Link Prediction Based Face Clustering Using Variational Attentional Graph Autoencoder

Presented to the Faculty of the Graduate School of The University of Texas at Arlington in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Harish Deepak Verlekar

UNIVERSITY OF
TEXAS
ARLINGTON

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

University of Texas at Arlington

DEC 2020

Copyright © by Harish Deepak Verlekar 2020
All Rights Reserved

ACKNOWLEDGEMENT

This project and my master's wouldn't have been possible without the help and guidance of many people. I would like to thank everyone who has helped me in someway or the other throughout this journey.

First and foremost, I would like to express my gratitude to my supervisor Dr. Vassilis Athitsos for believing in me and guiding me throughout.I would also like to thank Dr. Farhad Kamangar and Dr. William Beksi for being a part of my thesis committee and providing me valuable feedback on my research.I would also like to the thank the Department of Computer Science and Engineering department at University of Texas at Arlington for providing with all the infrastructure and support.

I would like to thank my lab mates Ashiq, Saif and Reza for constantly motivating me and sharing their knowledge with me which helped me pass through the hurdles of research.

I would like to thank my parents for their constant support they have shown me through this research.Also I would like to show my deepest gratitude to my family and friends who were always there whenever I needed them through this journey.

Abstract

In this work, we address the problem of clustering faces according to their individual identities present inherently in the dataset.The current clustering frameworks are either based on some heuristic method or require labelled data for training the models,also some of them make assumptions on data distribution or shape of the clusters.We have framed the problem of forming clusters to that of link prediction on graphs and learn how to do that in a completely unsupervised way by proposing to use Variational Graph Autoencoders and use Graph Attentional Network as the Encoder. We call this network as Variational Attentional Graph Autoencoder(VAGAE).Our framework is not based on any assumptions of data distribution or shape of clusters and learns without any use of labelled data.

We solve the problem in the following way, we first extract features from a feature extractor which has been trained in an unsupervised way using Convolutional Adversarial Autoencoders and have compared the results of it with a pre-trained Inception Resnet feature extractor trained using FaceNet triplet loss algorithm. We then generate Instance Subgraphs(ISG) for each instance by finding the K-Nearest neighbours for each instance upto 2-hops and connect the edges to the instance only if it satisfies an approximate rank-order distance threshold. We then pass the ISG's to the Variational Graph Autoencoder which uses a Graph Attention Network as an encoder to learn general graph structure features from ISG's and perform link prediction. We then transitively merge the link prediction prediction results to form final clusters.

We evaluate our results on the first 50 faces from the Youtube Faces dataset and show that the results are decent enough or even better in some metrics compared to the other clustering methods.We have also evaluated our method on Link based Face Clustering via Graph Convolution Network and have shown that we get decent performance on data taken from same distribution for train-test, even though we train our model using unsupervised learning.

Keywords : Face Clustering, Graph Variational Autoencoders, Graph Attention Networks

TABLE OF CONTENTS

List of Figures

List of Tables

Acronyms

1. VAGAE - Variational Attention Graph Autoencoders
2. VGAE - Variational Graph Autoencoders
3. GCN - Graph Convolution Networks
4. GAT - Graph Attention Network
5. ISG - Instance Subgraph
6. AROC - Approximate Rank Order Based Clustering
7. GAAE - Graph Attentional Autoencoders
8. LBFGCN - Link Based Face Clustering via Graph Convolution Networks

Glossary

1. Merge ISG's : Clustering results obtained by directly merging ISG's obtained from algorithms without passing through VAGAE.
2. Pure ISG's : ISG's having nodes from the same class.
3. Impure ISG's : ISG's having nodes from different class.

Chapter 1

Introduction

1.1 Introduction

With the exponential growth of media and technology exabytes of data is generated each day, Facebook alone generates around 4 peta bytes of data every day out of which 350 mn are images alone[1]. With such huge data available it becomes imperative to develop methods which can mine useful information from this data and generate insights which can be used to improve our decisions and is applicable in every sector possible, one such problem is face clustering. It deals with clustering large number of unlabelled face images into individual identities present in the data[2]. It has been extensively studied in the past[2, 3, 4, 5] and can be used in application like, Video Analytics[4],tagging photos on social media or photo albums[5], criminal investigations[6].

In most of the use-cases, the data involves unbalanced identities and varying densities; A video might consist of many faces occurring at various frequencies. Similar are the cases with social-media and photo albums. Conventional clustering algorithms like K-Means[7] and Spectral Clustering[8] have impractical assumptions on data distribution and do not work well on data having face clusters which vary significantly in size, shape and density[3]. Other methods which do not have assumptions on the data distributions or shape of the clusters are the Rank-Order[5] based methods such as the Approximate Rank order based clustering(AROC)[2] which form clusters by calculating a rank order based score. These are also referred to as Linkage Based methods and they do not have any assumption on data distribution and also achieve higher accuracy.One of the recent papers, Linkage Based Face Clustering via Graph Convolution Networks[3] proposed a method to predict the linkages by classifying them as positive or negative nodes using graph convolution networks and then clustering them by transitive merging. Their method however involves training the Graph Convolution Network with a labelled training data to learn to predict the linkages.There can be cases or domains where labelled data is unavailable due to which the clustering problem cannot make use of non-heuristic based architectures to generate better clustering results.

We propose an architecture which makes use of Variational Graph Autoencoders[9] to learn these link predictions which can make use of the unlabelled data. Our contributions are the following a.) We propose to improve upon the Instance Subgraph(ISG) Generation algorithm[3] by incorporating the rank order based distance[5, 2] threshold to prune the nodes to generate ISG's having nodes belonging to the same class b.) We propose to use Variational Graph Autoencoders with Graph Attention Network as the Encoder to learn the linkages between nodes and perform the task of link predictions in an unsupervised way. c.) We have presented a study on the use of Adversarial Autoencoders[10] for the use of learning feature extractors from face images and compared it with the Inception Resnet trained using FaceNet[11] triplet loss algorithm as a feature extractor. d.) We have also performed an Ablation Study to show if the contributions proposed are actually helping get good clustering results and what are the drawbacks of our framework.

Although we are experimenting our framework on a face dataset we believe that this can be used for any data domain since we have not constrained it to learn only face features.

1.2 Thesis Outline

The thesis is organized as follows:

Chapter 1 provides a general introduction to the problem statement and proposed method.

Chapter 2 introduces the necessary background on various Autoencoders, Graph neural Networks and their various architectures.

Chapter 3 talks about various clustering methods.

Chapter 4 explains our proposed scheme.

Chapter 5 explains about the dataset used for the experiments.

Chapter 6 explains our experiment training and testing setup used.

Chapter 7 provides the experimental results of the proposed scheme and it analyses how it performs in comparison to the current architectures.

Chapter 8 provides an ablation study to show how our contributions lead to better clustering results

Chapter 9 concludes our work and gives the future work which can be done to improve this scheme.

Chapter 2

Background

2.1 Autoencoders

Autoencoders are models of unsupervised artificial Neural Netoworks which learn to compress or expand the input data by reconstructing the input from the latent representation. They are used to learn the feature embeddings in an unsupervised way.They are used in applications like dimensionality reduction instead of PCA since they learn the non-linearities in the data distribution due to the non-linear activations functions. Infact an Autoencoder without the non-linear activation functions approximates to PCA[12].

The autoencoder architecture consists of an encoder and decoder. The encoder and decoder is made of 1 or more dense layers of neural networks. the autoencoders mostly used in present day research compresss the input data into a hidden latent embedding using an encoder.[12] The decoder then reconstructs it back to its original dimension.The reconstruction loss is calculated either using the mean-squared error or binary cross entropy and the gradients propagated for optimizing the weights.

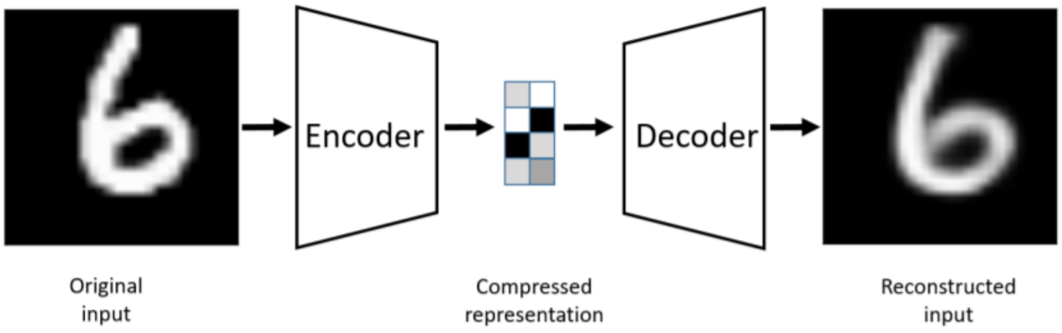


Figure 2.1: Vanilla Autoencoders[12]

2.1.1 Convolutional Autoencoders

Convolution Neural Networks(CNN) have shown extraordinary performance in image processing tasks[13]. If autoencoders are used for processing images it makes sense to use convolutions layers instead of fully connected layers in the architecture. Just using convolution layers in the encoder , transposed convolution layers in the decoder and fully connected layers help in architecting a superior model than the stacked or vanilla autoencoders[14].

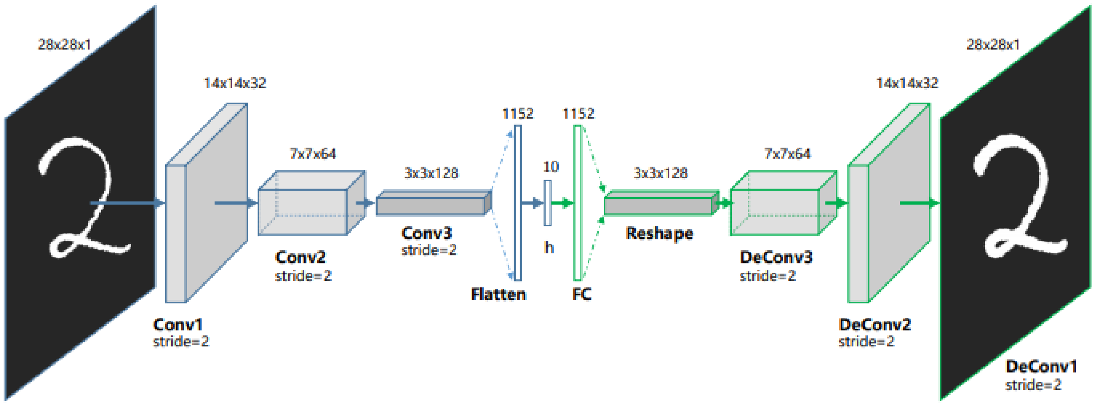


Figure 2.2: Convolutional Autoencoders[14]

2.1.2 Variational Autoencoders

Variational Autoencoders fall into the category of Regularized Autoencoders. They have shown a major improvement in the representation capabilities of the latent embeddings and follow the Variational Bayes Inference[12, 15].Given an observed dataset $x \in X$ and a latent variable $z \in Z$ we assume a probabilistic decoder $p_\phi(x|z)$ parameterized by ϕ following any distribution(usually the data distribution) p conditioned on an unobserved latent variable z. We also assume a probabilistic encoder $q_\theta(z|x)$ parameterized by θ following normal distribution.

The way the network is learnt is by making sure the latent embeddings follow a standard normal distribution.The loss function consists of two terms one is the reconstruction term which minimizes the reconstruction loss between the input and the output and second is the KL-Divergence which forces the latent embeddings to follow a standard normal distribution. The second term is a regularizing constraint which helps in making the latent space continuous and helps in making sure that similar points are clustered together. Variational Autoencoders because of the continuous latent space are used for data generation.[12]

2.1.3 Adversarial Autoencoders

Adversarial Autoencoder[10] is a probabilistic autoencoder which uses the framework of generative adversarial networks to match aggregated posterior of the hidden code vector to any arbitrary prior distribution[10].The beauty of this type of autoencoders is that it does not learn any explicit parameters to match the prior distribution rather it makes use of discriminators to learn it implicitly.In our work we have shown how Adversarial Autoencoders can also be used to learn the features and act as a feature extractor by learning from unlabelled data rather than training a feature extractor like FaceNet[11]. We show the results of our clustering pipeline by making use of both Adversarial Autoencoders and FaceNet.

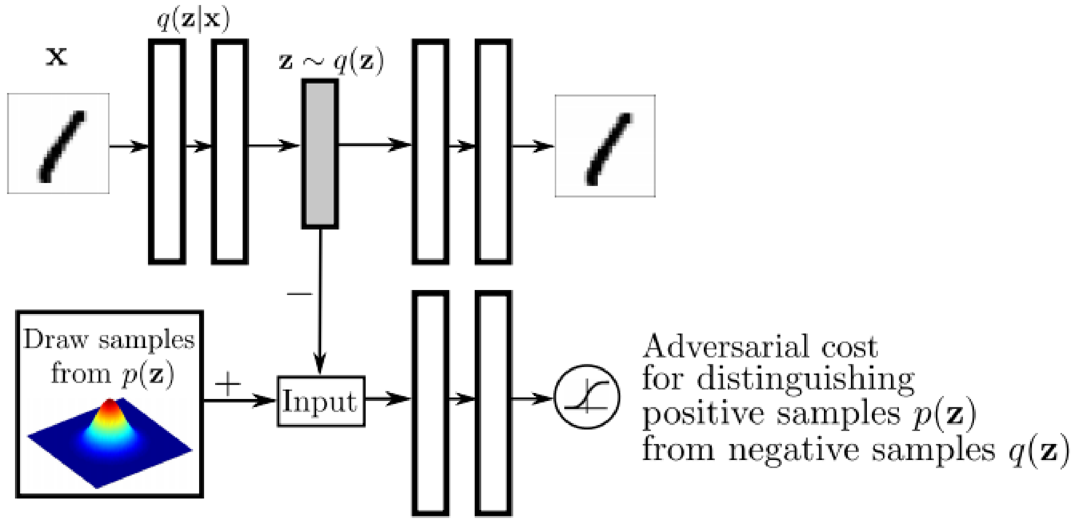

$q(\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
$\mathbf{z} \sim q(\mathbf{z})$
$-$
Draw samples from $p(\mathbf{z})$
$+$
Input
Adversarial cost for distinguishing positive samples $p(\mathbf{z})$ from negative samples $q(\mathbf{z})$

Figure 2.3: Adversarial Autoencoders[10]

Let x be the input and z be the latent code vector, $p(z)$ be the prior distribution which has to be imposed on the latent embeddings, $q(z|x)$ be the encoding distribution which defines the aggregated posterior distribution $q(z)$ and $p(x|z)$ be the decoding distribution[10]. Adversarial Autoencoders are trained by matching the prior distribution $p(z)$ to the aggregated posterior $q(z)$ and it does that by the use of discriminator. The training can be divided in 3 stages.In the first stage input is given to the encoder and the latent embedding generated is assigned label as fake and along with a sample from $p(z)$ which is labelled as real is passed to discriminator to learn fake vs real classification. In the second stage the encoder is again passed images and the latent embedding is passed to the decoder to generate the reconstruction loss and the encoder and decoder is optimized.In the 3rd stage we check whether the encoder is generating latent embeddings which follow the prior distribution by passing it to the discriminator and checking whether it is able to classify it as a sample from the prior distribution.At each stage the loss is calculated and gradients propagated separately.

