TÉCNICO LISBOA

A Mixture-of-Experts approach to deep image clustering

Estimating latent sizes and number of clusters

Pedro de Távora Barroco e Moreira dos Santos

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Doutor Pedro Filipe Zeferino Tomás
Doutora Helena Isabel Aidos Lopes Tomás

Examination Committee

Chairperson: Doutora Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Doutor Pedro Filipe Zeferino Tomás
Member of the Committee: Doutor Luís Miguel Parreira Correia

December 2020

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acima de tudo quero agradecer à minha família e aos meus orientadores por toda a paciência e por me guiarem neste último ano de mestrado. Um gigante obrigado à minha família por me aturar e fingir que percebiam o que eu estava a dizer quando falava em geração de imagens de zeros e uns ou quando eu dizia que a minha tese ia mudar o mundo por razões completamente descabidas e esotéricas. Um muito obrigado aos meus orientadores Pedro Tomás e Helena Aidos por me ouvirem quase dia sim dia sim e por estarem sempre disponíveis dia e noite (até com uma filha bebé ao colo) para me tirarem dúvidas que de certeza que podiam esperar mais uns dias. Sem os professores não teria gostado tanto desta tese e o trabalho teria sido infinitamente mais difícil.

Por fim gostaria de agradecer aos meus amigos que me apoiaram durante todo estes 5 anos de técnico. Obrigado aos que me forçaram a sair de casa durante os primeiros anos de licenciatura quando andava a desesperar com Álgebra, aos do técnico a quem sinto que posso contar tudo e ter conversas super profundas num dia e no dia seguinte estarmos a ir para o urban, aos da suíça que me ensinaram que é possível ter uma vida social, ser extremamente humilde e mesmo assim atingir excelência não só académica como pessoal, e aos dos Maristas que estiveram lá para isto tudo e muito mais. Um muito obrigado a todos vós, isto não teria sido possível sem o vosso apoio e esta tese também é vossa

Abstract

Deep clustering is a field with many widespread applications, ranging from Biology to Marketing. It differs from classical clustering since it uses deep learning algorithms (typically autoencoders) to perform representation learning on the raw data. Despite the importance of these deep representation learning algorithms, little to no priority is given to their structure, leading most theoretical works to attain poor performances in real-life applications. Moreover, while generative clustering approaches can also be used for data generation, most bodies of work do not allow specialization of parts of the network to different clusters, making it hard to influence the type of data that is generated. Hence this thesis aims at exploring a novel deep clustering technique based on a mixture of variational autoencoders (VAEs), the experts, in which each VAE models a cluster. A manager network is then used to estimate a relative importance score to each of the experts, effectively attaining a soft clustering. The main contributions of this work are fourfold: (i) it is a generative approach to clustering; (ii) it has a fully data dependant architecture, removing the need for most hyperparameter selection; (iii) by relying on Principal Component Analysis, it can estimate the dimensionality of the VAEs bottleneck layers, simplifying the model; and (iv) by relying on HDBSCAN (instead of a classical Gaussian Mixture Model, as in N2D) it allows to estimate the number of clusters automatically. The proposed approach is evaluated on five image datasets and compared against state-of-the-art deep clustering approaches. Results show that it surpasses those solutions, becoming the definitive baseline for deep image clustering.

Keywords

Deep image clustering, Image Generation, Mixture of experts model, Pretraining architecture, number of clusters, dimensionality of the autoencoders latent space

Resumo

Deep clustering é um campo com as mais variadas aplicações, que vão da Biologia ao Marketing. Difere do clustering clássico, pois usa algoritmos de deep learning (geralmente autoencoders) para realizar feature learning nos dados originais. Apesar da importância desses algoritmos de feature learning, pouca ou nenhuma prioridade é dada à sua estrutura, levando a que a maioria dos trabalhos teóricos atinja desempenhos maus em aplicações reais. Além disso, as abordagens de agrupamento generativo atuais geralmente não permitem a especialização de partes da rede para diferentes clusters, tornando difícil influenciar o tipo de dados que é gerado. Esta tese visa explorar uma nova técnica de clustering profundo baseada em uma Mistura de especialistas de Autoencoders Variacionais (VAEs). Nesta abordagem cada especialista modela um cluster e uma rede gerente é usada para estimar uma pontuação de importância relativa para cada um dos especialistas, atingindo efetivamente um agrupamento soft. As principais contribuições deste trabalho são quatro: (i) é uma abordagem generativa para agrupamento de dados; (ii) possui uma arquitetura totalmente dependente de dados, eliminando a necessidade de seleção de hiperparâmetros; (iii) usando Análise de Componentes Principais, podemos estimar a dimensionalidade da dimensão latente dos VAEs, simplificando o modelo; e (iv) usado o HDBSCAN (em vez de um modelo de mistura gaussiana clássico, como no N2D), podemos estimar o número de clusters automaticamente. A abordagem proposta é avaliada em cinco conjuntos de dados de imagens e comparada com abordagens de clustering profundo de última geração provando que o algritmo proposto supera a maior parte das soluções, tornando-se uma referência para deep clustering.

Palavras Chave

Agrupamento neuronal de imagens, Geração artificial de imagens, Modelo de mistura de especialistas, Arquitetura de pré-treino, número de grupos, dimensão do espaço latente do autoencoder

Contents

1 Introduction **2**

1.1 Motivation . . . 3

1.2 Ojectives . . . 4

1.3 Main contributions . . . 4

1.4 Thesis outline . . . 5

2 Background on Unsupervised Learning **7**

2.1 Classical clustering . . . 8

2.1.1 Automatic selection of the number of clusters . . . 11

2.2 Deep clustering . . . 16

2.2.1 Basic building blocks of deep dimensionality reduction . . . 17

2.2.2 Autoencoder based clustering . . . 22

2.2.3 Generative model based clustering . . . 27

2.2.4 Direct group optimization clustering . . . 28

2.2.5 Mixture of experts clustering . . . 29

2.3 Evaluation metrics . . . 31

2.3.1 Clustering metrics . . . 31

2.3.2 Image quality metrics . . . 33

2.4 Summary . . . 36

3 Architecture overview **39**

3.1 Clustering methodology . . . 40

3.2 Overview of the training (MoE) architecture . . . 41

3.3 Pre-training Architecture . . . 42

3.3.1 Latent dimension finder . . . 43

3.3.2 UMAP . . . 45

3.3.3 HDBSCAN for automatic cluster and outlier detection . . . 46

3.3.4 Manager Training . . . 47

3.4 Dynamic architecture features . . . 48

3.4.1 Convolutional architecture . . . 48
3.4.2 Depth of the architecture . . . 48
3.4.3 Activation functions . . . 49
3.5 Summary . . . 51
4 Experiments **52**
4.1 Datasets . . . 53
4.2 Choice of autoencoder for the MoE . . . 54
4.3 Loss construction for the VAEs . . . 56
4.4 Pretraining . . . 57
4.4.1 Data-driven latent space dimension finder . . . 58
4.4.2 Usage of UMAP on the pretraining stage . . . 59
4.4.3 Usage of HDBSCAN on the pretraining stage . . . 60
4.4.4 Manager training . . . 64
4.5 Training . . . 68
4.5.1 Data dependant latent space for the experts . . . 69
4.5.2 Influence of α and β in the training stage . . . 71
4.5.3 Expertise of experts . . . 72
4.6 Final accuracy results . . . 75
4.6.1 Comparison to the mixture of expert's methods . . . 76
4.6.2 Comparison to autoencoder based methods . . . 76
4.7 Summary . . . 77
5 Conclusion **78**
5.1 Review of the work done . . . 79
5.2 Future work . . . 79
5.3 Closing remarks . . . 80

List of Figures

2.1 Dendogram for clustering US states based on distance . . . 9
2.2 Example of centroid optimization based clustering . . . 10
2.3 Example of distribution optimization based clustering . . . 11
2.4 DBSCAN algorithm . . . 14
2.5 Hypersphere inscribed on a hypercube in a three dimensional space . . . 14
2.6 Autoencoder general architecture . . . 17
2.7 Reparameterization trick . . . 20
2.8 WAE-GAN general architecture . . . 20
2.9 GAN general architecture . . . 21
2.10 DEC-based methods general architecture . . . 23
2.11 DEPICT general architecture . . . 25
2.12 MoE general architecture . . . 29

3.1 Diagram of the pretraining architecture . . . 42
3.2 Explained variance ratio for a VAE trained on the MNIST dataset . . . 44
3.3 Diagram of the Dynamic latent space finder . . . 44
3.4 All projections for the MNIST dataset . . . 46
3.5 Diagram of the full architecture . . . 47
3.6 Diagram of the Convolutional blocks . . . 48
3.7 Diagram of the Autoencoder-like architecture . . . 49
3.8 ReLU activation function . . . 50
3.9 SWISH activation function . . . 50
3.10 MISH activation function . . . 51

4.1 Example images of the MNIST dataset . . . 53
4.2 Example images of the FMNIST dataset . . . 53
4.3 Example images of the USPS dataset . . . 53

4.4 Example images of the CIFAR10 dataset . . . 54
4.5 Example images of the COIL20 dataset . . . 54
4.6 Histograms for all datasets . . . 57
4.7 Images from COIL20 and CIFAR10 . . . 58
4.8 Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) for the USPS dataset . . . 61
4.9 Images from the noise vector in the USPS dataset . . . 62
4.10 UMAP for the MNIST dataset . . . 63
4.11 Images from the noise vector in the MNIST dataset . . . 64
4.12 UMAP for the FMNIST dataset . . . 65
4.13 Found clusters for the 'Trouser' cluster . . . 66
4.14 Found clusters for the 'Bags' cluster . . . 66
4.15 Found clusters for the 'Sneaker' cluster . . . 66
4.16 Found clusters for the 'Ankle Boot' cluster . . . 67
4.17 Found clusters for the 'Sandals' cluster . . . 67
4.18 Found clusters for the 'T shirt' cluster . . . 67
4.19 Found clusters for the 'Pullover' cluster . . . 68
4.20 Found clusters for the 'Dress' cluster . . . 68
4.21 UMAP for the COIL20 dataset . . . 69
4.22 Different images of sevens and fours . . . 70
4.23 Reconstructed images . . . 73
4.24 Images generated from the experts trained on the USPS dataset . . . 73
4.25 Images generated from the expert specialized in fours . . . 74
4.26 Images generated from the expert specialized in fives . . . 74
4.27 Latent dimension for the expert specialized in ankle boots . . . 74

Acronyms

ACAI Adversarially Constrained Autoencoder Interpolation

ACC Clustering Accuracy

AE Autoencoder

ARI Adjusted Rand Score

ASPC-DA Adaptive Self-Paced Deep Clustering with Data Augmentation

CNN Convolutional Neural Network

DA Data Augmented

DAC Deep adaptive image clustering

DAE Denoising Autoencoder

DAMIC Deep Autoencoder Mixture Clustering

DBC Discriminatively Boosted Image Clustering

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DCEC Deep Convolutional Embedding Clustering

DCN Deep Continuous Clustering

DEC Deep Embedding Clustering

DEPICT Deep Embedded Regularized Clustering

DMC Deep Manifold Clustering

DynAE Deep Clustering using a Dynamic Autoencoder

GAN Generative Adversarial Network

GMM Gaussiam Mixture Model

GMM-MML Gaussiam Mixture Model with Minimum Message Length

GMVAE Gaussian Mixture Variational Autoencoder

HDBSCAN Hierarchical Density-Based Spatial Clustering of Applications with Noise

InfoGAN	Information Maximizing Generative Adversarial Network
JULE	Joint Unsupervised Learning of Deep Representations and Image Clusters
KL	Kullback–Leibler
MoE	Mixture of Experts
MSE	Mean Squared Error
N2D	Not Too Deep Clustering
NMI	Normalized Mutual Information
PCA	Principal Component Analysis
SS	Silhouette Score
T-SNE	T-Student Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection for Dimension Reduction
VaDE	Variational Deep Embedding
VAE	Variational Autoencoder
WAE	Wasserstein Autoencoder

1

Introduction

Contents

1.1 Motivation 3
1.2 Ojectives 4
1.3 Main contributions 4
1.4 Thesis outline 5

1.1 Motivation

"Chaos is merely order waiting to be deciphered."
The Double, José Saramago.

The act of making sense of noise is a primordial human desire. From the beginning of life to modern society, we long for making sense of our relationships, our experiences, and our feelings. In recent years, with the advent of portable sensors, we can measure almost anything: from our heartbeat to the distance we run, to the sugar levels in our bloodstream. We can collect unprecedented amounts of data from nearly everything in our lives. With this amount of data, it is necessary to have mechanisms that make order out of chaos. Currently, there are two contrasting ways to analyze raw data. We can interpret it in a supervised or unsupervised manner.

The supervised methodology uses a human expert to analyze each sample of the data, providing knowledge to the algorithm. The two primary purposes for this type of analysis are regression and classification. Regression consists on the estimation of continuous variables such as stocks, based on information such as past values, and classification is the assignment of discrete variables (classes) based on the earlier associations made from the training data and their labels. We can see supervised learning as a system generating a mathematical formula that best fits input to output while learning to generalize to new unseen data. The main advantage of a supervised approach is that we account for semantic meaning for the data and usually have better performance than unsupervised methods. However the better performance of the supervised methods may be misleading: since human experts set performance with inherent human biases, supervised methods also have this inherent bias built into them, making the performance on supervised labels better than unsupervised methods. The disadvantages of the supervised method are that we need a human expert to categorize the data (which on large datasets may take years). We also force the algorithm to learn the human interpretation of the data, having no new insights about the data.

In contrast, the unsupervised method uses only the data to bring order by clustering into different groups (no tags or human intervention). Its main aim is to explore the underlying patterns and predict the output. Here we provide the machine with data and ask to look for hidden features and cluster it in a way that makes sense. The advantages of the unsupervised approach are that it takes only the input the data, and it provides insights that may not have been initially thought of by human experts. However, the main disadvantage of these methods is the lack of semantic interpretability of the data clustering: just because it makes sense for the algorithm does not mean that humans can understand the assignments given. Despite these apparent advantages of unsupervised learning, most state of the art research in machine learning is based on supervised learning, leaving clustering research with a scarce number of papers in comparison. Clustering is a widespread methodology in several areas such as image segmentation

[Coleman and Andrews, 1979, Pappas and Jayant, 1989, Chuang et al., 2006], Marketing [Harrigan, 1985, Srivastava et al., 1981, Arimond and Elfessi, 2001], Genetics [Sleegers et al., 2004, Solovieff et al., 2010], Biology [de Queiroz and Good, 1997, Gönen and Margolin, 2014, Damian et al., 2007], Fake news identification [Hosseinimotlagh and Papalexakis, 2018, Parikh and Atrey, 2018] and medicine [Ames et al., 2019] however, in this dissertation, we will exclusively deal with image clustering.

Image clustering is a subset of clustering that focuses on the grouping of sets of images. It can have many uses such as face recognition [Schroff et al., 2015], database image retrieval [Shyu et al., 2004], medical diagnosis [Li et al., 2011] and cancer detection [Taher and Sammouda, 2011]. This type of clustering is a tough task since even small images may have hundreds or even thousands of correlated pixels, making phenomena such as the curse of dimensionality more apparent and harder to handle. There are very few clustering algorithms solely dedicated to images, and their performances, interpretability, theoretical formulation, and overall architectural decisions leave much to be desired.

Despite dealing with image data, there are several challenges that we must overcome to extract meaningful information from any high dimensional dataset. The first challenge is called the curse of dimensionality. The curse of dimensionality [Bellman, 1957] is a blanket term to refer to phenomena that arise when analyzing data in a high dimensional space, compared to a low dimensional space such as two or three dimensional. The main idea of this ”Curse” is that when we add dimensions to a problem, the volume of the space increases exponentially, making the data distribution sparse and the underlying relationships hard to analyze, due to their similarities. The ”Curse” appears in many fields that require very high dimensional data, from dynamic programming to machine learning and databases.

1.2 Ojectives

The objective of this dissertation is to review the state-of-the-art in regards to clustering and to, either improve existing methods or to develop a novel clustering methodology using a deep learning approach. Ideally, this methodology should be able to determine the appropriate number of clusters automatically and be generative, i.e., be able to generate new data based on previous observations and for each cluster. This allows us to generate artificial data for representation/analysis purposes. Since no two datasets are equal, another goal was having an architecture with as few static parameters as possible: the parameters must be data dependant.

1.3 Main contributions

We built an original architecture based on the mixture-of-experts framework. Each expert models and generates a separate cluster of data, and the data distribution is controlled by a manager. We used

a fully dynamic architecture in which the depth of the network, the latent size of the experts, and the number of experts used to model the data are all data dependant. Hence, the main contributions of this dissertation are as follows:

- **Generative approach to clustering -** The architecture is generative, i.e., we can generate data from each of the clusters separately. This ability to create per-cluster data presents a new approach for understanding not only how the algorithm groups data, but what prominent features from the data are used when making these decisions.
- **Mixture of experts** - The proposed architecture surpasses state-of-the-art mixture of experts clustering approaches by several percentage points, becoming the *de-facto* model for this type of framework.
- **Automatic clustering estimation** - The proposed model also features an automatic number-of-centroid finder based on the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) and the Not Too Deep Clustering (N2D) architecture. With this feature in place, there is no need to hardcode the appropriate number of clusters and to perform previous data exploration.
- **Automatic autoencoder latent size finder** - One of the most crucial problems when using autoencoder-like architectures is knowing the number of neurons in the latent space often being left as a hyper-parameter. We devised a methodology based on Principal Component Analysis (PCA) decomposition that eliminates the need for guesswork. An estimate of the optimal latent size is made based on the explained variance of the PCA decomposition of the data at the autoencoder bottleneck.
- **Comparison of autoencoders in the architecture -** We also compared several autoencoders to determine the best architecture for clustering purposes, always relating each result to the theoretical background of each architecture.

1.4 Thesis outline

The remaining chapters of this thesis are split as follows. The second chapter provides an overview of classical and deep clustering architectures, separated by their approach to the clustering problem and their handling of the data. We report on the cornerstones that most state-of-the-art clustering algorithms use, and attempt to extract powerful insights from each one. The chapter also discusses automatic cluster detection algorithms, evaluation metrics for image compression, clustering and image complexity. The third chapter provides a theoretical overview of the architecture: We talk about the basic formulation of the architecture, it's pretraining and training stages and discuss the several improvements

that were made when compared to other clustering architectures. We also discuss the building blocks of the fully data-dependant architecture and the several options for activation functions available to us. The fourth chapter provides practical results that justify our design choices when building the architecture and introduces the datasets used when analyzing our model. We provide answers to questions such as what autoencoders to use, the loss function associated with said autoencoders, why use a dynamic latent space dimension on the autoencoder architecture, why use data augmentation, among others. We discuss the results obtained in the pretraining and training stages, along with several variations and comparisons of performance at each step. In this chapter, we also compare our architecture with several state-of-the-art approaches among several clustering sub-groups, providing several metrics that justify our approach and results. Finally, the fifth chapter presents the main contributions of the thesis and outlines future research directions.

2

Background on Unsupervised Learning

Contents

2.1 Classical clustering 8
2.2 Deep clustering 16
2.3 Evaluation metrics 31
2.4 Summary 36

The task of clustering data in an unsupervised manner is not new. Since its inception, several models have been analyzed and proposed. These methods do not present a unified framework, but rather several interpretations on what clustering data means. When dealing with the grouping of data, we can have different definitions for what constitutes a group in the space, different methodologies for finding the clusters, and different understandings on how to measure the distance on the data space.

These models can be separated on a high level by classical clustering and deep clustering, as described in the next sections.

2.1 Classical clustering

Classical clustering algorithms involve all methodologies that do not use deep learning. These were the original methodologies to perform clustering and often are the final step in deep learning clustering approaches. We can sub-divide these methods into four main groups: Connectivity-based clustering, centroid-based clustering, density-based clustering, and distribution-based clustering.

Connectivity-based clustering Connectivity-based clustering [Murtagh and Contreras, 2012], also called hierarchical clustering, is set on the idea that the closer two data points are in a manifold, the more related they are. These algorithms define a distance function between objects, a maximum distance by which a connection between points constitutes a cluster, and hierarchically clusters them, using a dendrogram (See the example in figure 2.1).

By changing the maximum length by which the algorithm specifies a cluster, we can change the resolution of the clustering from a more global view to a more localized view (hence the denomination of hierarchical). Hierarchical clustering algorithms mainly vary in how the distance function is defined and can be agglomerative (where it starts with single elements and aggregates them into clusters) or divisive (dividing a complete dataset into partitions). There are several implementations of these kinds of algorithms made by most commercial code distributors such as MATLAB, SAS and Mathematica. Still, the underlying principle stays the same: by selecting a distance, ϵ and a point p_i, all points in the neighbourhood of radius ϵ of point p_i belong to the same cluster.

The advantages of these types of algorithms are threefold. Due to its inherent structure, we do not need to specify the exact number of clusters, it is easy to implement in a real-world setting, and the produced dendrogram allows the user to gain further insights on the data being analyzed. However, these algorithms also have several drawbacks: they are susceptible to outliers (i.e. a data points far away from the manifold can skew the clustering results) and do not scale well to datasets with many elements, due to computational constraints. Since we use a dendrogram, we also cannot recover from a lousy clustering made at a previous hierarchical level. Due to these drawbacks, they are often considered to

be obsolete, despite also being the theoretical foundation for cluster analysis. Nonetheless, they inspired several methodologies, such as density-based approaches.

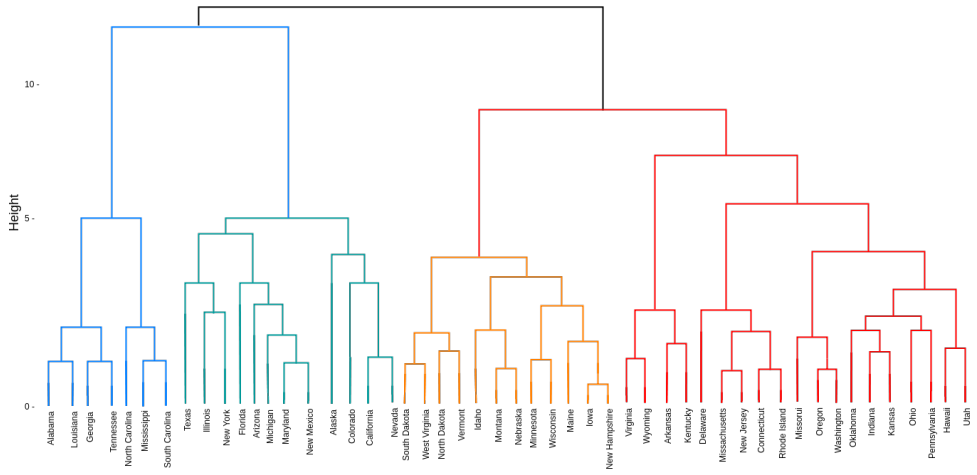


Figure 2.1: Dendogram for clustering US states based on distance
https://online.visual-paradigm.com

Centroid-based clustering In centroid-based clustering, we optimize the cluster centres instead of the distance between individual data points (As seen in Figure 2.2). Contrasting with the hierarchical clustering approach, in this case, we have to specify the number of clusters that we want to fit, which is considered a significant drawback of these types of algorithms. Centroid based algorithms attempt to minimize the intra-cluster distance while maximizing the inter-cluster reach. The most famous example of this clustering type is the K-Means [Lloyd, 1982], which attempts to optimize a set of cluster centres with circular unoptimized borders. There are several other methods and variants of centroid-based methods such as Min-Max K-Means [Tzortzis and Likas, 2014], Accelerated K-Means [Elkan, 2003], and Coresets [Tukan et al., 2020]. The main advantages of this clustering type are that it is less sensitive to outliers and more accessible to geometrical interpretation than the hierarchical approach. However, since we are not optimizing the cluster borders, this type of clustering does not work well on clusters with irregular shapes (since the shape of the groups made by the algorithms is mostly circular).

Distribution-based clustering In distribution-based clustering, we assume that we can express the whole dataset as a set of probability distributions, where points on the same cluster are likely to have been generated from the same probability distribution. The main advantage of this method is that it has a robust theoretical background and closely emulates how artificial data is generated: trough sampling of random points from a distribution. However, since these methods are prone to overfitting, we need some additional constraints on the model's complexity, such as the number of probability distributions the algorithm needs to model. One of the most straightforward models to come out of this category is

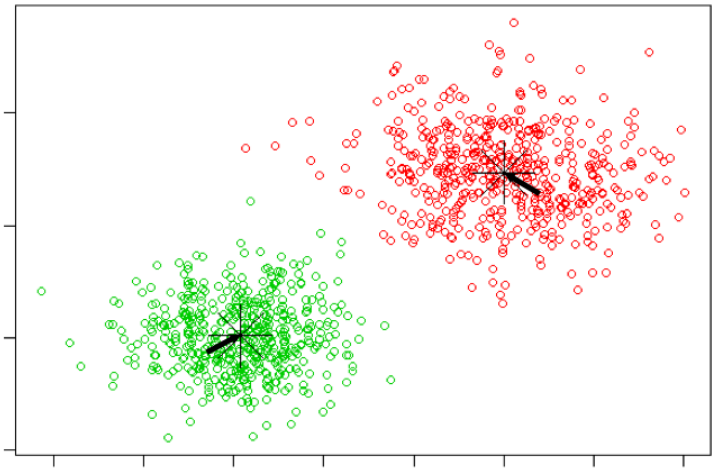


Figure 2.2: Example of centroid optimization based clustering
https://blogs.oracle.com/bigdata/k-means-clustering-machine-learning

the Gaussiam Mixture Model (GMM) [Murphy, 2012].

