

```
    Input: $A(\mathcal{S}_{c_i})$ adjacency matrix, $H(\mathcal{S}_{c_i})$ feature matrix,
    Output: A list of edges, A list of similarity score
    Require: $N_e$: the number of epochs for VGAE
             $N_d$: the number of epochs for Discriminator
 1: for $i = [1, 2, 3, \ldots, N_e]$ do
 2:     Generate $\mu$ and $\log \sigma$ by the encoder including GCN($A(\mathcal{S}_{c_i})$, $H(\mathcal{S}_{c_i})$ layers
 3:     Generate latent matrix $Z$ from $\mu$ and $log\sigma$
 4:     Reconstruct $\hat{A}$ by $\sigma(ZZ^T)$
 5:     for $i = [1, 2, 3, \ldots, N_d]$ do
 6:         Sample m entities $\{z_f^1, z_f^2, z_f^3, \ldots, z_f^{|V|}\}$ from latent feature matrix $Z$
 7:         Sample m entities $\{z_r^1, z_r^2, z_r^3, \ldots, z_f^{|V|}\}$ from Gaussian Random Noise $p_z$
 8:         Update the adversarial model through Eq. 3.12
 9:     end for
10:     Update the encoder with its stochastic gradient by total loss (Eq. 3.8)
11: end for
12: Generate new latent feature matrix $Z$
13: Gather similarity scores and edges between each center node and its neighbors
```

3.7 Link Merging

Given the similarity scores and edges(c_i, v_j), the similarity scores should be utilised for finding final clusters. For every instances c_i $\{i|i = 1, 2, \ldots, N\}$, c_i is connected to $v_j(j \in N_i)$ if the similarity score between c_i and v_j is above a threshold. The final clusters can be found by merging every connected edges.

Chapter 4

Dataset and Preprocessing

4.1 Dataset

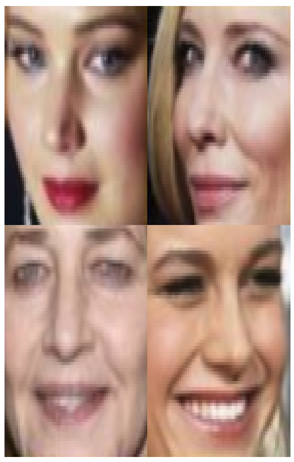
4.1.1 MS-CELE-1M dataset

Microsoft Celebrity One(1) Million(MS-CELE-1M) dataset [8] is a large-scale dataset of celebrities face images with more than 10K identities. To be more specific, there are 10M number of face images with 100K number of identities, moreover the dataset are offered with diverse pose variations, face expression, countless angles, and various styles. The dataset also have been offered as open data set.

Numerous research has extensively used these celebrities facial images for various face image applications. For example, one representative of the diverse applications is face recognition(face detection) and feature extraction of facial images, in addition, many face clustering research have been used. Therefore, This is a great dataset for comparison with previous face clustering models. This is a reason why this paper selected this MS-CELE-1M dataset. The example face images of the MS-CELE-M1 dataset is provided as shown in Fig. 4.1 (a).

4.1.2 Youtube Faces Dataset

Youtube faces data set[29] is shared for research of unconstrained face recognition. The data set is generated from more than 1.5K different identities in


(a) MS-CELE-M1

(b) Youtube Face

Figure 4.1: Left: face images of the MS-CELE-1M dataset Right: face iamges of youtubeface dataset

more than 3K videos, which were provided from YouTube. To be more specific, The data set includes 1,595 different people in 3,425 videos whose range of video clip duration is from 48 frames to 6,070 frames. The average duration of the clip is 181.3 frames. The example face images of Youtube face dataset is provided as shown in Fig. 4.1 (b)

Using the faces data set can be equal to dealing with problems of video analysis related to faces because each video frame also is considered as an image. In this point of view, the Youtube faces data set has been famous for studying face recognition issues.

4.2 Preprocessing face images

4.2.1 Feature Extraction

To solve the face clustering problem, face images should be transformed to low dimensional feature vectors, which is n-dimensional vectors of numerical features which can represent an object's characteristics or measurable property. Each pixel of an image is also considered as a feature in the case of image data. For example, if there is a 64 x 64 face image, 4,096 pixels can be considered as individual properties. However, these countless features(all pixels) are difficult to accurately classify the images or distinguish the images from the others. In other words, a vector with countless features hardly catches the facial characteristics. To solve the problem, lower-dimensional vectors with key features should be extracted from raw image data with high dimensional features such as whole pixels. vectors with key features allow the images to be accurately distinguished. How are these vectors converted from face images?

The solution to this issue is to utilise neural networks of face recognition. The use of Convolutional Neural Network(CNN) for this problem has been used in much research. CNN enable the facial images to be converted to spatial features with several advantages, which are sharing weights, creating latent features, using an image as an input. However, like conventional deep neural networks, the deeper network layers are the more possibility increases regarding a vanishing gradient problem or an exploring problem.

4.2.2 DenseNet

To overcome the problems of CNN, skip connections between layers is proposed. The skip connection is that outputs of a current layer are feed into inputs of the next layers after skipping some layers between the current layer and the next layer. This ResNet architecture allows the CNN networks to have more layers by solving the vanishing problem such as ResNet50.

To improve the performance of the ResNet, DenseNet[11] is suggested for the improvement of model performance. DenseNet has an additional skip connection between layers compared to ResNet as illustrated by Fig. 4.2. Each layer can obtain collective knowledge from each layer image. Therefore, to achieve improvement in the accuracy of the model, This paper will make use of the DenseNet-121 for feature extraction to represent the face images in vector space. There is the DenseNet-121 Structure which is used as shown in Table 4.1. By using the DenseNet-121, 256-dimensional feature vectors were extracted for allocating the face images to feature space(latent vector space).

t-distribution Stochastic Neighbor Embedding(t-SNE) was used to display latent image vectors in a scatter plot. The t-SNE is a nonlinear dimensionality reduction method in which the high dimensional vectors are transformed into low dimensional vectors. For the purpose of showing the plot, the dimen-

Layers	**DenseNet-121**
Convolution	7×7 conv, stride 2
Pooling	3×3 max pool, stride 2
Dense block(1)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer(1)	1 x 1 conv 2 x 2 average pool, stride 2
Dense block(2)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer(2)	1 x 1 conv 2 x 2 average pool, stride 2
Dense block(3)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Transition Layer(3)	1 x 1 conv 2 x 2 average pool, stride 2
Dense block(4)	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$
Classification Layer	7 x 7 global average pool 1000D fully-connected, softmax

Table 4.1: DenseNet-121 architectures for feature extraction. Note the each "conv" layer indicates the sequence Batch Normalisation-ReLU-Convolution

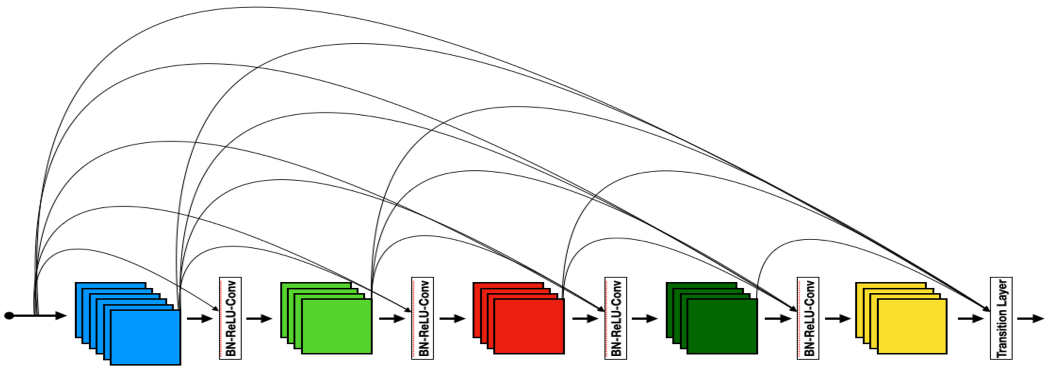
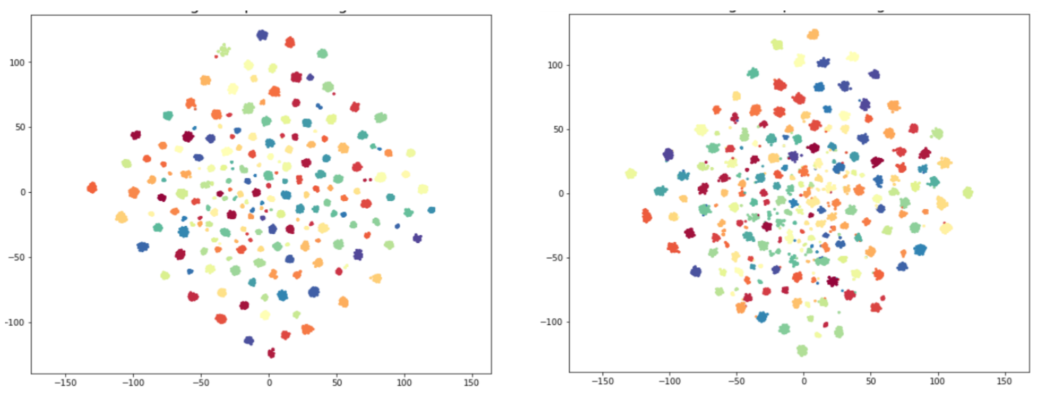