In our implementation we use convolution layers in the encoder and transposed convolution layer to the decoder and also add some noise to the input to make the latent embeddings z generated more robust.

2.2 Graph Neural Networks

Many of the important real world data such as social media, computer or telecommunication networks,protein interaction networks are represented in terms of graphs[16].A lot of useful information can be extracted by learning from these graph structures and since then a number of papers have explored the possibility of generalizing neural networks to work on arbitrarily structured graph data[16].Graph Neural Networks are used to encode the structure of the graph data and learn many useful features[17].Individual node features are learnt by aggregating the neighbouring node features and accordingly the network is optimized.The result is such that similar nodes are clustered together whereas dissimilar nodes are clustered far away.

2.2.1 Graph Convolution Networks

Convolutions Neural Networks in the recent years have shown a lot of progress and have produced amazing results in the area of image processing. Applying convolutions of graphs is non-trivial because of the presence of irregular graph structure and varying

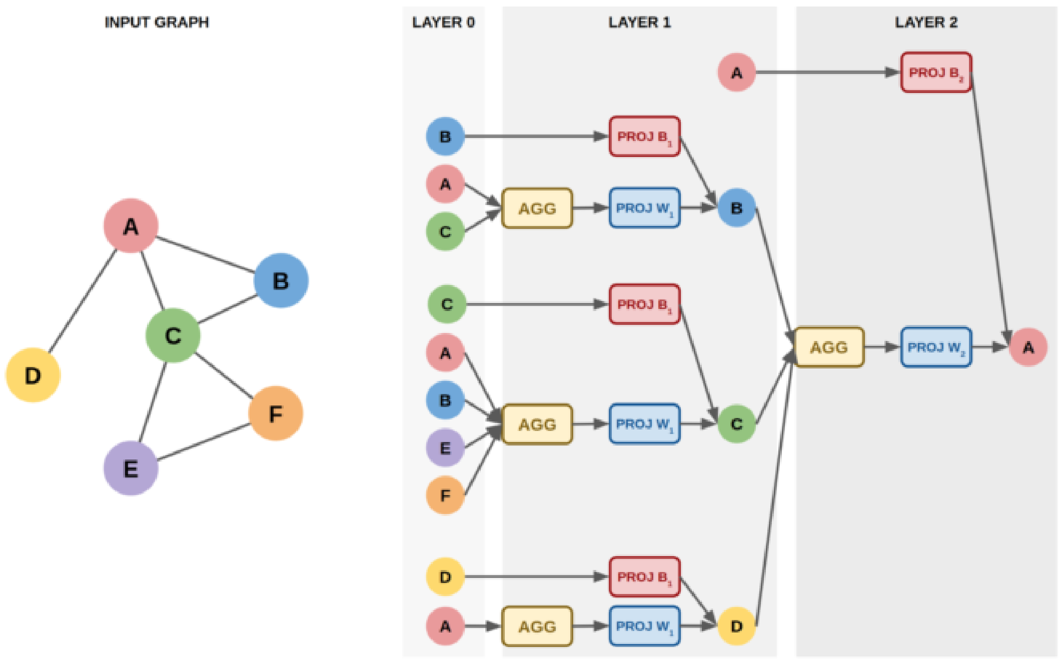


Figure 2.4: Graph Neural Networks[17]

number of neighbours for each node. In recent years there has been considerable research to designing graph convolution neural networks[18, 19].Graph Convolution networks can be divided into two approaches spatial based and spectral based[20]. Spectral based are depend upon fourier transforms where as spatial based work by attending to the nodes and its neighbours.

Kipf and Welling[21] proposed a multi-layer Graph Convolution Network with a layer-wise propagation rule as follows: $H^{l+1} = \sigma(\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}H^{l})W^{l}))$ where $\widetilde{A}$ is the adjacency matrix, $\widetilde{D}$ is the degree matrix, H^{l} is the feature matrix, for layer l ,W^{l} is the trainable weight matrix for layer l and $\sigma(.)$ denotes an activation function[21].This method was applied on some standard graph datasets such as Citeseer,Cora, Pubmed and NELL and state of the art results were obtained[21].

2.2.2 Graph Attention Networks

Attention mechnanism's have shown great performance in sequence based tasks and are very useful since they work with variable size inputs[22].It was primarily used for machine translation tasks and had shown a great performance[23]. It started getting extended to many other areas like computer vision[24] and Graph Neural Networks. In most of the prior work the edge weights were defined explicitly, Petar et.al proposed to learn these weights which he called it as the attention weights by using the multi-head self attention mechanism[25]. By finding the weights implicitly while aggregating the node features in the convolution step and giving equal weight or explicit weights to all nodes it will now learn the weights and perform aggregation on the node features accordingly. It is a form of spatial GCN and works on both inductive as well as transductive graph problems.

The attention co-efficients are calculated in the following way, consider a node 'i' and node 'j' as its neighbouring node. One learnable linear $a(.)$ transformation is applied on the node features of 'i' and 'j', e_{ij} parameterized by a weight matrix $W \in \mathbb{R}^{\mathbb{F}' \times \mathbb{F}}$ where F is the input dimension and F' is the output dimension[22] as shown in (2.1). The graph structure is injected into the mechanism by performing masked attention by computing e_{ij} only for the neighbours of node i, N_i in the graph as shown in (2.2).

$$e_{ij} = a(W\overrightarrow{h_i}, W\overrightarrow{h_j}) \tag{2.1}$$

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in N_i} exp(e_{ij})} \tag{2.2}$$

Once the attention coefficients are obtained they are used to compute a linear combination of the features corresponding to them, and this forms the final representation output for every node[22]. To stabilize the learning process of the co-efficients they use multi-head attention[22].

$$\overrightarrow{h_i'} = \|_{k=1}^{K} \sigma(\sum_{j \in N_i} \alpha_{ij}^k W^k \overrightarrow{h_j}) \tag{2.3}$$

In equation (2.3) $\|$ represents concatenation,K represents number of multi-head attentions, α_{ij}^k are the attention co-efficients computed by the k-th attention mechanism(a^k) and W^k is the corresponding weight matrix for the computation of input linear transformation.

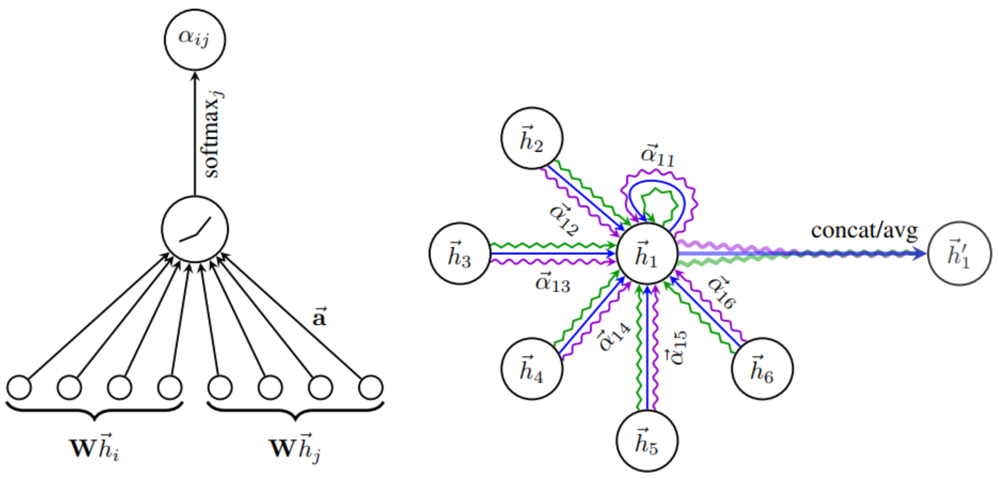


Figure 2.5: Graph Attention Networks[22]

2.2.3 Variational Graph Autoencoders

Kipf et al. proposed the use of Variational autoencoders for graphs for learning the feature embeddings in an unsupervised way[9].The concept is very similar to that of an variational autoencoder where the latent embeddings are mapped to a standard normal distribution for regularization and generation. The encoder in this model is ideally a Graph Convolution Network[18], input to it are the latent features and the adjacency matrix. The decoder is simply a dot-product decoder which uses the features to compute the dot product of the features given by the GCN and reconstruct the adjacency matrix.This model has proved to get state of the art results on link predictions task which deal with predicting the link between two member nodes.

Chapter 3

Literature Review

3.1 Approximate Rank Order Based Clustering

In this work the authors have proposed to perform face clustering in a more efficient and a scalable manner in large-scale clustering scenarios. An approximate rank-order based clustering approach is proposed which achieves better accuracy than other clustering algorithms such as k-means[7] and spectral clustering[2]. They improve upon the scalability issues of Rank-Order clustering approach proposed by Zhu et al.[5].It follows the procedure of agglomerative hierarchical clustering where every sample is intially initialized as separate clusters and then iteratively merged to get final set of clusters.

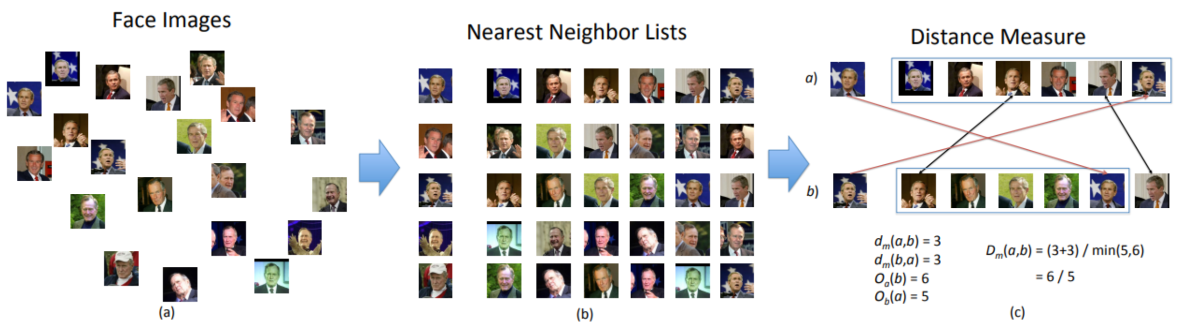


Figure 3.1: Approximate Rank-Order Clustering[2]

In this algorithm first for every sample K-Nearest Neighbour lists are found using approximate KNN methods such as FLANN library which makes use of randomized k-d tree algorithm to compute a short list of nearest neighbours. Then a distance metric is calculated which is given as follows

$$d_m(a,b) = \sum_{i=1}^{min(O_a(b),k)} I_b(O_b(f_a(i)),k) \tag{3.1}$$

This distance metric calculates the distance between two samples by considering the ranking of the faces of the neighbour list of one sample in the neighbour list of the other sample.The $f_a(i)$ in equation (3.1) is the i-th face in the neighbour list of a and $O_b(f_a(i))$ refers to the rank of $f_a(i)$ in b's neighbour list. $I_b(x, k)$ is an indicator function with a value 0 if face x is in face b's top k-nearest neighbours and 1 otherwise. This metric is used to define the asymmetric distance between two faces. To calculate a symmetric metric the below function is defined[2]:

$$D_m(a, b) = \frac{d_m(a, b) + d_m(b, a)}{min(O_a(b), O_b(a))} \tag{3.2}$$

The symmetric rank order distance gives low values if two faces are close to each other and higher values if they are the farther away. Once the distances are computed, based on a threshold the nearby clusters are transitively merged and final clusters are obtained.

The above paper makes no assumption on data distribution or cluster shapes which other algorithms like K-Means and spectral clustering assume and hence perform far better than them. Also as mentioned before it is scalable and can be used for large datasets. One drawback of this algorithm is that it is based on a heuristic approach and hence we propose a framework which learns to cluster the images to improve the metrics.

3.2 Linkage Based Face Clustering via Graph Convolution Network

This paper makes an attempt to solve the drawback of the previous paper which is based on heuristic by proposing to use graph convolution network to predict the link between nodes and form clusters.[3] It does that in the following way, the unlabelled face images are first passed onto a feature extractor such as ArcFace[26] or FaceNet[11] and embeddings of each face are extracted. For every face instance a using the extracted embeddings an Instance Pivot Subgraph(IPS) is constructed which is then passed through a graph convolution classfier network to predict the links and form clusters by transitively merging them.

For forming the IPS they propose an algorithm which has 3 steps. In the first step

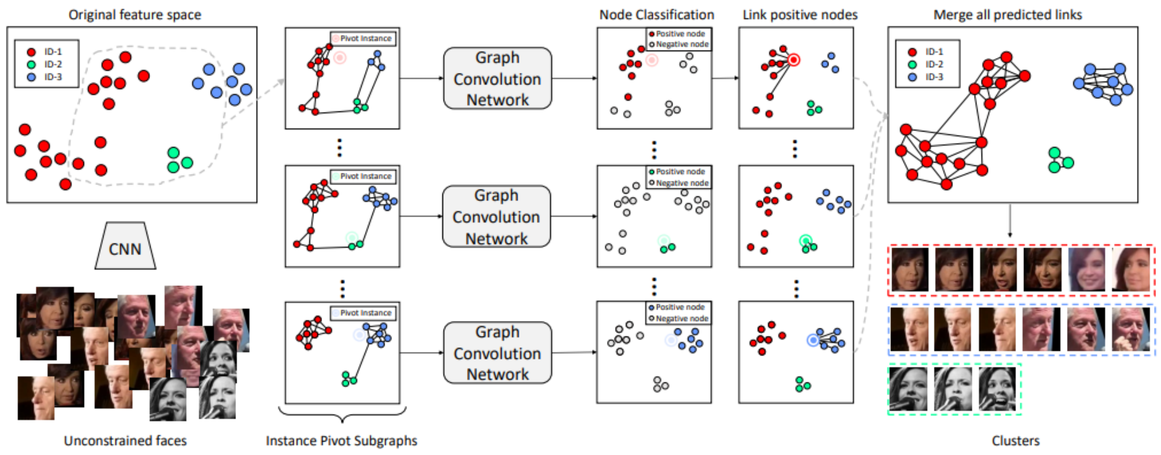


Figure 3.2: Linkage Based Face Clustering using Graph Convolution Networks[3]

neighbours upto h-hops are taken as nodes for the IPS. It is important to note here that the instance node instance is not included as a part of IPS[3]. In the second step the node feature normalization is done by subtracting the node feature p from the features of the nodes included in the IPS. Step 3 deals with adding the edges among the nodes of the IPS which is done by finding the top u neighbours of the nodes of the IPS out of all the face instances but connecting the neighbour of a node to a node which is present in the IPS. After the construction of the IPS for each face instance they are sent through a Graph Convolution Network for classifiying whether the IPS nodes for a instance should be connected or not. The way this GCN network is trained is a labelled face dataset is chosen, labels of positive and negative nodes in the IPS are assigned according to the actual labels in the dataset. Once the model is trained the model is then tested on other datasets for which the clustering needs to be performed. In short different datasets for training and clustering are used[3].