In GMM, we attempt to model the data using a fixed number of Gaussian distributions using the expectation-maximization algorithm. It iteratively optimizes cluster centres (means) and cluster shapes (covariances) to fit the data better. In this particular instance, we have a significant advantage over the K-Means since we do not impose circular cluster shapes (due to the covariance matrix). However, we still have the strong assumption of a fixed number of Gaussians, since in most real-world datasets there is no fixed mathematical probability distribution. This algorithm generated multiple variants such as the Gaussiam Mixture Model with Minimum Message Length (GMM-MML) [Figueiredo and Jain, 2002], that eliminates the need to select the appropriate number of Gaussians and Multivariate GMM [Murphy, 2012] that extends the ideas in GMM to a high dimensional space.

Density-Based Clustering The final type of clustering methodology analyzed in this work is Density-based clustering [Kriegel et al., 2011]. Here clusters are defined by regions with higher data point density than the rest of the space, and areas with lower point density represent noise/outliers. The main advantages of this type of clustering are its ability to identify outliers and cluster groups of irregular size data, while not making strong assumptions about the structure of the data. To this effect, algorithms of this type use what is called ϵ neighbours of data points and the definition of data density. Given a point p, it's ϵ neighbourhood is the set of points inside a circle/sphere/hypersphere of a radius of length ϵ from point p. The point density is calculated by dividing the number of points on a specific ϵ neighbourhood by the area/volume of this neighbourhood. Density-based methods assume that points that are close together (same neighbourhood) and with similar local density belong to the same cluster. This assumption can often lead to mislabelled data points, but it presents a fully unsupervised clustering methodology, unlike

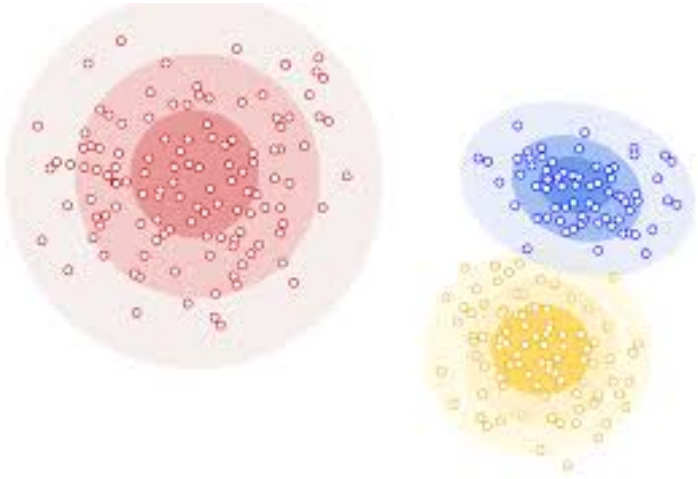


Figure 2.3: Example of distribution optimization based clustering
https://developers.google.com/machine-learning/clustering/clustering-algorithms

most of the other methods. The most widespread density clustering methodologies are Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and it's Hierarchical counterpart HDBSCAN.

2.1.1 Automatic selection of the number of clusters

Deeply allied with classical clustering approaches is the topic of automatic clustering selection. Since most algorithms have challenges regarding overfitting, hardcoding the number of clusters is a necessary evil, leading to algorithms that cannot be genuinely categorized as unsupervised. It is an ongoing research topic among data scientists and concerns itself with giving the algorithm power over the number of clusters. The objective is to make the model find a suitable cluster distribution for the data that is interpretable by humans and machine alike. By applying automatic cluster selection, we can have novel representations of the data and cluster it in a genuinely unsupervised manner. There are several ways to choose the best amount of groups for a clustering algorithm and can be separated into three distinct classes: data-analysis methods, algorithms, and dimensionality reduction methods.

Data analysis methods Data analysis methods often look at several unsupervised metrics such as the silhouette score [Rousseeuw, 1987], the intra-cluster and inter-cluster sum of squares [Aloise et al., 2009], and gap statistics, and the best values of these metrics are used to select the appropriate number of clusters. A human expert often makes this selection to decide what is the best number of clusters to input to the algorithm when it is not known.

Sum of squares is a way of selecting the optimal number of clusters in which we maximize the between cluster sum of squares while minimizing the inter-cluster sum of squares for different values of the number of clusters.

Gap statistics compare the intercluster variation for a different number of clusters with their expected values, taking only into consideration the input data.

When selecting the appropriate number of clusters, we often employ the elbow method, which states that we must choose the number of groups at the 'elbow' of the metric curve. This cutoff is done because the elbow is the point where we get an optimal tradeoff between getting better results and increasing the complexity of the model. The main problem with data analysis methods is that we need to analyze the data; i.e., it is not an automatic process. It can be time-consuming (since we are doing a grid-search-like process to find the best number of clusters), and there is no guarantee that we achieve the optimum amount of clusters.

Algorithm based methods Algorithm-based methods are the most versatile since they can be deployed automatically to find the correct number of clusters. In this section, we have many options, such as PCA and its nonlinear alternatives, several clustering algorithms such has HDBSCAN and DBSCAN, and variations to popular clustering algorithms such as GMM-MML [Figueiredo and Jain, 2002].

- **PCA** [Pearson, 1901] is a technique based on matrix factorization. It uses an orthogonal transformation to convert a set of correlated vectors into linearly independent components called principal components.

  Let X^T be the data matrix with zero mean where the rows represent the samples, and the columns represent the features. We can decompose this matrix into a set of its eigenvalues and eigenvectors like so:

  $$X = W\Sigma V^T \tag{2.1}$$

  where W is the matrix of eigenvectors of the covariance matrix XX^T, Σ is a positive definite diagonal matrix where the elements are the eigenvalues of the matrix and V is the matrix of eigenvectors of X^TX. So, if we want a PCA that does not perform dimensionality reduction, we can use:

  $$Y^T = X^TW = V\Sigma^TW^TW = V\Sigma^T \tag{2.2}$$

  where the first column of Y^T is the scores of each component related to the first principal component, the second column has the second principal component scores, and so on. Suppose we want to use PCA to perform dimensionality reduction. In that case, we will use the number of principal components where we keep a certain percentage of the variance (usually greater than 70%). This algorithm can be applied to selecting the optimal number of clusters [Divya and Devi, 2018] [Kaya et al., 2017] by using the number of groups to be equal to the number of principal components. The main problem with PCA is that it does not take into account nonlinear relationships among the

features. Some variants take into account these relationships, such as kernel PCA [Mika et al., 1999] that uses the kernel trick to model nonlinearities and deepPCA [Chan et al., 2015] that use an artificial neural network to model nonlinearities before applying PCA on the output layer.

- **HDBSCAN and DBSCAN** can naturally define the appropriate number of clusters in its architecture via density estimation. Density-based spatial clustering of applications with noise (DBSCAN) [Ester et al., 1996] is one of the most widespread density-based clustering algorithms. The algorithm groups together points that are tightly packed together (points with many neighbours), marking as outliers, points that lie alone in low-density regions (whose nearest neighbours are far away). To further explain this algorithm, we need some notation.

  - A core point is a point that has at least min_points in a distance ϵ around it.
  - A directly reachable point is a point that is in a neighbourhood of range ϵ of a core point.
  - All non-reachable points are noise/outlier points

  So, to use DBSCAN, we need to tune two hyperparameters: the epsilon and the min_points. As seen in figure 2.4, we start the algorithm by visiting an arbitrary point, analyze its neighbourhood, and categorizing the point into a core, boundary(reachable), or outlier point. By repeating this procedure for all points in the dataset, we manage to get a set of core, outlier, and border points. We can then compute the cluster assignments (a cluster consists of a group of core points with border points acting as the inter-cluster frontier). A big assumption that this algorithm has is that clusters must have close to the same point density (min_points). Thus, the authors of DBSCAN also have an algorithm supporting varying cluster densities called HDBSCAN [McInnes et al., 2017] that uses a dendrogram to select the most persistent clusters, allowing for picking varying density clusters based on cluster stability.

- **GMM-MML** [Figueiredo and Jain, 2002] is a distribution based clustering approach that uses the Minimum message length [Oliver et al., 1996] and other information theory criteria applied to a GMM. In a nutshell, we multiply each component by an importance term, which can be set to zero, virtually allowing us to eliminate specific components for the mixture. The intuition behind this process is that the algorithm will only select the optimal number of features when clustering a dataset, automatically setting the optimal number of clusters.

Dimensionality Reduction methods It has been discovered that in most machine learning applications, it is not desirable to use raw high dimensional datasets. High dimensional datasets are often challenging to perform computations on and are sparse in the high-dimensional manifold. When a measure such as a Euclidean distance is defined using many coordinates, there is little difference in the spaces

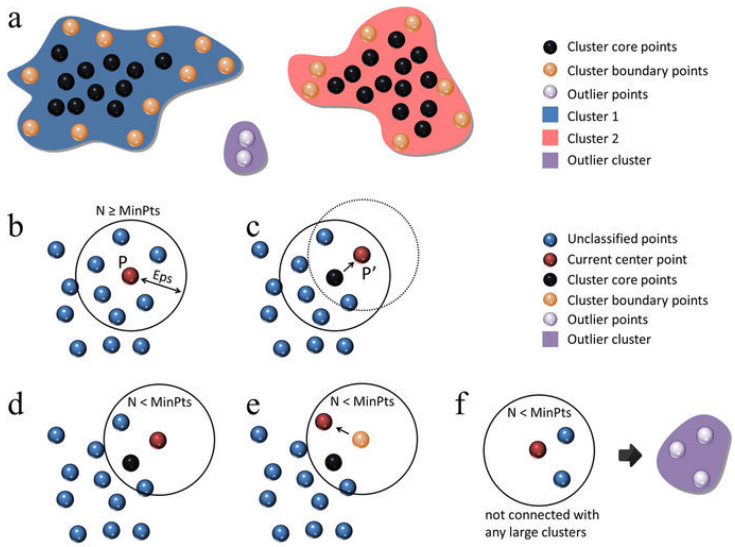


Figure 2.4: DBSCAN algorithm
https://www.researchgate.net/figure/Schematic-drawings-of-the-DBSCAN-clustering-algorithm-Panel-a-shows-the-clustering_fig$_3$08750501

between different pairs of samples. This phenomenon can be illustrated with the following formulation: let's assume that we have a hypersphere inscribed inside a hypercube in a euclidean space.

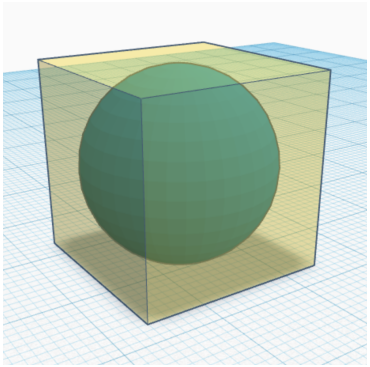


Figure 2.5: Hypersphere inscribed on a hypercube in a three dimensional space

The volume of said hypersphere can be written as V_{Sphere}

$$V_{Sphere} = \frac{2r^d \pi^{d/2}}{d\Gamma(d/2)} \tag{2.3}$$

With the volume of the hypercube being V_{cube}

$$V_{cube} = 2r^d \tag{2.4}$$

where d is the dimensionality of the data, and r is the radius of the hypersphere and 2r are the edges

of the hypercube. As we increase the dimensionality of the data the volume of the hypersphere becomes irrelevant when compared to the volume of the hypercube, and since the distance between the centre and the edges of the cube is $r\sqrt{d}$ it increases in an unbounded manner for a fixed r we can say that when the dimensionality of the data increases the data is skewed to the corners of the hypercube and is infinitely far away from each other, making the euclidean distance practically useless in high dimensional spaces. Research [Aggarwal et al., 2001] has shown that the use of L_1 or even L_k with fractional k improves the usability and interpretability of results in high dimensions, alleviating the effect of the curse of dimensionality. However, at this time, the best action, when presented with high dimensional datasets, is still to select the features that best describe the data or, as an alternative, project it into a lower-dimensional space. The process of choosing the most prominent features is often called feature selection (or dimensionality reduction), and the process of projecting the elements into a lower-dimensional space is called feature projection. Since feature selection automatically leads to loss of information (since features are dropped based on usability), it is often common to first perform feature projection of the data and then apply feature selection based on some predefined score. Artificial neural networks are experts in this: By using a setup that involves compressing the data to a latent space and clustering on the latent area, rather than grouping on the original data space, we have fewer parameters to group and make the network parameters data dependant, capturing better dependencies when compared to classical approaches. In fact, by analyzing the most prominent deep clustering models, they use a deep learning model to perform feature selection, and they use a classical clustering algorithm to perform the act of in fact grouping the data.

Dimensionality reduction techniques solve this: by leveraging feature selection and feature projection, we can observe the data and via visual analysis, find the appropriate number of cluster. To this effect, manifold learning algorithms such as T-Student Stochastic Neighbor Embedding (T-SNE) or Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) can be used.

- **T-SNE** T-distributed Stochastic Neighbour Embedding [Maaten and Hinton, 2008] is a manifold learning technique to visualize high dimensional data in a two or three-dimensional space. It models each high-dimensional feature by a two or three-dimensional point in such a way that nearby points model similar objects, and dissimilar objects are modelled by distant point locations.

  It works in two main phases. In the first phase, we construct a pairwise probability distribution over high dimensional objects such that similar items have a higher probability associated with them, and different objects have lower probabilities. In the second phase, the algorithm defines an equivalent probability distribution in the low dimensional manifold. It then uses the Kullback–Leibler (KL) Divergence to minimize the dissimilarity between the two distributions concerning the points' location.

  When compared to PCA, T-SNE models nonlinearities and preserves local and global structure

better. However, it scales quadratically with the data, so it is only limited to raw datasets with a reduced number of features. It is also non-deterministic, which means that we can have multiple plots with the same hyperparameters for different runs of the algorithm, which may skew results in an undesirable manner. Because of these disadvantages, it is usually used only for visualization purposes.

- **UMAP** Uniform Manifold Approximation and Projection [McInnes et al., 2018] is also a manifold learning technique that makes three simple assumptions about the data. Firstly, we assume that the data is uniformly distributed on the Riemann manifold. Secondly that the Riemann metric is constant, or it can be approximated as a constant and thirdly that the manifold is locally connected. If all of these assumptions can be made about the data, we can use UMAP. This algorithm is based on finding the best projection of the data where its fuzzy topological set best matches the fuzzy topological set of the data.

  UMAP uses a k-neighbour based graph algorithm to compute the nearest neighbours of points. UMAP first constructs a weighted k-neighbour graph at a high level, and from this graph, a low dimensional manifold is calculated. This low dimensional layout is optimized to have as close a fuzzy topological representation to the original as possible based on cross-entropy between the two manifolds. It has several essential hyperparameters that influence performance. The first is the number of neighbours to consider as local. By increasing this parameter, we provide a tradeoff between how much local structure is preserved versus how much global structure is captured. As we are primarily concerned with integrating the local structure into our embedding, we will typically choose lower values for the number of neighbours. However, a low number of neighbours is highly dependant on the number of points on the dataset. The second parameter is the number of components that the algorithm should use, which is usually set to be the number of clusters we are seeking to find. UMAP also requires the minimum allowed separation between points in the embedding space. Lower values of this minimum distance will capture the real manifold structure more accurately but may lead to dense clouds that make visualization difficult.

  The UMAP approach preserves the global structure of the data better, and it is faster compared to T-SNE. It also has better theoretical fundaments to be a dimensionality reduction technique, whereas T-SNE is mainly used only for visualization purposes **??**. However, it lacks maturity, so best practices and parameters have not yet been identified.

2.2 Deep clustering

In recent years, advances in deep learning architectures over a multitude of fields, the availability of vast amounts of data, and the increase of processing power of computers have galvanized the use of

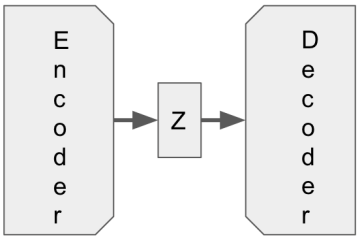


Figure 2.6: Autoencoder general architecture

artificial neural networks for computational tasks. By leveraging a neural network, we can perform feature selection and clustering in an end to end manner that surpasses most classical clustering approaches. For most deep clustering algorithms, the training procedure goes as follows: we train a deep architecture to learn feature representations of the data at a lower dimension than the raw data has and then use a shallow classical clustering algorithm to then cluster the data. It then makes sense first to analyze the basic building blocks of these methodologies, i.e. the deep algorithms that allow us to perform representation learning on the raw data.

2.2.1 Basic building blocks of deep dimensionality reduction

When performing deep clustering, some standard architectures are used to perform dimensionality reduction to the original dataset. In this subsection, we will briefly explain each of them, as they are essential stepping stones to the full-fledged algorithms.

Autoencoders Autoencoder (AE)s [Schmidhuber, 2015] are a type of Neural Network used for efficient data compression and dimensionality reduction in an unsupervised manner. It is composed of 3 main blocks: an encoder, a latent dimension, and a decoder. The encoder aims to learn a low dimensional representation of the input data, by discarding input noise and attempting to learn relevant features and patterns of the data. On the opposite side, we have a decoder that does the opposite: from the low dimensional code, it attempts to reconstruct the data as close as possible to the input data. The latent dimension is the reduced code that we get by encoding the data.

It is trained to copy it's input to its output, passing the information through a bottleneck, and the objective function is a distance function (such as Mean Squared Error (MSE)) between the input and output of the network.

$$L_{rec} = \frac{1}{N}||x_i - \tilde{x_i}||_2 \tag{2.5}$$

Sparse autoencoders It had been shown [Zeng et al., 2018, Meng et al., 2017] that autoencoders that enforce a sparse latent code perform better on classification and clustering tasks. This autoencoder

has several hidden layers z that is bigger than the input and output dimensions instead of less but fewer neurons in the latent layer are allowed to be active for each activation. This constraint is emulated by adding a L_1 sparsity penalty on the latent code of the autoencoder.

$$L = L_{rec} + \lambda \sum_i |h_i| \tag{2.6}$$

where h_i are the terms of the code layer

Denoising autoencoders Most autoencoders constraint the latent space by either adding sparsity constraint or physically, by having a lower number of neurons that the input and output dimensions. Denoising autoencoders [Lu et al., 2013, Gondara, 2016] try to achieve an accurate reconstruction by changing the reconstruction criterion.

Denoising Autoencoder (DAE)s take an input partially corrupted and attempt to recover the uncorrupted data as the training procedure. Doing this can extract useful and invariant information about the data that can be useful for coding. To corrupt the input, several approaches can be taken. We can add noise such as isotropic Gaussian noise, salt, and pepper noise or even masking noise to the input before processing.

Variational autoencoders Unlike classical autoencoder models, Variational Autoencoder (VAE)s [Kingma and Welling, 2013] are generative models, i.e., much like Generative Adversarial Network (GAN) [Goodfellow et al., 2014] we can sample the latent space and generate new samples. Despite the VAE having autoencoder in its name, the two should not be confused. The autoencoder maps the input data into a latent variable space z, often compressing the data, and then proceeds to decode the latent variable back into the input data. In contrast, the variational autoencoder tries to predict the probability distribution of the input variable, and from this distribution, get the input data back.

So we have a dataset that has an associated probability density function $P(X)$, a latent vector z with an associated probability density function $P(z)$ and a set of deterministic functions $f(z;\theta)$ that map the latent variable back to the dataset X, where θ is the optimization parameter. More precisely, we are attempting to maximize the probability of each X_i in the dataset via the following expression:

$$P(X) = \int_z P(X|z;\theta)P(z)dz \tag{2.7}$$

where $f(z;\theta) = P(X|z;\theta)$ due to the maximum likelihood principle. In VAEs, the choice of output distribution is often made Gaussian with mean $f(z;\theta)$ and covariance matrix equal to the identity multiplied by a scalar σ (hyperparameter). The choice of this function is because, with this distribution, we can easily perform gradient descent and train the model to make adjustments to θ to make the model approxi-

mate $P(X)$. Despite this theoretical background being solely focused on standard Gaussians, several authors have also already shown how to optimize this type of networks to several other probability distributions [Jin et al., 2019], [Loaiza-Ganem and Cunningham, 2019].

VAEs must deal with two problems to solve equation 2.7. It first needs to know how to define the latent variable z and how to deal with the integral over z. For the first problem, we assume that we can sample z from a simple distribution (for example, $N(0, I)$ where I is the identity matrix). We can assume this because we use a neural network: we can use the first layers to map these simple variables to the proper suited latent variable z. For the second problem, we take a shortcut: in practice, most values of $P(X|z)$ are zero, or pretty close to zero, so they do not contribute almost anything for the estimate of P(X). The objective of the VAE should be to sample values of z that were likely to produce X and calculate P(X) from just those. This sampling has a consequence: we need a new function $Q(z|X)$ that from an X it gives us z's that are probable to have generated the X, giving us the following loss function:

$$log(P(X|z)) - D[Q(z|X)||P(z)] \tag{2.8}$$

The first term measures the decoding part of the VAE, i.e., measures the probability of the X that we have at the end being generated from a given z. The second part- the KL divergence term- measures the encoding part of the VAE: measuring the similarities between the encoding function $Q(z|X)$ and the probability of a specific latent variable z.

So far, it has been explained the overall functioning of a VAE. Still, there is one problem: the sampling operation is nondifferentiable, so we cannot train the end-to-end model with the current architecture. Since we cannot sample the latent variable z directly, so we have to use the reparameterization trick. Instead of sampling z, we sample a vector e from a simple distribution $N(0, I)$ and multiply the Covariance matrix and sum the mean of the Z distribution after the sampling equation. This method is in figure 11, and it allows us to train the method using gradient descent.

Beta-VAEs β-VAE [Higgins et al., 2016] is a variant of the regular VAE architecture in which we multiply the KL-Divergence term by a constant $\beta > 1$ to build more disentangled representations. It also presents a normalized beta that depends not only on the input dimension but also on the latent space of the VAE.

$$\beta_{norm} = \frac{z_{dim}}{input_{dim}} \tag{2.9}$$

In [Higgins et al., 2016] they prove that β_{norm} achieves close to maximum disentanglement for the datasets. Despite having an excellent theoretical background supporting them, VAEs have one considerable downside: the generated images often lack definition and are blurry. This lack of clarity can be solved by using another approach to generative autoencoders: Wasserstein autoencoders.

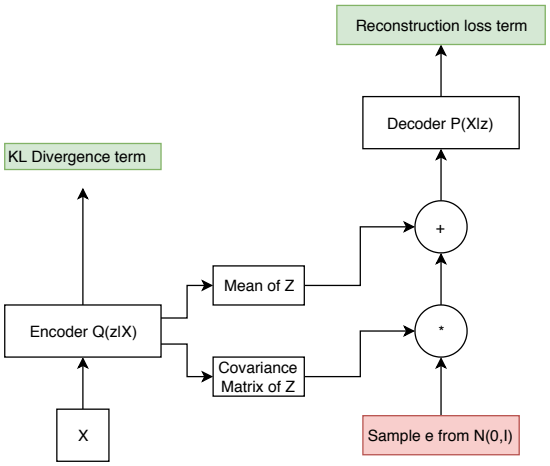


Figure 2.7: Reparameterization trick

Wasserstein autoencoders Wasserstein autoencoders [Tolstikhin et al., 2017] minimize a penalized version of the Wasserstein distance (instead of the KL divergence) between the model and target distributions:

$$L(Z) = \lambda * D_Z(q_Z||p_Z) + L_{recon} \tag{2.10}$$

In [Tolstikhin et al., 2017] they propose 2 different formulations for the penalty $D_Z(q_Z||p_Z)$: GAN-based and MMD-based. GAN-based uses adversarial training to estimate the distribution P_Z. It uses the regular autoencoder network and a discriminator network that attempts to guess from which probability distribution the data belongs. It is either from the input distribution P_Z, or the encoder distribution Q_Z.