Figure 4.2: a 5-layer DenseNet

sionality of the extracted feature vectors is two(2) because the scatter plot has 2 dimensions. For example, the training dataset has 13,180 instances of 200 different people, and the testing dataset has 13,810 instances of 200 different people. The example is shown in Fig. 4.3. Each colour means different identities.



(a) MS-CELE-1M train dataset

(b) MS-CELE-1M test dataset

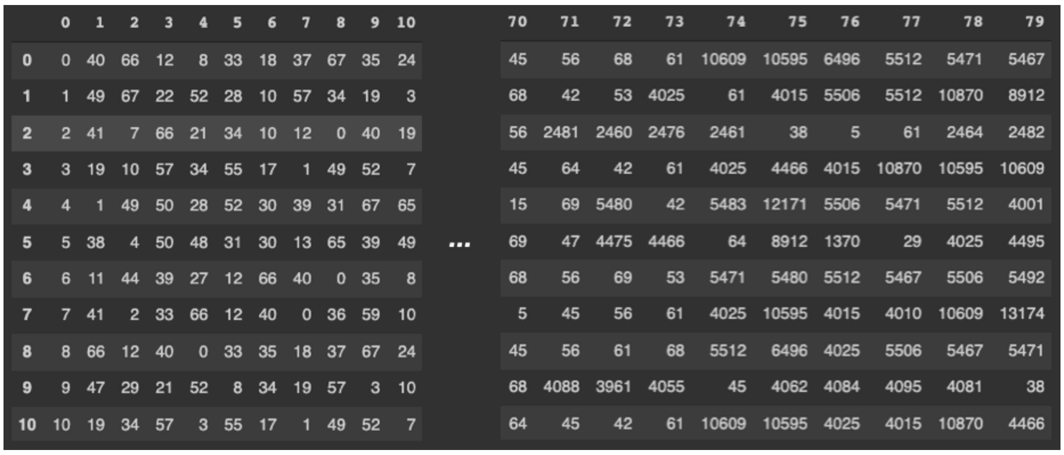
Figure 4.3: Left: scatter plot of t-sne dimension reduction for training dataset(first 200 clusters), Right: scatter plot of t-sne dimension reduction for testing dataset(first 200 clusters)

4.2.3 k-Nearest Neighbors matrix

It is necessary that the k-Nearest Neighbors(kNNs) of each node is computed bacause a sub-graph are generated based on the kNNs of a center nod, which are likely to connect to the center node. In this paper, kNNs were computed by a feature vector extracted from a face image.

The kNN was computed on the basis of cosine similarity, which is between two vectors of an inner product. To be specific, if a value of cosine similarity between two feature vectors is close to 1, the two feature vectors are similar and close to each other in its vector space, whereas if a value of cosine similarity between them is close to 0, the two feature vectors are dissimilar and far from each other in its vector space.

For the sake of building kNNs, this paper utilised Facebook AI Similarity Search(Faiss) library, which enable the code to quickly search for nodes with cosine similarity. As a result, by using the 256 dimension feature vectors extracted from the face images, the cosine similarity among all nodes was calculated, then kNN(k=80) of each node was computed in order of closest distance(cosine similarity close to 1) through Faiss as shown in Fig. 4.4.



	0	1	2	3	4	5	6	7	8	9	10		70	71	72	73	74	75	76	77	78	79
0	0	40	66	12	8	33	18	37	67	35	24		45	56	68	61	10609	10595	6496	5512	5471	5467
1	1	49	67	22	52	28	10	57	34	19	3		68	42	53	4025	61	4015	5506	5512	10870	8912
2	2	41	7	66	21	34	10	12	0	40	19		56	2481	2460	2476	2461	38	5	61	2464	2482
3	3	19	10	57	34	55	17	1	49	52	7		45	64	42	61	4025	4466	4015	10870	10595	10609
4	4	1	49	50	28	52	30	39	31	67	65		15	69	5480	42	5483	12171	5506	5471	5512	4001
5	5	38	4	50	48	31	30	13	65	39	49	...	69	47	4475	4466	64	8912	1370	29	4025	4495
6	6	11	44	39	27	12	66	40	0	35	8		68	56	69	53	5471	5480	5512	5467	5506	5492
7	7	41	2	33	66	12	40	0	36	59	10		5	45	56	61	4025	10595	4015	4010	10609	13174
8	8	66	12	40	0	33	35	18	37	67	24		45	56	61	68	5512	6496	4025	5506	5467	5471
9	9	47	29	21	52	8	34	19	57	3	10		68	4088	3961	4055	45	4062	4084	4095	4081	38
10	10	19	34	57	3	55	17	1	49	52	7		64	45	42	61	10609	10595	4025	4015	10870	4466

Figure 4.4: k-NNs(k=80) matrix form. There are only nodes' number in order of closest distance. The number of rows means center node numbers.

Chapter 5

Experiment Setup

Two main problems of implementation are requiring high computational cost and a lack of memory space. The high computational cost often appears that considerable complexity is required by many layers and structures of neural networks which include convolution neural networks and graph convolution networks. In addition, the shortage of memory space basically is a cause of high dimensional parameters. If the architecture of the model has deep neural network depth, each layer should basically need numerous parameters for learning weights of the networks. Therefore, it is important that hardware specification and appropriate software are selected for developing the proposed model.

In this chapter, the hardware specification will be presented, moreover, programming language and frameworks will be explained after the specification.

5.1 Hardware Specification

both CPU and GPU have been used for the implementation of training and testing regarding the deep learning model because GPU running should be utilised for treating the computational cost problem. Especially, for GPU running in the Python environment, the Pytorch CUDA and cuDNN programming library has been installed and utilised. To use better computer specification, this paper use Colab, which enable to implement a Python code in a browser with high-quality GPUs and RAM. The detailed hardware specification of Colab is presented as shown in the following table 5.1

Table 5.1: Hardware Specification

Item	Specification
System	Macintosh
Python	3.7.11 version
Frameworks	Pytorch-GPU
CPU	Intel(R) Xeon(R) CPU @ 2.20GHz
GPU	Tesla P100-PCIE-16GB
RAM	13 GB

5.2 Programming Language and Library

Python is an interactive objected oriented programming language with remarkable use of significant indentation. In recent years, Python has become famous in the data science field because Python has the numerous libraries that data scientists easily utilise for the analysis or development of deep learning models. This study uses Python language.

In addition, in this paper, several libraries have been used such as Pandas(easily data analysis), Numpy(offering comprehensive mathematical functions), and Matplotlib (comprehensive visualisation library). For the execution of baseline models, this project has utilised the sklearn library which contains various clustering models such as kmean, DBSCAN, and spectral clustering.

Especially, to implement the deep learning model, this study uses PyTorch library. The PyTorch allows users to easily develop deep neural networks(tape-based autograd system), and the Pytorch provides tensor computation with strong GPU acceleration. In this study, every deep learning model are executed by the Pytorch.

To speed up the development of the proposed model, Jupyter notebook with Python has been used for interactive computing at the development initial stage because results of Python programming can be easily shown in the virtual notebook in near real-time.

5.3 Baseline models

This study compared the proposed model with various baseline models. The baseline models are implemented by methods in the Sklearn library and previous work. Please refer to the following references if a detail description is needed.