We propose to address the drawback of the requirement of labelled images for training the Graph Network to do the learning of predicting links, and learn in a completely unsupervised way with the use of Variational Attentional Graph Autoencoders.

Chapter 4

Proposed Work

4.1 Overview

4.1.1 Problem Overview

Given unlabelled set of N face images F where $F = [f_1, ..., f_N]$, we need to cluster the face images according to their individual identities in an completely unsupervised way without the need any labelled face images even for the feature extraction.We solve this problem by using an Adversarial Autoencoder[10] for extracting the features $X = [x_1, ..x_N] \in \Re^{NXD}$ from the face images in an unsupervised way where D is the feature dimension, and use Variational Attentional Graph Autoencoders to predict linkages between nodes in the instance subgraphs and finally perform clustering by transitively merging them.

4.1.2 Motivation

The motivation behind this work stems from the idea of removing labelled data completely from the pipeline of clustering and still manage in getting decent clustering results without any assumptions on data distribution or cluster shapes.

4.1.3 Pipeline

The pipeline consists of the following stages.

1. We first preprocess the face images by using the MTCNN[27] face detector using the facenet-pytorch[28] library.The images are resized to 160 X 160.

2. The images are then split into train-test using 80:20 ratio. We propose to use Adversarial Autoencoders(AAE) for training the feature extractor to extract features from face

images in an unsupervised way, the feature extractor in the AAE setup is nothing but the encoder, the embeddings of which are going to be used in our clustering pipeline.

3. Using these embeddings, we propose an Instance Subgraph(ISG's) generation algo-

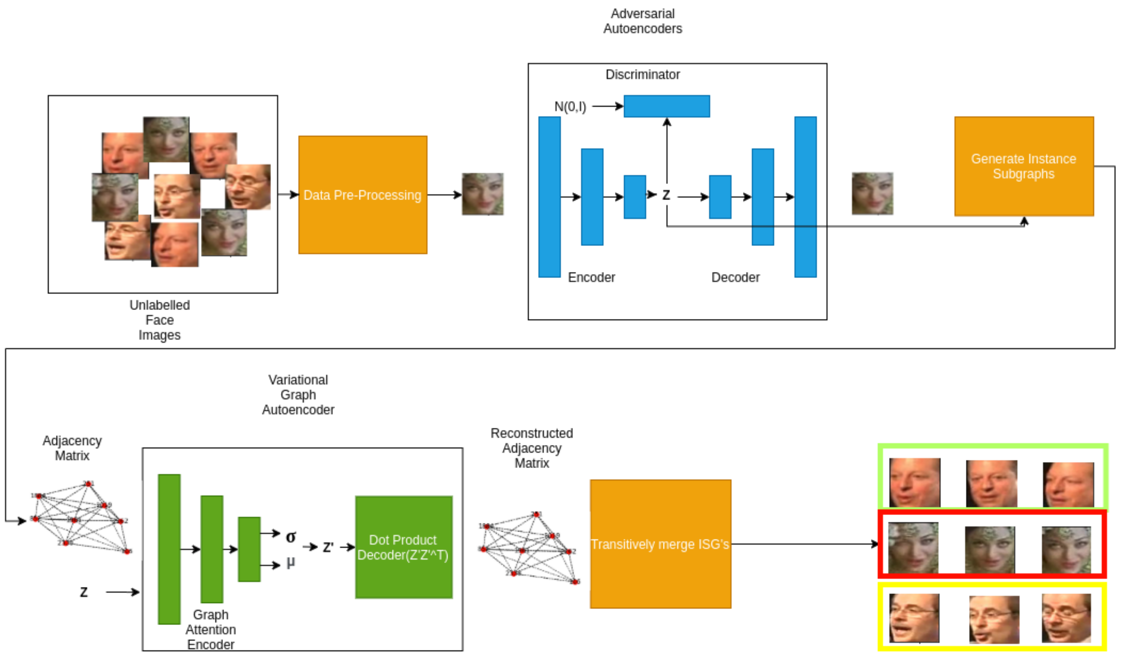


Figure 4.1: Unsupervised face Clustering Using Variational Graph Autoencoders

rithm for every face image similar to the algorithm proposed by Wang et. al[3]. We have improved upon by integrating the Rank-Order based Distance to connect the edges to generate pure ISG's which is imperative for our pipeline.

4. The train set ISG's are then passed through an Variational Graph Autoencoder(VGAE)[9] where instead of using Graph Convolution Network[18] as the encoder we propose to use Graph Attention Network[22] to learn a weight matrix which would help in predicting the linkages in the test graphs.

5. The Decoder of VGAE gives out the adjacency matrix and since we are simply taking the dot product it gives us the similarity scores of whether two edges should be connected or not.We can then have only those edges connected between the instance and other nodes whose scores are above a threshold value on our test set and then transitively merge the the instance node links to get our final clusters.

4.2 Constructing Instance Graphs(ISG) using Rank Order Based Distance

For generating the Instance Subgraphs we have improvised the Instance Pivot Subgraph generation Algorithm proposed by Wang et. al[3] to generate Instance Subgraphs(ISG) which have less number of impure ISG's; Out of the total number of ISG's generated majority of the ISG's should have nodes which belong to the same class and even if they have dissimilar nodes if possible they should be farther away from the instance node. To do that we have integrated the Rank Based Distance proposed by Zhu et al.[5] and Otto et al.[2] to generate pure clusters.

The major differences between our algorithm and the one proposed by Wang et. al[3] is as follows:

1. We include the instance node as a part of our sub-graph.
2. We do not normalize the node features by subtracting the instance node features from all the other node features.
3. We integrate Rank Based Distance to generate pure ISG's.

4.2.1 Steps for Generating ISG

Step 1. For every instance in the dataset find $'K'$ nearest neighbours this will be the $1 - hop$ neighbours from the instance, apply Rank Based Order Distance to prune the nodes.
Step 2. For each of the $1 - hop$ neighbours generated in step 1 find $'M'$ nearest neighbours, this will be the $2 - hop$ neighbours from the instance.Apply Rank Based Order Distance to prune the nodes.All these nodes form our final set of nodes in an ISG.
Step 3. To connect the edges we find $'U'$ Nearest neighbours for each of the nodes generated in Step 2 out of all the face instances and only connect those which are present in ISG nodes generated in Step 2.

Following the above steps to generate ISG's leads to the generation of pure ISG's which are well connected. One important thing to note here is that due to the addition of the

Algorithm 1 Rank Based Order Distance(RBOD):

Input: Input the K-NN lists of both nodes in an edge, let them be knn1 and knn2.
Output: distance,limit

```
1: distance = 0
2: Find the rank of node1 in the K-NN list of node 2, let's call this limit.
3: for instance upto limit do
4:     distance + = rank of knn2[instance] in knn1
5: return distance,limit
```

Rank Based Order Distance constraint for Node Pruning there are many Single Node ISG's which are generated which is a drawback, but it should be minimized by carefully tuning the hyperparamters K,M,U, and Rank-Threshold which is used for applying the rank based distance metric for node pruning.

4.2.2 Rank Based Order Distance(RBOD) for Node Pruning:

We used the Rank Based Order Distance proposed by Zhu et al.[5] and Otto et al.[2] to prune our nodes, the algorithm works by finding the ranks of the instance neighbours and other instance neighbours and calculating the distance. The Algorithm 1 calculates the asymmetric Rank Based Order Distance between two nodes.

4.2.3 Node Pruning

The Algorithm 2 takes as input a list of source nodes and target nodes, the embeddings list, the number of neighbours 'k' and the threshold which will be used in calculating the rank based distance.It outputs the pruned source nodes and pruned target nodes. For every pair of source and target node it finds the asymmetric rank distance for both the combinations in a pair and calculates the final symmetric distance which is then compared to that of the threshold which has been set. If the distance is less than the threshold then the nodes are appended to the list of Pruned Source Nodes and Pruned Target Nodes and both the lists are returned after finding for all nodes.

Algorithm 2 Node Pruning

Input: nodeList(List of Nodes):[[sourceNodes,targetNodes]],embeddings,k,rankthreshold
Output: PrunedSourceNodes,PrunedTargetNodes

```
 1: PrunedSourceNodes,PrunedTargetNodes = [],[]
 2: for nodes in nodelist do
 3:     topkNode1 = findTopk(nodes[0],embeddings,k)
 4:     topkNode2 = findTopk(nodes[1],embeddings,k)
 5:     distance1,lim1 = rankBasedDistance(topkNode1,topkNode2)
 6:     distance2,lim2 = rankBasedDistance(topkNode2,topkNode1)
 7:     finDist = (distance1+distance2)/min(lim1,lim2)
 8:     if $finDist \leq rankthreshold$ then
 9:         PrunedSourceNodes.append(nodes[0])
10:         PrunedTargetNodes.append(nodes[1])
11: return PrunedSourceNodes,PrunedTargetNodes
```

4.2.4 Generate Nodes

The Algorithm 3 generates the final list of nodes given the embeddings, value k for finding the nearest neighbours, the instance for which nodes need to be generated,j is used by the Node Pruning algorithm to find j nearest neighbours, and the threshold for rank order based distance. It does that by finding the nearest neighbours and performing node pruning on them.

4.2.5 Generate Instance Sub Graphs

Algorithm 4 is an aggregated algorithm which uses the other 3 algorithms to generate the final ISG's by first finding the nodes and then connecting them to form edges as explained in the steps before. We input the embeddings, the number of 'k', 'm' neighbours for generating the nodes and number of 'u' nodes for connecting the edges, j is used by the Node Pruning algorithm to find j nearest neighbours and rankthreshold is used in the

Algorithm 3 Generate Nodes:

Input: embeddings,k,instance,j,threshold
Output: uniqueNodes

```
1: knns = topK(embeddings,k,instance)
2: sourceNodes,targetNodes = nodePruning(knn,embeddings,j,threshold)
3: uniqueNodes = findUniqueNodes(sourceNodes,targetNodes)
4: return uniqueNodes
```

Node Pruning algorithm so that most of the dissimilar nodes are eliminated from the ISG.

4.3 Variational Graph Attention Autoencoder on ISG

Once the ISG's are generated we pass them to the Variational Graph Attention Autoencoder(VAGAE) for learning the generalized weight matrix W which would help us in link prediction. The encoder in VAGAE is made up of an Graph Attention Network which uses the attention mechanism to understand which neighbouring nodes it should focus on more to transform the features of the nodes by aggregating them. The decoder simply performs Inner Dot Product of the generated features matrix Z' by the encoder to reconstruct the adjacency matrix which gives us a similarity score between the two nodes and helps us predict whether the two nodes should be connected or not. Using a Variational Autoencoder helps in making the latent space regularized and more robust in encoding the similar face embeddings near to each other which helps in improving the clustering results. We have conducted experiments where Variational Graph Autoencoder has outperformed the Vanilla Graph Autoencoder.

4.4 Link Merging

Once we get the adjacency matrices for each instance from the VAGAE we connect the instance to only those edges of the subgraph which are above a link threshold. Once we obtain the predicted instance linkages for all instances we transitively merge them to

Algorithm 4 Generate Instance Subgraphs(ISG):

Input: embeddings,k,m,u,j,rankthreshold
Output: isgList

```
 1: isgList = []
 2: for i,embed in enumerate(embeddings) do
 3:     finNodes,finedges = [], dict()
 4:     uniqueNodes = generateNodes(embeddings,k,i,j,rankthreshold)
 5:     finNodes.append(uniqueNodes)
 6:     for node in uniqueNodes do
 7:         uniqueNodes = generateNodes(embeddings,m,node,j,rankthreshold)
 8:         finNodes.append(uniqueNodes)
 9:     for fNode in finNodes do
10:         unns = generateNodes(embeddings,u,fNode,j,rankthreshold)
11:         ipsCommNodes = intersection(unn,finNodes)
12:         finedges[fNode] = ipsCommNodes
13:     isgList.append((finNodes,finedges))
14: return isgList
```

form final clusters.

Chapter 5

Dataset

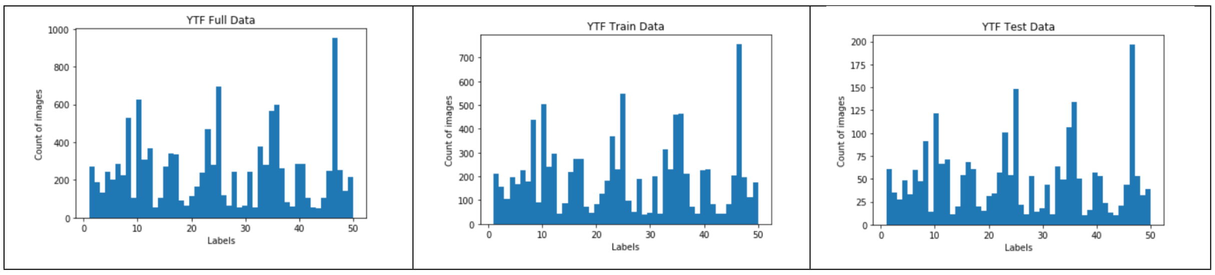
5.1 Youtube Faces

This dataset consists face images taken from youtube videos. Each face identity has face images taken from 1 or more videos. The distribution is as follows. Number of videos per person[29]:

No of videos	1	2	3	4	5	6
No of people	591	471	307	167	51	8

In our experiments we are just considering the first 50 face images of the dataset for clustering, which consists of 12468 face images. Since we are using the aligned images from the YTF dataset most the the images are already cropped and aligned. The data imbalance distribution is as follows. The data has been split into train, test in a 80:20 ratio and the model has been evaluated on full and test data.