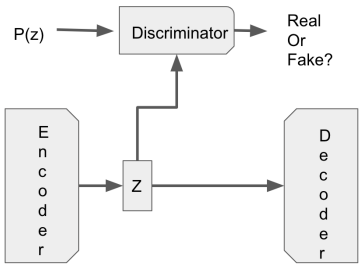


Figure 2.8: WAE-GAN general architecture

Compared to the regular GAN, we moved the adversarial part of the network to the autoencoder's latent space. If P_Z has an appropriate shape (such as a multimodal Gaussian), it is much easier to train/match distributions in this case.

MMD-based Wasserstein Autoencoder (WAE) uses a positive definite kernel K and the maximum mean discrepancy

$$MMD_K(P_Z, Q_Z) = || \int_Z K(z,)dP_Z(z) - \int_Z K(z,)dQ_Z(z)||_{H_k} \tag{2.11}$$

,where H_k is the Hilbert space of K, to perform regularization.

WAEs produce better quality samples compared to regular VAEs (i.e., less blurry) [Tolstikhin et al., 2017], since the WAE does not introduce random noise into the system, and are more robust to the choice of hyperparameters in the network.

GAN Generative Adversarial Network [Goodfellow et al., 2014] is another type of generative model, but which introduces a novel concept: adversarial training. The idea of adversarial training is based on the concept of competition: we have two networks/models that compete with each other, trying to find the flaws in each other, and consequently improve on each other. Hence a generator network is trained to generate a sample that fools the classifier (for example, if the real data is images of cats, the generator network tries to make images of cats that fool the discriminator into believing that the image is a cat). We also have a discriminator network that receives data from the real dataset and the generator at random and tries to distinguish actual data and artificial data produced by the generator network.

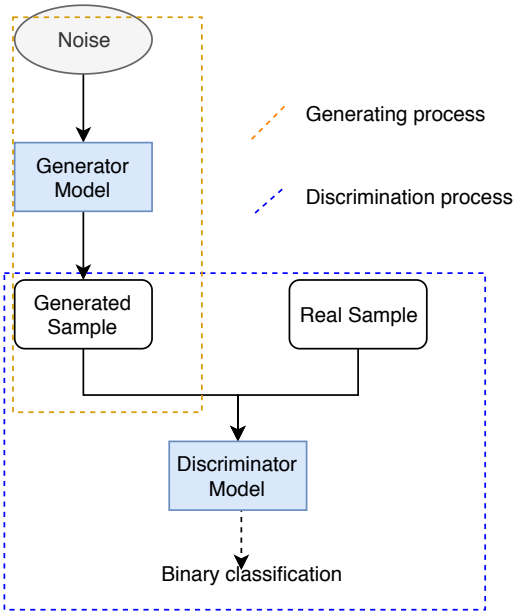


Figure 2.9: GAN general architecture

Goodfellow explains the concepts of GANs in a more formal way [Goodfellow et al., 2014]. We have a prior noise variable $p_z(z)$ that we pass through a mapping function to the dataset space x via the generator function $G(z;\theta_g)$. A neural network/multi-layered perceptron can model this function G with parameters θ_g, and it gives us the data sampled in a probability p_g that tries to be as close as possible to the distribution of the real dataset p_x. We also define another function that can be expressed as a neural network $D(x,\theta_d)$ that outputs a scalar representing the probability of a data point being real (i.e., coming from the actual distribution p_x instead of from p_g).

Our objective is to play a minimax game between G and D in the following manner: we train D to maximize the probability of assigning the correct labels, and we train G to minimize the differences between the real and artificial:

$$min_G max_D L(D,G) = E_{x\ p_{data}(x)}[logD(x)] + E_{z\ p_z(z)}[log(1 - D(G(z))] \tag{2.12}$$

where the first term reflects the maximization of D and the second term reflects the minimization of G. All of these approaches compress the data into a low-level manifold (either directly or via probabilistic methods). It would then make sense to use this low-level manifold to cluster the data instead of the high dimensional dataset. However, despite autoencoders being shown to perform well at many feature representation tasks, they do not explicitly preserve the data's distances in the representation that they learn. This lack of distance preservation may cause a problem for clustering methodologies since the gap between data points is crucial for most algorithms to group data.

Deep clustering can be divided into three main branches: autoencoder based, generative model-based, and direct cluster optimization-based.

2.2.2 Autoencoder based clustering

The main idea of autoencoder based clustering is to use an autoencoder to reduce the dimensionality of the data while using a clustering loss to group the low dimensional space. Most of these approaches use a pretraining scheme based on the reconstruction loss before applying the clustering loss.

Deep Embedding Clustering (DEC) DEC [Xie et al., 2016] is often regarded as the definitive baseline for autoencoder based clustering. It uses the standard AE reconstruction loss coupled with a cluster assignment hardening loss derived from K-Means clustering. The training has two separate phases: manifold searching step and clustering step.

The first phase consists of the compression of data using the autoencoder reconstruction loss to generate meaningful low-dimensional representations. This compression is done by training the AE to reconstruct the data from its input, passing trough a bottleneck layer that has a lower number of neurons than the input layer. This step is done to transform the data into a lower-dimensional manifold, reducing the curse of dimensionality and making clustering easier. When the manifold is appropriately defined, K-Means is used on the low dimensional space to initialize the initial cluster assignments.

The second phase - clustering step consists of first defining two distinct probability distributions: p and q. q defines a soft clustering assignment function based on Student's t-distribution as follows:

$$q_{ij} = \frac{\left(\frac{1+||z_i+\mu_j||^2}{\alpha}\right)^{\frac{-\alpha+1}{2}}}{\sum_{j'}\left(\frac{1+||z_i+\mu_{j'}||^2}{\alpha}\right)^{\frac{-\alpha+1}{2}}} \tag{2.13}$$

where q_{ij} is the probability of assigning sample i to cluster j. This kernel is used to measure the similarity between the data sample and the cluster center in the low level manifold. The auxiliary distribution p can be defined as:

$$p_{ij} = \frac{\frac{q_{ij}^2}{f_j}}{\sum_{j'}\frac{q_{ij}^2}{f_{j'}}} \tag{2.14}$$

where f_j is the soft cluster frequencies. This distribution was built with three objectives: improve cluster purity, put more emphasis on data points assigned with high confidence, and prevent large clusters from distorting the hidden space. It is updated after a set number of itterations to reflect the current state of the manifold. The clustering process consists of using a KL divergence term between the soft clustering assignments q and the auxiliary distribution p on the bottleneck layer of the autoencoder, making over the course of this step q as close as possible to p, fine-tuning the assignments and the latent representations iteratively.

This methodology was one of the first that used deep networks to perform clustering using neural networks, giving rise to what is called DEC-based methods. These methods can be defined as any method that follows the process described above and a general architecture like Figure 2.10: An autoencoder-like network where the latent dimension has a clustering loss associated with it, making it simultaneously a latent and clustering layer.

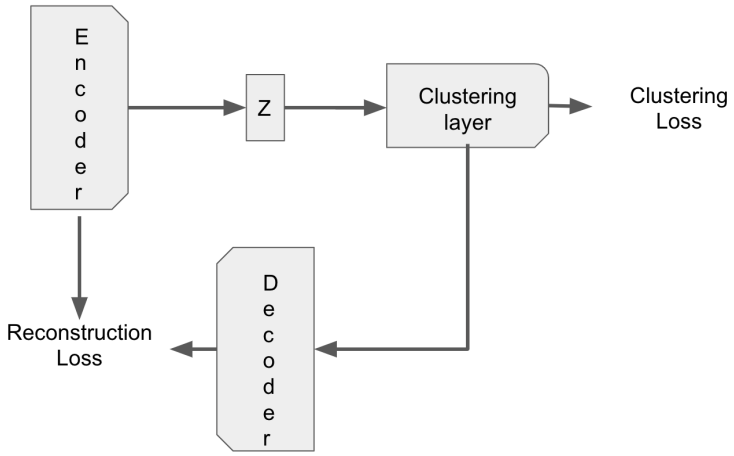


Figure 2.10: DEC-based methods general architecture

Examples of DEC-based methods are, but are not limited to Discriminatively Boosted Image Clustering (DBC) [Li et al., 2018] and Deep Convolutional Embedding Clustering (DCEC) [Guo et al., 2017b] that use a fully convolutional architecture for image data, DEC-Data Augmented (DA) and DCEC-DA [Guo

et al., 2018] that use data augmentation to improve the low level manifold representation, Deep Continuous Clustering (DCN) [Yang et al., 2017], which treats dimensionality reduction and clustering in one unified loss, claiming that simultaneously optimizing both tasks improves the clustering performance, and Deep Manifold Clustering (DMC) [Chen et al., 2017], which assumes that each cluster lies on a separate low-level manifold and that points that are close together in the manifold present similar characteristics, using a locality preserving loss and a clustering loss in the same objective function to learn and cluster structure-preserving representations.

Deep Embedded Regularized Clustering (DEPICT) Deep embedded regularized clustering [Dizaji et al., 2017] improves the baseline set by DEC using several essential changes.

- **Fully Convolutional network** DEPICT is specialized in image data clustering. In order to improve image clustering the architecture uses convolutional layers on the encoders, and strided convolutional layers in the decoder.
- **Multiple encoder network** As seen in figure 2.11, the architecture consists of two encoders and one decoder. One of the encoders is described as 'noisy', since in the output of each layer Dropout is used to randomly set some of the output neurons to zero, improving the generalization of the decoder network and the quality of the low-level manifold. The other encoder is 'clean' and shares it's weight with the decoder, improving reconstruction capability and latent space interpretability. The latent layers of the encoders have a softmax layer to produce clean and noisy cluster assignments.
- **Clustering process** DEPICT jointly optimizes the reconstruction loss and the cross-entropy loss between the clean and noisy assignments, which lead to a balanced cluster assignment loss. Both clustering layers obtain cluster noisy and clean cluster assignments from the latent spaces of the encoders using a softmax function. We assume that the 'real' assignments of the data are given by the clean encoder q and the model is predicting labels using the noisy encoder p. Let i and k represent clean samples and clusters, and i' and k' represent noisy samples and clusters. We can perform an expectation-maximization algorithm to iteratively refine the approximation to the underlying data pdf q and approximate the clean pdf p according to the following formulas:

$$q_{ik} = \frac{\frac{p_{ik}}{(\sum_{i'} p_{i'k})^{1/2}}}{\frac{\sum_{k'} p_{ik'}}{(\sum_{i'} p_{i'k'})^{1/2}}} \tag{2.15}$$

$$min - \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} q_{ik} log(p_{ik}) \tag{2.16}$$

inherently making the clustering assignments noise-independent.

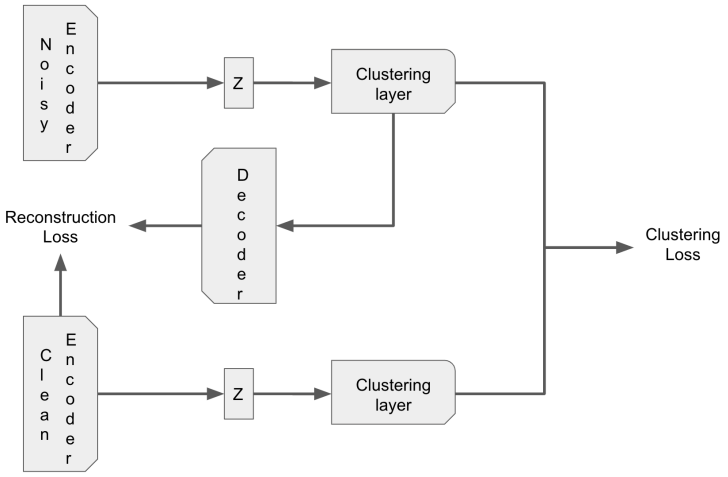


Figure 2.11: DEPICT general architecture

N2D N2D [McConville et al., 2019] is one of the best algorithms in terms of performance and simplicity. It uses an autoencoder to solely learn low-level feature representations of the data, with the clustering itself being left to a shallow algorithm such as K-Means. The secret behind the success of this algorithm is the manifold learning method used - UMAP. It has been shown [McInnes et al., 2018] that this technique not only better preserves the local structure of the data when compared to T-SNE but, by selecting the adequate hyperparameters, can also preserve the global structure of the data. To this effect, we can project the data to a space that has the same dimensionality of the input, but with parameters that give less importance to the local structure and more importance to the global structure, making the clusters more separable and compact. The clustering process goes as follows: it uses an autoencoder to learn feature representations of the data, applies a manifold learning technique (UMAP) on the latent space of the autoencoder to determine a more clusterable manifold of the data, improving the accuracy and the quality of the discovered clusters, and finally uses a shallow clustering algorithm (K-Means or GMM) to perform the clustering itself. The advantages of this algorithm are its simplicity and performance. However, there are still some questions regarding the use of UMAP as a valid clustering preprocessing step, since it is quite new and has not been properly studied in the clustering scope.

DynAE Deep Clustering using a Dynamic Autoencoder (DynAE) [Mrabah et al., 2020] uses a dynamic loss function to balance reconstruction and clustering loss. The main idea of this approach is that we start by reconstructing every data point separately but move towards the network attempting to reconstruct the average data point for each cluster (i.e. each cluster centre).

The pretraining of this network uses Adversarially Constrained Autoencoder Interpolation (ACAI) which can be loosely compared to a GAN with some fundamental alterations. It consists on a game of competitions between two adversarial networks, namely a critic and an autoencoder where the autoen-

coder generates samples that are not equal but have a set of semantic characteristics from the inputs according to the following equation:

$$x_a = g_{\theta_d}(\alpha f_{\theta_e}(x_1) + (1-\alpha) f_{\theta_e}(x_2)) \tag{2.17}$$

where α is a constant between 0 and 1, randomly generated at each training epoch and x_a is the reconstructed data point by the decoder (g_{θ_d}), whose latent representation by the encoder (f_{θ_e}) is an interpolation between two randomly sampled data points x_1 and x_2. The critic's task is to regress the interpolation coefficient α, and the autoencoder is to fool the critic, by considering the interpolated points to be as realistic as possible. This pretraining technique allows for the interpolations to look realistic, making the latent space as smooth as possible, as shown by [Berthelot et al., 2018].

The training of the network starts with using K-Means on the latent space to initialize clustering labels. The architecture of the DynAE is similar to DEC, consisting of an autoencoder architecture with a clustering layer on the latent dimension, that outputs soft probabilities for each data point. The main difference is on the objective of the autoencoder: instead of reconstructing each data point separately, we reconstruct the image from each cluster centre.

At the beginning of each training loop, we characterize every data point in conflicted and non-conflicted based on the soft probability assignments. Points are defined as conflicted (belong to S) if the distance between their highest probabilities (h_{i1} and h_{i2}), given by the clustering layer, is higher than a certain threshold β_1 or if their highest probability is lower than a certain threshold of β_2.

$$S = x_i \in X | h_{i1} < \beta_1 \text{ or } (h_{i1} - h_{i2}) < \beta_2 \tag{2.18}$$

The unconflicted points are used to refine the cluster centroids while the conflicted wait until the centroids are better constructed. Over time both β_1 and β_2 are progressively reduced until all points become unconflicted. Let L be the loss function for this architecture:

$$L = \sum_{i=1}^{N} \begin{cases} ||x_i - \tilde{x_i}||_2^2 & \text{if } x_i \in S \\ ||g(\sigma(x_i)) - \tilde{x_i}||_2^2 & \text{otherwise} \end{cases}$$

As we can see, in the loss above, the unconflicted points optimize the network, and in the conflicting points, we get the closest unconflicted neighbour $g(\sigma(x_i))$ and use that to optimize the network. This architecture achieves excellent results in terms of performance, and it presents novel ideas, such as optimizing the cluster centroids instead of each sample separately.

2.2.3 Generative model based clustering

As the name suggests, generative model-based clustering uses VAEs or GANs in the architecture. These types of models can sample the representation space to generate new samples from the underlying data distribution. The next paragraph presents an overview of generative-based clustering techniques.

Variational Deep Embedding (VaDE) When coding a standard VAE, the function prior used to regularize the manifold is a one dimensional Gaussian with a certain mean μ and variance σ^2. If we assume that the underlying data is generated from a set of probabilities, this single-Gaussian-constraint can be too restrictive to model the data correctly. VaDE [Jiang et al., 2017] uses the probabilistic approach of the VAE, coupled with a mixture of Gaussians prior, instead of the aforementioned single Gaussian. The algorithm first assumes that the data can be generated from a mixture of N Gaussian distributions and that each independent cluster is modelled by a singular Gaussian, much like a standard GMM algorithm. The optimization involves minimizing the sample reconstruction loss, done by using a reconstruction loss such as MSE between input and output, and the KL divergence between the mixture of Gaussians and the variational posterior to learn a uniform latent space that allows for sampling and restricting the latent space to be modelled by a mixture of Gaussians. At the end of the optimization, every data sample is associated with a soft probability for each cluster.

The main advantage of the VaDE is that it is theoretically very sound, and its performance is quite good, using the advantages of deep neural networks allied with the fundamentals of GMMs. A very similar approach is the Gaussian Mixture Variational Autoencoder (GMVAE) [Dilokthanakul et al., 2017], which uses the same principles but states that the KL-Divergence term is anti clustering. Since we are forcing the data to be close to a gaussian distribution, we are not worried about learning disentangled representations. To this effect, it adds a consistency violation term that minimizes conditional entropy between the reconstructions and the input data, making the clusters more compact and separable on the low dimensional space.

Information Maximizing Generative Adversarial Network (InfoGAN) The InfoGAN [Chen et al., 2016] is not a clustering algorithm per se, but more of an extension of the traditional GAN architecture. When generating new samples, the vanilla GAN is remarkably effective but provides no control over the generated images i.e. they are entirely random. The InfoGAN solves this problem by adding control variables that are learned throughout the optimization procedure and allow control over several aspects of the generated image such as style, thickness and type for the MNIST dataset. Similar to the GAN training procedure, we have two adversarial networks in a zero-sum game in which the generator is generating data that is as real as possible. The discriminator is attempting to distinguish between real

and generated data, but the InfoGAN used control variables mixed with noise (instead of just noise) as input to the generator. It then optimizes the common objective of the GAN plus a term that maximized the mutual information between the control variables and the generated data. This process has an attractive property: it manages to disentangle the latent space of the GAN, making it more separable. Suppose we decide to use discrete one-hot vectors in this disentangled latent space. In that case, we can attribute to each data point a soft probability to belong to a cluster, effectively clustering the data points.

Close to this architecture, we also have CatGAN [Springenberg, 2016], which uses a categorical aware model and a hierarchical evolutionary learning algorithm to minimize the distances inside clusters. ClusterGAN [Mukherjee et al., 2019] [Dizaji et al., 2019] samples vectors from one hot encoded and continuous vectors from a normal distribution and an inverse network, that can project the data into the latent space using a clustering loss. Both of these methods are very time consuming to train but achieve excellent clustering performance.

2.2.4 Direct group optimization clustering

Cluster optimization algorithms only use clustering loss to optimize the latent space, skipping over the reconstruction loss.

JULE Joint Unsupervised Learning of Deep Representations and Image Clusters (JULE) [Yang et al., 2016] is set on a very fundamental idea: good representations are beneficial to image clustering, and clustering results provide supervisory signals to representation learning. It starts by computing an affinity matrix A between all data points in the dataset. Then at timestep t, the process is as follows:

1. Initially we assume that each image on the dataset belongs to one individual cluster.
2. A set of Images I is fed trough a Convolutional Neural Network (CNN) that is equal in architecture to a regular encoder network and an initial set of labels p^0 is produced.
3. Based on the affinity matrix A, the two clusters with the highest affinity among them are joined into a singular cluster of dimension 2.
4. Step 2 is repeated, this time not only with the set of images, but also the probabilities in the affinity matrix calculated in the previous step and at the end, step 3 is repeated.
5. Step 4 is repeated until we reach a number of clusters that we predefined as a stopping point

This algorithm iteratively improves the clustering labels and feature representations of the network, allowing for excellent performance in image datasets. Still, JULE needs to calculate an undirected affinity matrix, which causes it to suffer from computational and memory complexity issues.

DAC Deep adaptive image clustering (DAC) [Harchaoui et al., 2017] assumes that the relationship of pairwise images is binary, i.e. an image cannot belong to more than one cluster. With this assumption in mind, we can convert the task of clustering data into a binary pairwise classification model: when we compare two images either they belong or do not belong to the same cluster.

We use a CNN to generate feature labels a batch of images, resulting in a low dimensional space of the dataset. We then calculate the cosine similarity between the image features in the batch and select training samples based on the value for the similarity: the more different two samples are, the better they are for training the network. The output of the CNN has a size that is equal to the number of clusters and is constrained to be close to one-hot encoded, so when the network is fully trained, it will output class probabilities for each image.

2.2.5 Mixture of experts clustering

Mixture of Experts (MoE) clustering can be considered a subset of autoencoder based clustering, and in some cases, can also be perceived as a generative approach. However, due to the main focus on this paper being an architecture of this kind, it required a separate section. MoE clustering is based on a set of independent neural networks called experts, and a balancing system called a manager [**?**]. The experts' objective is to specialize in one subset of the data while the manager aims to ensure that the correct expert is chosen for each data point. The next paragraph highlights the most notable contributions in the field of mixture-of-experts clustering.

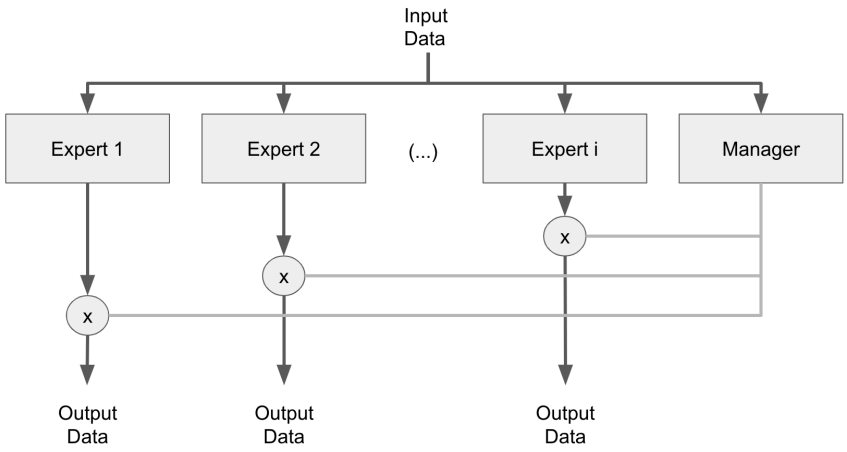


Figure 2.12: MoE general architecture

Deep Autoencoder Mixture Clustering (DAMIC) The DAMIC [Chazan et al., 2019] architecture uses the vanilla MoE architecture with a set of autoencoders as experts and a deep neural network as a manager. We are first introduced here to the concept of influence: the manager determines, based on

the input data, which is the most influential expert by giving a score between 0 and 1 to each expert. The data in each expert id multiplied by this constant and the output data is the sum of all outputs of the experts multiplied by their respective influences.

The pretraining phase of the DAMIC architecture consists of first training an autoencoder on the full dataset and getting initial cluster labels using K-Means on the latent space. This initial set of labels is used to pretrain the manager of the architecture, biasing the data that each expert will specialize.

The training loss is optimized by minimizing the reconstruction loss multiplied by the soft labels assigned by the manager to each expert (importance). If we let n be the number of data points and k the number of experts, the goal is to minimize:

$$L = -\sum_{t=1}^{n} log(\sum_{i=1}^{k} p(c_t = i) e^{-d(x_t, \tilde{x}_t)}) \tag{2.19}$$

where $p(c_t = i)$ is the probability of the manager assigning sample t to an expert i and $-d(x_t, \tilde{x}_t)$ is the reconstruction loss between the input and output of each expert.

The main advantages of this architecture are the apparent lack of a regularization term. It prevents cluster collapse between experts and does not require an organized latent space during the clustering. However, due to the lack of regularization, not only it is heavily dependant on the pretraining finding a good initialization, but it cannot recover from bad pretraining, and its performance is not state of the art. Nonetheless, it is a good starting point for subsequent architectures.

MIXAE MIXAE [Zhang et al., 2017] uses a different approach when regularizing the experts. It uses sample entropy and batch entropy to regularize the architecture. The sample-wise entropy motivates a sparse assignment of cluster labels so that each sample receives a dominant label.

$$E_s(p^{(i)}) = -\sum_{k=1}^{K} p_k^{(i)} log(p_k^{(i)}) \tag{2.20}$$

This term achieves its minimum only if the vector of probabilities is one-hot encoded, which raises a problem. At the extreme, the network can use only one expert to model the full dataset, which worsens the clustering performance. To prevent this and ensure a uniform distribution of data we calculate the batch-wise entropy between the clustering assignments:

$$E_s(\tilde{p}^{(i)}) = -\sum_{k=1}^{K} \tilde{p}_k^{(i)} log(\tilde{p}_k^{(i)}) \tag{2.21}$$

where

$$\tilde{p}^{(i)} = \frac{1}{B}\sum_{i=1}^{B} p^{(i)} \tag{2.22}$$

i.e., it is the mean of assignments over the batch. The system achieves equilibrium when the manager assignments are as one hot as possible, given a uniform assignment over the experts.