- **Kmeans:** https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans
- **DBSCAN:** https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift
- **Spectral:** https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering
- **Meanshift:** https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift
- **HAC:** https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering
- **ARO:** https://github.com/KunpengWang/approximate-rank-order-clustering
- **LGCN:** https://github.com/yl-1993/learn-to-cluster/tree/master/lgcn

Chapter 6

Experimental Results

This section will provide the results of experiments that are carried out for comparison between the proposed model and baseline model. The results of clustering are based on clusters having the best link prediction, which link is connected if the link prediction score is above a predefined link threshold. Results regarding various graph auto-encoder models will be presented. In addition, training model setup will be explained in this chapter. This chapter is organized as follows:

The evaluation metrics will be explained in Section 6.1. The training model setup will be described in Section 6.2.2. Results for various graph auto-encoder are presented in Section 6.3.1. The section 6.3.2 provides the results of comparing the proposed model with baseline models.

6.1 Evaluation Metrics

To measure the performance of the proposed method, this study adopts three main evaluation metrics: pairwise clustering metric, Bcubed clustering metric, and normalized mutual information(NMI).

Pairwise Clustering Evaluation: This study have evaluated the proposed model by using the pairwise clustering evaluation metrics defined in rank order face clustering[19]. In this metric, ”pair” means possible pairs in predicted clusters and correct labels. This pairwise evaluation metric allows the model to be evaluated in terms of clusters corresponding to known identity labels and can measure for clustering quality based on predicted identity labels. This pairwise metric includes the pairwise precision, pairwise recall, and pairwise f1 score. Definitions of these pairwise metrics are provided as follows:

Pairwise Precision(Eq. 6.1) is defined as the ratio between the total number of possible pairs in all clusters and the total number of correct pairs in the clusters. In Figure 6.1, (A1, A2) and (A2, B1) are possible pairs in the left cluster, whereas (A1, C1) and (A2, C1) are impossible pairs. In this case, the total number of correct pairs of clusters is 2, and the total number of pairs in all clusters is 6. Therefore, pairwise precision is 2/6(abt 0.333).

$$Pairwise\ Precision = \frac{the\ total\ correct\ pairs\ of\ clusters}{Total\ pairs\ in\ all\ clusters} \tag{6.1}$$

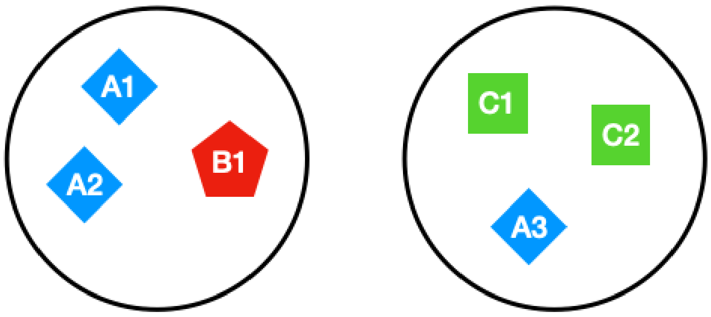


Figure 6.1: Diagram of a possible clustering configuration. There are two clusters(left and right circle); A1, A2, and A3 have same an identity, and B1 has different identity. C1 and C2 also have a different identity.

Pairwise Recall(Eq. 6.2) is defined as the ratio between the total number of possible pairs in the same identity and the total number of correct pairs in the clusters. In Figure 6.1, (A1, A2), (A2, A3), and (A3, A1) are possible pairs, while (A1, B1) and (A2, B1) are not possible pairs. In this case, the total number of correct pairs of clusters is 2, and the total number of pairs in the same identity is 4. Hence, pairwise recall is 2/4(0.5).

$$Pairwise\ Recall = \frac{the\ total\ correct\ pairs\ of\ clusters}{Total\ pairs\ of\ all\ identities\ in\ dataset} \tag{6.2}$$

Pairwise F1 score is defined in Eq. 6.3 ,which summarising the pairwise precision and the pairwise recall.

$$Pairwise\ F1\ score = \frac{2(PairwisePrecision \times PairwiseRecall}{(PairwisePrecision + PairwiseRecall)} \tag{6.3}$$

B-cubed Clustering Evaluation: The B-cubed score[1] is used as a clustering evaluation metric. The Bcubed F-measure is a practical measurement of clustering system effectiveness because this metric satisfies all constraints of clusters. Let this study denote $L(i)$ and $C(i)$ as identity label and cluster label respectively. The correct elements are defined as follows:

$$Correct(i,j) = \begin{cases} 1, if\ L(i) = L(j)\ and\ C(i) = C(j) \\ 0, otherwise \end{cases} \tag{6.4}$$

The *Bcubed Precision* and *Bcubed Recall* are respectively defined as Eq. 6.5. Like pairwise F1 score, the *Bcubed F1 score* is defined as Eq. 6.6.

$$\begin{aligned} Bcubed\ Precision &= E_i[E_{j:C(j)=C(i)}[Correct(i,j)]] \\ Bcubed\ Recall &= E_i[E_{j:L(j)=L(i)}[Correct(i,j)]] \end{aligned} \tag{6.5}$$

$$Bcubed\ F1\ score = \frac{2(BcubedPrecision \times BcubedRecall}{(BcubedPrecision + BcubedRecall)} \tag{6.6}$$

Normalized Mutual Information(NMI): The NMI is based on entropies of identities and clusters. Let this study denote L and C as labels(identity) and clusters. The NMI is defined as Eq. 6.7.

$$\begin{aligned} H(L) &= \sum_{l \in L} -P(l)logP(l) \\ H(C) &= \sum_{c \in C} -P(c)logP(c) \\ I(L,C) &= H(L) - H(L|C) \\ NMI(L,C) &= \frac{I(L,C)}{\sqrt{H(L)H(C)}}, \end{aligned} \tag{6.7}$$

where $H(L)$ and $H(C)$ mean the entropies for identities and clusters, and $I(L,C)$ is the mutual information.

6.2 Training Model Setup

In this section, the experimental setup will be explained with a description of auto-encoders, which are used in similarity density estimator and link prediction estimator. Firstly, loss functions that are used in the auto-encoders will be described in Section 6.2.1. Secondly, structures of the auto-encoders will be presented with their training results in Section 6.2.2.

6.2.1 Loss function

For optimizing models and discriminator, loss function are used according to Table 6.2. The loss functions contain four terms such as reconstruction loss, KL divergence loss, discriminator loss and regularization loss. At first, the mean square roots(MSE) (Eq.6.8) is difference between input adjacency matrix A and output adjacency matrix $\hat{A}$, which is predicted from a decoder. This loss means how similarly the adjacency matrix $\hat{A}$ is made from extracted latent feature matrix Z. Secondly, KL divergence is the difference between Gaussians for encoder and noise, which has random Gaussian distribution. The KL divergence was simplified and applied to the loss function as described in Eq. 6.9. The detailed mathematical proof can be seen in Appendix A. Thirdly, the discriminator loss is used for training the discriminator. The discriminator loss has two probabilities: the incorrect prediction of a latent matrix Z(fake) and the correct prediction of Gaussian random noise(real). For using gradient descent(minimising the loss), the mathematical formation is applied as described in Eq. 6.10. Lastly, to train the encoder, the regularised loss(Eq. 6.11) is applied.

$$\textit{reconstruction loss } \mathcal{L}_{recon} = \frac{1}{N}\sum(A-\hat{A})^2 \tag{6.8}$$

$$\textit{simplified KL divergence } \mathcal{L}_{kl} = -\frac{1}{2}(1+(log\sigma)^2-\mu^2-\sigma^2) \tag{6.9}$$

$$\begin{aligned} &\textit{Let } Z_{latent} \textit{ be a latent feature matrix} \\ &\textit{Let } Z_{noise} \textit{ be a Gaussian random noise} \\ \textit{discriminator loss } \mathcal{L}_{disc} &= -log(Z_{noise}) - log(1-Z_{latent}) \end{aligned} \tag{6.10}$$