Table 5.1: Data Distribution of YTF Full, Train and Test of YTF data of first 50 classes



5.2 IJB-B

This dataset consists of 67000 face images, 7000 face videos, and 10000 non-face images. We have taken a random subset from the IJB-B subtask which consists of 512 identities out of which in our subset we have 506 identities and 3635 images.Looking at the histogram plot 5.1 below the distribution seems largely imbalanced.

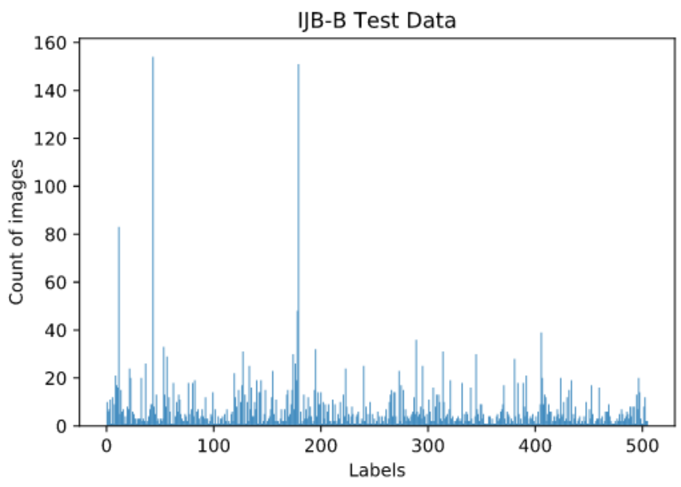


Figure 5.1: Data Distribution of IJB-B subset of taken from 512 subtask consisting of 506 classes

5.3 CASIA Dataset

The CASIA dataset is annotated with 10,575 unique people with 494,414 images in total.We have taken a random subset of 11365 images and 5554 identities.The histogram plot 5.2 shows that majority of the per identity images are less than 5.

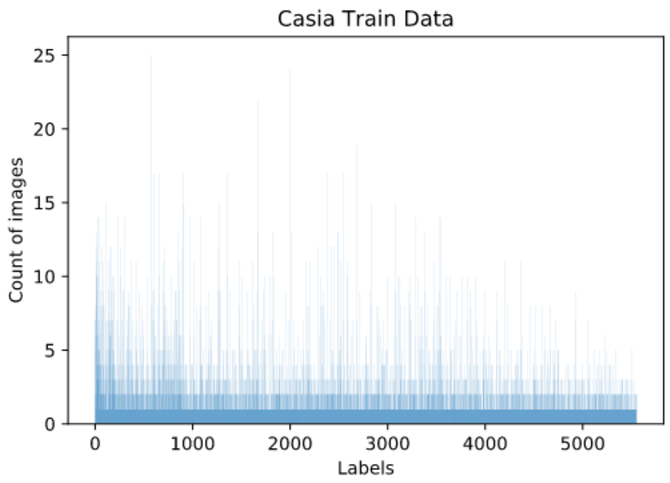


Figure 5.2: Data Distribution of CASIA subset consisting of 5554 classes

Chapter 6

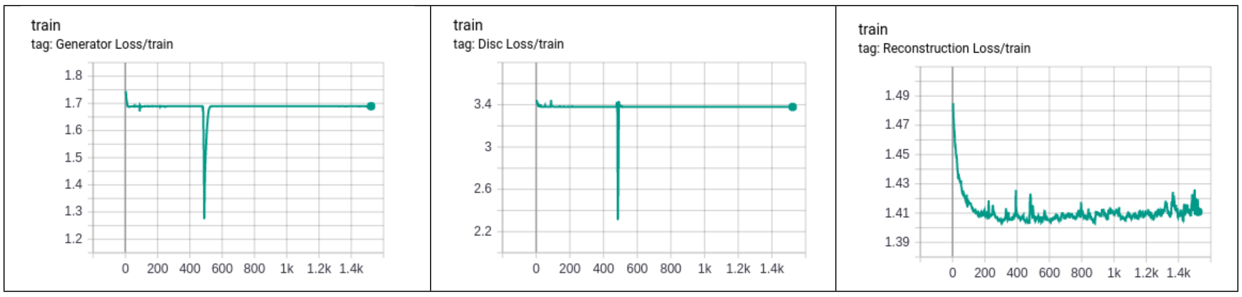
Training and Testing Setup

6.1 Training and Testing Setup

6.1.1 Adversarial Autoencoders

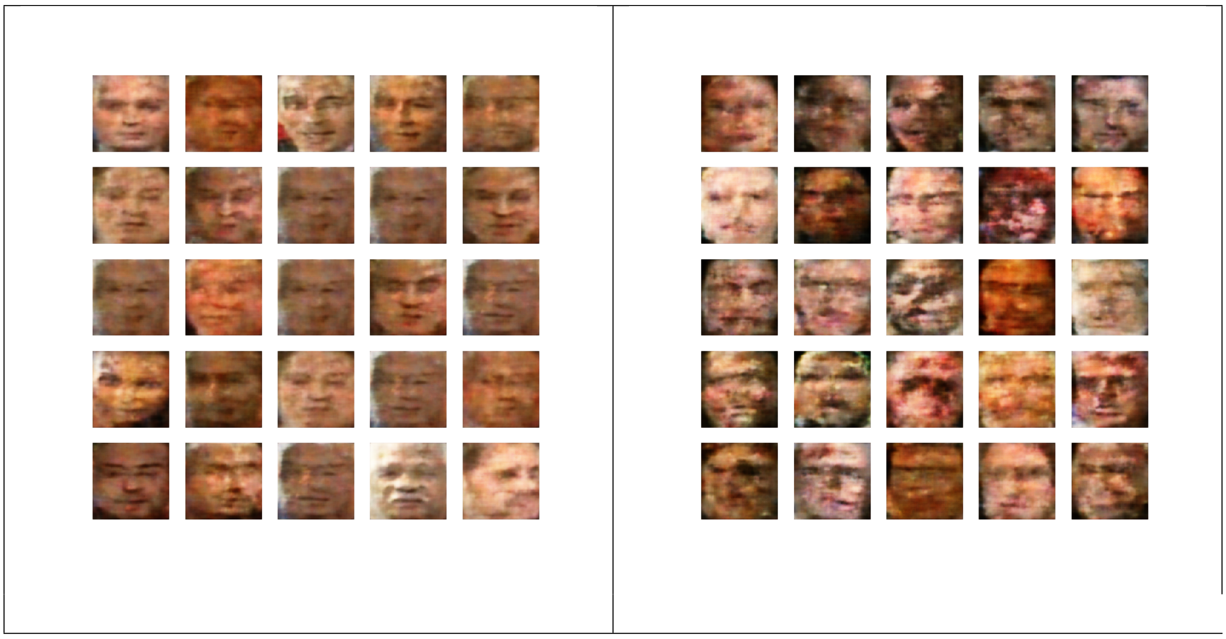
Adversarial Autoencoders[10] consists of 3 networks, the encoder,decoder and the discriminator. In our experiments we have used a Convolution Encoder which consists of 4 convolution layers with dropout of 0.3 and Leaky ReLu activation function, the convolution layers are followed by a fully connected layer which generates the latent embedding of 128 dimension. The Decoder consists of 4 transposed convolution layers with Batch Normalization and ReLu activation function between layers and sigmoid activation function at the end of the last layer. The input to the decoder which is 128 dimensional latent embedding is first passed through a fully connected layer. The discriminator consists of 5 fully connected layers having Dropout and LeakyReLu activation functions. The Adam optimizer is used for training the model using mini batch gradient descent.

Figure 6.1: Generator, Discriminator and Reconstruction Loss Curves(X Axis - Epochs, Y Axis - Loss)



The best encoder model is chosen on the basis of the best K-Means accuracy. The highest K-Means[7] accuracy which we got is 67.16% at 120 epoch.As we see from Figure 6.2 and Figure 6.3 better reconstruction or generation results are not directly proportional to good K-Means Clustering results.

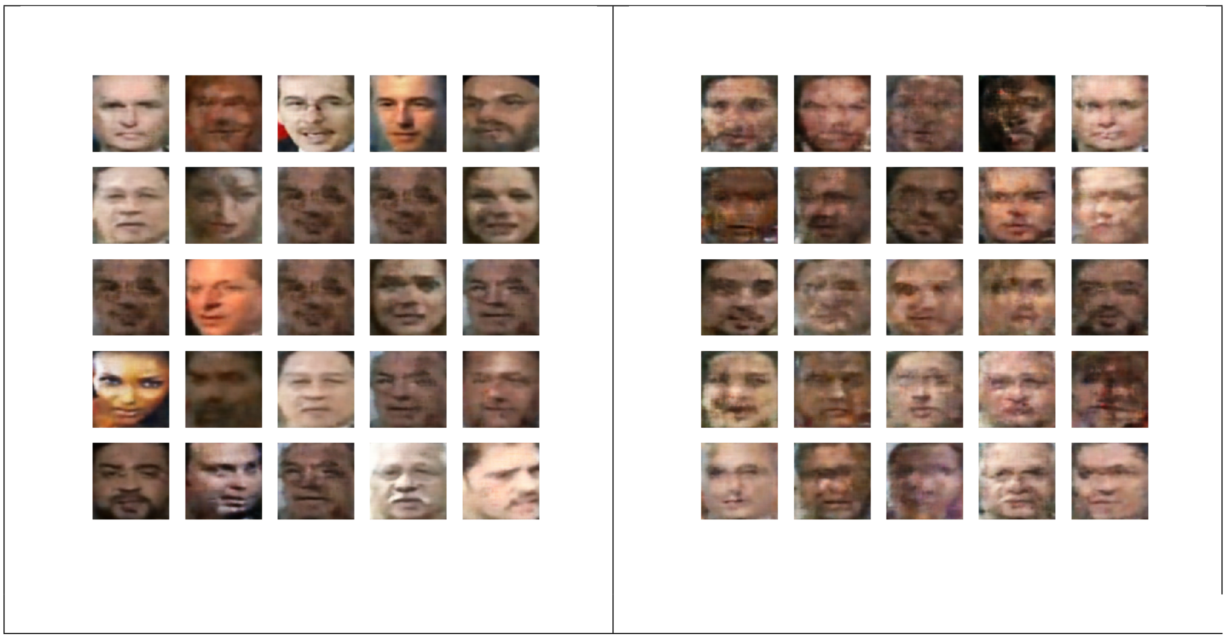
Figure 6.2: Reconstruction and Generated Face Images on Model based on Best K-Means Accuracy



6.1.2 Variational Attention Graph Autoencoder

We use 3 layers of Graph Attention[22] layers with 8 heads in all the layers for performing multi-head attention.Out of the 3 layers 2 layers are used for generating the mean and the standard deviation. In the first layer the output from the multi-attention heads is concatenated while in the last layer the output from the multi-attention heads is averaged. The Graph Attention encoder outputs 64 dimensional latent embeddings.We use ELU activation function and dropout in between layers. The decoder is simply a inner product decoder which reconstructs the adjacency matrix.The Adam optimizer is used for optimizing the weights using stochastic gradient descent.

Figure 6.3: Reconstruction and Generated Face Images on Model of Last Epoch



6.1.3 Variational Graph Convolution Autoencoder(VGAE+GCN)

We use 3 layers of Graph Convolutions in the encoder out of which 2 layers are used for finding the mean and the standard deviation. The encoder outputs latent embeddings of 64 dimensions. The Adam optimizer is used for optimizing the weights using stochastic gradient descent.Dropout layer are used along with ReLU activation function.

6.1.4 Graph Attention Autoencoder

We use 2 layers of Graph Attentions in the encoder.It outputs latent embeddings of 256 dimensions. The Adam optimizer is used for optimizing the weights using stochastic gradient descent.Dropout layer are used along with ReLU activation function.

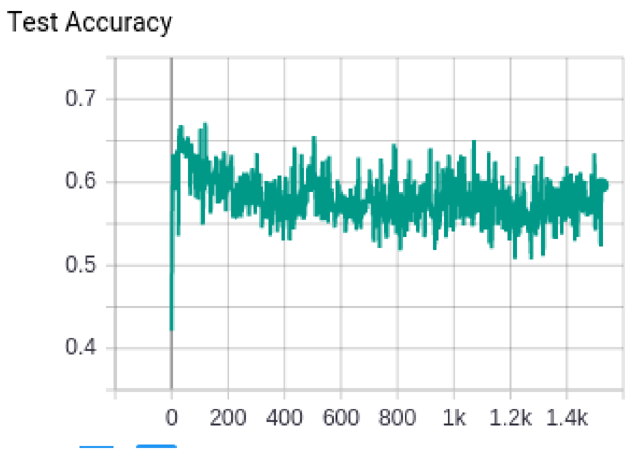
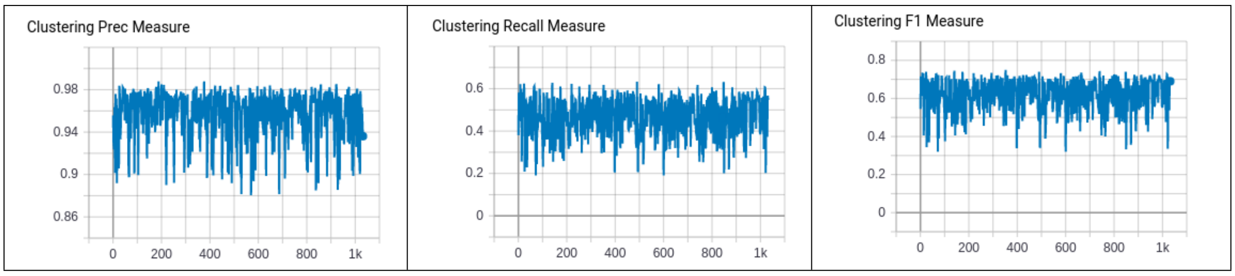


Figure 6.4: K-Means Accuracy on Adversarial Autoencoders on Test Dataset (X Axis - Epochs, Y Axis - Accuracy)