This architecture presents several advances compared to DAMIC, namely that it does not require pretraining while preventing cluster collapse. However, the performance is still far from state of the art.

MoeSim-VAE MoeSimVAE [Kopf et al., 2020] is the newest approach in regards to a mixture of expert clustering. It uses a single encoder and multiple decoder networks, instead of the regular MoE framework to model the experts and a gating network as manager.

This network receives the latent vectors of each expert, instead of the input dimension, and outputs class probabilities using as a loss function a similarity metric between the latent representations. The experts are modelled using VAEs, which enforces a Gaussian latent space, making it more interpretable. The use of a similarity matrix allows for domain-specific data to be used, and the use of VAEs makes the generation of new data easier. However, the number of experts needs to be decided beforehand, and once again, the performance leaves much to be desired.

2.3 Evaluation metrics

2.3.1 Clustering metrics

The are several metrics in which we can evaluate a clustering algorithm. These metrics allow us to compare performances between algorithms and can be divided into supervised and unsupervised metrics.

Supervised metrics Supervised metrics use the original labels of the dataset and compare them to the clustering model's output. The three main metrics in this group are the Clustering Accuracy (ACC) [Cai et al., 2010], Normalized Mutual Information (NMI) [Strehl and Ghosh, 2002] and Adjusted Rand Score (ARI) [Rand, 1971].

- **ACC** The clustering Accuracy measures the proportion of data points for which the obtained clusters can be correctly mapped to the correct classes. This mapping can be obtained using the Hungarian algorithm [Kuhn, 1955].

$$ACC(y_{true}, y_{pred}) = max_T(\frac{\sum_{i=1}^{N} 1(y_{true}(i) = T(y_{pred}(i)))}{N}) \tag{2.23}$$

where y_{true} are the ground truth labels, y_{pred} are the predicted labels, N is the total number of samples, and finally, T is the best one-to-one mapping that matches the clustering indexes to the ground truth labels.

- **NMI** Normalized Mutual Information is an information-theoretic measure based on the mutual information between y_{true} and y_{pred}, normalized using the entropy of the labels.

$$NMI(y_{true}, y_{pred}) = \frac{I(y_{true}, y_{pred})}{\frac{1}{2}(H(y_{true}) + H(y_{pred}))} \tag{2.24}$$

  where I is the mutual information, and H is the entropy functions.

- **Adjusted Rand Index** ARI is the corrected-for-chance version of the Rand index [Rand, 1971]. The Rand index measures how correct the clustering is.

  It can be computed using the following formula:

$$Rand = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.25}$$

  where TP are the true positives, TN are the true negatives, FP are the false positives, and FN are the false negatives.

The first two metrics are defined on an interval between 0 and 1 and the last one from a range between -1 and 1, where the higher the score, the better the clustering performance. The main disadvantage of these models is that we are forced to adhere to the labels imposed by the human experts that analyzed the algorithm. We can have models that can separate the data into meaningful clusters but still have bad performances on supervised metrics, since the groups formed are not the ones that were expected. Unsupervised metrics solve this hard interpretability issue.

Unsupervised Metrics unsupervised metrics do not use the ground truth labels, using the input data and the predicted labels y_{pred}. The number of available unsupervised clustering metrics is negligible when compared to supervised clustering metrics. Still, we have some standard baselines in regards to unsupervised clustering such as the Silhouette Score [Rousseeuw, 1987], the Perplexity [Brown et al., 1992], and the Calinski Harabaz Index [Caliński and Harabasz, 1974].

- **Silhouette Score** This score measures how compact and separable the various clusters are. A compact cluster means that objects inside a cluster are very close together, and separable clusters mean that the clusters are far from each other. The Silhouette score is in the interval -1 to 1, where the highest the value, the more separable and compact the clusters are. The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. It can be calculated using the following formula:

$$SS = \frac{b-a}{max(a,b)} \tag{2.26}$$

- **Perplexity** The Perplexity score measures how well a given probability model predicts an unseen sample. It is usually used in NLP tasks to evaluate language models. Let $S = s_1...s_N$ be the test set with N data points. We can then define perplexity for a binary distribution as:

$$PP(S) = 2^{-\frac{1}{N}log(P(s_1,...,s_n)))} = 2^{-H(S)} \tag{2.27}$$

  Where $H(S)$ is the entropy of the test set. With this metric,the higher the perplexity, the better the model predicts on the test data.

- **Calinski Harabaz Index** This index, or variance ratio criterion is the ration between the dispersion among clusters and the dispersion in the cluster for all present clusters. Let E be a dataset of size n_e which has been clustered into k clusters. We can build two matrices W_k and B_k representing the group dispersion and the within-cluster dispersion:

$$W_k = \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)(x - c_q)^T \tag{2.28}$$

$$B_k = \sum_{q=1}^{k} \sum_{x \in C_q} n_q (c_q - c_E)(c_q - c_E)^T \tag{2.29}$$

  where C_q is the set of points in cluster q, c_q is the centre of cluster q, c_E is the center of cluster E and n_q is the number of points in cluster q. We can then build the index s by computing the ratio between the traces of these matrices:

$$s = \frac{tr(B_k)(n_E - k)}{tr(W_k)(k-1)} \tag{2.30}$$

  This score in unbounded, where the higher it is, the more compact and separable the clusters are.

The main disadvantage of these metrics is that we do not consider the interpretability of the clusters. We can have very separable and compact groups but do not produce meaningful interpretations or even interpretations that we want to achieve when analyzed. In this dissertation, we will only use one unsupervised metric: the Silhouette score.

2.3.2 Image quality metrics

Image quality assessment plays a vital role in different image processing applications such as image acquisition, image compression, image restoration, among other subjects. Image quality assessment is

necessary because images may contain different types of noise like a blur, noise, contrast change, etc. In our case, we can use these metrics to relatively compare different reconstruction algorithms for our data. These metrics are essential when choosing which autoencoder model to use: we want a model that generates not only meaningful clusters but also generated high fidelity images.

There are two main ways in which we can access the quality of an image: subjective and objective methods.

Subjective methods This set of methodologies uses a set of humans to subjectively access the quality of the images, rating them on a scale from 0 to 10. The two most general approaches are the Double stimulus impairment scale (DSIS) [De Simone et al., 2010] and pair comparison (PC) [Zhang et al., 2017].

- **DSIS** In DSIS, we compare a set of distorted pictures to the reference and rate it from 0 to 10. This distortion can come in the form of blurring, contrasting, adding elements to the image, among others.
- **PC** Pair comparison (PC) methods compare pairs of images to determine the best and worst ones. We can use several tournament-style methods to rank the image quality, such as round-robin and swiss tournament methods.

These methodologies are useful to evaluate the quality of images with several degrees of distortion, but, there are several caveats when performing this type of evaluation. The test environment must be strictly controlled and remain invariant throughout the experiment, which can present several challenges. The test subjects must be carefully selected and must be enough in numbers to achieve statistical significance. When these conditions cannot be met, it is better to go for objective image quality assessment.

Objective methods In these methods we do not use humans to evaluate the images, but several metrics that can correlate a value to the impact that a specific distortion has to the human visual system. These methods can be such as PSNR [Horé and Ziou, 2010], SSIM [Horé and Ziou, 2010], or FID [Heusel et al., 2017] where we compute a value that allows us to infer about the quality of the image.

- **PSNR** Peak signal-to-noise ratio represents the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the reconstructed signal. Because different signals have a vast dynamic range, PSNR is expressed in terms of the logarithmic decibel scale. It can be easily expressed using the MSE formula: If we have an $m \times n$ grayscale image, the MSE can be calculated using the following formula:

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [X_{ij} - \hat{X}_{ij}]^2 \tag{2.31}$$

where X is the original image and $\hat{X}$ is the reconstructed image. Based on this we can calculate the PSNR. Let MAX_I be the maximum possible pixel value for the image and MSE the mean squared error for the image. The PSNR is evaluated as:

$$\text{PSNR} = 10 * log_{10} \left[\frac{\text{MAX}_I^2}{\text{MSE}} \right] \tag{2.32}$$

This metric can be used to study the quality of the algorithm. The larger the PSNR, the better the quality of the reconstructed image.

- **SSIM** The structural similarity index measure is a perception-based model that considers reconstruction errors as a three-component equation. It takes into account structural information, luminance masking, and contrast masking to estimate compression quality. Structural information is the idea that the closer pixels are in an image, the more interdependencies they have. These dependencies carry essential information about objects such as edges and textures in the picture. Luminance masking is a phenomenon whereby image distortions tend to be less visible in bright regions. Contrast masking is a phenomenon whereby distortions become less visible where there is a significant activity or "texture" in the image.

  The difference with other techniques such as MSE or PSNR is that these approaches estimate absolute errors. In contrast, SSIM takes into account errors that are more perceptive to the human visual system.

  SSIM is calculated between several windows of fixed size $N \times N$. For any two windows x and y, the luminance is calculated using:

$$l(x, y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \tag{2.33}$$

  where μ_x and μ_y are the average of x and y and c_1 is a constant that stabilizes the division. Contrast can be calculated using:

$$c(x, y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \tag{2.34}$$

  where σ_x^2 and σ_y^2 are the variances of x and y and c_2 is a constant that stabilizes the division. Structure can be calculated using:

$$s(x, y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \tag{2.35}$$

  where σ_{xy} is the covariance between x and y and $c_3 = c_2/2$. With these three terms we can calculate the SSIM using:

$$\text{SSIM}(x, y) = l(x, y) * c(x, y) * s(x, y) \tag{2.36}$$

This equation gives us a score between -1 and +1, where the higher, the closer the original and the reconstructed image are related.

- **FID** The Frechet Inception Distance score is a metric that calculates the distance between real and generated image datasets.

  The score summarizes how similar the two datasets are in terms of statistics on computer vision features of the raw images calculated using the inception v3 model [Szegedy et al., 2015]. Lower scores indicate the two groups of images are more similar or have more similar statistics, with 0 meaning that the two datasets are identical. The FID score is usually used to evaluate the quality of images generated by generative adversarial networks, and lower scores have been shown to correlate well with higher quality images.

  Having two image datasets x and y we can calculate the FID score like:

$$FID = ||\mu_x - \mu_y||^2 + Trace(\sigma_x + \sigma_y - 2 * \sqrt{\sigma_x * \sigma_y}) \tag{2.37}$$

  where the μ represents the means of the datasets and σ the covariance matrices.

- **Shannon entropy as a measure of image complexity** In information theory, Shannon's entropy is used to measure the information in a set of symbols, such as pixels in an image. The following equation can define it for a grayscale image with k gray levels, where p_k is the probability of a gray level k of appearing in the image:

$$H(X) = -\sum_{k} P_k log_2(p_k) \tag{2.38}$$

  The entropy measures the variations between the pixels in an image and gives a score: if we have a single-toned grayscale image (k=1), the entropy reaches its minimum but, if grayscale values are equally probable in an image, the entropy maximizes. We can use this metric to measure image complexity: the higher the entropy, the higher the possible grayscale values the picture can take, and consequently, the higher the image complexity.

2.4 Summary

In this chapter, we presented the baselines and theoretical work that inspired our model. We talked about the struggles of clustering in high dimensions and ways to alleviate this phenomenon, covered

different metrics for clustering and compressing image data and provided an overview of popular deep and classical clustering approaches, giving their respective advantages and disadvantages, and their possible use in real-world scenarios.

In Table 1, we provide an overview of the performance of deep clustering algorithms on the MNIST dataset. To collect this data, we consulted several surveys, such as [Aljalbout et al., 2018, Min et al., 2018a] [Nasraoui and N'Cir, 2019] and [Min et al., 2018b]. As seen from the table, the type of result by which the performance is inferred varies greatly from model to model. It can be an average of multiple runs, the best of a subset of runs or even unspecified. This discrepancy of result recording leaves a lot of margins and reduces the credibility of algorithms since there isn't a real base standard for evaluating clustering performance.

Still, at the time of this dissertation's writing, the best algorithms for clustering are N2D, DynAE and Adaptive Self-Paced Deep Clustering with Data Augmentation (ASPC-DA), all autoencoder-based methods. The MoE based methods are far from the state-of-the-art (by MNIST standards), with very few works using this framework to perform clustering. In the next section, we will explore the basic building blocks of our proposed architecture, giving justifications behind the implementation of each component.

Table 2.1: Comparison of state-of-the-art methods based on network architecture and MNIST results reported in the original papers. AE - Autoencoder; RBM - Restricted Boltzman Machine; AAE - Adversarial Autoencoder; GAN - Generative Adversarial Network; MLP - Multi-Layer Perceptron; CNN - Convolutional Neural Network; CAE - Convolutional Autoencoder; VAE - Variational Autoencoder; SAE - Stacked Autoencoder

Method	**Arch.**	**MNIST**		
		Acc	NMI	Type of result
CatGAN [Springenberg, 2016]	GAN	95.7%	–	–
DEC [Xie et al., 2016]	MLP	84.3%	–	best 20 trials
JULE [Yang et al., 2016]	CNN	–	0.91	avg 3 trials
InfoGAN [Chen et al., 2016]	GAN	95.0%	–	–
GMVAE [Dilokthanakul et al., 2017]	VAE	96.9%	–	best
DMC [Chen et al., 2017]	AE	–	0.86	avg
DAC [Harchaoui et al., 2017]	AAE	94.1%	–	median 10 trials
IMSAT [Hu et al., 2017]	MLP	98.4%	–	avg 12 trials
DCN [Yang et al., 2017]	AE	83.0%	0.81	–
VaDE [Jiang et al., 2017]	VAE	94.5%	–	best 10 trials
IDEC [Guo et al., 2017a]	AE	88.1%	0.87	–
DEPICT [Dizaji et al., 2017]	CAE	96.5%	0.92	avg 5 trials
DAC [Chang et al., 2017]	CNN	97.8%	0.94	–
DCEC [Guo et al., 2017b]	CAE	89.0%	0.89	–
[Figueroa and Rivera, 2017]	VAE	85.8%	0.82	best trial
MIXAE [Zhang et al., 2017]	AE	85.6%	–	–
LRAE [Chen et al., 2018]	AE	60.7%	0.62	–
CCNN [Hsu and Lin, 2018]	CNN	–	0.88	–
DCC [Shah and Koltun, 2018]	AE	96.2%	0.91	–
SpectralNet [Shaham et al., 2018]	MLP	97.1%	0.92	–
[Tzoreff et al., 2018]	AE	97.4%	–	–
DEC-DA [Guo et al., 2018]	AE	98.5%	0.96	–
DBC [Li et al., 2018]	CAE	96.4%	0.92	–
ClusterGAN [Mukherjee et al., 2019]	GAN	95.0%	0.89	best 5 trials
ASPC-DA [Guo et al., 2019]	AE	**98.8%**	**0.97**	avg 5 trials
LTVAE [Li et al., 2019]	VAE	86.3%	0.83	best 10 trials
ClusterGAN [Dizaji et al., 2019]	GAN	96.4%	0.92	avg 5 trials
[Yang et al., 2019b]	AE	97.8%	0.94	avg 10 trials
BAE [Chen and Huang, 2019]	CAE, SAE, AAE	83.7%	0.81	–
[Lin et al., 2019]	AE	74.3%	–	avg 10 trials
N2D [McConville et al., 2019]	AE	97.9%	0.94	–
DAMIC [Chazan et al., 2019]	AE	89.0%	0.87	avg 5 trials
IIC [Ji et al., 2019]	CNN	98.4%	–	–
DGG [Yang et al., 2019a]	VAE	97.6%	–	–
DynAE [Mrabah et al., 2020]	AE	98.7%	0.96	–
MoE-Sim-VAE [Kopf et al., 2020]	VAE	97.5%	0.94	–
VIB-GMM [Uğur et al., 2020]	VAE	96.1%	–	best 10 trials
DERC [Yan et al., 2020]	AE	97.5%	0.93	–
S3VDC [Cao et al., 2020]	VAE	93.6%	–	avg 5 trials

3

Architecture overview

Contents

3.1 Clustering methodology 40
3.2 Overview of the training (MoE) architecture 41
3.3 Pre-training Architecture 42
3.4 Dynamic architecture features 48
3.5 Summary 51

In this chapter we will introduce the MoE architecture, justifying the main concepts and implementation decisions along the way such as the loss function of the architecture, the need for pretraining, the training stage, the architecture of the experts and how we made the architecture fully data dependant.

3.1 Clustering methodology

We start by formally introducing our clustering methodology. Suppose a data set $X = \{x_1, x_2, \cdots, x_N\}$, composed of N samples lying on and-dimensional feature space, $x_i \in \mathcal{X}^d$. We formally assume that the set of N samples were generated through K models. As each model is statistically different, it generates samples on different regions of the feature space, each corresponding to a different scenario. A natural way to approach to clustering under this assumption relies on the use of a mixture of experts to formally divide the feature space into different regions, each represented by a different cluster c_j. Hence, the experts' objective is to model individual clusters, whereas the manager has a task to split the data among the experts. If done right, this approach encourages cooperation between experts to ensure meaningful subspaces and makes each expert model a separate cluster, allowing for the separation of groups for subsequent analysis.

Let $\mathcal{Y}$ be the space of one-hot encoded vectors representing the assignments made by the network. In the mixture of experts framework, we wish to learn a function $f : \mathcal{X}^d \to \mathcal{Y}$ through a set of K experts, each specialized in a different scenario. A manager is then assigned the job of selecting the best expert for each particular case:

$$f(x_i) = \sum_{k=1}^{K} \pi_k(x_i) f_k(x_i) \tag{3.1}$$

where $\pi_k(x_i) \in [0; 1]$ represents the manager decision regarding sample x_i, which is constrained such that $\sum_k \pi_k(x_i) = 1$, and $f_k(x_i)$ is the output of expert k. Despite these probabilities being continuous we want each probability to be as close to zero or one as possible (one-hot encoded probability vectors π). Accordingly, the proposed model includes K specialized experts, each assigned to the modelling of a distinct cluster c_k.

To construct f, we wish to learn a latent space representation that can accurately model the data within each cluster. In this section, we will provide a theoretical overview of the architecture from a top-down perspective. The full architecture can be divided into two sub-groups: the mixture of experts architecture and the pretraining architecture, fundamental to achieve a good performance consistently.

3.2 Overview of the training (MoE) architecture

The training architecture is represented in Figure 3.5. It consists onset of K experts controlled by a manager that, depending on the input data x_i, gives an importance value $\pi_k(x_i) \in [0; 1]$ to each expert on the model, as per the DAMIC architecture [Chazan et al., 2019]. However, this approach has a central problem: if left as-is we have cluster collapse: the experts compete instead of cooperating, leading to one expert specializing in the whole dataset, reducing clustering performance. To attenuate this effect we took a page from the MIXAE architecture [Zhang et al., 2017] and changed the vanilla loss function by adding batch entropy and sample entropy.

The MoE loss function is then composed of 3 elements. The first one is the weighted sum of the reconstruction losses of each expert averaged over the batch:

$$R(\theta) = \frac{1}{\eta} \sum_{i=0}^{\eta} \sum_{k=0}^{K} p_k^i * d(x_i, \hat{x}_k^i) \tag{3.2}$$

where p_k^i is the importance associated with sample i for expert k, $d(x_i, \hat{x}_k^i)$ is a loss function such as MSE or binary crossentropy coupled with a normalized KL loss term, η is the batch size, and K is the number of clusters.

The second part of the loss function is the sample wise entropy averaged over the batch, which forces the distribution of probabilities to follow a one-hot vector encoding:

$$S(\theta) = -\frac{1}{\eta} \sum_{i=0}^{\eta} \sum_{k=0}^{K} p_k^i * log(p_k^i) \tag{3.3}$$

The third one is the batch-wise entropy that ensures that we have a balanced distribution of labels, and prevents cluster collapse in the network:

$$B(\theta) = \sum_{k=0}^{K} \hat{p}_k * log(\hat{p}_k) \tag{3.4}$$

Where $\hat{p}_k$ is the mean of all predicted importances for the manager in a batch for a specific expert.

So the overall loss function will be as follows:

$$L(\theta) = R(\theta) + \alpha S(\theta) + \beta B(\theta) \tag{3.5}$$

Where α and β are hyperparameters to be tuned, despite this architecture presenting several improvements over vanilla architectures, it still has some downsides. A non-pretrained architecture presents very variable performance when compared to a pretrained one, mainly due to the random initialization of the network's weights, making it unreliable for clustering efforts. This architecture also requires a priori knowledge about the number of experts to model the data, which must be determined before training

commences. Finally, it is also required the size of the latent space of the experts. To address these effects, we made a pretraining procedure.

3.3 Pre-training Architecture

The pretraining architecture (Figure 3.1) is where most of the innovation of this architecture is made.

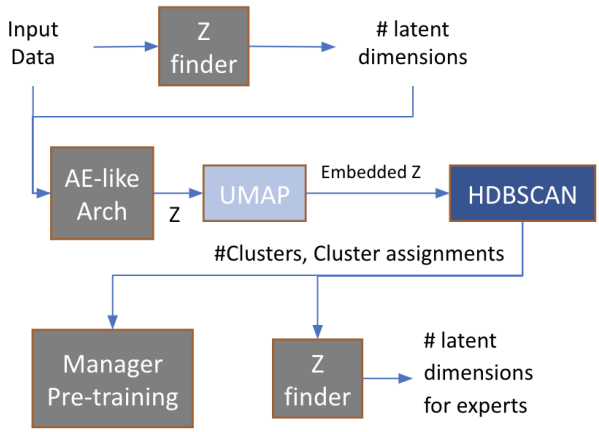


Figure 3.1: Diagram of the pretraining architecture

As previously stated, the objective of the pretraining procedure is finding two distinct values: the optimal number of clusters by which to cluster the data and the optimal size for the latent dimension for each of the experts, and train the network to ensure consistent results over different iterations of the algorithm. The pretraining task goes as follows:

1. Train an autoencoder-like architecture with a predetermined large latent dimension size (e.g. 100) on the full dataset.
2. Apply a latent dimension finder on the latent space of the ae-like architecture and find the optimal latent dimension size
3. Retrain the ae-like architecture with this new latent size on the input dataset.
4. Apply UMAP on the latent manifold of the trained ae-like architecture.
5. Apply HDBSCAN on the manifold outputted from the UMAP block, getting a vector of cluster assignments for each datapoint and the number of clusters.
6. Excluding the noisy samples, train the manager using the provided labels and original data.
7. Use the latent dimension finder on each subset of data (each cluster found in the HDBSCAN block) find the optimal latent dimension for each expert

8. Using the found number of clusters and optimal latent dimensions construct the MoE architecture.

9. Run the MoE architecture with the pretrained manager.

In the next sections, we will dive into the pretraining process in more detail.

3.3.1 Latent dimension finder

Given a set of data points X, we must first find the optimal number of variables by which we can encode the data. Currently, there are no best practices regarding layer size, generally being left to the discretion of each person (a hyperparameter essentially), and in most works, it is left, dataset independent. We argue that this parameter must be variable and data dependant, mainly because a bottleneck that is suited for a 16x16 image will not be the same as one suited for a 128x128 image.

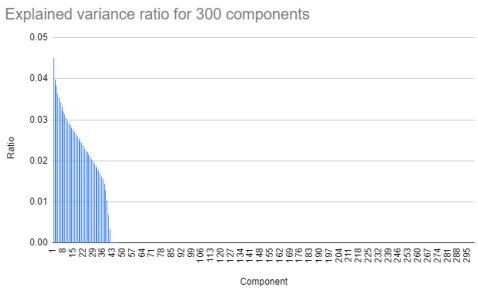
Suppose we use the PCA on the latent space from a VAE or a WAE with an L_1 constraint. In that case, we can ensure that the data follows a multivariate gaussian distribution with zero mean, unitary standard deviation and that each component is as uncorrelated to the others as possible. With this setup, we can infer three main assumptions about the data.

- Features of the data must be linearly dependant
- Features with high variance are essential in the dataset
- The principal components of the data are orthogonal

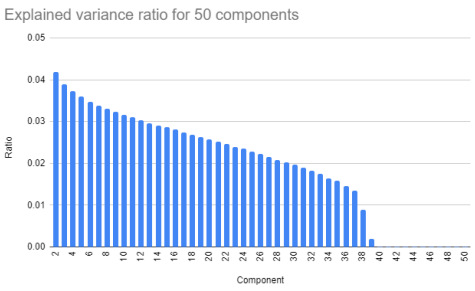
With these 3 points in mind, as seen from chapter two of this body of work, we can theoretically use PCA on the latent dimension of the autoencoder-like architecture, where the number of nodes in the autoencoder, i.e., the number of Gaussian mixtures modelling the data, is equal to the number of principal components, and analyze the explained variance ratio of each principal component to determine the most influential components. This procedure allows us to determine a data dependant bottleneck for almost any autoencoder based network.