Table 6.1: Various loss functions for update weights of Encoder

Model	loss function
GAE	$\mathcal{L}_{recon}$
VGAE	$\mathcal{L}_{recon} + \mathcal{L}_{kl}$
Attention-based VGAE	$\mathcal{L}_{recon} + \mathcal{L}_{kl}$
Adverserial VGAE	$\mathcal{L}_{recon} + \mathcal{L}_{kl} + \mathcal{L}_{reg}$
Adverserial Attention-based VGAE	$\mathcal{L}_{recon} + \mathcal{L}_{kl} + \mathcal{L}_{reg}$
Discriminator	$\mathcal{L}_{disc}$

$$regularisation\ loss\ \mathcal{L}_{reg} = -log(Z_{latent}) \tag{6.11}$$

6.2.2 Training Auto-encoder

This section will explain a detailed architecture of the variational auto-encoder including its optimizer, which was implemented. The encoder of the auto-encoder has 3 layers of graph convolution network(GCN). One out of three layers is used for generating mean matrix μ and log standard deviation matrix log σ. input and output dimensions of the first layer are 256, 128 dimensions respectively, and input and output dimensions of the second layer are 128, 64 dimensions respectively. The last layers have equal dimensionalities between the input and the output layer. These layers are used with 64 dimensions for calculating mean matrix μ and log standard deviation matrix log σ respectively. The reduced dimensionality(64 dimensions) can make the feature vectors to be distinguishable and have meaningful features. To specifically explain the GCN layer, batch normalisation is carried out before computing GCN layers. Then, each GCN layer computes $H^{k+1} = \sigma([H^k||Agg(A, H^k)])W$. After each GCN layer, the output matrix of each layer goes through ReLU activation function and dropout layer. the dropout has 0.2 probability. weights and biases also are used. Decoder of the auto-encoder used $A = \sigma(ZZ^T)$, where σ is Sigmoid function. Lastly, the Adaptive Moment Estimation(Adam) optimizer is used for training layers. The Adam optimizer has 0.01 learning rate. The detailed mathematical formation is presented in Eq. 6.12.

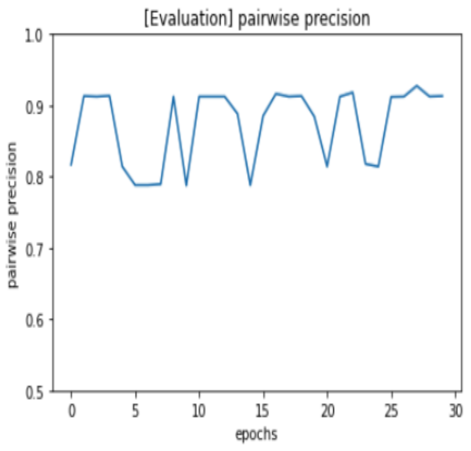
$$\begin{aligned} \hat{m_t} &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ \hat{v_t} &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v_t} + \epsilon}} \hat{m_t} \end{aligned} \tag{6.12}$$

where η is a learning rate and ϵ is 10^{-8}. The Adam optimizer can compute the decaying averages of past and past squared gradients m_t and v_t respectively.

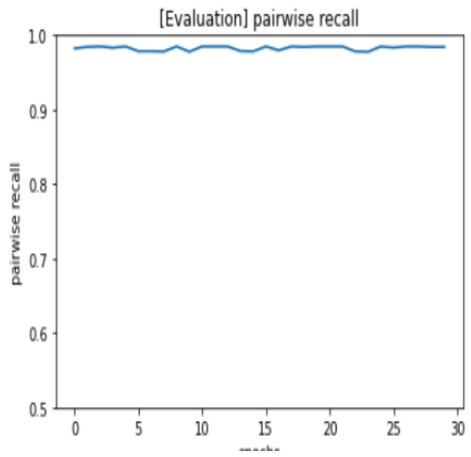
Discriminator Option: In the case of the discriminator option, the linear neural networks are used for the discriminator. The discriminator network has 3 layers. The input layer has 64 dimensions and 128 dimensions as input and output dimensions individually. The hidden layer has 128 dimensions, and the output layer uses 128 dimensions and 64 dimensions as input and output dimensions. ReLU activation functions and dropout(0.1 possibility) are used among layers. The increased dimension layers allow the model to calculate possible combinations among features.

Attention base GCN Option: The attention base GCN options has three GCN layers. Two out of three layers are used for finding the mean matrix μ and standard deviation matrix log σ which are the same as the variational auto-encoder. Unlike the variational auto-encoder, attention scores among nodes are calculated in each layer. LeakyReLU activation function is used for attention score. This attention base GCN also is used with multi-head attention. the number of attention heads is eight(8), and the multi-head attentions are concatenated instead of averaging. The output of the graph attention goes through a nonlinear activation function(sigmoid).

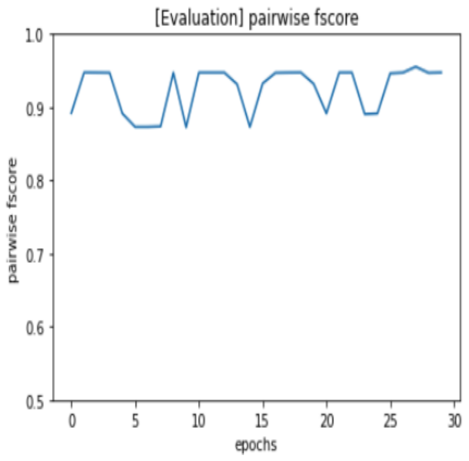
The following graphs are the results of training the simple variational model. The model's evaluation metrics will be explained in the section 6.1. The first graphs show the pairwise precision, recall, and f1 score as shown in Fig. 6.2. The second graphs are the Bcubed precision, recall, and f1 score as illustrated by Fig. 6.3.



(a) Pairwise Precision

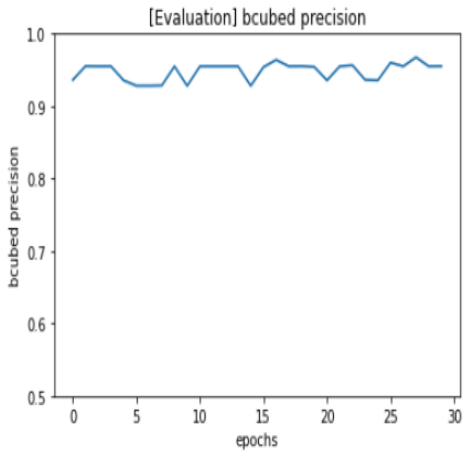


(b) Pairwise Recall

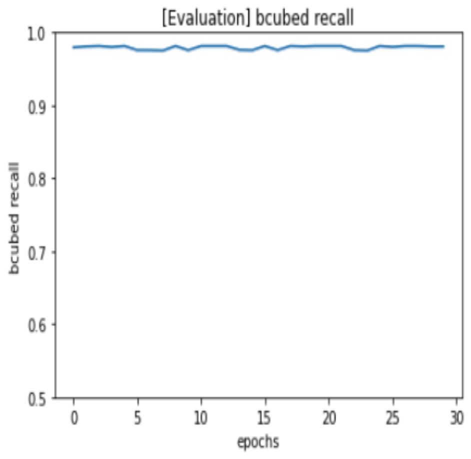


(c) Pairwise F1 score

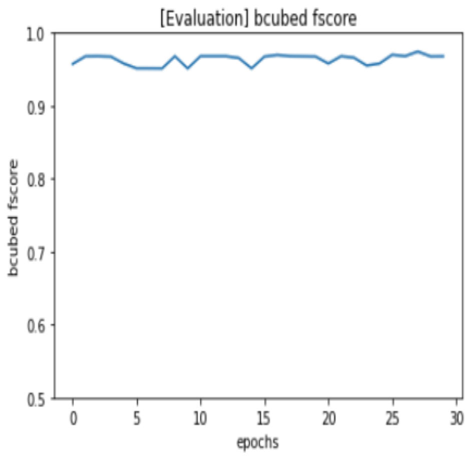
Figure 6.2: For variational auto-encoder, train pairwise precision, recall, and f1 score graph (X axis - the number of epochs, Y axis - score)