Figure 6.5: Pairwise Precision, Recall and F1 Score Curves(X Axis - Epochs, Y Axis - Scores for VAGAE)



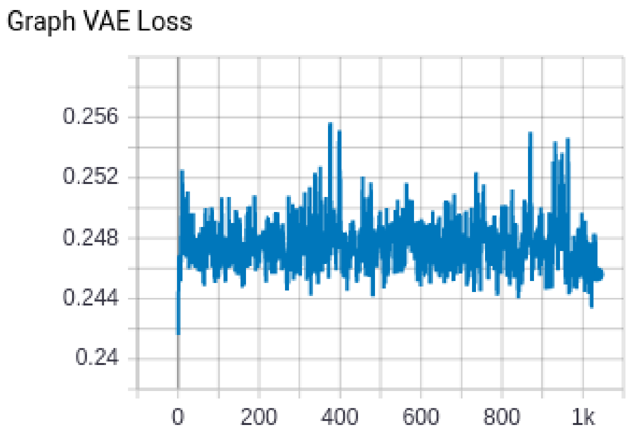
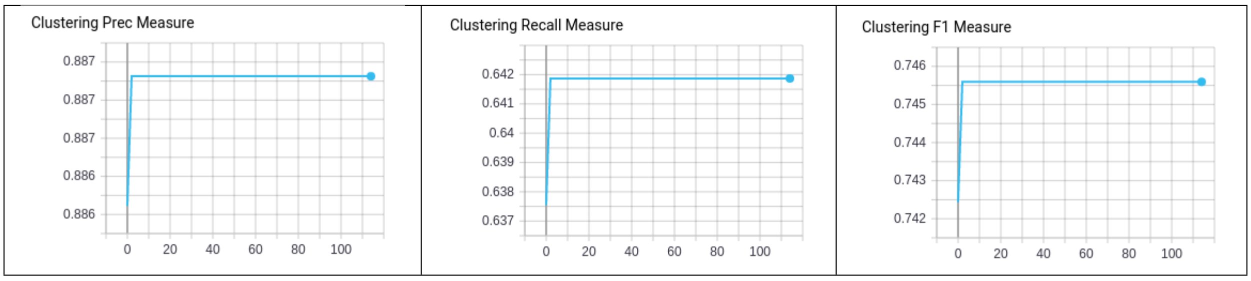


Figure 6.6: Variational Attention Graph Autoencoders(VAGAE) Loss curve on Test Dataset (X Axis - Epochs, Y Axis - Loss)

Figure 6.7: Pairwise Precision, Recall and F1 Score Curves(X Axis - Epochs, Y Axis - Scores) For VGAE+GCN



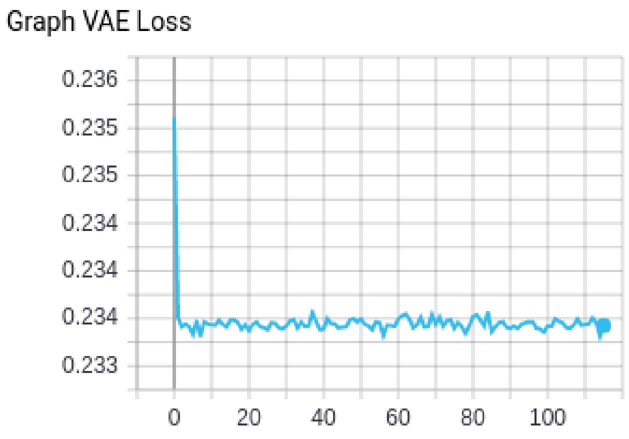
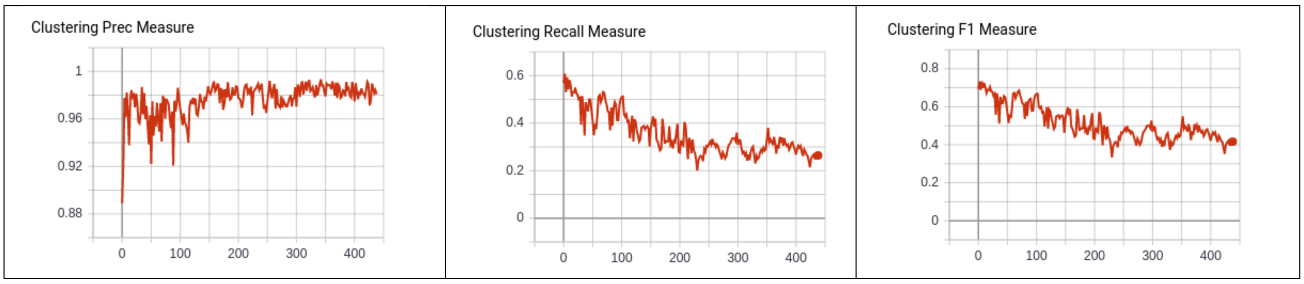


Figure 6.8: VGAE+GCN Loss curve on Test Dataset (X Axis - Epochs, Y Axis - Loss)

Figure 6.9: Pairwise Precision, Recall and F1 Score Curves(X Axis - Epochs, Y Axis - Scores)Graph Attention Autoencoder



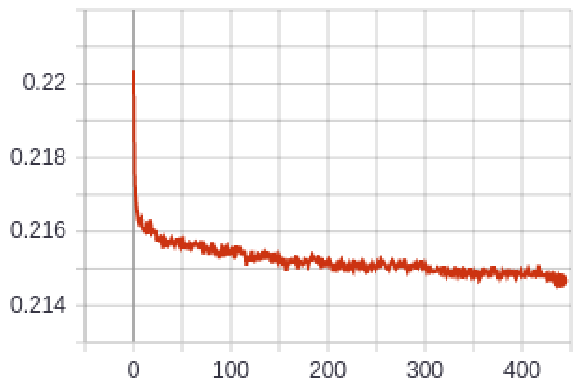


Figure 6.10: Graph Attention Autoencoder Loss curve on Test Dataset (X Axis - Epochs, Y Axis - Loss)

Chapter 7

Experimental Results

This section provides the results of the experiments we have performed on our proposed model and also compared it with other existing clustering models.For Graph Autoencoders we have chosen the best model based on a fixed linking threshold and the best f1 score and then while testing model is fine tuned on other threshold values for best results.

We have evaluated our results using the pairwise f1, pairwise precision and pairwise recall measures which are external measures for evaluation clustering quality relying on the identity labels[2]. The above measures can be defined as follows.

Pairwise Precision : It is defined as the ratio of the total number of pairs of samples within a cluster which belong to the same class to the total number of same cluster pairs within a dataset[2].

$$PairwisePrecision = \frac{TotalCorrectPairsOfclusters}{TotalPairsInAllClusters} \tag{7.1}$$

Pairwise Recall: It is defined as the ratio of the number of same class pairs within a cluster to the total number of same-class pairs within a dataset.[2]

$$PairwiseRecall = \frac{TotalCorrectPairsOfclusters}{TotalPairsOfSameClassPairsInDataset} \tag{7.2}$$

Pairwise F1: Pairwise precision and Pairwise recall can be summarized using the F-measure, defined as [2]

$$PairwiseF1 = \frac{2X(PairwisePrecisionXPairwiseRecall)}{(PairwisePrecision + PairwiseRecall)} \tag{7.3}$$

Comparisons between our model and the state-of-the-art model has been made using B-Cubed F1 measure.B-Cubed F1 Measure works by computing a precision and recall for each individual item and then taking the total precision and recall to be a weighted average of these[30]. The weighted averaged Precision and Recall is then used for calculating the B-Cubed F1 Measure. in (7.6) and (7.7) w_i is assigned equal weight of 1/N[31].

$$B-CubedPrecision_i = \frac{NumberOfCorrectElementsInTheClusterContainingEntityi}{NumberOfElementsInTheClusterContainingEntityi} \tag{7.4}$$

$$B-CubedRecall_i = \frac{NumberOfCorrectElementsInTheClusterContainingEntityi}{NumberOfElementsInTheTrueClusterContainingEntityi} \tag{7.5}$$

$$FinalPrecision = \sum_{i=1}^{N} w_i * B - CubedPrecision_i \tag{7.6}$$

$$Dat \tag{7.7}$$

$$B-CubedF1 = \frac{2X(FinalPrecisionXFinalRecall)}{(FinalPrecision + FinalRecall)} \tag{7.8}$$

So what do precision and recall represent in clustering results. Precision measures the purity of the cluster and recall measures the completeness of the cluster[32].

7.1 Adversarial Autoencoders vs FaceNet

We have proposed to use to train our feature extractors using Adversarial Autoencoders in an unsupervised way, on the unlabelled data from the same distribution as that of the test set, instead of training a classifier network on labelled dataset of a similar domain data from a different distribution first and then using it as a feature extractor. We show that even though the the feature extractor is trained in an unsupervised way it is able to give us almost the same results as compared to the FaceNet features extractor trained in a supervised way for AROC. The TSNE representation of AAE and Facenet on YTF data for 50 classes can be seen in Figure 7.1 and Figure 7.2. It can be clearly seen that even though Adversarial Autoencoders were trained in an unsupervised way it managed to learn similarity embeddings between face images.

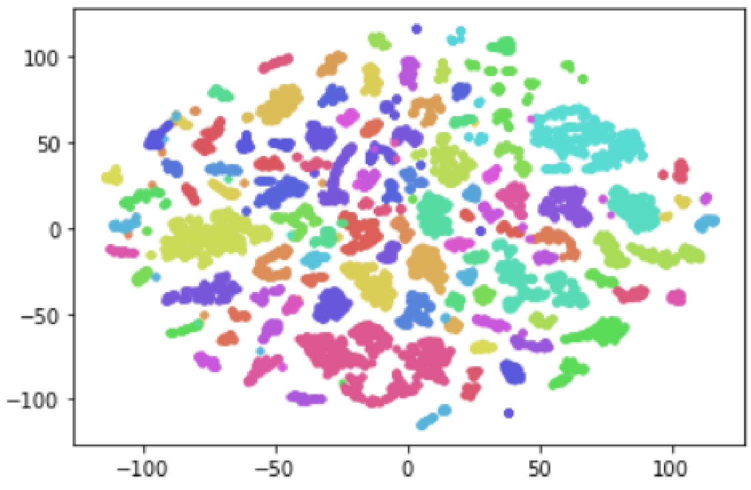


Figure 7.1: TSNE Representation of AAE on YTF[Train + Test] Data[50 classes]

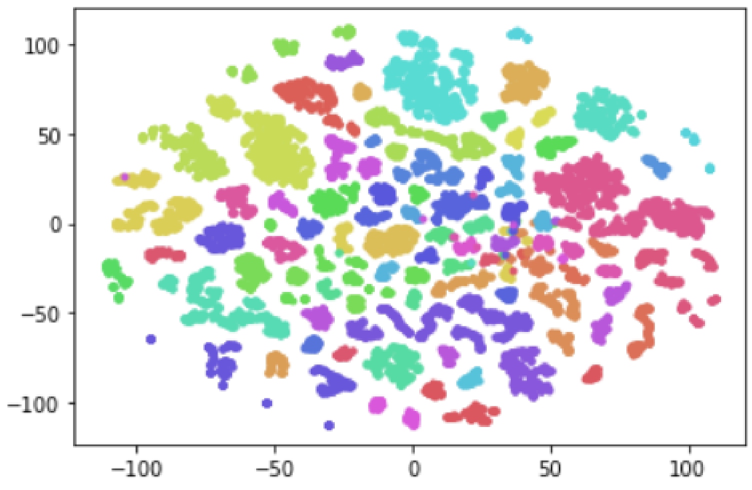


Figure 7.2: TSNE Representation of Facenet on YTF[Train + Test] Data[50 classes]

The results as shown in table 7.1 and table 7.2 show the comparitive results of features extracted using a model trained using Adversarial Autoencoders vs FaceNet architecture on K-Means[7] Approximate Rank Order Clustering(AROC)[2] and our framework. The results clearly indicate how k-means suffers because of its dependency of having convex shaped clusters[3]; K-Means shows a dramatic improvement in metrics when FaceNet features are used compared to Adversarial Autoencoder features. Since AROC and our model makes no such assumptions it there is not much performance difference on using any of these features extractors.

Table 7.1: Performance Evaluation using Adversarial Autoencoder Features on YTF Test Dataset for first 50 classes

Models	Pairwise Precision	Pairwise Recall	Pairwise F1
K-Means	0.7186	0.6254	0.6688
AROC	0.9535	0.66491	0.7834
VAGAE	0.9309	0.6321	0.7529

Table 7.2: Performance Evaluation using FaceNet Features on YTF Test Dataset for first 50 classes

Models	Pairwise Precision	Pairwise Recall	Pairwise F1
K-Means	0.8856	0.7204	0.7945
AROC	0.9624	0.6613	0.7839
VAGAE	0.9247	0.5719	0.7067

7.2 AROC vs VAGAE vs K-Means

As we can see in the Table 7.1 our model has achieved decent metrics compared to the AROC. There is just a 2-3% difference between the two models. Compared to K-Means our model performs far better having almost a 10% boost in F1 measure when features are taken from the feature extractor trained on Adversarial Autoencoders.In Table 7.3 we can see that both the models suffer from the problem of the generation of singleton clusters or small clusters having 1-5% of the samples compared to the actual clusters.Our model is generating 40-45% more singleton clusters compared to AROC which can be the reason for degraded performance.Compared to the actual number of clusters in the dataset which is 50 our model generates 168 out of which 100 clusters have less than 12 samples(Small Clusters) and 68 clusters have samples greater than 11 samples(Best Clusters) compared to AROC which has 69 small clusters and 67 best clusters.One important thing to note here is that out of the 2294 samples in our test set 2185 samples got clustered using our model and 2239 samples got clustered in AROC in the best clusters.

We have also done a evaluation on the full YTF dataset[Test+Train] on 50 classes where in we achieve a better Pairwise Precision compared to AROC but the Pairwise F1 measure goes down owing to a bad Pairwise Recall score which is because of the formation of large number of small clusters and also more number of best clusters compared to AROC.We can see the results in Table 7.4 and Table 7.5.

The possible reason for a lower performance can be that since our architecture is dependent on many hyperparameters including the ISG data generation process, using better hyperparameter tuning techniques or algorithms can improve the performance and might even beat the AROC model.Possible improvement for eliminating the singleton clusters would be to train a Fully-Connected Neural Network on the the best clusters data and then classify each sample in the small clusters into these best clusters.