To prove that the optimal number of components is bottleneck-independent, we did a PCA transform on the latent dimension of a VAE trained on all datasets with a latent size of 300, 50 and 40. The PCA transform had the same number of components as the latent dimensions, and the results for the MNIST dataset are in figure 3.2.

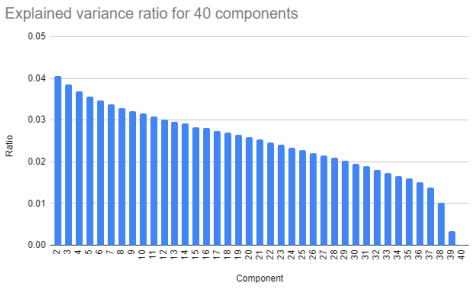
As shown from figure 3.2, the optimal number of dimensions does not change: for a large enough dimension, it uses the minimum number of components to encode necessary features (high variance), allowing us to find the optimal latent size for autoencoders. In practice, we attempt to retain around 90% of the explained variance for clustering and 95% for data reconstruction. The latent dimension finder block is represented in figure 3.3



(a) Explained variance ratio for 300 components



(b) Explained variance ratio for 50 components



(c) Explained variance ratio for 40 components

Figure 3.2: Explained variance ratio for a VAE trained on the MNIST dataset

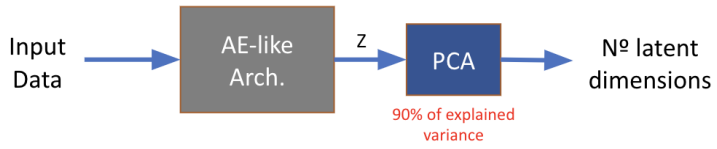


Figure 3.3: Diagram of the Dynamic latent space finder

We use this block not only to determine the optimal dimension for each expert but also for all networks involved in pretraining the MoE architecture. We first train an autoencoder-like architecture with a very high latent space dimension (100 neurons, for example) on the full dataset and obtain the low-level manifold representing the full dataset. We then use a PCA to transform the data at the latent dimension level and analyze the explained variance ratios for all the components. By selecting several components that account for 90/95 % of the explained variance, we can obtain a number for the optimal dimensions for each dataset, discarding possible irrelevant dimensions.

3.3.2 UMAP

After determining the optimal latent space, we retrain the ae-like architecture with the optimal latent dimension. We use an UMAP transform on the latent dimension of the encoder to make the low dimensional space more clusterable and overall improve clustering results, as seen in [McConville et al., 2019] and initially approached in chapter two of this document.

In N2D [McConville et al., 2019], it was shown that the use of UMAP on the latent space of a dense autoencoder, without dimensionality reduction, improves classification, achieving the state of the art results in most image datasets. The reason for this behaviour is that UMAP can cluster the data taking into account the global structure of the data, providing better inter-cluster separability and intra-cluster compactness. This separability can be visually seen by comparing the manifold projections (obtained using T-SNE to prevent skewing of results) of the MNIST dataset, the latent space of a VAE trained on the MNIST dataset, the UMAP transform of the MNIST dataset and finally the UMAP transform of the latent space of a VAE trained on the MNIST dataset (Figure 3.4)

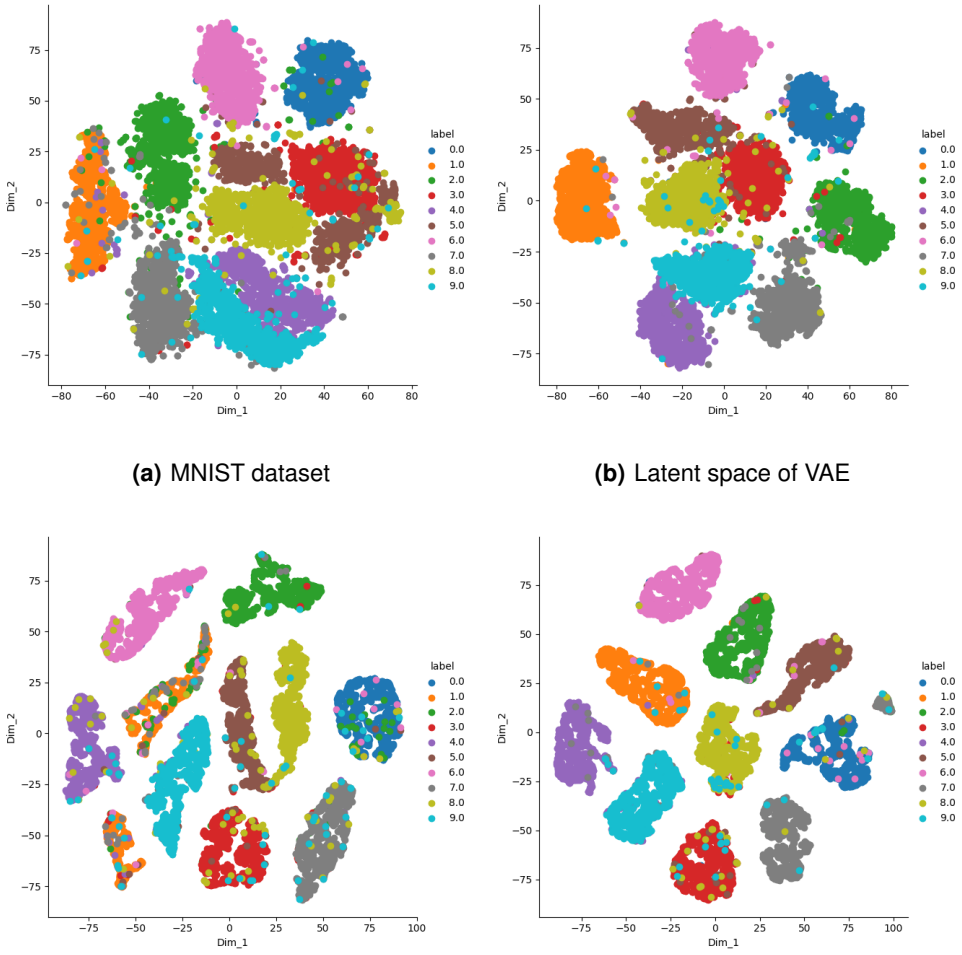
As seen from figure 3.4, the clusters become progressively more compact, and the intracluster distance becomes more significant, allowing for better distinction between clusters and better clustering performance.

Dynamic number of neighbours The 'n_neighbours' parameter in UMAP controls how the algorithm balances local versus global structure in the data. It does this by constraining the number of neighbours that UMAP will look at when attempting to learn the structure of the data. A more significant number of neighbours will give us a more global view of the manifold of the data, and a smaller number of neighbours will provide us with a more localized look but, this is dependant on the size of the dataset. If we have a small dataset where each cluster has only 20 samples, it makes sense to have a low number of neighbours (less than 20) to capture the data structure correctly. However, suppose we have a big dataset with 1000 samples per cluster, such as MNIST. In that case, 20 neighbours will give us a localized view of the data, possibly creating undesired subclusters, making sense to have a dynamic parameter. We came up with the following expression for the dynamic number of neighbours:

$$n\_neighbours = max\{int\Big(\frac{dataset\_size}{300}\Big), 100\} \tag{3.6}$$

The number of neighbours is equal to the integer division of the dataset size by 300. The maximum number of neighbours caps at 100 due to memory constraints: for large datasets and number of neighbours, the UMAP algorithm exhausts all available memory.

After UMAP, we then apply a HDBSCAN algorithm to the clusterable low-level manifold to get the number of experts K for the MoE network, as well as the clustering assignments for the data points.



(a) MNIST dataset

(b) Latent space of VAE

(c) UMAP on the original MNIST dataset

(d) UMAP on the latent space of the VAE

Figure 3.4: All projections for the MNIST dataset

3.3.3 HDBSCAN for automatic cluster and outlier detection

Most works in deep clustering, be it MoE based architectures such as DAMIC [Zhang et al., 2017] or otherwise, such DynAE [Mrabah et al., 2020] use a shallow clustering algorithm at the end of the pretraining stage to get initial clustering assignments. The two most widespread methodologies are the K-Means and the GMM. Still, both of them have several problems such as no resistance to outliers and points between clusters being mislabelled. Other issues include frontiers that do not support highly irregular cluster shapes and the fact that the number of clusters needs to be fed to the algorithm. The usage of HDBSCAN as the final clustering algorithm of the pretraining stage presents two distinct advantages when compared to a regular GMM or KMeans approach. On the one hand, it automatically detects the appropriate number of clusters that best fit the data, removing the need to know a priori how many

groups the data has, or to find the best amount of clusters using silhouette coefficients/gap statistics. On the other hand, it presents a noise vector: points which the algorithm considers outliers are put in a separate cluster that is not labelled (in the images it appears with the cluster number -1). This outlier detection mode can be useful if we do not pretrain the network in noisy points: it allows the experts to initialize their latent spaces better, and the conflicting data can be introduced during training, with less skewed experts.

3.3.4 Manager Training

The final step is training the manager network using the labels obtained from the HDBSCAN to gain insights about the overall structure of the data. In the previous step, the data points can be labelled as noise. These points are not used when training the manager, improving the distinction between clusters of the manager network. We also train K latent dimension finders on each subset of data to dynamically assign different bottlenecks to each expert.

At the end of the pretraining phase, we have three different elements: the weights of the manager, the latent space dimension of the experts and the number of experts in the training architecture. We pass these elements onto the training architecture as seen in figure 3.5, solving the problems brought up in the MoE architecture.

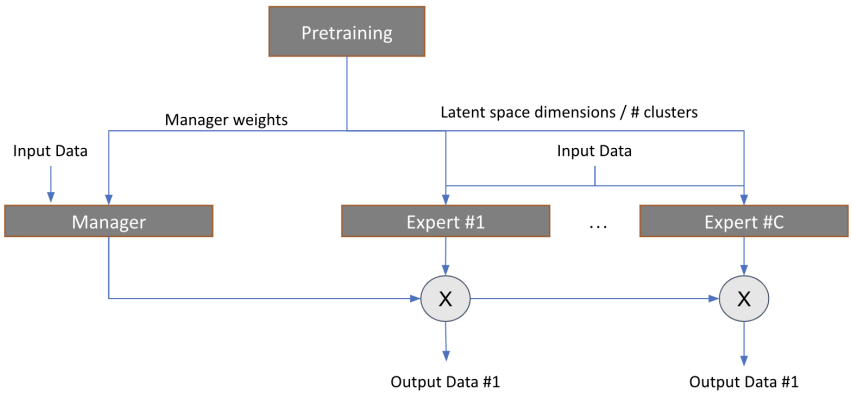


Figure 3.5: Diagram of the full architecture

By not training the experts, we hope to learn better representations for each cluster using the reconstruction loss while progressively hardening the assignments using the entropies. The experts are not trained and rather their weights randomly initialized. In the next sections, we discuss the particulars of the fully dynamic architecture and the architecture of the building blocks.

3.4 Dynamic architecture features

In this section, we discuss the overall architecture for all autoencoders featured in our algorithm. This architecture is the same, whether we are talking about AEs, VAEs, or WAEs.

3.4.1 Convolutional architecture

It has been shown [He et al., 2015] [Szegedy et al., 2014a] that by mixing different kernel sizes we can have more dynamic architectures that better model image features, so we created two distinct blocks: general and specific. General blocks reconstruct general/significant features of the image, and specific blocks reconstruct particular features. This distinction between general and specific blocks is important since different kernel sizes/number of filters can map different features of an image. Smaller kernel sizes map smaller image details whereas larger kernel sizes map overall image structure, improving latent space separability and image reconstruction. The two most widespread choices for filters are 3x3 or 5x5 due to memory and simplicity sake. We chose to use two 3x3 blocks instead of 5x5 since both have the same receptive field, but the two 3x3 blocks have less mathematical operations, leading to lower training times as found in [Szegedy et al., 2014b]. We also use Batch Normalization [Ioffe and Szegedy, 2015] before the activation function, according to [Sønderby et al., 2016]. These lead to the two primary blocks of our autoencoder-like networks, as seen in figure 3.6.

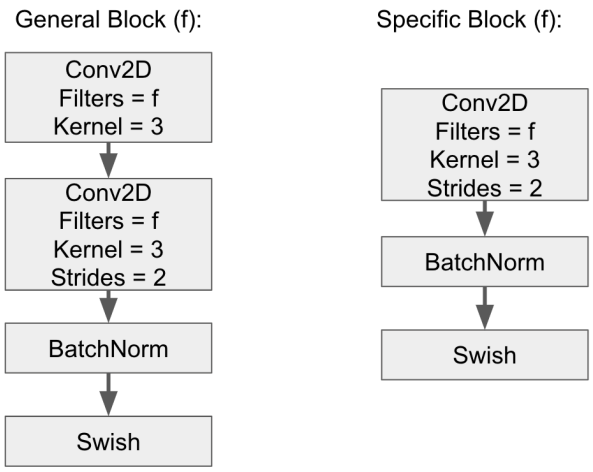


Figure 3.6: Diagram of the Convolutional blocks

3.4.2 Depth of the architecture

A network that works well for a 16x16 image dataset such as USPS may not work as well for a dataset such as COIL20 that has 128x128 sized images. To tackle this, we made the depth of the network, and

the filters depend on the image size. Assuming that an image has dimensions DxD, we can calculate the number of blocks in the encoder like:

$$N_{Blocks} = log_2(D) - 1 \tag{3.7}$$

This amount of blocks in the encoder assures us that no matter how big the image is, before the latent space of the encoder we have data that is 2x2xF in size, where F is the number of filters in the last layer of the encoder. The number of general and specific blocks is given by:

$$N_{general} = \lceil N_{Blocks}/2 \rceil \tag{3.8}$$

$$N_{specific} = N_{Blocks} - N_{general} \tag{3.9}$$

The number of filters is given by multiplying D by two every for each block of the encoder. This allows us to get an approximately equal number of general and specific blocks in our architecture. The autoencoder-like architecture can be summarized like so:

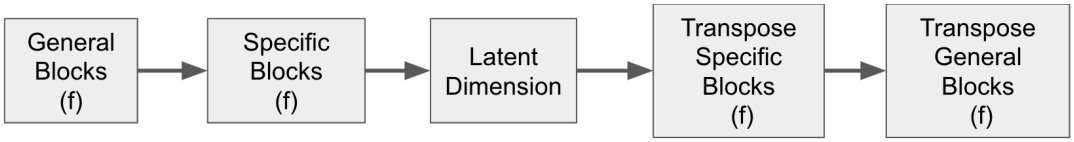


Figure 3.7: Diagram of the Autoencoder-like architecture

Where each General block is repeated $N_{general}$ times and each Specific block is repeated $N_{specific}$ times.

3.4.3 Activation functions

There are a wide variety of possible activation functions to choose from for the architecture. Recently there have been functions that achieve excellent results in image-related tasks, and in this section, we will talk about three: ReLU, Swish, and Mish.

ReLU or Rectified linear unit [Nair and Hinton, 2010] is a function where all inputs lower than zero are mapped to zero, zeroing out the neuron, whereas for inputs greater than zero ReLU is linear. It looks as follows (Figure 3.8).

This function is the most widespread activation function with several advantages such as faster convergence times, faster run times and sparser models (more zero-valued neurons), which results in concise models that often have better predictive power and less overfitting/noise. The main disadvantage of

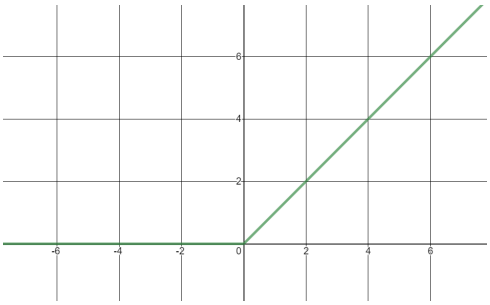


Figure 3.8: ReLU activation function

this function is what is called vanishing gradient. In such methods, each of the neural network's weights receives an update proportional to the partial derivative of the error function concerning the current weight in each iteration of training. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may ultimately stop the neural network from further training.

Swish was proposed in 2017 [Ramachandran et al., 2017] by the Google Brain team, and its formulation is as follows

$$Swish(x) = x * sigmoid(x) \tag{3.10}$$

This activation function presents several advantages when compared to the ReLU activation function. Like ReLU, Swish is bounded below (meaning as x approaches negative infinity, y comes to some constant value) but unbounded above (meaning as x approaches positive infinity, y approaches infinity), as seen in figure 3.9. However, unlike ReLU, Swish is smooth (it does not have sudden motion changes or a vertex). Additionally, in Swish there is not always a singularly and continually positive (or negative) derivative throughout the entire function.

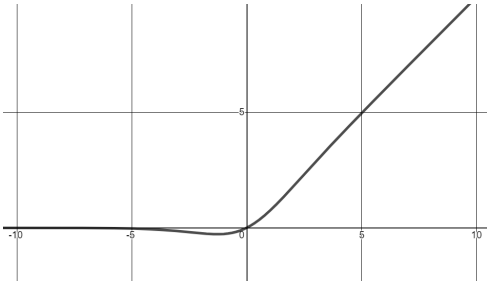


Figure 3.9: SWISH activation function

The authors highlighted that the Swish function's main advantages are the simplicity and improved accuracy as the Swish does not suffer vanishing gradient problems. It also provides useful information propagation during training and outperforms the ReLU activation function on image classification and

segmentation tasks [Ramachandran et al., 2017].

Mish [Misra, 2019] was elaborated in 2019, and can be explained as follows:

$$MISH = x * tanh(softplus(x)) \tag{3.11}$$

The advantages of Mish are as follows: it is unbounded above, like the ReLU and Swish activation functions, it is bounded below, and it has C^{∞} order of differentiability, resulting in a smoother optimization space (Figure 3.10).

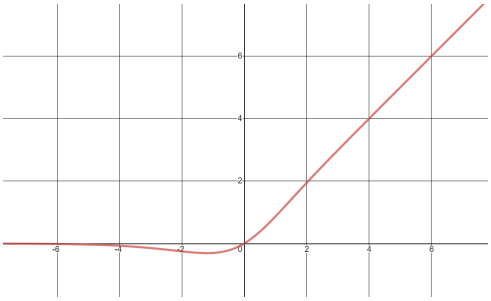


Figure 3.10: MISH activation function

The MISH activation function has been extensively compared to ReLU and SWISH and has gotten the best results in all image classification, segmentation and generation tasks with clean and noisy samples for neural networks with a large number of layers [Misra, 2019]. For this reason, we chose to use MISH as the activation function for all architectures in this thesis.

3.5 Summary

In this chapter, we presented a general overview of the architecture. We talked about the several difficulties of a vanilla MoE architecture and how, by tweaking the loss function, we can overcome them. Still, this was not sufficient, so a pretraining methodology had to be implemented. We used several well-known techniques such as UMAP, HDBSCAN and PCA to help determine several parameters of the architecture. We also talked about the architecture of the autoencoders: they must have an architecture that is fully data dependant and dynamic, depending on the input and finally, we approached the topic of selecting the best activation function for our network based on the task at hand.

4

Experiments

Contents

4.1 Datasets 53
4.2 Choice of autoencoder for the MoE 54
4.3 Loss construction for the VAEs 56
4.4 Pretraining 57
4.5 Training 68
4.6 Final accuracy results 75
4.7 Summary 77

This chapter serves as a way for us to not only give numerical strength to our approach but also as a way to justify the significant design decisions made throughout the conception of the model.

4.1 Datasets

We compare our architecture with an array of various clustering algorithms in 5 baseline datasets: MNIST [LeCun et al., 2010], FMNIST [Xiao et al., 2017], USPS [Hull, 1994], CIFAR10 [Krizhevsky et al., 2009] and COIL20 [Nene et al., 1996].

- **MNIST**: A dataset of 70000 images of handwritten digits separated into ten classes. Each sample is a 28x28 grayscale image.

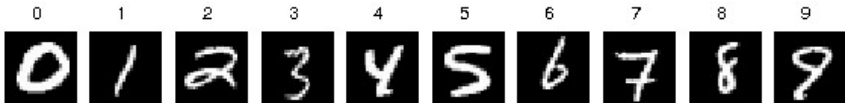


Figure 4.1: Example images of the MNIST dataset

- **FMNIST**: A dataset of 70000 images of fashion items separated into ten classes. Each sample is a 28x28 grayscale image.

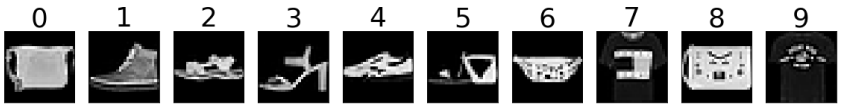


Figure 4.2: Example images of the FMNIST dataset

- **USPS**: A dataset of 9298 images of handwritten digits separated into ten classes. Each sample is a 16x16 grayscale image.

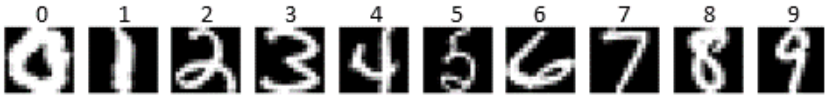


Figure 4.3: Example images of the USPS dataset

- **CIFAR10**: A dataset of 60000 images of handwritten digits separated into ten classes. Each sample is a 32x32x3 RGB image.

Figure 4.4: Example images of the CIFAR10 dataset

- **COIL20**: A dataset of 1440 images of several objects separated into ten classes. Each sample is a 128x128 grayscale image.

Figure 4.5: Example images of the COIL20 dataset

All samples were standardized, and the FMNIST and MNIST samples were padded from 28x28 to 32x32 to fit the architecture. The COIL20 dataset was downsampled from 128x128 to 64x54 using an antialiasing filter so as not to overcomplicate the architecture.

4.2 Choice of autoencoder for the MoE

To simplify the MoE architecture, we chose to use only one type of autoencoder throughout the pre-training and training stages. When selecting this basic block of the architecture, we had five options: a regular autoencoder, a variational autoencoder, a deterministic Wasserstein autoencoder, a random Wasserstein autoencoder, and a β variational autoencoder.

The chosen architecture must have the following characteristics:

- **A clusterable and well defined latent space.** It is measured by the Silhouette Score (SS), which measures the compactness and separability of clusters. It is defined in the interval between -1 and 1 where the higher the score, the more compact and separable the clusters are.
- **Provide high fidelity images.** This is measured by the image quality metrics such as the PSNR, the SSIM and the FiD which measure the quality of image reconstruction by the networks.
- **Be generative** i.e. we can sample data from each expert.

We started by training each architecture for 1000 iterations with a learning rate of $5e^{-4}$, a batch size of 100, and a large latent size of 100 neurons. The average results of 5 runs for each architecture in each dataset is present in tables 4.1 to 4.5, where the best results for each metric are highlighted in green.

Table 4.1: Unsupervised metrics for USPS

USPS	SSIM	PSNR	FID	SS
AE	0.981	29.100	0.041	0.117
VAE	0.868	19.346	0.414	0.061
Det WAE	0.854	19.245	0.559	0.040
Rand WAE	0.929	23.292	0.143	-0.031
β VAE	0.191	6.956	27.479	0.006

Table 4.2: Unsupervised metrics for MNIST

MNIST	SSIM	PSNR	FID	SS
AE	0.989	34.999	0.035	0.069
VAE	0.935	29.475	0.339	0.023
Det WAE	0.853	19.236	0.598	0.348
Rand WAE	0.964	28.820	0.137	0.012
β VAE	0.367	7.963	51.089	-0.025

Table 4.3: Unsupervised metrics for FMNIST

FMNIST	SSIM	PSNR	FID	SS
AE	0.883	28.167	1.614	0.066
VAE	0.860	21.642	5.908	0.0009
Det WAE	0.755	21.628	7.128	0.009
Rand WAE	0.820	24.632	3.861	0.014
β VAE	0.407	11.357	81.383	0.060

Table 4.4: Unsupervised metrics for COIL20

COIL20	SSIM	PSNR	FID	SS
AE	0.941	34.733	3.919	0.073
VAE	0.894	30.503	7.720	0.090
Det WAE	0.899	31.042	8.134	0.018
Rand WAE	0.781	24.683	33.027	0.030
β VAE	0.631	19.137	76.267	0.258

Table 4.5: Unsupervised metrics for CIFAR10

CIFAR10	SSIM	PSNR	FID	SS
AE	0.649	21.978	0.0523	-0.048
VAE	0.562	17.120	48.288	-0.053
Det WAE	0.497	18.089	58.245	-0.006
Rand WAE	0.608	20.668	36.127	-0.005
β VAE	0.069	2.408	426.049	-0.170

Table 4.6: Sum across all datasets

Sum	SSIM	PSNR	FID	SS
AE	4.443	149.0	5.6613	0.028
VAE	4.119	118.1	62.669	0.092
Det WAE	3.858	109.2	74,664	0.409
Rand WAE	4.102	122.1	73,295	0.020
β VAE	1.665	47.82	662.3	0.129

From the tables, we can see that the autoencoder presents the overall best results. However, since we also want a generative model in our architecture, it makes sense to select the VAE as the basic model in our architecture instead of a vanilla AE. It presents the second-best overall results and is also generative, making it ideal for our proposed task. Another good option is the WAE.