(a) bcubed Precision



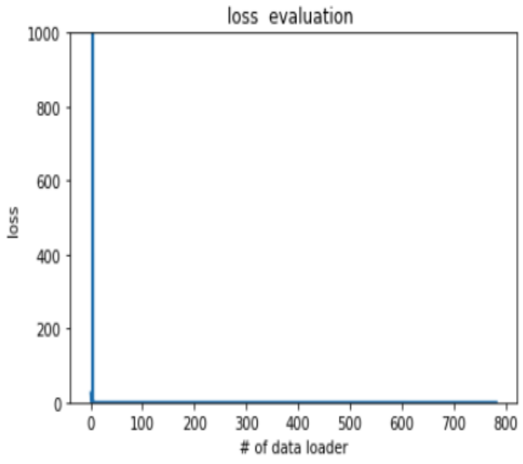
(b) bcubed Recall



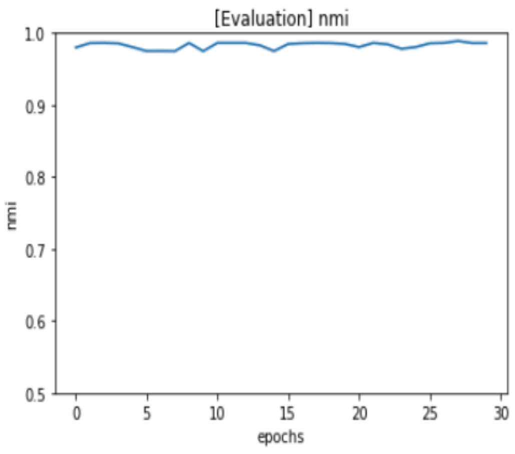
(c) bcubed F1-score

Figure 6.3: For variational auto-encoder, train bcubed precision, recall, and f1 score graph (X axis - the number of epochs, Y axis - score)

The last graphs regarding normalised mutual information(NMI) and loss can be found in Fig. 6.4.



(a) training loss



(b) NMI

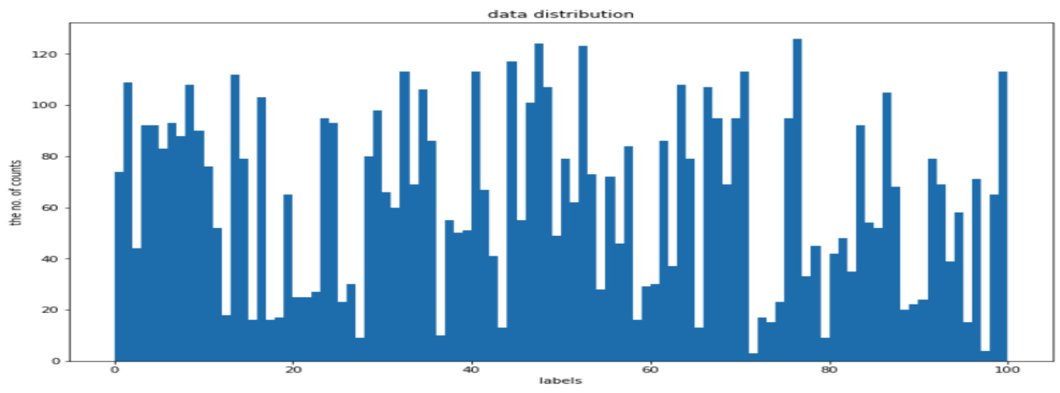
Figure 6.4: For variational auto-encoder, train loss and normalised mutual information (X axis - the number of epochs, Y axis - score)

6.3 Results

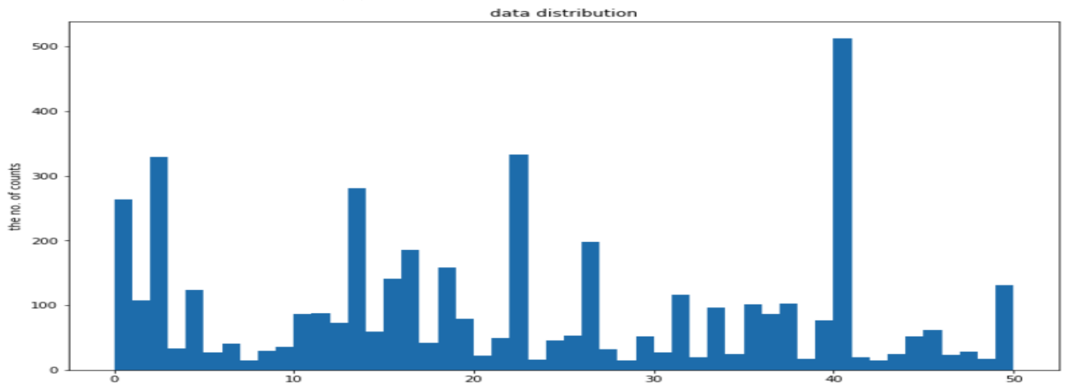
This section will show the results of comparing the proposed model with various graph auto-encoder and baselines. The main code of the proposed model is presented in Appendix B

Dataset: As explained in Section 4.1, this study used two datasets: MS-CELE-M1 and Youtube face. Because these datasets are too large scale, this experiment makes use of sliced datasets of these two datasets. To be specific, given MS-CELE-M1, the training and test dataset have 6,270 instances with 100 classes(identities) and 6,683 instances with 100 classes(identities). These feature vectors' dimensions are 256 dimensions. Regarding Youtube face, the training and test dataset has 4,413 faces and 4,560 faces with the first 50 classes respectively. Figure 6.5 shows histograms for each label data. The histograms present that the datasets are imbalanced. In addition, the features of MS-CELE-1M data are more distinguishable than the feature of Youtube Face data as shown in Fig. 6.6. Youtube face dataset is difficult to be separated by using features extracted from Youtube video because the Youtube face is made by capturing a moving object(person) at a certain frame and has various angles of faces.

Parameter Selection: There are three parameters of generating sub-



(a) MS-CELE-1M label for test



(b) Youtube Face labels for test

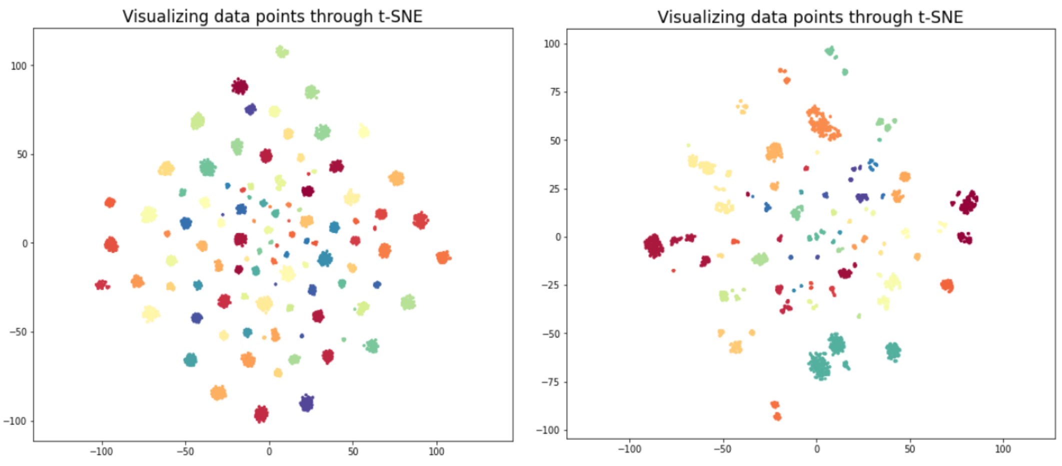
Figure 6.5: Up: Histogram of sliced MS-CELE-1M dataset (first 100 clusters), Down: Histogram of sliced Youtube face dataset (first 50 clusters)

graph: the number of hops h, the number of picked nearest neighbours in each hop h_i, and the number of possible connection to neighbouring nodes is u. Because the sliced datasets are small scale, this experiment chose $h_1 = 6, h_2 = 5$, and $u = 5$ for MS-CELE-M1 dataset and $h_1 = 10, h_2 = 5$, and $u = 5$ for Youtube face dataset. In the attention models, the number of the multi-head is 8.

6.3.1 Result: Comparison Various Graph Auto-encoders

Model comparison: This section compare the proposed method with various graph auto-encoders. These graph auto-encoders are briefly described below:

(1) VGAE[13], The variational graph auto-encoder used mean μ and stan-



(a) MS-CELE-1M test data (b) Youtube face data

Figure 6.6: Left: a scatter plot for features of sliced MS-CELE-1M dataset (first 100 clusters), Right: Scatter plot for features of sliced Youtube face dataset (first 50 clusters)

dard deviation σ with Gaussian Random Noise in order to create the latent feature matrix Z.
(2) VGAE[13] **with simple GCN**, This model apply simple GCN($GCN = AXW$) to the VGAE instead of concatenated GCN(Eq. 3.2).
(3) Adversarial VGAE[20], The adversarial VGAE is the VGAE adopting a discriminator. The model is optimized via a minimax game(loss function).
(4) Attention[25] **VGAE**, The attention VGAE uses self-attention GCN to utilse the relevance with neighbor nodes.
(5) Adversarial Attention-based VGAE[28], This model uses both a discriminator and attention-based GCN in the auto-encoder.