Table 7.3: AROC vs VAGAE Cluster Statistics

Models	Best Clusters(samples >11)	Small Clusters(samples <12)	Total Clusters	Total Samples in Best Clusters
AROC	67	69	136	2239/2494
VAGAE	68	100	168	2185/2494

Table 7.4: Performance Evaluation using Adversarial Autoencoder Features on YTF[Test + Train] Dataset for first 50 classes

Models	Pairwise Precision	Pairwise Recall	Pairwise F1
K-Means(146 clusters)	0.9695	0.3458	0.5098
AROC	0.9915	0.6482	0.7839
VAGAE	0.9923	0.5575	0.7139

Table 7.5: Cluster Statistics for YTF[Test + Train] Dataset for first 50 classes

Models	Best Clusters(samples >11)	Small Clusters(samples <12)	Total Clusters	Total Samples in Best Clusters
AROC	117	188	305	12113/12468
VAGAE	146	684	830	11384/12468

7.3 How VGAE is helping to improve the clustering results

.

The ISG's generated have some impurities in it which lead to poor results while merging. If we transitively merge the algorithmically generated ISG's or only its instance links, we get a lower bound on our clustering metrics. We remove these impurities by training a VAGAE for it to learn which nodes should be linked. As we can see in the results in Table 7.6 and Table 7.7 we get improved metrics where we get a big bump in the pairwise precision which goes up from 0.8869 to 0.9309 and also F1 measure goes up from 0.7450 to 0.7529 we have a slight decrease in the pairwise recall from 0.6423 to 0.6321. If we just merge the instance links of the ISG's then we get a far low f1 score of 0.6116 compared to VGAE merging of predicted instance links which gives us a F1 score of 0.7529 which clearly shows the power of VAGAE link prediction.

We can see how the VAGAE improves the purity of ISG's in the Figure 7.3 where we show how face images belonging to different class were removed. One drawback which was identified is that VAGAE also removes some face images belonging to the same class which might be the reason of a lower pairwise recall measure. Also to get even deeper insight we have shown in Figure. 7.4 how instance linked predictions are improved. The red boxes indicates nodes with dissimilar identities. In the first row images we can see how dissimilar identites were removed and replaced with similar identity; highlighted with a green box, which was identified from all of the Instance Subgraph Nodes.

Table 7.6: Performance Evaluation of transitively merging the generated ISG's vs instance link of generated ISG's vs instance link predictions from VAGAE

Models	Pairwise Precision	Pairwise Recall	Pairwise F1
Merge ISG's	0.8869	0.6423	0.7450
Merge Instance Links	0.8636	0.4734	0.6116
VAGAE	0.9309	0.6321	0.7529

Table 7.7: Clusters Statistics of transitively merging the generated ISG's vs instance link of generated ISG's vs instance link predictions from VAGAE

Models	Best Clusters(samples >11)	Small Clusters(samples <12)	Total Clusters	Total Samples in Best Clusters
Merge ISG	64	77	141	2214/2494
Merge Instance Links	69	166	235	2084/2494
VAGAE	68	100	168	2185/2494

7.3.1 Final Clustering Results

Figure 7.5 and Figure 7.6 show the final clustering results. In Figure 7.5 we see some random sample images visualized from the good clusters formed. In the first image in Figure 7.5 we can see how two different images of the same identity are clustered together, which means our pipeline did a fairly good job learning the face features.We can also see how even different pose face images of the same identity were clustered together. However in Figure 7.6 we can also see how overfitting to the background or the colour, saturation, and hue of the image caused different identity images to get clustered together.This problem is caused because of the extracting face features using Adversarial Autoencoders. Because of the absence of explicit supervision it just tends to look at the more general structure of the image rather than the specifics. A possible solution for the improvement can be to integrate a attention mechanism in the Adversarial Autoencoders which focuses on the face features to learn the embeddings rather than the overall general background.

7.4 Comparison With The State-Of-The-Art

The state-of-the-art paper when we started working was Linkage Based Face Clustering Using Graph Convolution Networks(LBFGCN)[3] and we have compared our model with it. As explained before, this framework trains a GCN network using labelled data and then tests it on (assumed)unlabelled data from another distribution. When compared to LBFGCN, we have train the graph network on unsupervised data and test the model using the data using the same training data distribution.For a fair evaluation, we have compared our model with LBFGCN using both the training strategies i.e. using same distribution data and different distribution data on train and test.

Table 7.8 shows the performance evaluation LBFGCN vs VAGAE when trained and tested on data from different distributions. The features of both the test data and train data have been extracted from Arcface[26] model. The train set consists of a subset 11656 images from CASIA face dataset and test set consists of 3544 face images from IJB-B dataset. As we can see VAGAE was not able to perform as well as LBFGCN when data distributions of test set and train set were different. This is one use case where supervised learning really helps.

Table 7.9 shows the performance evaluation LBFGCN vs VAGAE when trained and tested on data from same distribution. We can see that VGAE outperforms LBFGCN in terms of precision but performs poorly when it comes to recall. The pairwise f1 measure of VAGAE is close to LBFGCN even after it being trained in an unsupervised way.

Table 7.8: Performance Evaluation of LBFGCN vs VAGAE when trained and tested on data from different distributions

Models	B-Cubed Precision	B-Cubed Recall	B-Cubed F1
LBFGCN	0.8578	0.7501	0.8004
VAGAE	0.8564	0.4305	0.5730

Table 7.9: Performance Evaluation of LBFGCN vs VAGAE when trained and tested on data from same distribution

Models	B-Cubed Precision	B-Cubed Recall	B-Cubed F1
LBFGCN	0.8857	0.7977	0.7806
VAGAE	0.9259	0.5945	0.7241

Figure 7.3: Link Prediction results VAGAE: left column shows before link prediction results and right column shows after link prediction results

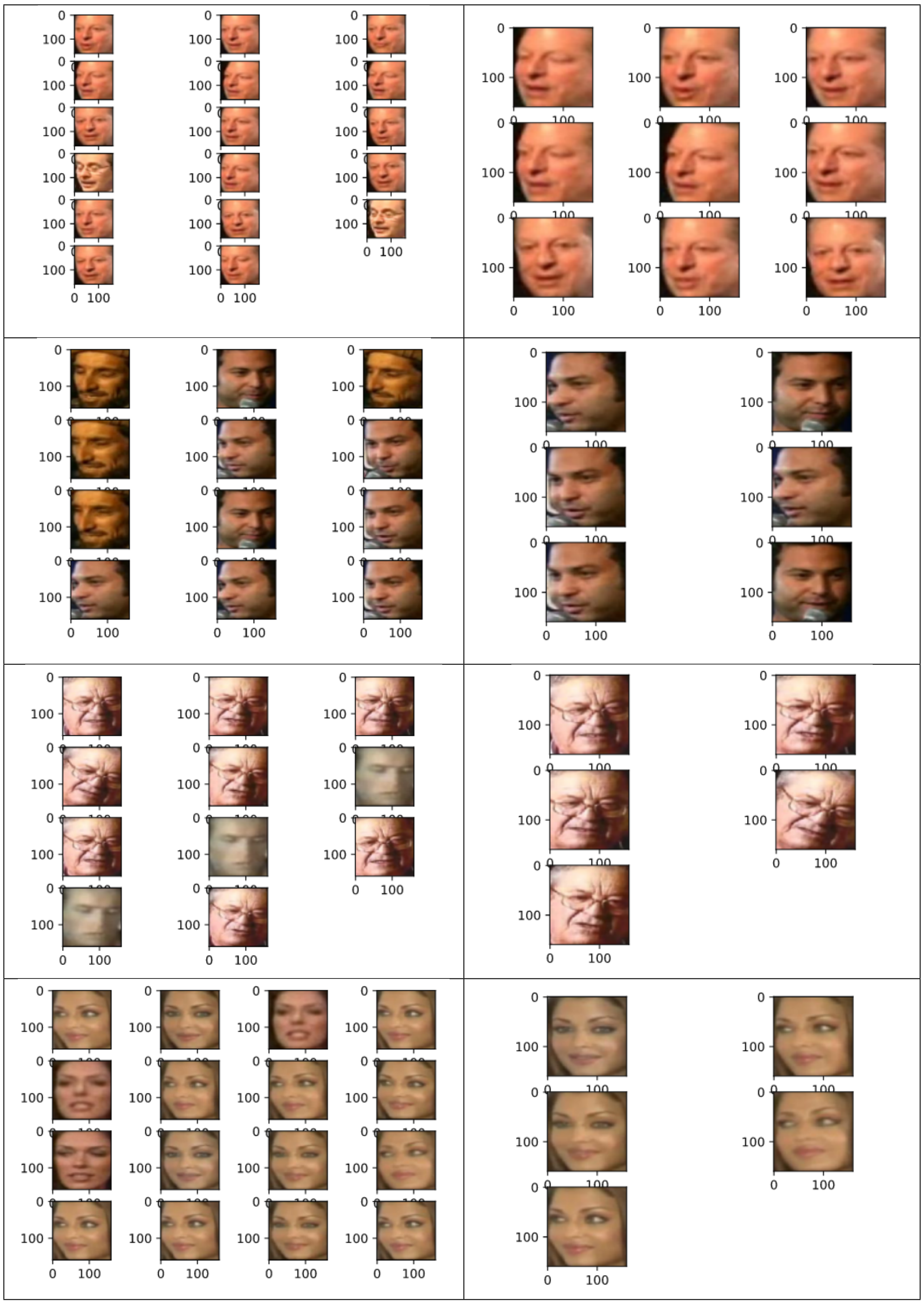


Figure 7.4: How instance link predictions are improved

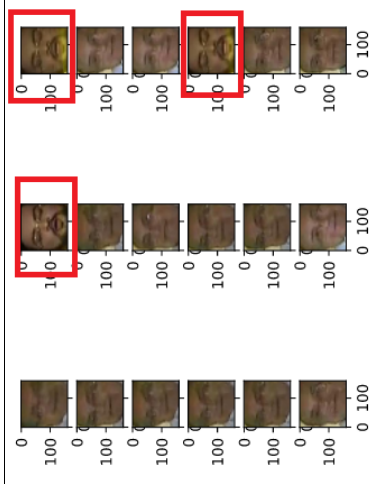
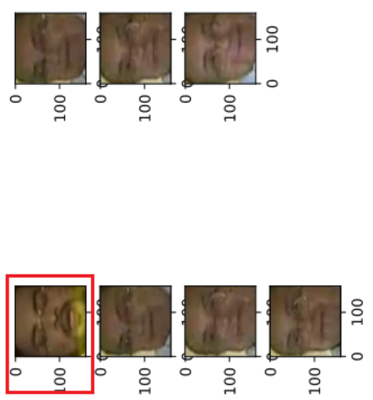
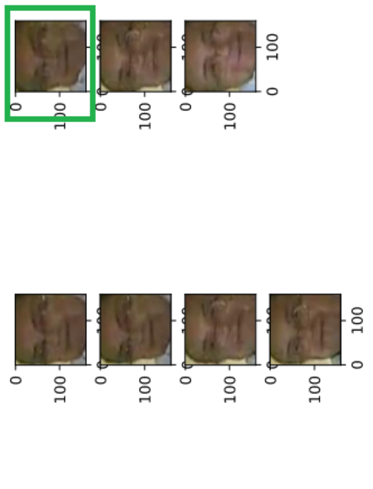
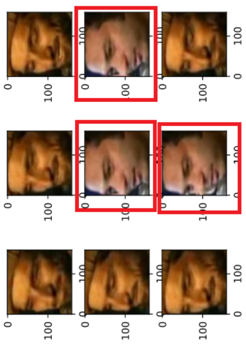
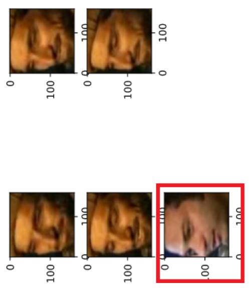
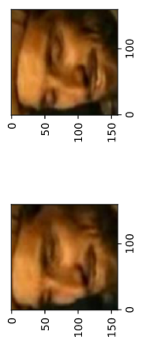
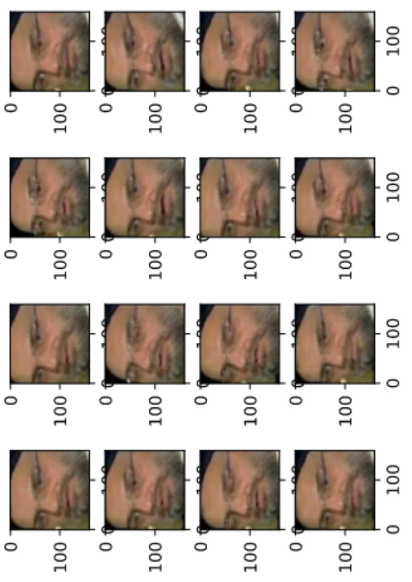
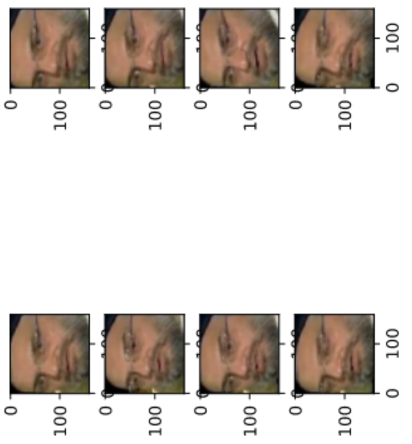
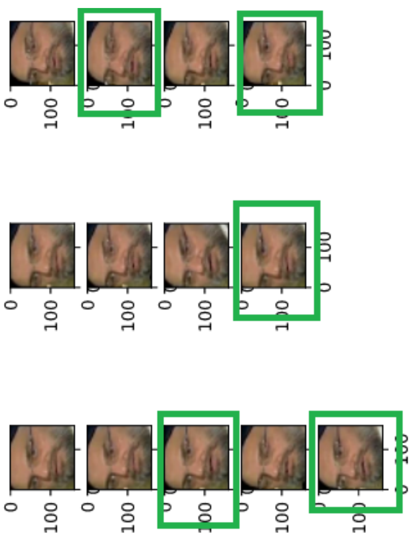
Sr No.	ISG Nodes	Instance Linked Nodes	VGAE Predicted Instance Linked Nodes
1			
2			
3			