Another critical decision that is worthy of exploration is the use of data augmentation in the model. Data augmentation consists of random rotations, shifts, and crops on an image. We use this method in supervised learning as a form of regularization. With data augmentation, we have more data and variability between samples, which allows for better feature generalization cluster distinction. The main idea is that when we perform augmentation, the augmented data shares the same manifold/probability distribution of the input data. This augmentation allows the data manifold/distribution to become smoother, especially in datasets with few samples.

To test the benefits of data augmentation in the VAE architecture, we once again trained it with and without data augmentation and compared their its unsupervised metrics. We used a rotation range of 10 degrees for all datasets, a zoom range of 0.1 and width and height shift ranges of 0.1. For the FMNIST, COIL20, and CIFAR10 dataset, we also use horizontal flips. The results are present in Table 4.7, where the best values are highlighted in green.

Table 4.7: Metrics for a VAE trained with augmentation

	USPS	MNIST	FMNIST	COIL20	CIFAR10
SS	-0.061	0.023	0.014	0.0009	-0.053
SS (DA)	0.125	0.069	0.037	0.146	-0.0528

So, as a result, we will use a vanilla VAE with data augmentation for all steps of this architecture.

4.3 Loss construction for the VAEs

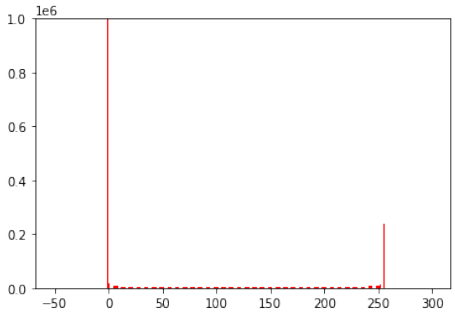
As previously said, the loss function for a regular VAE is the sum of two different terms: a KL divergence term and a reconstruction term. To determine the best reconstruction term for the data, we analyzed the average intensity distribution for each dataset and plotted one histogram for each dataset present in Figure 4.6. All histograms have been capped at 10^6 to check the full range of values.

As seen from Figure 4.6, the USPS and MNIST datasets have most values centred around zero or one, instead of being spread across all ranges of intensity. We assume that these datasets are binary and are generated by a multinomial Bernoulli distribution with unknown parameters. In contrast, the other datasets are generated by a multimodal Gaussian distribution with unknown parameters. Since we consider the USPS and MNIST datasets as binary, we use the binary cross-entropy (BCE) as a reconstruction loss and a sigmoid activation function as the output function of the VAEs. In contrast, the rest of the datasets have as a reconstruction loss the Mean squared error (MSE) and a linear loss function as an output function.

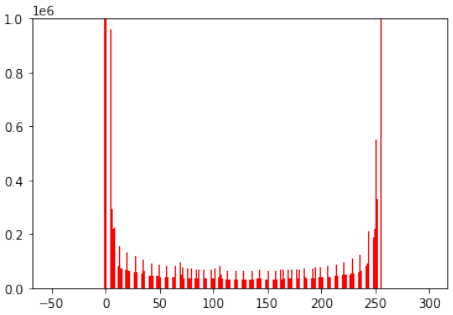
Another problem that we need to address is the KL vanishing [Zhu et al., 2020]. The variational regularization term causes some of the latent units to become inactive during training. The approximate posterior for some units are regularized towards its own prior, a phenomenon also recognized in the VAE setting. This regularization can be seen as a virtue of automatic relevance determination and a problem when many units collapse early in training before they learned a useful representation of the data. We alleviate the problem by initializing training using the reconstruction error only (corresponding to training a standard deterministic auto-encoder), and then gradually introducing the variational regularization term.

$$L(\theta) = R(\theta) + W * KL(\theta) \tag{4.1}$$

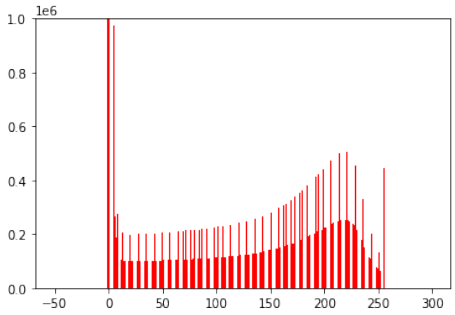
This introduction is done by adding a weight W that is multiplied by the variational regularization term and is increased from zero to one over a set number of epochs.



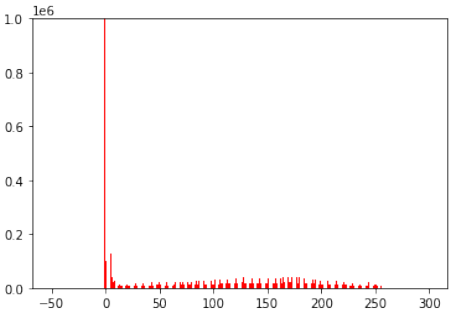
(a) Histogram for USPS dataset



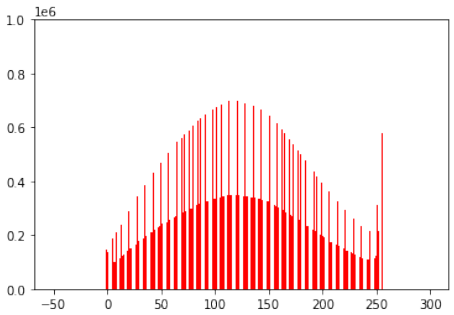
(b) Histogram for MNIST dataset



(c) Histogram for FMNIST dataset



(d) Histogram for COIL20 dataset



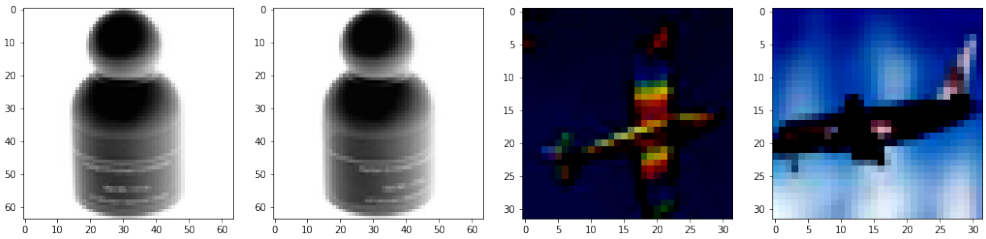
(e) Histogram for CIFAR10 dataset

Figure 4.6: Histograms for all datasets

4.4 Pretraining

The pretraining architecture is composed of 4 main parts:

- Selecting the appropriate latent space dimension for the VAE;
- Using UMAP on the means vector of the trained VAE;
- Using HDBSCAN to get the pretrain labels for the data points;
- Training the manager network with the HDBSCAN labels.



(a) Images from a cluster in COIL20 **(b)** Images from a cluster in CIFAR10

Figure 4.7: Images from COIL20 and CIFAR10

4.4.1 Data-driven latent space dimension finder

To find the optimal VAE latent dimension for each dataset, we used a PCA on the means vector of the pretrained VAE instead of on the latent space vector. This alternative is justified because, on the VAE's latent space, we introduce Gaussian noise, which may skew the PCA results, adding components that are not needed. We assume that the data is generated by zero variance Gaussians, preserving the multivariate Gaussian properties of the architecture without accounting for noise components. We also added an L_1 constraint of 10^{-4} to make sure that all components are as uncorrelated as possible. The results for the application of a PCA in the means vector of the pretrained VAE using 90% explained variance are present in Table 4.8

From Table 4.8, we can see that the intrinsic dimension of a dataset depends on several factors, such as the number of features, the variability and the complexity of the data. For example, when comparing USPS and MNIST, two datasets with handwritten digits, two different optimal latent dimensions (18 and 23) are attained due to dataset size and input dimensionality.

It is also dependant on the variability of the data, as seen by comparing the COIL20 and CIFAR10 datasets (see Figure 4.7). The COIL20 samples have 20 objects rotated 360 degrees in the same lighting and background conditions. On the other hand, CIFAR10 objects have different backgrounds, orientations, and colours, having more inter-class variability, resulting in a higher latent dimension size when compared to the COIL20 dataset (28 vs 34).

Finally, by comparing MNIST and FMNIST, we can see that the data's complexity also plays a role in the optimal latent dimension. These datasets have the same number of features (image dimension)

Table 4.8: Dimensions for each dataset

	USPS	MNIST	FMNIST	COIL20	CIFAR10
Z dimension found by PCA	18	23	29	28	34
Shannon entropy	3.171	2.652	4.845	5.343	7.262
Nº of features	256	1024	1024	4096	3072
Dataset size	9298	70000	70000	1440	60000

and dataset size. Still, if we refer to the Shannon entropy table for all datasets, we can see that MNIST has a lower entropy (complexity) than FMNIST. Most binary datasets typically have lower complexity than grayscale ones (there are no prominent textures present and edges are better defined), which can be observed by the results in Table 4.8. Binary-valued datasets (USPS and MNIST) have lower values for entropy (and consequently less complexity) than grayscale datasets (FMNIST and COIL20) and coloured datasets (CIFAR10). The complexity's effect is reflected in the optimal dimension for the MNIST vs FMNIST datasets (23 vs 29).

4.4.2 Usage of UMAP on the pretraining stage

To show the benefit of using an UMAP-transform over the latent space of a VAE architecture, we use the silhouette score measures for each dataset in 3 different cases: on the raw dataset without any transform (Original), with a VAE encoding, and with a UMAP transform with default parameters on the VAE encoded space. The UMAP transform does not reduce the dimension of the space, i.e., it has precisely the same number of components that the latent space vector. The best silhouette scores for each dataset are, once again, highlighted in green in Table 4.9.

Table 4.9: Silhouette Scores for several instances on all datasets

Dataset	Original	VAE	VAE + UMAP
USPS	0.112	0.161	0.745
MNIST	0.047	0.072	0.776
FMNIST	0.049	0.039	0.248
CIFAR10	-0.056	-0.044	-0.058
COIL20	0.164	0.200	0.652

As we can see from Table 4.9, the silhouette score marginally improves when we encode the data using the VAE, and but jumps when we UMAP-transform this encoded latent space. We did not use UMAP directly on the dataset due to memory constraints (the program uses all the available RAM and crashes). These results solidify that the use of UMAP after the autoencoder training proves beneficial to the pretraining clustering algorithm.

Usage of L_1 metrics in the UMAP algorithm In chapter 2, in the introduction of the dimensionality reduction methods, we have noted that the Euclidean distance collapses in high dimensions. To make the algorithm robust for high dimensions, we use the manhattan distance on the UMAP-projection of the data. We also $L1$ normalized the data before UMAP transforming it. In Table 4.10 we compare the two approaches.

As we can see from Table 4.10, a UMAP with manhattan distance and L_1 regularization provides a better structured latent space than a UMAP with default parameters, so we used the latter approach.

Table 4.10: Silhouette Scores for several instances on all datasets

Dataset	Default UMAP	$L1$ regularization + L_1 UMAP
USPS	0.745	0.768
MNIST	0.776	0.748
FMNIST	0.248	0.308
CIFAR10	-0.058	-0.056
COIL20	0.652	0.598

4.4.3 Usage of HDBSCAN on the pretraining stage

In this section, we will evaluate and interpret the performance of the HDBSCAN in several datasets. Since the results of this algorithm lack semantic interpretability (i.e., there is no guarantee that the algorithm separates dresses from shoes in the FMNIST dataset), this is a crucial step to justify our architecture's performance.

The results for the clustering accuracy with HDBSCAN and GMM are present in Table 4.11. Similar to UMAP, the metric used to perform clustering in the HDBSCAN is the Manhattan (L_1) metric, the minimum cluster size for the algorithm is dynamic and equal to $\frac{dataset\_size}{50}$, and the algorithm to decide the minimum spanning tree is the 'leaf' method. For the GMM, we used the ground truth number of clusters and individual covariance matrixes for each cluster.

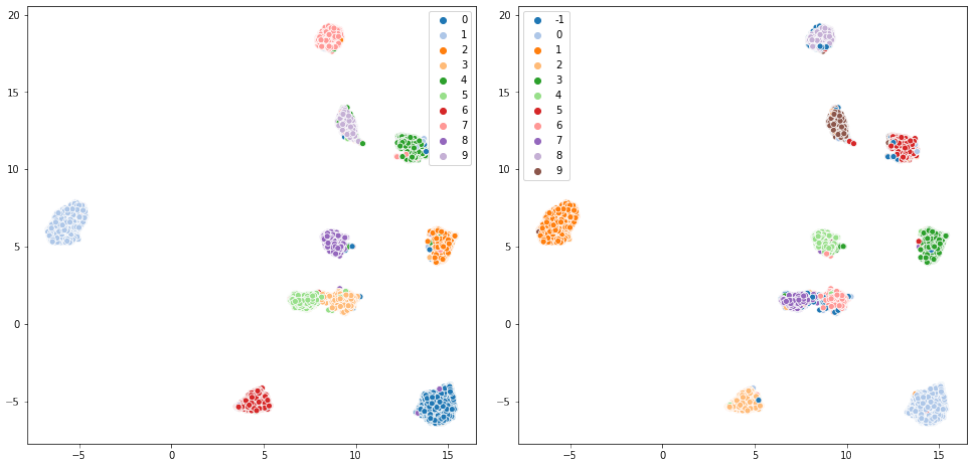
Table 4.11: Results of clustering with HDBSCAN and GMM on each of the datasets

	USPS	MNIST	FMNIST	COIL20	CIFAR10
Nº experts	10	10	11	20	3
% of dataset labeled	0.97	0.96	0.48	1	0.18
Accuracy on labeled dataset	0.967	0.978	0.673	1	0.436
Accuracy on full dataset	0.945	0.945	0.381	1	0.175
GMM accuracy for the dataset	0.962	0.956	0.583	1	0.276

From the accuracy results alone, we can see that the accuracy of labelled samples by HDBSCAN is higher than the GMM on the whole dataset. Still, there are datasets such as FMNIST and CIFAR10, where the percentage of labelled samples is low (less than 60% of the entire dataset), which may skew the manager training process.

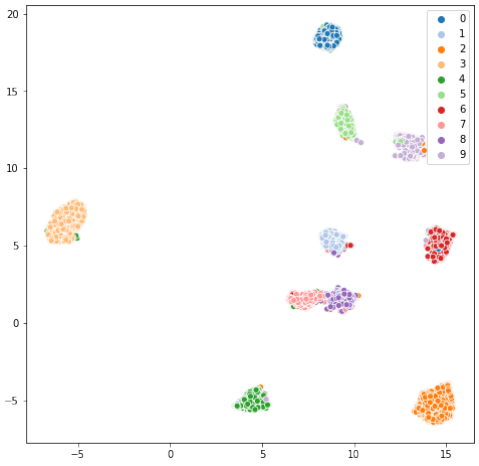
USPS The HDBSCAN algorithm found 10 clusters for this dataset. To visualize them, we changed the number of components in the UMAP algorithm to two dimensions to plot them. The rest of the pretraining architecture remains the same, and the plots for the ground-truth labels, GMM labels, and HDBSCAN labels are present in Figures 4.8(a), 4.8(c) and 4.8(b) where for the HDBSCAN the label -1 represents points that were not mapped to a cluster (belong to the noise vector).

As we can see from the figures, all digits have been correctly separated into groups, with the noisy



(a) UMAP for the USPS with groundtruth labels

(b) UMAP for the USPS with HDBSCAN



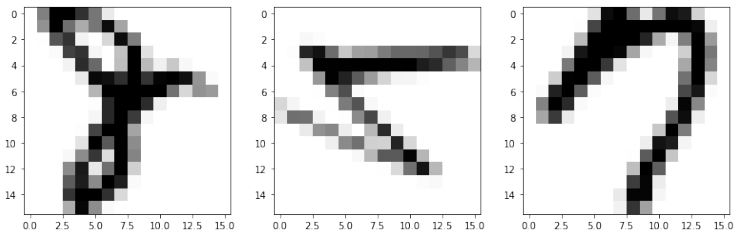
(c) UMAP for the USPS with GMM

Figure 4.8: UMAP for the USPS dataset

samples/mislabeled samples usually in the edges of clusters. In Figure 4.9, we can see that labels in the noise vector sometimes are indecipherable even for a human expert, making it also difficult for a deep learning algorithm to get it 100% right.

MNIST The two dimensional UMAP projections for the MNIST dataset with groundtruth, GMM and HDBSCAN labels are present in Figures 4.10(a), 4.10(c) and 4.10(b) where once again for the HDBSCAN the label -1 represents points that were not mapped to a cluster (belong to the noise vector).

Once again, the digits were all correctly separated, with the noisy/mislabeled (Figure 4.11) samples being in the edges of the clusters. Both USPS and MNIST datasets are relatively simple, with binary



(a) (b) (c)

Figure 4.9: Images from the noise vector in the USPS dataset

data, well-defined edges and data that is easily separated. Next, we analyze FMNIST - a dataset that is much harder when compared to the USPS and MNIST.

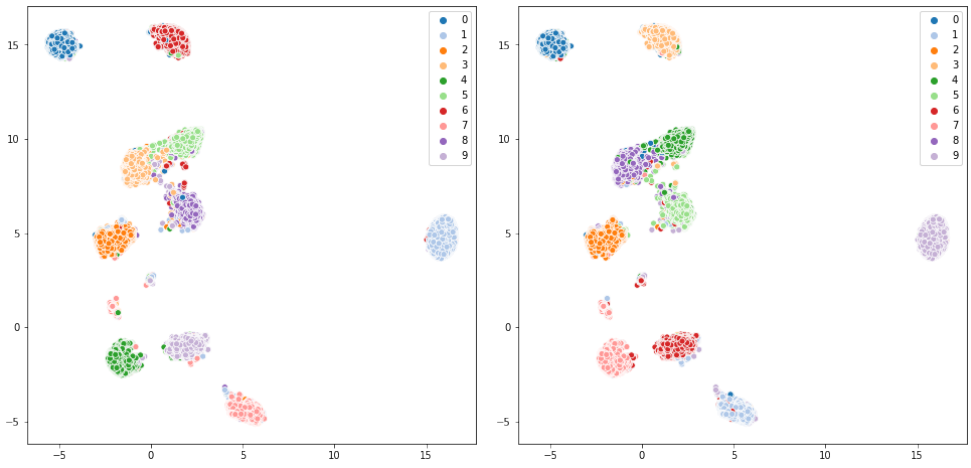
FMNIST Clustering this dataset is a more challenging task than the previous ones. As we can see from 4.12(a) we have ten ground-truth clusters, but they are of highly irregular shapes, with overlapping values and overall less separable data. At first glance, we can distinguish four main groups: Bags, Trousers, shoewear, and clothes.

The 'Trousers' cluster has a regular shape and is far from the rest of the groups on the two dimensional manifold. This distance can be explained by the fact that trousers, in general, have a distinctive shape that cannot be easily confused by one of the other clothing items. As such, HDBSCAN managed to find and isolate this cluster easily. However, since this cluster has a relatively irregular shape, the GMM algorithm also captures some of the dresses in the trousers clusters (seen in Figure 4.13).

The 'Bags' cluster has been separated into two sub-clusters: bags with a visible strap and bags without a visible strap (Figure 4.14). Both algorithms separated the 'Bags' group in the same way, making sense since the algorithm has no semantic information of the images (i.e., does not know what a bag is). This lack of semantic knowledge, coupled with the two types of bags being different from a visual standpoint, makes the clustering algorithm split it into two.

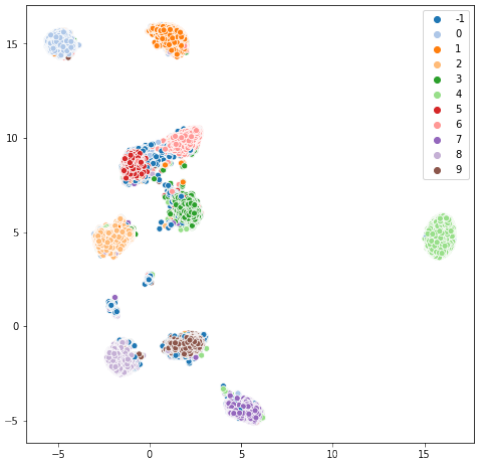
The 'Shoewear' group is originally divided into 3: ankle boots, sneakers and sandals. However (as seen in Figures 4.12(b) and 4.12(c)), GMM clusters the group into 3 but HDBSCAN clusters it into 4. Two of the clusters have been correctly identified: Sneakers (Figure 4.15) and ankle boots (Figure 4.16). The main difference between the ground truth labels and the clustered labels is that, similarly to the bags, the sandals have been split into high heeled and flat sandals (seen in Figure 4.17). This split is, once again, caused by the lack of semantic interpretability that most, if not all, cluster algorithms possess.

In the clothes group, we have five ground-truth clusters: 'Dresses,' 'Pullovers,' 'Coats,' 'Shirts' and 'T-shirts.' As seen in Figures 4.12(b) and 4.12(c) the GMM and HDBSCAN algorithms only found 4 clusters. This difference in cluster numbers is somewhat expected since the data is not very separable in this



(a) UMAP for the MNIST with groundtruth labels

(b) UMAP for the MNIST with HDBSCAN

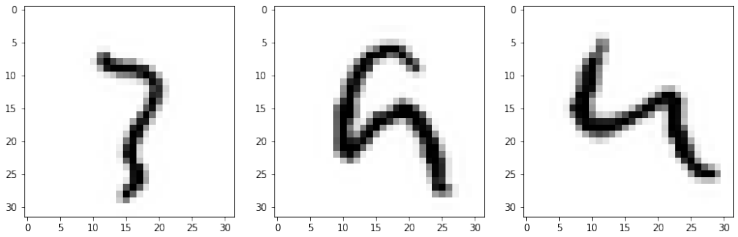


(c) UMAP for the MNIST with GMM

Figure 4.10: UMAP for the MNIST dataset

case. There is a severe overlap between several classes such as Shirts and T-shirts, and pullovers and coats, making their distinction extremely difficult. From Figure 4.18, we can see that, in both algorithms, we have a cluster that manages to get most of the short-sleeved items of clothing (such as T-shirts, tops and short-sleeved shirts), a cluster for long-sleeved items (Figure 4.19) for items such as pullovers and shirts and a cluster that gets long pieces of clothing like dresses and long coats (Figure 4.20). The extra cluster for the HDBSCAN is a residual one, with a meagre amount of samples that are exclusively dresses (and not long pieces of clothing).

The advantage of HDBSCAN in this case is the ability to classify data as noise and allowing for very irregular cluster shapes. It allows for more robust expert initialization as fewer mislabeled points, which



(a) **(b)** **(c)**

Figure 4.11: Images from the noise vector in the MNIST dataset

can lead to bad converging points when training the MoE architecture.

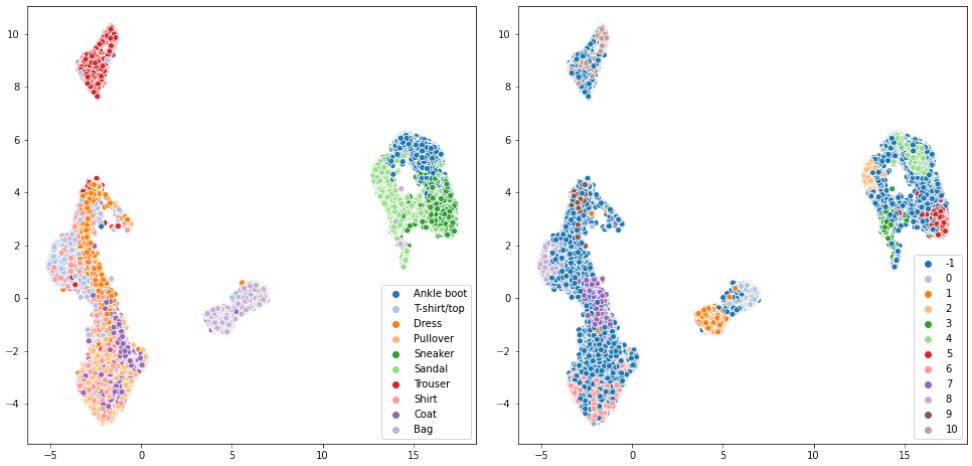
Since this algorithm, like all cluster algorithms, lacks the context of the label (it does not know what separates an image of a short-sleeved shirt from a t-shirt for example) there are many points classified as noise and the performance is far worst than the USPS and MNIST dataset. Still, and just at the pretraining stage, the values of accuracy shown here are very close to state of the art.