Results: The result of various graph auto-encoders can be seen in Table. 6.2. (1) The VGAE has excellent clustering performance in both pairwise and bcubed evaluation metrics, moreover, the training and testing time is acceptable. (2) In the VGAE with simple GCN, the reduced number of parameters is seen compared with the first VGAE model(using concatenated GCN). However, most evaluation metrics slightly decrease. This means that the concatenated GCN(GraphSage[9]) bring better performance because concatenated feature elements act as additional features(increasing feature dimen-

Table 6.2: benchmark for various graph auto-encoders (M1-CELE-1A 100 clusters)

Method	Train Time (sec)	Pair Pre	Pair Rec	Pair F1	bcubed Pre	bcubed Rec	bcubed F1	NMI	Test Time (sec)	No. of Para
VGAE (94 clusters)	16.72	0.8065	0.9905	0.8891	0.9110	0.9878	0.9479	0.9806	11.90	98,624
VGAE with simple GCN (106 clusters)	18.41	0.7889	0.9860	0.8765	0.9201	0.9847	0.9513	0.9804	23.15	49,472
Adversarial VGAE (96 clusters)	24.96	0.8331	0.9892	0.9045	0.9283	0.9858	0.9562	0.9829	12.93	131,712
Attention VGAE (101 clusters)	2695.80	0.8641	0.9870	0.9215	0.9285	0.9841	0.9555	0.9837	256.14	1,549,632
Adversarial Attention based VGAE (109 clusters)	2622.87	0.7368	0.9800	0.8412	0.8972	0.9785	0.9361	0.9733	234.54	**1**,582,720

sionality). (3) In Adversarial VGAE, the pairwise and bcubed precision are slightly higher than the VGAE, whereas the number of parameters increases and training time also roses. (4) Attention-based VGAE bring better performance compared with others, however the number of parameters, training time, and test time considerably increase. This result is caused by the calculation of attention score and its weights in multi heads. (6) Like attention-based VGAE, the number of parameters and time-consuming roses, however this model brings disappointing performance as seen in the result table.

Discussion: As a result, it seems that the attention-base VGAE can bring better performance, however, there is a disadvantage in memory space because of too many parameters. In addition, this model needs to take a lot of execution time. This trade-off made a decision that the proposed model considered a method without the self-attention structure. On the other hand, it is evident that the adoption of a discriminator and concatenated GCN allows the model to be improved. Therefore, this study uses the adversarial VGAE with concatenated GCN as a base model of the proposed method.

6.3.2 Result: Proposed models vs baseline

Model comparison: This section compare the proposed method with various graph auto-encoders. These graph auto-encoders are briefly described below:
(1) K-means[16], the k-means is usually used clustering method. K-means algorithm required deciding the number of clusters. Therefore, the performance is depended on the number of clusters.
(2) DBSCAN[6], DBSCAN is commonly used for complex data manifold. The algorithm is carried out based on the maximum distance(epsilon) between two samples and the number of samples in a neighbourhood.
(3) HAC[24], Hierarchical clustering merges close clusters by linkage criteria, then cut the cluster based on a predefined distance.
(4) Spectral[18], The spectral clustering utilises the eigenvalues of matrices built from the graph.
(5) ARO[19], The ARO performs clustering with an approximate nearest neighbours by using the rank order distance.
(6) MeanShift[4], Meanshift performs clustering by shifting points with the highest density of data points.
(7) LGCN[27], Linkage based face clustering use linkage prediction via graph convolution networks by merging sub-graphs. In this experiment, the pre-trained model was used for test.
(8) Adversarial VGAE, This is the proposed model which includes a discriminator to regularise the latent feature vectors.
(9) Adversarial VGAE using similarity density, This is the adversarial VGAE using similarity density to prune sub-graphs. The similarity densities of each instance are calculated by an additional adversarial VGAE model.
(10) Adversarial Attention-based VGAE using similarity density, The adversarial VGAE using similarity density(Model No.(9)) adopted attention-based GCN.

Results: Hyper-parameters were turned for all baseline methods. The result of the comparison with baselines can be seen in Table. 6.3 and 6.4. Table 6.3 shows the results of the MS-CELE-1M dataset, and the table 6.4 indicate the results of the Youtube face dataset. (1) Given the true number of clusters, the K-means obtains high F1 scores (pairwise and bcubed) and nmi in MS-

Table 6.3: benchmark for baseline (M1-CELE-1A first 100 clusters)

Method	Pair Pre	Pair Rec	Pair F1	bcubed Pre	bcubed Rec	bcubed F1	NMI	Time (sec)
kmeans (100 clusters)	0.9755	0.9259	0.9501	0.9655	0.9344	0.9497	0.9823	12.2809
DBSCAN (108 clusters)	0.9254	0.9891	0.9562	0.9616	0.9827	0.9720	0.9895	3.4354
Spectral (97 clusters)	0.4997	0.8062	0.6170	0.6979	0.8273	0.7571	0.9112	26.3207
ARO (86 clusters)	0.7035	0.9959	0.8245	0.8918	0.9960	0.9410	0.9697	24.9140
MeanShift (3192 clusters)	1.0000	0.1393	0.2446	1.0000	0.1357	0.2389	0.7435	384.5784
HAC (125 clusters)	0.7753	0.9922	0.8705	0.8972	0.9867	0.9398	0.9789	9.1503
LGCN (76 clusters)	0.7462	0.9999	0.8546	0.8422	0.9991	0.9140	0.9682	87.3746
Adversarial VGAE (96 clusters)	0.8645	0.9892	0.9226	0.9287	0.9857	0.9563	0.9843	11.9607
Adversarial VGAE with similarity density (105 clusters)	0.7528	0.9867	0.8540	0.9134	0.9842	0.9475	0.9779	12.9607

Table 6.4: benchmark for baseline (Youtube Face first 50 clusters)

Method	Pair Pre	Pair Rec	Pair F1	bcubed Pre	bcubed Rec	bcubed F1	NMI	Time (sec)
kmeans (50 clusters)	0.9144	0.6480	0.7585	0.8778	0.7348	0.8000	0.9140	6.5579
DBSCAN (50 clusters)	0.6860	0.9667	0.8025	0.7963	0.9564	0.8690	0.9364	3.2351
Spectral (50 clusters)	0.2840	0.3449	0.3115	0.4342	0.4481	0.4410	0.6756	14.0789
ARO (15 clusters)	0.1290	0.9179	0.2261	0.4858	0.9474	0.6422	0.6400	57.4009
MeanShift (603 clusters)	1.0000	0.4650	0.6348	1.0000	0.4449	0.6158	0.8386	115.0168
HAC (66 clusters)	1.0000	0.9482	0.9734	1.0000	0.9218	0.9593	0.9813	6.0407
LGCN (183 clusters)	0.9567	0.6690	0.7874	0.9433	0.7874	0.8583	0.9226	65.6727
Adversarial VGAE (76 clusters)	0.9428	0.7977	0.8642	0.9408	0.7417	0.8295	0.9261	31.7047
Adversarial VGAE with similarity density (77 clusters)	0.6772	0.8498	0.7537	0.8628	0.7754	0.8168	0.9172	39.3435
Adversarial Attention based VGAE (74 clusters)	0.4334	0.8508	0.5742	0.7808	0.7755	0.7781	0.8828	449.8156

CELE-1M, while the model have overall lower performance in Youtube Face. However, these performances depend on the predetermined number of clusters. If the number of clusters is unknown, it is difficult to gain these performances. (2) DBSCAN achieve high performance in both datasets, however, DBSCAN requires the dataset that different clusters have similar density. (3) Spectral clustering has poor scores because the data is imbalanced. This method needs large computation and memory space by calculating eigenvalue decomposition. (4) In MS-CELE-1M, ARO gains good scores, whereas the performance of Youtube Face is bad. This method is dependent on the number of neighbour nodes. (5) The Meanshift failed to be optimized by the bandwidth. In addition, the methods require a lot of time to shift data. (6) HAC has excellent performance. Unlike kmeans, this does not need to determine the number of clusters, but this requires the criteria of distance to cut the clusters. (7) LGCN has good performance in both datasets. (8) although the number of predicted clusters is not perfect, the adversarial VGAE obtains noticeable performance in both datasets, where this requires enough time to merge the link predictions to find final clusters. (9) Although the precision and F1 score decrease, the recall of this is improved compared to the simple adversarial VGAE(Model No. (8)) in the Youtube Face dataset. It seems that the pruning edges of sub-graphs made an effect on the recall score in the Youtube face. (10) The adversarial attention-based VGAE has overall poor scores. Applying both the attention and similarity density(pruning) does not help improve the model performance.