Figure 7.5: Random sample images from good clusters

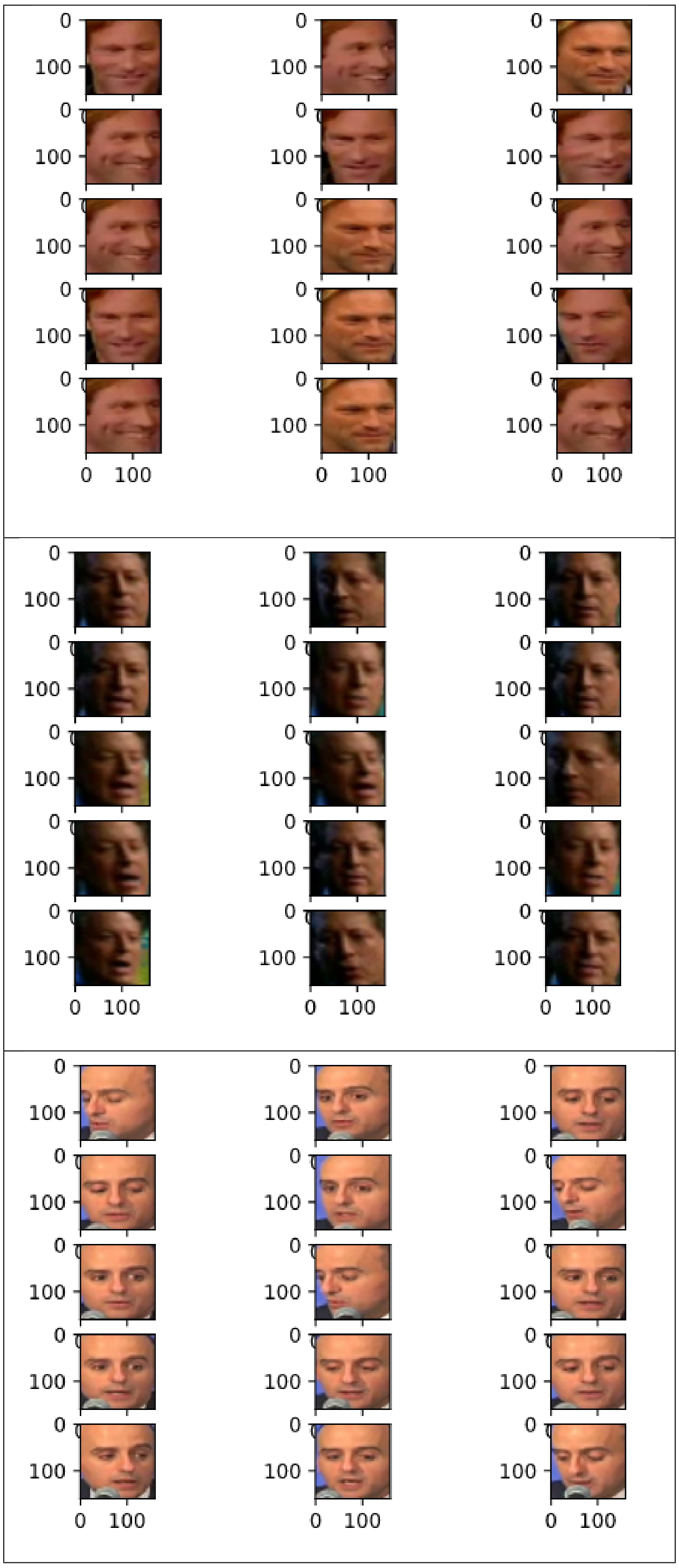
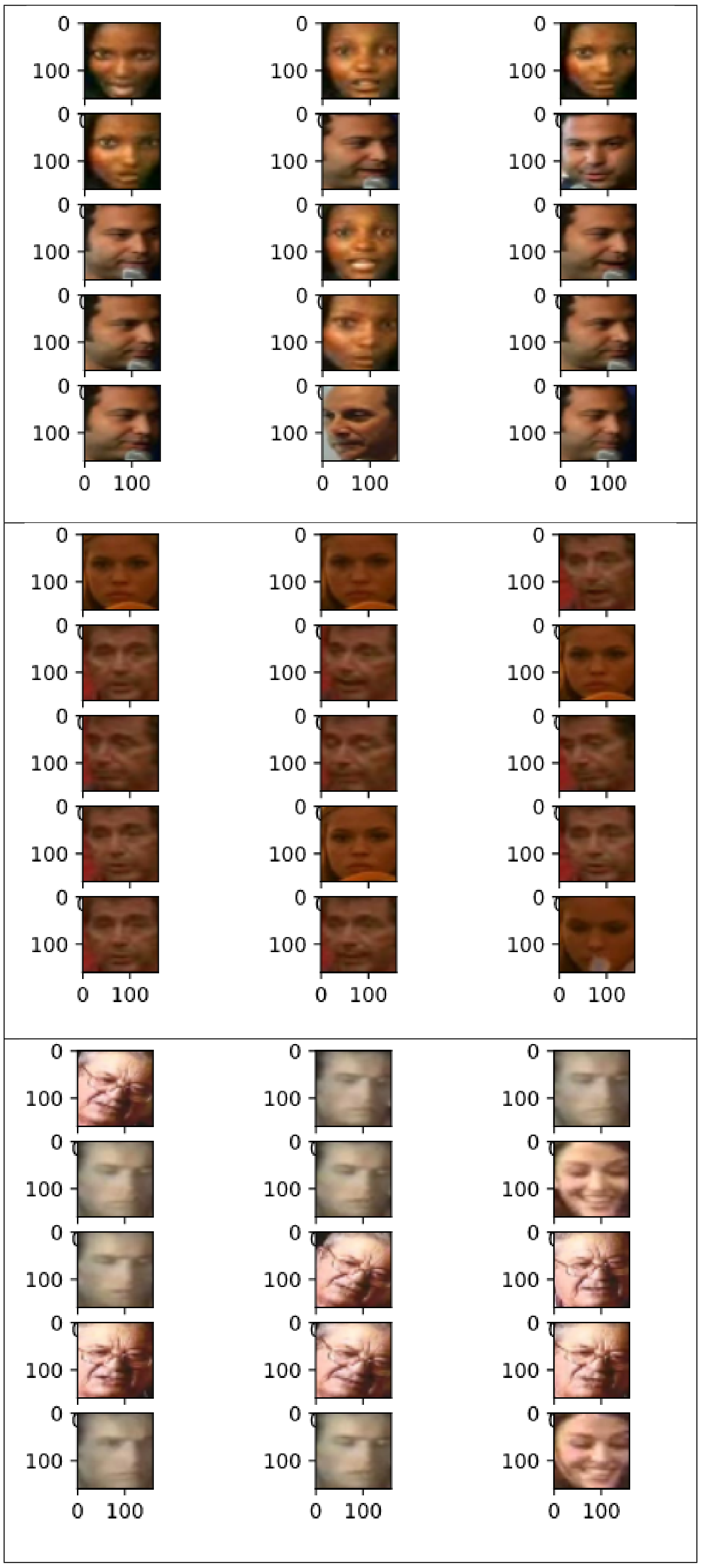


Figure 7.6: Random sample images from bad clusters



Chapter 8

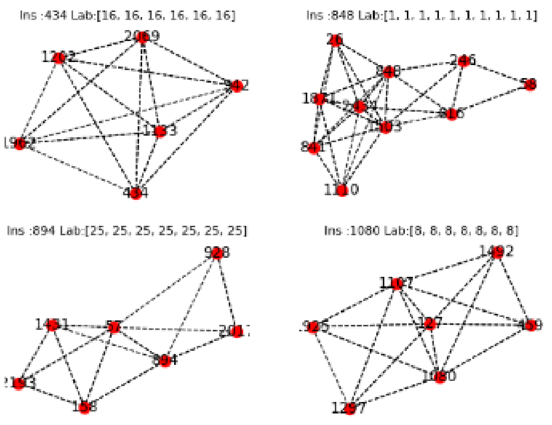
Ablation Study

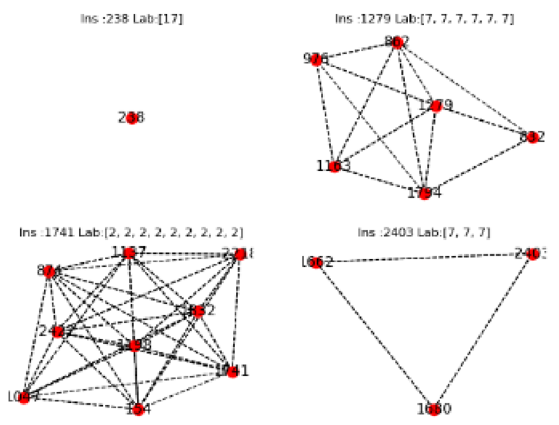
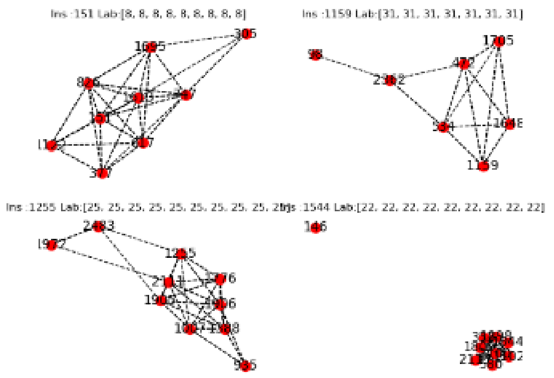
8.1 Hyperparameter Tuning and Sensitivity Analysis of Generating Instance Subgraphs

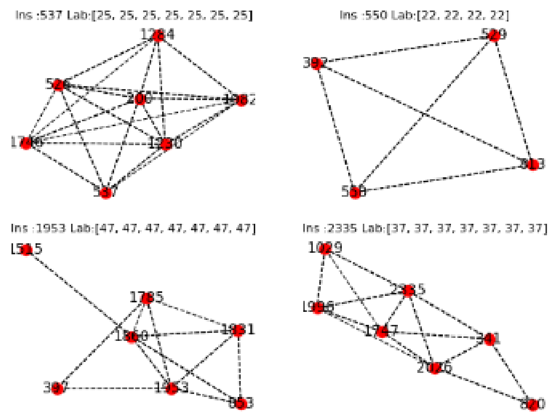
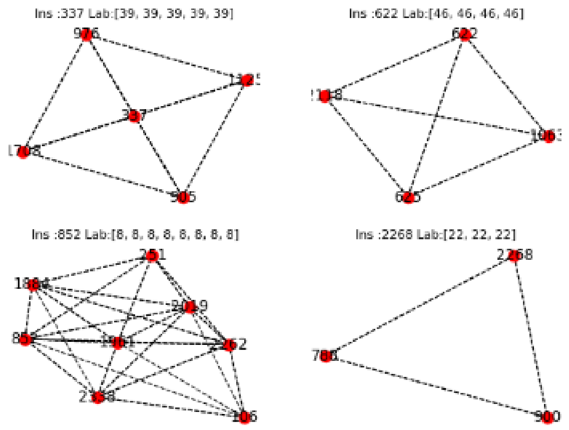
We conducted a thorough analysis of the effect of the hyperparameters K,M,U and Rank threshold on instance graph generation.We can see the results of the study in Table 8.1, decreasing K leads single node ISGs going up by a small amount while increasing them leads to their decrease by a small amount. Increasing M leads to an increase in impure ISG's and decrease in single ISG's while decreasing M leads to vice-versa. Decreasing M leads to slight decrease in number of edges.Changing U has no effect on single or impure clusters but leads to considerably less number of edges.Increasing the Rank Threshold leads to disconnected graphs with a considerable increase in impure clusters and decrease in Single Node ISG's. We first had an assumption that we need to find hyperparameters which given us a perfect balance between impure clusters and single node clusters, where both of them had to be less and also make sure that ISG's generated are not disconnected to get better clustering results using VAGAE. But as we can see at Rank Threshold 40 even though the graphs are not disconnected we got clustering metrics on the lower side than having Rank Threshold as 50. In Table 8.2 and Table 8.3 we can see how changing the Rank Threshold from 50 to 40 changed the pairwise F1 measure from 0.7529 to 0.6993 which shows the how sensitive the hyperparameters are to the clustering results.It also indicates that we should ensure that we have less number of Single ISG's because it affects the overall pairwise recall measure.Less number of single node ISG's will ensure that more ISG's are merged leading to more number of samples in best clusters. Setting

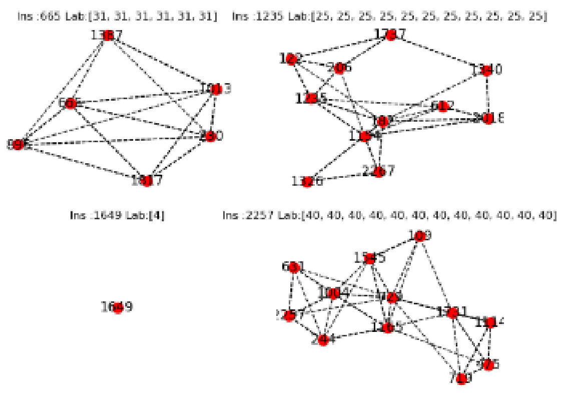
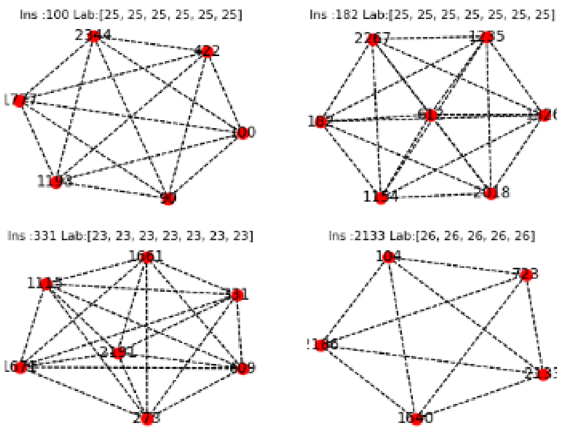
K as 14, M as 3 , U as 5 Rank Threshold as 50 we were able to generate decent ISG's which gave us decent clustering results.

Table 8.1: Hyperparameter Tuning Analysis of Generating Instance Subgraphs

Sr No	K	M	U	Rank Threshold	No of Nodes	No of Edges	Impure ISG's	Single ISG's	Comments
1	14	6	10	15	1-14	1-140	52	110	Many Single node ISG's are formed.
 Ins :434 Lab:[16, 16, 16, 16, 16, 16] Ins :848 Lab:[1, 1, 1, 1, 1, 1, 1, 1, 1, 1] Ins :894 Lab:[25, 25, 25, 25, 25, 25, 25] Ins :1080 Lab:[8, 8, 8, 8, 8, 8, 8]									
2	7	6	10	15	1-13	1-129	51	122	Single ISG's went up on decreasing K.Number of edges went down by a small margin.