COIL 20 We also analyzed the COIL20 dataset in this section. It is a sparse dataset with only 1440 images of 20 different objects, making for very local clusters with small dimensions and irregular shapes, as seen in 4.21(a). As seen in Figures 4.21(a) to 4.21(c) the manifold learning algorithm can separate each distinct cluster in a way that allows both clustering algorithms to correctly distinguish each cluster, reaching 100% accuracy on this dataset.

4.4.4 Manager training

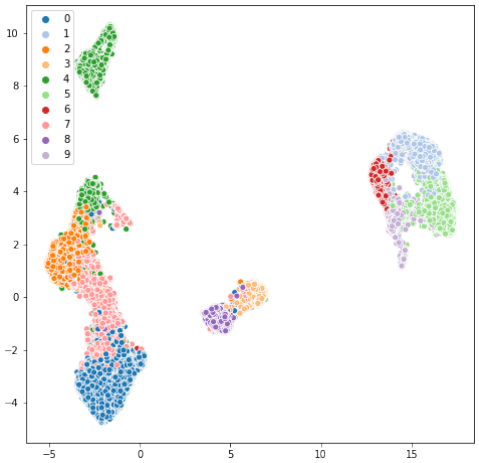
For the manager training, we took two distinct approaches: we used the GMM labels as a baseline to train the manager and the HDBSCAN labels (except the noisy samples) to train another architecture. The results for each dataset are present in Table 4.12. Each manager was trained with a learning rate of 10^{-3}, batch size of 100 and for 1000 epochs with the loss being the kullback-Leiber distance between the soft assignments produced by the manager and the soft cluster labels produced by the shallow clustering methods. This loss function choice was not random. We wanted to train the manager with a soft probability output to make samples that are close to two clusters (for example a four that looks like a 9 in the MNIST dataset) remain in between two clusters. This soft assignment reduces sample misclassification and allows for better expert initialization in the training phase.

As we can see, training the manager using only the HDBSCAN labels makes the manager's accuracy improve on most datasets. This performance improvement is mainly because the HDBSCAN has far fewer mislabelled samples, reducing possible errors in the manager training. However, there were two datasets in which the performance of the GMM surpassed HDBSCAN: COIL20 and CIFAR10. We



(a) UMAP for the FMNIST with groundtruth labels

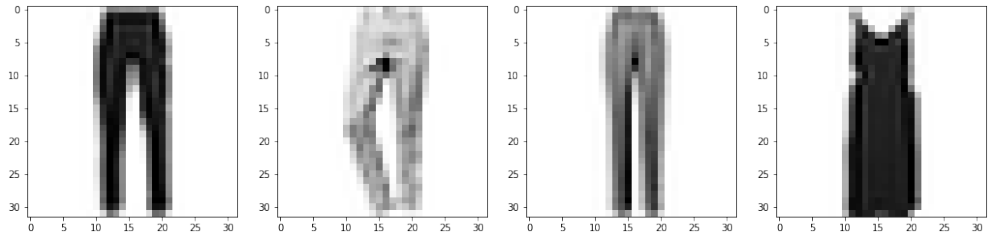
(b) UMAP for the FMNIST with HDBSCAN



(c) UMAP for the FMNIST with GMM

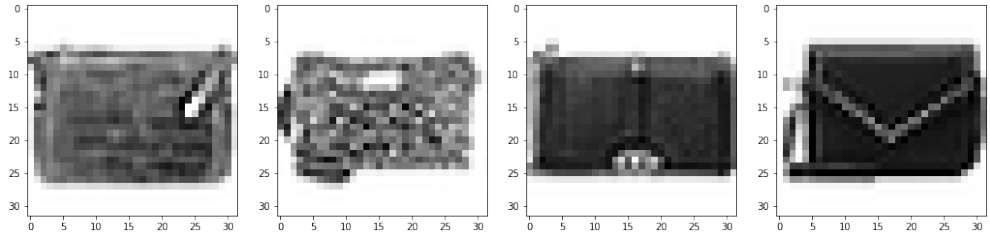
Figure 4.12: UMAP for the FMNIST dataset

attribute this increase in performance to the fact that the manifold of COIL20 is sparse, with clusters very far apart and with a low amount of points, making an algorithm such as GMM that only optimizes centres and borders of clusters easily envelop the data groups. In contrast, an algorithm based on the density of points and hierarchical topologies such as HDBSCAN may have more difficulties. On CIFAR10, both performances are bad, but the GMM had the optimal number of clusters given to it, which increases performance when compared to HDBSCAN.

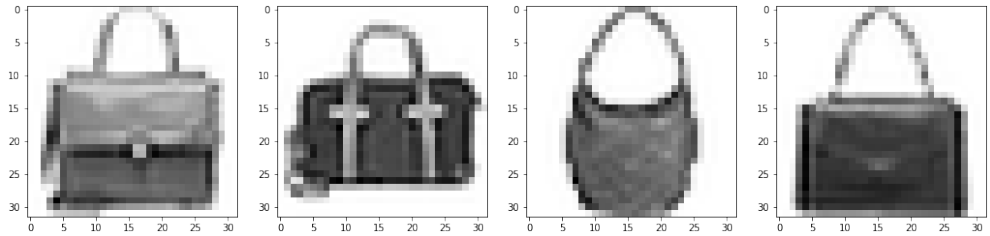


(a) Images from cluster 10 of HDBSCAN **(b)** Images from cluster 4 of GMM

Figure 4.13: Found clusters for the 'Trouser' cluster

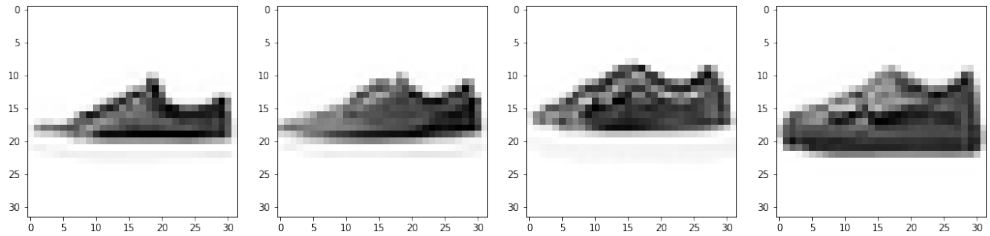


(a) Images from cluster 0 of HDBSCAN **(b)** Images from cluster 3 of GMM



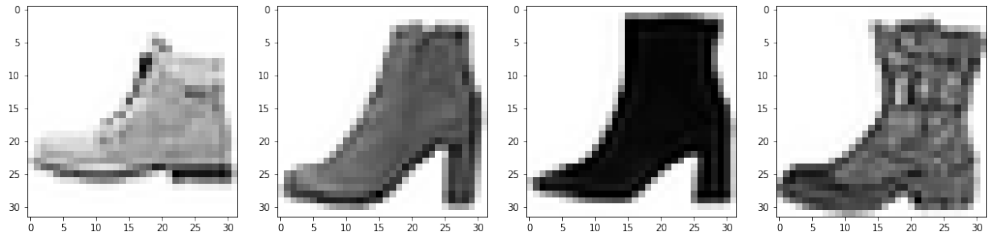
(c) Images from cluster 1 of HDBSCAN **(d)** Images from cluster 8 of GMM

Figure 4.14: Found clusters for the 'Bags' cluster



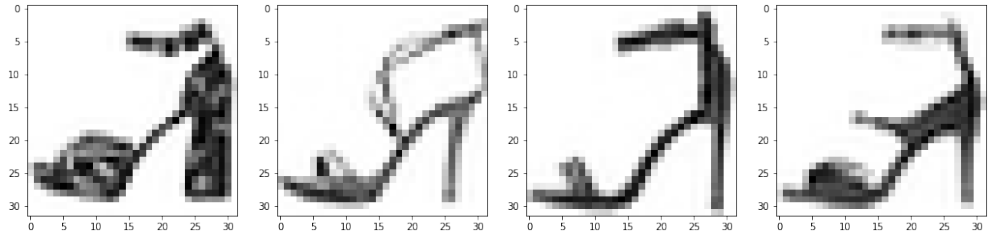
(a) Images from cluster 5 of HDBSCAN **(b)** Images from cluster 5 of GMM

Figure 4.15: Found clusters for the 'Sneaker' cluster

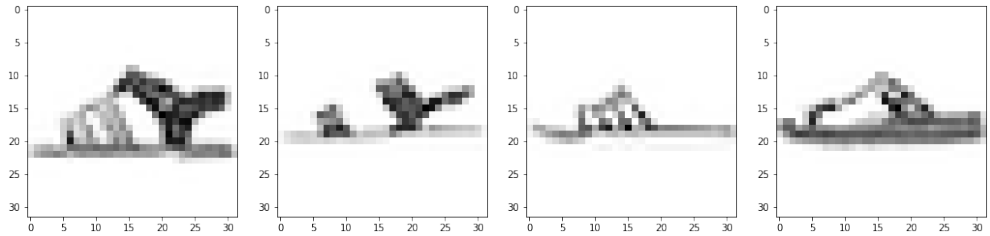


(a) Images from cluster 4 of HDBSCAN **(b)** Images from cluster 1 of GMM

Figure 4.16: Found clusters for the 'Ankle Boot' cluster

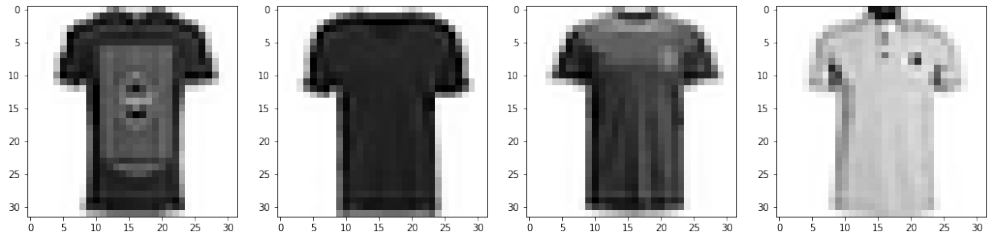


(a) Images from cluster 2 of HDBSCAN **(b)** Images from cluster 6 of GMM



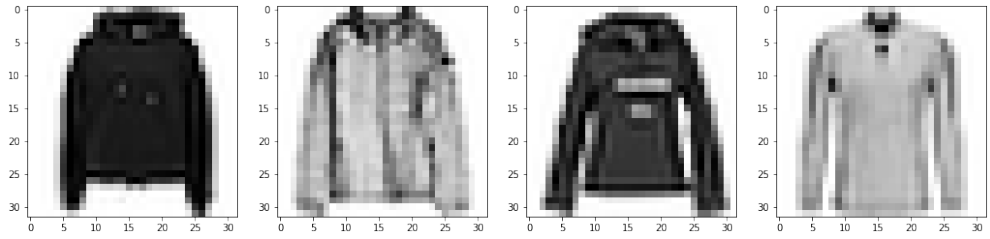
(c) Images from cluster 3 of HDBSCAN **(d)** Images from cluster 9 of GMM

Figure 4.17: Found clusters for the 'Sandals' cluster



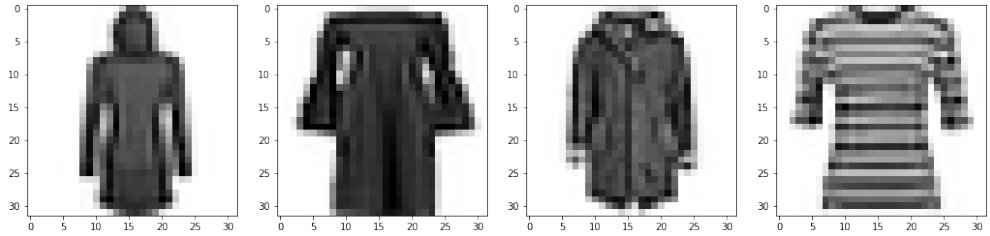
(a) Images from cluster 8 of HDBSCAN **(b)** Images from cluster 2 of GMM

Figure 4.18: Found clusters for the 'T shirt' cluster



(a) Images from cluster 6 of HDBSCAN **(b)** Images from cluster 0 of GMM

Figure 4.19: Found clusters for the 'Pullover' cluster



(a) Images from cluster 7 of HDBSCAN **(b)** Images from cluster 7 of GMM

Figure 4.20: Found clusters for the 'Dress' cluster

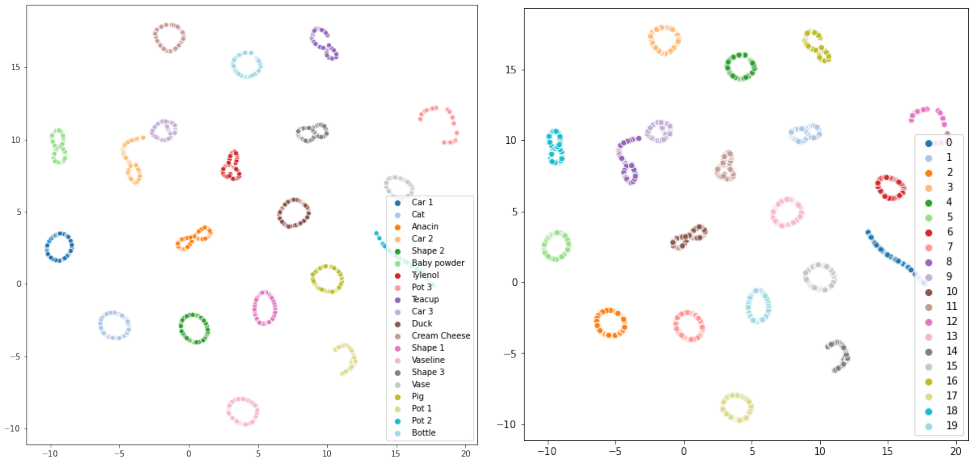
Table 4.12: Results of manager training on each of the datasets

	USPS			MNIST			FMNIST		
	ACC	NMI	ARI	ACC	NMI	ARI	ACC	NMI	ARI
GMM manager	0.968	0.917	0.937	0.961	0.915	0.918	0.592	0.678	0.503
HDBSCAN manager	0.971	0.926	0.943	0.967	0.922	0.927	0.623	0.685	0.528

	COIL20			CIFAR10		
	ACC	NMI	ARI	ACC	NMI	ARI
GMM manager	1	1	1	0.281	0.181	0.083
HDBSCAN manager	0.958	0.957	0.914	0.210	0.108	0.056

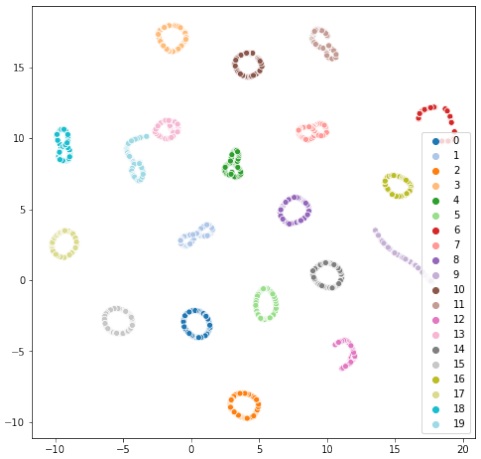
4.5 Training

The training procedure is composed of only two phases: Finding the optimal latent dimension of the experts based on the data and training the architecture. It is in this step where the cluster assignments are progressively hardened and where the expert specialization occurs. Here we will visually assess the quality of the final clustering, the expertise of the experts in image reconstruction and image generation and the final accuracy results of the architecture.



(a) UMAP for the COIL20 with groundtruth labels

(b) UMAP for the COIL20 with HDBSCAN



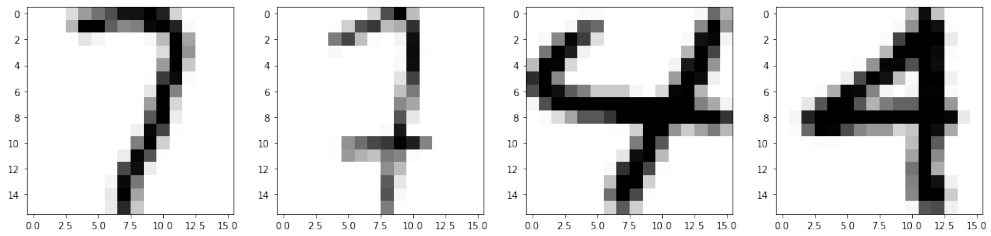
(c) UMAP for the COIL20 with GMM

Figure 4.21: UMAP for the COIL20 dataset

4.5.1 Data dependant latent space for the experts

We trained the dimension finder on each of the found subsets of data for the GMM and HDBSCAN labels. We did this so that each expert fully adapts to its input data (some clusters may need more latent space neurons than others to encode the data assigned to them). Since we are going to use the experts to reconstruct and generate data, we aimed to capture 95% of the explained variance instead of the previous 90% threshold. We present the values for all datasets in tables 4.13 to 4.17.

Once again, the optimal dimensions depend on data complexity, dataset size, and variability. Typically, and as seen in the tables, datasets, where each cluster has a low amount of points, have a smaller



(a) Different images of sevens **(b)** Different images of fours

Figure 4.22: Different images of sevens and fours

!!t]

Table 4.13: Optimal dimensions for each subset of data for the USPS dataset

Correct Label	7	8	0	1	6	9	2	5	3	4
GMM points	778	693	1549	1284	834	849	955	720	809	827
GMM entropy	2.759	3.602	3.795	1.911	3.289	3.026	3.483	3.542	3.539	3.023
GMM optimal z	13.9	15.0	16.0	10.0	16.0	14.1	16.0	16.0	16.0	16.0
Correct Label	0	1	6	2	8	4	3	5	7	9
HDBSCAN points	1549	1284	834	955	691	800	751	648	711	784
HDBSCAN entropy	3.173	3.147	3.185	3.225	3.147	3.267	3.137	3.173	3.072	3.187
HDBSCAN optimal z	15.0	16.0	15.0	15.5	15.0	15.0	15.0	15.0	15.0	15.0

amount of dimensions required for the encoding, as seen in Table 4.16. The COIL20 dataset needed a latent dimension size of 28 for the whole dataset, but, when split into classes, each class only needs around 3-7 neurons, significantly reducing the per expert complexity. It is also dependant on the variability of the data, as seen from the USPS dataset. Digits that can be written in multiple ways such as sevens and fours (Figure 4.22) require more encoding dimensions than other numbers such as zeros or ones.

However, there is a caveat to this rule. As we can see from Table 4.13, symmetric digits like eight, zero, and threes also require less encoding dimensions. This lack of encoding dimensions is because the autoencoder mainly detects edges/patterns in the data. Since these digits are symmetric, these detected patterns can be reused multiple times in the same image, making optimal encoding take a lower latent space dimension.

During the training of the architecture, it was noted that due to the way the loss functions was constructed (namely the batch entropy and KL loss per expert) we require all experts to be of the same dimensions. This requirement is because the higher the expert dimension (i.e. a higher number of Gaussian components), the more significant the KL loss and the more data we can encode per expert. This unbalance between encoding capabilities of the experts resulted in some experts encoding multiple classes (leading to cluster collapsing), decreasing the performance of the algorithm.

Table 4.14: Optimal dimensions for each subset of data for the MNIST dataset

GMM label	0	1	2	3	4	5	6	7	8	9
GMM points	6903	6274	7126	6860	6317	6775	6945	7619	7201	7980
GMM entropy	3.238	2.364	2.888	2.693	2.75	2.919	2.514	2.573	2.887	1.798
GMM optimal z	20.7	18.5	21.0	20.8	20.6	20.9	20.8	21.0	20.5	13.9
HDBSCAN label	0	1	2	3	4	5	6	7	8	9
HDBSCAN points	6903	6860	7126	6428	7766	6715	6169	6180	6545	6318
HDBSCAN entropy	2.64	2.651	2.658	2.652	2.641	2.653	2.663	2.649	2.648	2.649
HDBSCAN optimal z	17.2	21.0	21.0	21.0	21.0	21.0	13.9	21.0	21.0	21.0

Table 4.15: Optimal dimensions for each subset of data for the FMNIST dataset

GMM label	0	1	2	3	4	5	6	7	8	9	10
GMM points	15344	7076	8207	3106	11260	7434	3222	7232	3701	3418	-
GMM entropy	5.842	4.87	5.176	5.54	3.947	3.759	4.342	5.239	5.508	3.07	-
GMM optimal z	20.1	24.3	18.3	21.3	20.6	21.9	23.9	23.4	20.3	19.7	-
HDBSCAN label	0	1	2	3	4	5	6	7	8	9	10
HDBSCAN points	2894	3669	2155	2045	3877	3036	4763	3133	3252	1858	2987
HDBSCAN entropy	4.792	4.812	4.833	4.846	4.845	4.828	4.839	4.864	4.821	4.836	4.822
HDBSCAN optimal z	24.9	25.5	24.1	24.8	25.4	23.9	24.5	25.0	24.4	24.5	23.2

Table 4.16: Optimal dimensions for each subset of data for the COIL20 dataset

GMM label	0	1	2	3	4	5	6	7	8	9
GMM points	72	72	72	72	72	72	72	72	72	72
GMM entropy	4.898	5.319	5.846	5.569	6.111	4.578	6.963	5.161	5.923	6.455
GMM optimal z	5.8	6.2	3.0	2.0	6.8	6.3	2.0	5.9	6.1	2.5
HDBSCAN label	0	1	2	3	4	5	6	7	8	9
HDBSCAN points	72	72	72	72	72	72	72	72	72	72
HDBSCAN entropy	6.455	5.161	4.755	5.569	4.472	4.21	6.177	4.898	4.207	4.476
HDBSCAN optimal z	2.3	3.5	6.0	1.0	3.7	4.6	5.7	5.4	3.4	3.1
GMM label	10	11	12	13	14	15	16	17	18	19
GMM points	72	72	72	72	72	72	72	72	72	72
GMM entropy	4.472	5.244	6.622	4.476	6.088	4.755	6.177	4.21	4.341	4.207
GMM optimal z	3.0	3.6	4.6	3.1	6.0	5.0	3.0	4.7	5.0	4.21
HDBSCAN label	10	11	12	13	14	15	16	17	18	19
HDBSCAN points	72	72	72	72	72	72	72	72	72	72
HDBSCAN entropy	5.319	6.111	6.963	5.923	6.622	6.088	5.244	5.846	4.341	4.578
HDBSCAN optimal z	6.4	6.4	3.0	5.5	3.8	7.4	4.3	3.4	3.4	6.4

To take this into account, we made the latent space for each expert the average of the dimensions for all classes in the dataset. Averaging is not ideal but is a good compromise between dynamic experts and equal experts architecture.

4.5.2 Influence of α and β in the training stage

The final step on the training stage is training the MoE architecture. To do this, we have to balance two parameters during training. A setting α that controls how sure we need to be in our predictions

Table 4.17: Optimal dimensions for each subset of data for the CIFAR10 dataset

GMM label	0	1	2	3	4	5	6	7	8	9
GMM points	5840	4153	8420	5258	15313	4339	1835	1220	7543	6079
GMM entropy	6.988	7.447	7.19	7.419	7.328	7.097	7.083	7.401	7.327	7.267
GMM optimal z	21.3	23.5	19.8	25.1	20.2	18.9	27.0	22.0	22.7	21.1
HDBSCAN label	0	1	2	3	4	5	6	7	8	9
HDBSCAN points	4659	2025	4281	-	-	-	-	-	-	-
HDBSCAN entropy	7.275	7.27	7.268	-	-	-	-	-	-	-
HDBSCAN optimal z	20.3	18.2	21.7	-	-	-	-	-	-	-

(making the prediction vector one-hot encoded) and a setting β that makes sure that our predictions are uniformly distributed among all the datasets (to prevent cluster collapse). Intuitively, we should initially prioritize the entropies vs the reconstruction loss to encourage fair use of autoencoders while avoiding the case where all autoencoders are equally optimized for each input, i.e., they do not specialize in each class. Asymptotically, we should prioritize minimizing the reconstruction error to promote better learning of the manifolds for each cluster and reduce α and β to ensure every data sample assignment to only one autoencoder. To accommodate both phases, we split the training in two: the first half of training is focused in clustering, with α and β terms (i.e. the entropies) contributing to the loss. The main objective of this phase is to make sure that each expert specializes in one cluster centre, and the manager guesses the correct label the maximum number of times as possible. The second half of training is focused on getting the experts to learn the underlying manifold of their assigned set of data. To this effect, we predict the hard clustering assignments using the manager and train each separate expert with its subset of the data.