Overall, the baseline methods (such as Kmeans, DBSCAN, etc) have worse performance in Youtube face than in MS-CELE-1M, whereas the models using graph auto-encoders achieve remarkable scores in both datasets. It could be inferred that graph auto-encoder is an efficient and accurate method in Youtube face.

Discussion: As shown above, the adversarial VGAE using similarity density does not bring satisfactory performance in the experiment. However, there are still several important insights from the results. Firstly, the adversarial variational graph auto-encoder is an efficient and accurate method in the dataset, which are unclear and not easily separable among instances. This is because the graph convolution network makes nodes to be more distinguishable. It can be seen that scores of adversarial VGAE are better than baseline in Youtube Face dataset. Secondly, pruning edges of the sub-graphs improves recall scores. it means that the method has good performance on instances in the same label groups.

However, there are still several disadvantages of graph auto-encoder methods. Firstly, the precision score does not be improved in the case of pruning edges of sub-graphs. The similarity density requires the predetermined maximum number of neighbours. Model performance depends on these parameters, therefore it should be enhanced for robust performance. Secondly, in the case of using similarity density, one additional graph auto-encoder model is needed for computing the density. It means that training time and memory space become double. Lastly, the probability model(such as the variational graph auto-encoder.) has a limitation of training. It is hard to close zero training loss during the training model because the predicted value is made by probabilistically calculating a value in the range of standard deviation.

Although the performance of the proposed method is not perfect, there is the possibility to obtain better evaluation scores. In the small scale data, sub-graphs have a limited number of edges, so pruning edges is likely to remove necessary edges of the graphs. It is believed that if data is a large scale, unnecessary edges for clustering probably be eliminated by the similarity density.

Consequently, this study fulfilled the aim for unsupervised learning face clustering by the adoption of graph auto-encoder. It is meaningful because some conventional clustering algorithms and GCN based face clustering are unpractical. In the GCN based face clustering, label data is used for training GCN layers, but there is no label data in the real world. In addition, the conventional clustering should require unknown hyperparameters before execution. For example, kmeans need the number of clusters before model implementation. In the real world, the number of clusters cannot be known in many cases. In this regard, utilising graph auto-encoder is a great way to solve clustering problems without labels.

However, this still is not fully unsupervised learning clustering because only clustering is unsupervised learning. The feature extraction from face images is supervised learning such as DenseNet. Therefore, by using an auto-encoder based on pre-trained CNN[10], even feature extraction should be carried out by an auto-encoder for fully unsupervised learning clustering.

Chapter 7

Project Management

During this project, the primary tool used was Microsoft Excel because the Gantt chart was made by this tool. The Gantt chart has been used for the management of schedules. To utilise the Gantt chart, the following have been used in order to proceed without any delay.

- three kinds of sub-tasks are created such as Long, Medium, and Short term tasks.
- Schedule date: The start date and end date are written for sub-tasks.
- The progress bar of sub-tasks are displayed in the Gantt chart.
- The strikethrough function of sub-tasks is added. This is used when the sub-task is completed.

These items helped easily check my progress, therefore it is easy to notice what tasks should be followed up and finished. Especially, weekly base sub-tasks have been set, and to complete all sub-tasks is considered as a weekly goal. This scheduling method strongly supported finishing tasks without delay. Nevertheless, if the schedule is delayed, the more challenging week goal is made or the weekends are spent for finishing tasks to catch up with the schedule.

At the initial stage, this project spent a lot of time revising the initial proposed work because an unexpected problem happens in the implementation of the proposed method. To be specific, this project suggested node selection by similarity density based on the rank-order[34], in which candidate instances are selected for generating sub-graphs. However, in the past experiments which were done, results of this instance selection method showed very poor performance of the model, therefore other ways to utilise density similarity were studied to apply it to the model. Finally, it was decided that the similarity density is used for pruning edges of sub-graphs.

Chapter 8

Conclusion

In recent years, prior works have proposed face clustering adopting graph convolution networks. However, these studies have focused on the supervised or semi-supervised learning algorithm, which requires label data. In addition, these studies used inefficient sub-graphs, which contain all edges of sub-graphs. Therefore, this study proposed adversarial variational graph auto-encoder and pruning sub-graphs for face clustering. Especially, pruning sub-graphs is carried out based on nodes with higher similarity density. The adversarial variational auto-encoders effectively solve the face clustering problem without labels and achieve excellent performance. Moreover, in the case of pruning sub-graphs, the recall score was improved although the method did not bring overall satisfactory performance. It is evident that this is a meaningful study because the method is based on unsupervised learning with deep learning for face clustering. However, there are some limitations such as difficulty to train the model, an additional model for similarity density, and poor performance. Future work should therefore include follow-up work designed to develop the model for improvement of the performance and execution with large scale datasets and other kinds of data.

Bibliography

[1] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval*, 12, 2009.

[2] M. T. Chen Jie and X. Cao. FASTGCN : FAST LEARNING WITH GCN vis importance sampling. *Iclr 2018*, pages 1–15, 2018.

[3] W. L. Chiang, Y. Li, X. Liu, S. Bengio, S. Si, and C. J. Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 257–266, 2019.

[4] D. Comaniciu and P. Meet. Mean shift analysis and applications. *Proceedings of the IEEE International Conference on Computer Vision*, 2(2):1197–1203, 1999.

[5] J. Cui, F. Wen, R. Xiao, Y. Tian, and X. Tang. EasyAlbum: An interactive photo annotation system based on face clustering and re-ranking. *Conference on Human Factors in Computing Systems - Proceedings*, (February 2015):367–376, 2007.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[7] S. Guo, J. Xu, D. Chen, C. Zhang, X. Wang, and R. Zhao. Density-Aware Feature Embedding for Face Clustering. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 6697–6705, 2020.

[8] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao. MS-celeb-1M: A dataset and benchmark for large-scale face recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9907 LNCS:87–102, 2016.

[9] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. volume 2017-December, 2017.

[10] X. Hou, L. Shen, K. Sun, and G. Qiu. Deep feature consistent variational autoencoder. *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, pages 1133–1141, 2017.

[11] W. Huang, T. Zhang, Y. Rong, and J. Huang. Towards Fast Graph Representation Learning. (Nips):1–12, 2018.

[12] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[13] T. N. Kipf and M. Welling. Variational Graph Auto-Encoders. (2):1–3, 2016.

[14] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–14, 2017.

[15] J. Lee, I. Lee, and J. Kang. Self-attention graph pooling. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:6661–6670, 2019.

[16] S. P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[17] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial Autoencoders. 2015.

[18] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. 2002.

[19] C. Otto, D. Wang, and A. K. Jain. Clustering Millions of Faces by Identity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(2):289–303, 2018.

[20] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. Adversarially regularized graph autoencoder for graph embedding. *IJCAI International Joint Conference on Artificial Intelligence*, 2018-July:2609–2615, 2018.

[21] C. Qi, J. Zhang, H. Jia, Q. Mao, L. Wang, and H. Song. Deep face clustering using residual graph convolutional network. *Knowledge-Based Systems*, 211:106561, 2021.

[22] Y. Ren, N. Wang, M. Li, and Z. Xu. Deep density-based image clustering. *Knowledge-Based Systems*, 197, 2020.

[23] A. Salehi and H. Davulcu. Graph Attention Auto-Encoders. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 2020-Novem:989–996, 2020.

[24] R. Sibson. Slink: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16, 1973.

[25] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, and Y. Bengio. Graph attention networks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pages 1–12, 2018.

[26] H. D. Verlekar. Link Prediction Based Face Clustering Using Variational Attentional Graph Autoencoder. 2020.

[27] Z. Wang, L. Zheng, Y. Li, and S. Wang. Linkage based face clustering via graph convolution network. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June(c):1117–1125, 2019.