 Ins :238 Lab:[17] Ins :1279 Lab:[7, 7, 7, 7, 7, 7] Ins :1741 Lab:[2, 2, 2, 2, 2, 2, 2, 2, 2] Ins :2403 Lab:[7, 7, 7]									
3	14	12	10	15	1-22	1-153	485	47	Increasing M leads to a considerable increase in the number of Impure ISG's and single ISG's went down.
 Ins :151 Lab:[8, 8, 8, 8, 8, 8, 8, 8, 8] Ins :1159 Lab:[31, 31, 31, 31, 31, 31, 31] Ins :1255 Lab:[25, 25, 25, 25, 25, 25, 25, 25, 25, 25] Ins :1544 Lab:[22, 22, 22, 22, 22, 22, 22, 22, 22]									
4	14	8	10	15	1-15	1-140	145	94	Decreasing the value of M led to decrease in the number of impure ISG's

 Ins :537 Lab:[25, 25, 25, 25, 25, 25, 25] Ins :550 Lab:[22, 22, 22, 22] Ins :1953 Lab:[47, 47, 47, 47, 47, 47, 47] Ins :2335 Lab:[37, 37, 37, 37, 37, 37, 37]									
5	14	3	10	15	1-15	1-133	22	139	Further decreasing the value of M leads to less edges while maintaining less number of impure ISG's but the number of single ISG's is on the higher side.
 Ins :337 Lab:[39, 39, 39, 39, 39] Ins :622 Lab:[46, 46, 46, 46] Ins :852 Lab:[8, 8, 8, 8, 8, 8, 8, 8] Ins :2268 Lab:[22, 22, 22]									

6	14	3	5	15	1-13	1-73	22	139	Decreasing U Considerably less number of edges
 Ins :665 Lab:[31, 31, 31, 31, 31, 31] Ins :1235 Lab:[25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25] Ins :1649 Lab:[4] Ins :2257 Lab:[40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40]									
7	14	3	14	15	1-13	1-169	22	139	Changing U has no effect on single or impure clusters, but we do get less number of edges.
 Ins :100 Lab:[25, 25, 25, 25, 25, 25] Ins :182 Lab:[25, 25, 25, 25, 25, 25, 25] Ins :331 Lab:[23, 23, 23, 23, 23, 23, 23] Ins :2133 Lab:[26, 26, 26, 26, 26]									

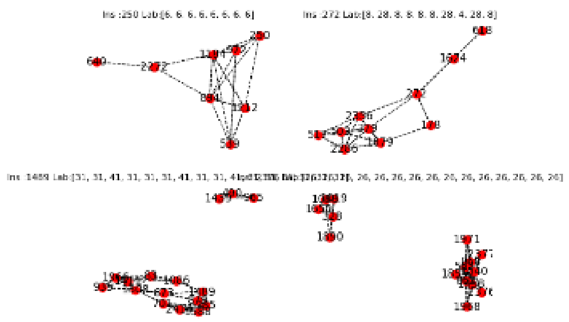
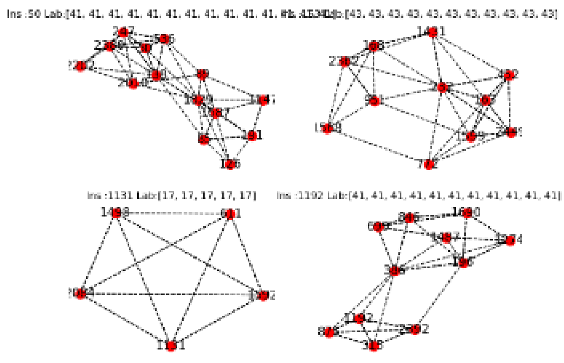
8	14	3	5	50	1-24	1-111	240	11	Increasing the Rank Threshold leads to disconnected graphs. Single ISG's went down considerably with increase in impure clusters.
									
9	14	3	5	40	1-19	1-90	110	16	No more disconnected graphs. Fairly good ISG representation.
									

Table 8.2: Effect of Rank Threshold on Accuracy

Hyperparameters	Pairwise Precision	Pairwise Recall	Pairwise F1
K=14,M=3,U=5,Rank-Threshold=40	0.9726	0.5459	0.6993
K=14,M=3,U=5,Rank-Threshold=50	0.9309	0.6321	0.7529

Table 8.3: Effect of Rank Threshold on cluster statistics

Hyperparameters	Best Clusters(samples >11)	Small Clusters(samples <12)	Total Clusters	Total Samples in Best Clusters
K=14,M=3,U=5,Rank-Threshold=40	72	94	166	2157/2494
K=14,M=3,U=5,Rank-Threshold=50	68	100	168	2185/2494

8.2 Graph Attentional Variational Autoencoder vs Graph Attentional Autoencoder

Using Variational Attentional Graph Autoencoders(VAGAE) gives us better performance than just using Graph Attentional Autoencoders(GAAE) especially on the Pairwise Precision which went up from 0.8964. The possible reason might be the regularization component which Variational Autoencoder imposes on the latent embeddings to make sure that the latent embeddings follow a standard normal distribution which causes it to generate better similarity embeddings which in turn lead to better clustering results.The cluster statistics are of GAAE are somewhat similar to VAGAE. We can see the results in Table 8.4 and Table 8.5

Table 8.4: Performance Evaluation of Variational Attentional Graph Autoencoders vs Graph Attention Autoencoders

Models	Pairwise Precision	Pairwise Recall	Pairwise F1
AGAE	0.8984	0.6340	0.7434
VAGAE	0.9309	0.6321	0.7529

Table 8.5: Clusters Statistics of Variational Attentional Graph Autoencoders vs Graph Attention Autoencoders

Models	Best Clusters(samples >11)	Small Clusters(samples <12)	Total Clusters	Total Samples in Best Clusters
AGAE	63	102	165	2150/2494
VAGAE	68	100	168	2185/2494

8.3 Performance Evaluation of Graph Attention Encoders vs Graph Convolution Encoders

We replaced the encoder in the Variational Graph Autoencoder with Graph Convolution Network instead of Graph Attention network.As we can see in the results in Table 8.6 and Table 8.7 Graph Attention Encoders perform better compared to Graph Convolution Networks. Infact GCN's are showing the same performance as directly merging the algorithmically generated ISG's. The reason being in the weights assigned to its neighbours which are used while aggregating are explicitly learned which leads to better performance. This is the first work where Graph Attention Networks have been used as Encoder in Variational Graph Autoencoder.

Table 8.6: Performance Evaluation of Graph Attention Encoders vs Graph Convolution Encoders

Models	Pairwise Precision	Pairwise Recall	Pairwise F1
GCN	0.8869	0.6423	0.7450
GAT	0.9309	0.6321	0.7529

Table 8.7: Clusters Statistics of Graph Attention Encoders vs Graph Convolution Encoders

Models	Best Clusters(samples >11)	Small Clusters(samples <12)	Total Clusters	Total Samples in Best Clusters
GCN	64	77	141	2214/2494
GAT	68	100	168	2185/2494

8.4 Performance Evaluation of Our Framework without using Rank Order Based Distance for Node Pruning

We check whether the Rank Based Order distance constraint which we have integrated in our framework for node pruning, is really improving the results. To test that we remove the constraint and generate instance link predictions on ISG's and test our clustering framework. As we can see in the results in Table 8.8 and Table 8.9 the performance of the framework after removing Node Pruning is far worse than compared to the results of our framework we get after integrating the Rank Order Based Distance constraint. Infact the lowerbound Pairwise F1 measure which is obtained by just merging the instance links of the ISG's is 0.2211. Also the clustering statistics reveal that VAGAE gives us 52 best clusters, whereas the actual number of clusters are 50 which is near to the actual number of clusters but only 1882/2494 samples getting clustered which indicates that a lot many singleton and small clusters are getting generated.

One important thing here to be noted would be how VAGAE is still managing to learn better link predictions even when without ROB. VAGAE w/o ROB takes the pairwise F1 to 0.6446 from 0.2211 which we would have got if we had merged teh instance links from the ISG's without predicting from the VAGAE. Integration of ROB improves the purity of the clusters which helps in getting better clustering results from our framework.

Table 8.8: Performance Evaluation of our framework without ROB distance

Models	Pairwise Precision	Pairwise Recall	Pairwise F1
Merge ISG w/o ROB	0.0534	0.7998	0.1001
Merge Instance Links w/o ROB	0.1371	0.5708	0.2211
VAGAE w/o ROB	0.8502	0.5191	0.6446
VAGAE	0.9309	0.6321	0.7529

Table 8.9: Clusters Statistics of Graph Attention Encoders vs Graph Convolution Encoders

Models	Best Clusters(samples >11)	Small Clusters(samples <12)	Total Clusters	Total Samples in Best Clusters
Merge ISG w/o ROB	23	31	54	2404/2494
Merge Instance Links w/o ROB	42	215	257	2146/2494
VAGAE w/o ROB	52	340	392	1882/2494
VAGAE	68	100	168	2185/2494

Chapter 9

Conclusion and Future Work

In this work, we presented a novel unsupervised learning framework for clustering which generated decent enough results compared to the heuristic methods and link prediction based methods.We show that our model can perform as well as the LBFGCN even though it is learning in a completely unsupervised manner and can outperform the heuristic method if the hyperparameters are tuned optimally.Currently this framework does not make use of any face specific constraints and hence it can be tried and used in other domain data such as product data where labelled images are hard to obtain.This framework will really shine where data labelling is a tedious task and the data to be clustered will always belong to the same distribution.

We plan to work on the following aspects in future.

1. Optimize the ISG genration algorithm to improve the sensitivity of the hyperparameters or use hyperparameter tuning algorithms to finetune them.
2. Try to integrate attention mechanism in the Adversarial Autoencoders so that it focus on face specific features rather than learning the features from the global structure of the image.
3. Try this framework out on other domains and evaluate the results.
4. Currently the VAGAE uses Stochastic Gradient Descent due to the implementation logic constraints. To make it scalable on large datasets mini-batch gradient descent will have to be integrated.

Bibliography

[1] *A Day in Data*. `https://www.raconteur.net/infographics/a-day-in-data/`. Accessed: 2020-10-12.

[2] C. Otto, D. Wang, and A. K. Jain. "Clustering Millions of Faces by Identity". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.2 (2018), pp. 289–303.

[3] Yali Li Zhongdao Wang Liang Zheng and Shengjin Wang. *Linkage-based Face Clustering via Graph Convolution Network*. 2019.

[4] M. T. Law M. Tapaswi and S. Fidler. "Video Face Clustering with Unknown Number of Clusters". In: *International Conference on Computer Vision (ICCV)* (2019).

[5] Chunhui Zhu, Fang Wen, and Jian Sun. "A rank-order distance based clustering algorithm for face tagging." In: *CVPR 2011*. IEEE Computer Society, 2011, pp. 481–488.

[6] Joshua C. Klontz and Anil K. Jain. "A Case Study of Automated Face Recognition: The Boston Marathon Bombings Suspects". In: *IEEE Computer* 46.11 (2013), pp. 91–94.

[7] J. Hartigan and M. Wong. "A k-means clustering algorithm." In: *JR Stat. Soc., Ser. C.* 1979.

[8] M. Jordan A. Ng and Y. Weiss. "On spectral clustering: Analysis and an algorithm". In: *Advances in neural information processing systems, 2(14),* 2001.

[9] Thomas N Kipf and Max Welling. "Variational Graph Auto-Encoders". In: *NIPS Workshop on Bayesian Deep Learning* (2016).

[10] Alireza Makhzani et al. "Adversarial Autoencoders". In: *International Conference on Learning Representations*. 2016. URL: `http://arxiv.org/abs/1511.05644`.

[11] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering." In: *CVPR 2015*. IEEE Computer Society, 2015, pp. 815–823.

[12] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*. 2020. arXiv: `2003.05991 [cs.LG]`.

[13] Andrew Zisserman Karen Simonyan. *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION*. 2015.

[14] Xifeng Guo et al. *Deep Clustering with Convolutional Autoencoders*. 2017.

[15] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[16] *GRAPH CONVOLUTIONAL NETWORKS.* `https://tkipf.github.io/graph-convolutional-networks/`. Accessed: 2020-10-13.

[17] *Food Discovery with Uber Eats: Using Graph Learning to Power Recommendations.* `https://eng.uber.com/uber-eats-graph-learning/`. Accessed: 2020-10-13.

[18] Thomas N Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[19] Pierre Vandergheynst Michaël Defferrard Xavier Bresson. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.* 2016.

[20] Zonghan Wu et al. *A Comprehensive Survey on Graph Neural Networks.* 2019.

[21] Max Welling Thomas N. Kipf. *Semi-Supervised Classification with Graph Convolutional Networks.* 2017.

[22] Petar Veličković et al. "Graph Attention Networks". In: *International Conference on Learning Representations* (2018). accepted as poster. URL: `https://openreview.net/forum?id=rJXMpikCZ`.

[23] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *ICLR 2015* (2015). URL: `https://arxiv.org/abs/1409.0473`.

[24] Kelvin Xu et al. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.* 2015.

[25] *Graph Attention Networks.* `https://petar-v.com/GAT/`. Accessed: 2020-10-14.

[26] Jiankang Deng et al. "ArcFace: Additive Angular Margin Loss for Deep Face Recognition". In: *CVPR.* 2019.

[27] Z. Li K. Zhang Z. Zhang and Y. Qiao. "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks". In: *IEEE Signal Processing Letters.* 2016.

[28] *Face Recognition Using Pytorch.* `https://github.com/timesler/facenet-pytorch`. Accessed: 2020-10-17.

[29] Soumyadip Sengupta Jun-Cheng Chen Carlos Castillo Vishal M. Patel Rama Chellappa and David W. Jacobs. "Frontal to Profile Face Verification in the Wild". In: *IEEE Winter Conference on Applications of Computer Vision (WACV).* 2016.

[30] Eric Shyu. "Latent Tree Structure Learning for Cross-Document Coreference Resolution". In: *Massachusetts Institute of Technology 2014.* 2014.

[31] E. Amigó, Gonzalo J., and J. et al Artiles. "A comparison of extrinsic clustering evaluation metrics based on formal constraints." In: *Inf Retrieval 12, 461–486 (2009).*

[32] Mirco Kocher and Jacques Savoy. "Author Clustering". In: *UniNE at CLEF 2017.*