[28] Z. Weng, W. Zhang, and W. Dou. Adversarial Attention-Based Variational Graph Autoencoder. *IEEE Access*, 8:152637–152645, 2020.

[29] L. Wolf, T. Hassner, and I. Maoz. Face recognition in unconstrained videos with matched background similarity. *Proceedings of the IEEE Computer*

Society Conference on Computer Vision and Pattern Recognition, pages 529–534, 2011.

[30] L. Yang, D. Chen, X. Zhan, R. Zhao, C. C. Loy, and D. Lin. Learning to Cluster Faces via Confidence and Connectivity Estimation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 13366–13375, 2020.

[31] L. Yang, X. Zhan, D. Chen, J. Yan, C. C. Loy, and D. Lin. Learning to cluster faces on an affinity graph. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:2293–2301, 2019.

[32] X. Zhan, Z. Liu, J. Yan, D. Lin, and C. C. Loy. Consensus-driven propagation in massive unlabeled data for face recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11213 LNCS:576–592, 2018.

[33] M. Zhang and Y. Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 2018-Decem(Nips):5165–5175, 2018.

[34] C. Zhu, F. Wen, and J. Sun. A rank-order distance based clustering algorithm for face tagging. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 481–488, 2011.

Appendix A

Mathematical Proof related to variational graph auto-encoder

A.1 Variation Auto-encoder Loss

$$
\begin{aligned}
&q_\phi(z|x) \textit{ is the conditional probability of Encoder}\\
&p_\theta(\hat{x}|z) \textit{ is the conditional probability of Decoder}\\
\log p_\theta(x) &= E_{z\sim q_\theta(z|x)}[\log p_\theta(x)]\\
&\textit{because } p_\theta(x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)} \text{ [Bayes' Rule]}\\
&= E_z\left[\log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)}\right]\\
&= E_z\left[\log \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)}\frac{q_\phi(z|x)}{q_\phi(z|x)}\right]\\
&= E_z\left[\log p_\theta(x|z)\frac{p_\theta(z)}{q_\phi(z|x)}\frac{q_\phi(z|x)}{p_\theta(z|x)}\right]\\
&= E_z\left[\log p_\theta(x|z)\right] - E_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z)}\right] + E_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]
\end{aligned}
\tag{A.1}
$$

$$\begin{aligned}\log p_\theta(x) &= E_z\Bigg[\log p_\theta(x|z)\Bigg] - D_{KL}(q_\phi(z|x)||p_\theta(z)) + D_{KL}(q_\phi(z|x)||p_\theta(z|x)) \\ &= E_z\Bigg[\log p_\theta(x|z)\Bigg] - \mathcal{L}_{KL\ divergence} + D_{KL}(q_\phi(z|x)||p_\theta(z|x)) \geq 0\end{aligned} \tag{A.2}$$

$$\therefore\ \textit{Tractable lower bound loss} = \ E_z\Bigg[\log p_\theta(x|z)\Bigg] - \mathcal{L}_{KL\ divergence} \tag{A.3}$$

A.2 Simplifying KL divergence

$$\begin{aligned}&\textit{Fake distribution } g(z) \sim N(\mu, \log\sigma^2) \textit{ from latent features}\\ &\textit{True distribution } p(z) \sim N(0,1) \textit{ from Gaussian Random Noise}\\ &\qquad g(z;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}\\ &\quad \log\ g(z;\mu,\sigma^2) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(z-\mu)^2}{2\sigma^2}\end{aligned} \tag{A.4}$$

$$\begin{aligned}\textit{Cross Entropy} &= -\int p(z)\log q(z)\mathrm{d}z\\ &= -\int p(z)\Big(-\frac{1}{2}\log(2\pi\sigma^2) - \frac{(z-\mu)^2}{2\sigma^2}\Big)\mathrm{d}z\\ &= \frac{1}{2}\log(2\pi\sigma^2)\int p(z)\mathrm{d}z + \frac{1}{2\sigma^2}\int p(z)(z-\mu)^2\mathrm{d}z\\ &= \frac{1}{2}\log(2\pi\cancel{\sigma^2})\cancel{\int p(z)\mathrm{d}z} + \frac{1}{2\cancel{\sigma^2}}\int p(z)z^2\mathrm{d}z\\ \textit{because }\ \sigma^2 &= E[X^2] - E[X]^2 \rightarrow E[X]^2 = \mu^2 + \sigma^2\\ &= \frac{1}{2}\log(2\pi) + \frac{1}{2}(\mu^2+\sigma^2)\end{aligned} \tag{A.5}$$

$$
\begin{aligned}
Entropy =& -\int q(z)\log q(z)\mathrm{d}z \\
=& -\int q(z)\Big(-\frac{1}{2}\log(2\pi\sigma^2) - \frac{(z-\mu)^2}{2\sigma^2}\Big)\mathrm{d}z \\
=& \frac{1}{2}\log(2\pi\sigma^2)\int q(z)\mathrm{d}z + \frac{1}{2\sigma^2}\int q(z)(z-\mu)^2\mathrm{d}z \\
=& \frac{1}{2}\log(2\pi\sigma^2)\cancel{\int q(z)\mathrm{d}z} + \frac{1}{2\cancel{\sigma^2}}\cancel{\sigma^2} \\
=& \frac{1}{2}log(2\pi) + \frac{1}{2}(\log\sigma^2) + \frac{1}{2}
\end{aligned}
\tag{A.6}
$$

$$
\begin{aligned}
KL\ divergence =& Cross\ Entropy - Entropy \\
=& \cancel{\frac{1}{2}\log(2\pi)} + \frac{1}{2}(\mu^2+\sigma^2) \\
& - \cancel{\frac{1}{2}\log(2\pi)} - \frac{1}{2}(\log\sigma^2) - \frac{1}{2}
\end{aligned}
\tag{A.7}
$$

$$
simplified\ KL\ divergence\ \mathcal{L}_{kl} = -\frac{1}{2}(1 + (log\sigma)^2 - \mu^2 - \sigma^2) \tag{A.8}
$$

Appendix B

Coding for Auto-encoder Model

```
class Auto_Encoder(nn.Module):
    def __init__(self, model_kwargs, model_opt, encoder, discriminator):
        super(Auto_Encoder, self).__init__()
        self.dout = model_kwargs['l4']['dout']
        self.EPS = 1e-15
        self.opt_decoder = model_opt['opt_decoder']
        self.opt_disc = model_opt['opt_disc']
        self.encoder = encoder
        if self.opt_disc==True:
            self.discriminator = discriminator

        self.opt_loss=model_opt['opt_loss']
        if self.opt_loss=='BCE':
            self.BCEloss=nn.BCELoss()
        elif self.opt_loss=='MSE':
            self.MSEloss=nn.MSELoss()

    def encode(self, A, X):
        Z = self.encoder(A, X)
        return Z

    def decode(self, Z):
        Zt=torch.transpose(Z,1,2)
        pred_A = torch.sigmoid(torch.matmul(Z,Zt))
        return pred_A

    def reconstruct_loss(self, A, pred_A, org_A):
        B, N, _ = pred_A.shape

        pred_A = pred_A.view(-1).float()
        true_A = A.view(-1).float()
        if self.opt_loss=='BCE':
            ent_loss = self.BCEloss(pred_A, true_A)

        elif self.opt_loss=='MSE':
            ent_loss = self.MSEloss(pred_A, true_A)

        if self.opt_decoder =='vgae':
            kl_divergence=((0.5/N)*(1+ 2*self.encoder.logstd -
            self.encoder.mean**2 -
            torch.exp(self.encoder.logstd)**2)).sum(1).mean()
        elif self.opt_decoder =='gae':
            kl_divergence=0

        recon_loss = ent_loss - kl_divergence

        return recon_loss, kl_divergence

    def discriminator_loss(self, Z):
        if self.opt_disc==True:
            real = torch.sigmoid(self.discriminator(torch.randn_like(Z)))
            fake = torch.sigmoid(self.discriminator(Z.detach()))
            real_loss = -torch.log(real + self.EPS).mean()
            fake_loss = -torch.log(1 - fake + self.EPS).mean()
            return real_loss + fake_loss
        else:
            print("Disc option is FALSE")

    def reg_loss(self, Z):
        if self.opt_disc==True:
            fake = torch.sigmoid(self.discriminator(Z))
            reg_loss = -torch.log(fake + self.EPS).mean()
            return reg_loss
        else:
            print("Disc option is FALSE")
```