By analyzing the magnitude of the losses for the training stage, the reconstruction + KL loss is of the order of 10^0, the sample entropy is of 10^{-2}, and the sample entropy is of the order of 10^0. To have all terms contributing the same to the loss, we chose $\alpha = 100$ and $\beta = 1$ for the first stage. The full training stage is composed of 200 epochs (100 for each stage) and each architecture was trained with a learning rate of 10^{-4} and batch size of 100

4.5.3 Expertise of experts

Expertise of reconstruction To determine how each expert specializes in the final architecture, we reconstructed an input image where every pixel value is equal to one. The results are present in Figure 4.23 where the leftmost image is the original image, and the rest are the reconstructions made by each expert.

From this figure, we can deduce that each expert is specialized in the reconstruction of a separate class of the dataset. It makes sense to view the clustering problem as a set of experts competing for the best reconstruction: given a new image we can cluster/perform classification of this new data by

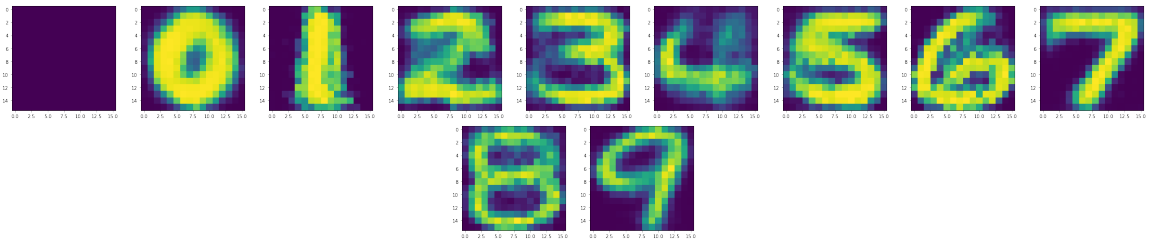


Figure 4.23: Reconstructed images

checking which expert presents the lowest reconstruction error.

Expertise of data generation By feeding random multivariate Gaussian vectors with the same length as the expert's latent dimension, we can generate new samples from the inherent distribution of the network. By applying this process to the experts trained in the USPS dataset, we obtained the samples in Figure 4.24, where each line represents the generated images from 10 random vectors taken from a multivariate Gaussian with zero mean and diagonal unitary covariance matrix.

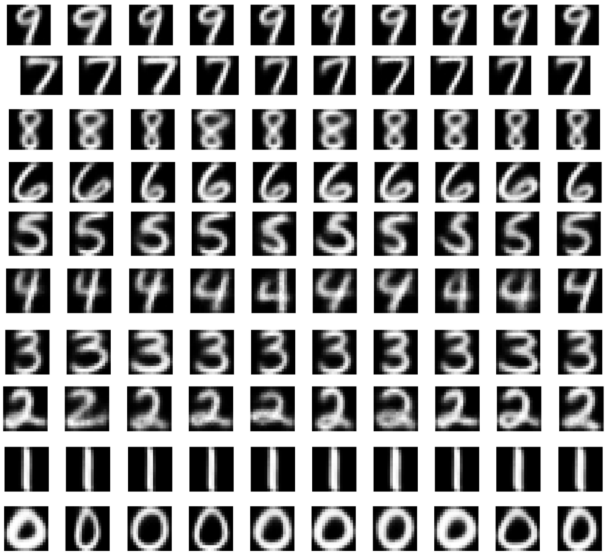


Figure 4.24: Images generated from the experts trained on the USPS dataset

As seen from the figure, every single expert once again specializes in one specific digit/class and provides high-quality image generation. One advantage of the latent space of the experts is that not only is it continuous, but by sampling it along its dimensions makes the data generated gradually change. For example, suppose we perform a walk between -1 and 1 on the first dimension of the variational autoencoder trained on the USPS dataset. In that case, we have the following images (Figures 4.25 and 4.26).

Figure 4.25: Images generated from the expert specialized in fours

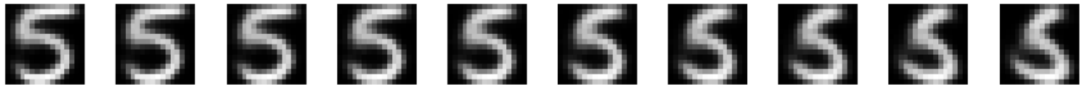


Figure 4.26: Images generated from the expert specialized in fives

As we can see, by gradually increasing the first dimension of the latent space, we can gradually change the style of writing of this digit, proving that this latent space is not only continuous but also directly affects palpable features of the image data.

Expertise of latent space To check the latent spaces of the experts, we use the FMNIST dataset. We chose this dataset for analysis because most experts in the clothes group model more than one class inside them. We once again used the UMAP transform with the parameters mentioned at the beginning of the chapter and with two components. Two components allow for the visualization of the latent space. We plotted the latent space of the expert that specialized in ankle boots (Figure 4.27).

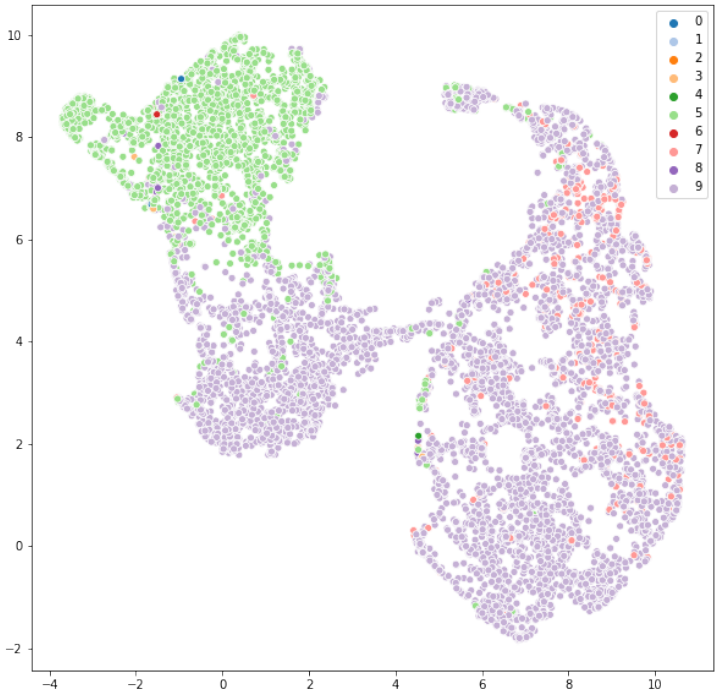


Figure 4.27: Latent dimension for the expert specialized in ankle boots

This cluster has two distinct types of ankle boots: with heel and without the heel. This distinction can

be seen by the two groups or light purple (cluster 9). The rightmost part represents the flat ankle boots, and the leftmost part represents the heeled ankle boots. There is a clear separation between these two sub-clusters, and we can also see that the heeled sandals (cluster 5) can also be confused with the heeled ankle boots. This confusion demonstrated that the latent space of each expert has semantic meaning and can be used to improve our knowledge further and to get new concepts of the data.

4.6 Final accuracy results

The average of five runs's final accuracy, NMI and ARI results for the whole architecture and for all datasets is present in tables 4.18, 4.19 and 4.20

Table 4.18: Accuracy for both methods for all datasets

	USPS	MNIST	FMNIST	COIL20	CIFAR10
Manager HDBSCAN	0.971	0.967	0.623	0.958	0.210
Manager GMM	0.968	0.961	0.592	1	0.281
MoE HDBSCAN	0.986	0.974	0.680	1	0.189
MoE GMM	0.980	0.967	0.598	1	0.328

Table 4.19: NMIs for both methods for all datasets

	USPS	MNIST	FMNIST	COIL20	CIFAR10
Manager HDBSCAN	0.926	0.922	0.685	0.957	0.108
Manager GMM	0.917	0.915	0.678	1	0.181
MoE HDBSCAN	0.960	0.949	0.720	1	0.082
MoE GMM	0.945	0.931	0.708	1	0.143

Table 4.20: ARIs for both methods for all datasets

	USPS	MNIST	FMNIST	COIL20	CIFAR10
Manager HDBSCAN	0.943	0.927	0.528	0.914	0.056
Manager GMM	0.937	0.918	0.0503	1	0.083
MoE HDBSCAN	0.971	0.944	0.570	1	0.055
MoE GMM	0.960	0.931	0.520	1	0.051

As we can see from the highlighted results, the usage of the HDBSCAN algorithm presents distinct advantages compared to the GMM, namely the increase of accuracy, NMI, and ARI scores to values that surpass the state of the art methods. This increase in performance can be explained by the fact that clusters are better defined in the pretraining stage and there are less mislabeled samples, leading to fewer errors when training the MoE architecture. However, there are still downsides to using HDBSCAN over GMM. Since we cannot specify the correct number of clusters, if we do not discover the correct number of clusters (like what happened to the CIFAR10 dataset), HDBSCAN performs worse than GMM. In the next sections, we will compare our architecture with several models from state of the art.

4.6.1 Comparison to the mixture of expert's methods

Table 4.21 presents the comparison between our architecture using the HDBSCAN algorithm and the DAMIC and MIXAE models. Cells marked with the symbol '-' indicate that in the original work, there was no information for the specified dataset.

Table 4.21: Accuracy for all mixture of experts models

	USPS	MNIST	FMNIST	COIL20	CIFAR10
DAMIC	-	0.89	0.60	-	-
MIXAE	-	0.85	-	-	-
MoE	0.99	0.97	0.68	1	0.19

As we can see, our model far surpasses these two architectures in available datasets and information. These results provide a way for our algorithm to be a definitive baseline in MoE based clustering: all of our design decisions are justified, and we have results to back them up, and this architecture has been testes in a plethora of datasets of varying dimensions and complexities.

4.6.2 Comparison to autoencoder based methods

Table 4.22 presents the comparison between our architecture using the HDBSCAN algorithm and the DEC,N2D and DynAE models.

Table 4.22: Accuracy for all autoencoder based methods

	USPS	MNIST	FMNIST	COIL20	CIFAR10
DEC	0.762	0.89	0.518	-	-
N2D	0.958	0.979	0.672	-	-
DynAE	0.981	0.987	0.591	-	-

We once again see that our algorithm presents the state of the art accuracy results on all given datasets from these results. It far surpasses DEC and is on par with the DynAE approach. N2D only outperforms it in the pretraining stage on the FMNIST dataset, which seems strange because our pretraining stage is essentially an N2D with a few modifications. Since there is no information on how the results on N2D were achieved (e.g. whether the results are an average across runs or a best case) and we couldn't reproduce these results with the authors' proposed architecture, we assumed that it is a best of multiple runs. Still, our pretraining method surpasses or equals the N2D approach for the rest of the datasets.

4.7 Summary

In this section, we performed several experiments to find the optimal design for the architecture. We started by analyzing several possibilities for architectures with several metrics before settling on a Variational Autoencoder as our primary building block. Next, we talked about which type of loss to use when associated with a VAE and depending on the distribution of the data and which activation function to use when training the pretraining and training stages. Following these crucial decisions we started by describing several parameters of the pretraining stage, namely selecting the appropriate latent dimension size for the VAE, justifying the usage of UMAP and HDBSCAN on the pretraining stage and the results of training the manager network with HDBSCAN vs GMM labels. Following the pretraining stage, we refer to several aspects of the training procedure, namely the latent space dimension for each of the experts, the influence of α and β in the training stage and analyzing the expertise of the experts.

5

Conclusion

Contents

5.1 Review of the work done 79

5.2 Future work 79

5.3 Closing remarks 80

5.1 Review of the work done

This thesis presented an architecture for clustering and image generation based on the mixture of experts framework that had several proposed baselines that it should have adhered to:

- The architecture must allow for the generation of data.
- The architecture must be fully data dependant and dynamically change based on the requirements needed from the input data.
- The architecture must automatically and, in an unsupervised manner, find the optimal number of clusters by which to model the data.

The proposed architecture not only achieved all the proposed points but significantly improved them. The architecture not only can generate new data but, due to the Mixture of Experts framework, can generate data from withing each cluster independently and allows for additional sub-clustering capabilities by using the manifold of the experts to perform further analysis. The architecture topology and depth are entirely data dependant, using shallower networks for smaller images and bigger networks for bigger images, leading to a network that can automatically change its topology based on the data. The latent dimension finder is a development that allows us to not only achieve but surpass the second proposed point, by finding a way to numerically find the optimal bottleneck size for all architectures, leaving a hyperparameter that had significant influence in the network's performance behind. Finally, and by using the HDBCAN algorithm allied with the manifold learning techniques of UMAP, we found a way to not only select the optimal number of clusters by which to model the data but, by using HDBSCAN's noise vector, found a way to isolate conflicted samples in the pretraining stage. This noise vector provided us with a way to increase pretraining performance and better initialize the manager of the final training architecture. Every single point was not only done but surpassed, all while achieving state-of-the-art performance in datasets of varying complexities, sizes and distributions which proves the robustness and adaptability of this algorithm.

5.2 Future work

This dissertation is the perfect starting point for multiple research directions. From a purely algorithmic perspective, it would be interesting to change the VAEs for WAEs and entirely run the whole architecture. WAEs supposedly have better reconstruction capabilities with slightly lower clustering performance, which would sacrifice performance for higher fidelity image generation. Another interesting approach is replacing the VAEs in the MoE architecture for β VAEs. This replacement would disentangle the latent

space, possibly making it more interpretable and provide further insights into each class. Another possible research avenue is training the experts with different latent dimension sizes: this would require coming up with some penalty term or changing the way α and $beta$ are made to account for this change. From a data analysis perspective, it would be interesting to extend this architecture to not only images but text corpora, videos, and even numeric datasets. This modification would require new architectures that have not been explored in this dissertation, such as RNNs. Still, since we did not impose any restrictions on the type of architecture to use in the MoE, it should be possible to achieve exciting results.

5.3 Closing remarks

Overall, an architecture was proposed with several objectives: it had to be fully dynamic and data dependant, based on the mixture of experts framework and specialized on image data clustering. Everything was built from the ground up: every architecture, every variation, every metric and the results we all made over nine months and the results far exceeded what was set to do. We achieved one of (if not the) best architecture for image clustering tasks and set a definitive baseline for the mixture of expert's clustering methodologies for data clustering and generation. At the beginning of this work, my knowledge of deep learning was shallow and mostly theoretical. This project allowed me to put my knowledge into practise and allowed me to use my imagination and problem-solving skills to design from scratch this architecture.

Bibliography

[Aggarwal et al., 2001] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In Van den Bussche, J. and Vianu, V., editors, *Database Theory — ICDT 2001*, pages 420–434, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Aljalbout et al., 2018] Aljalbout, E., Golkov, V., Siddiqui, Y., Strobel, M., and Cremers, D. (2018). Clustering with Deep Learning: Taxonomy and New Methods. *arXiv:1801.07648*.

[Aloise et al., 2009] Aloise, D., Deshpande, A., Hansen, P., and Popat, P. (2009). Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248.

[Ames et al., 2019] Ames, C. P., Smith, J. S., Pellisé, F., Kelly, M., Alanay, A., Acaroglu, E., Pérez-Grueso, F. J. S., Kleinstück, F., Obeid, I., Vila-Casademunt, A., et al. (2019). Artificial intelligence based hierarchical clustering of patient types and intervention categories in adult spinal deformity surgery: towards a new classification scheme that predicts quality and value. *Spine*, 44(13):915–926.

[Arimond and Elfessi, 2001] Arimond, G. and Elfessi, A. (2001). A clustering method for categorical data in tourism market segmentation research. *Journal of Travel Research*, 39(4):391–397.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

[Berthelot et al., 2018] Berthelot, D., Raffel, C., Roy, A., and Goodfellow, I. (2018). Understanding and improving interpolation in autoencoders via an adversarial regularizer. *arXiv preprint arXiv:1807.07543*.

[Brown et al., 1992] Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., Lai, J. C., and Mercer, R. L. (1992). An estimate of an upper bound for the entropy of english. *Computational Linguistics*, 18(1):31–40.

[Cai et al., 2010] Cai, D., He, X., and Han, J. (2010). Locally consistent concept factorization for document clustering. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):902–913.

[Caliński and Harabasz, 1974] Caliński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27.

[Cao et al., 2020] Cao, L., Asadi, S., Zhu, W., Schmidli, C., and Sjöberg, M. (2020). Simple, Scalable, and Stable Variational Deep Clustering. *arXiv:2005.08047*.

[Chan et al., 2015] Chan, T.-H., Jia, K., Gao, S., Lu, J., Zeng, Z., and Ma, Y. (2015). Pcanet: A simple deep learning baseline for image classification? *IEEE transactions on image processing*, 24(12):5017–5032.

[Chang et al., 2017] Chang, J., Wang, L., Meng, G., Xiang, S., and Pan, C. (2017). Deep Adaptive Image Clustering. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5880–5888. IEEE.

[Chazan et al., 2019] Chazan, S. E., Gannot, S., and Goldberger, J. (2019). Deep Clustering based on a Mixture of Autoencoders. In *IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.

[Chen et al., 2017] Chen, D., Lv, J., and Yi, Z. (2017). Unsupervised Multi-Manifold Clustering by Learning Deep Representation. In *AAAI Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, pages 385–391.

[Chen and Huang, 2019] Chen, P.-Y. and Huang, J.-J. (2019). A Hybrid Autoencoder Network for Unsupervised Image Clustering. *Algorithms*, 12(6):122.

[Chen et al., 2016] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 29*, pages 2172–2180.

[Chen et al., 2018] Chen, Y., Zhang, L., and Yi, Z. (2018). Subspace clustering using a low-rank constrained autoencoder. *Information Sciences*, 424:27–38.

[Chuang et al., 2006] Chuang, K.-S., Tzeng, H.-L., Chen, S., Wu, J., and Chen, T.-J. (2006). Fuzzy c-means clustering with spatial information for image segmentation. *computerized medical imaging and graphics*, 30(1):9–15.

[Coleman and Andrews, 1979] Coleman, G. B. and Andrews, H. C. (1979). Image segmentation by clustering. *Proceedings of the IEEE*, 67(5):773–785.

[Damian et al., 2007] Damian, D., Orešič, M., Verheij, E., Meulman, J., Friedman, J., Adourian, A., Morel, N., Smilde, A., and van der Greef, J. (2007). Applications of a new subspace clustering algorithm (cosa) in medical systems biology. *Metabolomics*, 3(1):69–77.

[de Queiroz and Good, 1997] de Queiroz, K. and Good, D. A. (1997). Phenetic clustering in biology: a critique. *The Quarterly Review of Biology*, 72(1):3–30.

[De Simone et al., 2010] De Simone, F., Goldmann, L., Lee, J.-S., Ebrahimi, T., and Baronicni, V. (2010). Subjective evaluation of next-generation video compression algorithms: A case study.

[Dilokthanakul et al., 2017] Dilokthanakul, N., Mediano, P. A. M., Garnelo, M., Lee, M. C. H., Salimbeni, H., Arulkumaran, K., and Shanahan, M. (2017). Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders. *arXiv:1611.02648*.

[Divya and Devi, 2018] Divya, V. and Devi, K. N. (2018). An efficient approach to determine number of clusters using principal component analysis. In *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, pages 1–6.

[Dizaji et al., 2017] Dizaji, K. G., Herandi, A., Deng, C., Cai, W., and Huang, H. (2017). Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5747–5756. IEEE.

[Dizaji et al., 2019] Dizaji, K. G., Wang, X., Deng, C., and Huang, H. (2019). Balanced Self-Paced Learning for Generative Adversarial Clustering Network. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4386–4395. IEEE.

[Elkan, 2003] Elkan, C. (2003). Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 147–153. AAAI Press.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[Figueiredo and Jain, 2002] Figueiredo, M. A. T. and Jain, A. K. (2002). Unsupervised learning of finite mixture models. *IEEE Transactions on pattern analysis and machine intelligence*, 24(3):381–396.

[Figueroa and Rivera, 2017] Figueroa, J. A. and Rivera, A. R. (2017). Is Simple Better?: Revisiting Simple Generative Models for Unsupervised Clustering. In *Second workshop on Bayesian Deep Learning (NIPS)*.

[Gondara, 2016] Gondara, L. (2016). Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246. IEEE.

[Gönen and Margolin, 2014] Gönen, M. and Margolin, A. A. (2014). Localized data fusion for kernel k-means clustering with application to cancer biology. In *Advances in neural information processing systems*, pages 1305–1313.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Guo et al., 2017a] Guo, X., Gao, L., Liu, X., and Yin, J. (2017a). Improved Deep Embedded Clustering with Local Structure Preservation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1753–1759.

[Guo et al., 2017b] Guo, X., Liu, X., Zhu, E., and Yin, J. (2017b). Deep Clustering with Convolutional Autoencoders. In *Neural Information Processing, ICONIP*, volume 10635, pages 373–382.

[Guo et al., 2019] Guo, X., Liu, X., Zhu, E., Zhu, X., Li, M., Xu, X., and Yin, J. (2019). Adaptive Self-paced Deep Clustering with Data Augmentation. *IEEE Transactions on Knowledge and Data Engineering*.

[Guo et al., 2018] Guo, X., Zhu, E., Liu, X., and Yin, J. (2018). Deep Embedded Clustering with Data Augmentation. In *Asian Conference on Machine Learning (ACML)*, pages 550–565.

[Harchaoui et al., 2017] Harchaoui, W., Mattei, P. A., and Bouveyron, C. (2017). Deep Adversarial Gaussian Mixture Autoencoder for Clustering. In *International Conference on Learning Representations (ICLR) - Workshop track*.

[Harrigan, 1985] Harrigan, K. R. (1985). An application of clustering for strategic group analysis. *Strategic Management Journal*, 6(1):55–73.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[Heusel et al., 2017] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Klambauer, G., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500.

[Higgins et al., 2016] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2016). beta-vae: Learning basic visual concepts with a constrained variational framework.

[Horé and Ziou, 2010] Horé, A. and Ziou, D. (2010). Image quality metrics: Psnr vs. ssim. pages 2366–2369.

[Hosseinimotlagh and Papalexakis, 2018] Hosseinimotlagh, S. and Papalexakis, E. E. (2018). Unsupervised content-based identification of fake news articles with tensor decomposition ensembles. In *Proceedings of the Workshop on Misinformation and Misbehavior Mining on the Web (MIS2)*.

[Hsu and Lin, 2018] Hsu, C.-C. and Lin, C.-W. (2018). CNN-Based Joint Clustering and Representation Learning with Feature Drift Compensation for Large-Scale Image Data. *IEEE Transactions on Multimedia*, 20(2):421–429.

[Hu et al., 2017] Hu, W., Miyato, T., Tokui, S., Matsumoto, E., and Sugiyama, M. (2017). Learning Discrete Representations via Information Maximizing Self-Augmented Training. In *International Conference on Machine Learning (ICML)*, pages 1558–1567.

[Hull, 1994] Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554.

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

[Ji et al., 2019] Ji, X., Vedaldi, A., and Henriques, J. (2019). Invariant Information Clustering for Unsupervised Image Classification and Segmentation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9864–9873. IEEE.

[Jiang et al., 2017] Jiang, Z., Zheng, Y., Tan, H., Tang, B., and Zhou, H. (2017). Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1965–1972.

[Jin et al., 2019] Jin, Y., Du, L., Gao, L., Xiang, Y., Li, Y., and Xu, R. (2019). Variational auto-encoder based bayesian poisson tensor factorization for sparse and imbalanced count data.

[Kaya et al., 2017] Kaya, I. E., Pehlivanlı, A. Ç., Sekizkardeş, E. G., and Ibrikci, T. (2017). Pca based clustering for brain tumor segmentation of t1w mri images. *Computer methods and programs in biomedicine*, 140:19–28.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kopf et al., 2020] Kopf, A., Fortuin, V., Somnath, V. R., and Claassen, M. (2020). Mixture-of-Experts Variational Autoencoder for clustering and generating from similarity-based representations. *arXiv:1910.07763*.

[Kriegel et al., 2011] Kriegel, H.-P., Kröger, P., Sander, J., and Zimek, A. (2011). Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240.

[Krizhevsky et al., 2009] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

[LeCun et al., 2010] LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

[Li et al., 2011] Li, B. N., Chui, C. K., Chang, S., and Ong, S. H. (2011). Integrating spatial fuzzy clustering with level set methods for automated medical image segmentation. *Computers in biology and medicine*, 41(1):1–10.

[Li et al., 2018] Li, F., Qiao, H., and Zhang, B. (2018). Discriminatively boosted image clustering with fully convolutional auto-encoders. *Pattern Recognition*, 83:161–173.

[Li et al., 2019] Li, X., Chen, Z., Poon, L. K. M., and Zhang, N. L. (2019). Learning Latent Superstructures in Variational Autoencoders for Deep Multidimensional Clustering. In *International Conference on Learning Representations (ICLR)*.

[Lin et al., 2019] Lin, X., Yang, X., and Li, Y. (2019). A Deep Clustering Algorithm based on Gaussian Mixture Model. *Journal of Physics: Conference Series*, 1302:032012.

[Lloyd, 1982] Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE Trans. Inf. Theory*, 28:129–136.

[Loaiza-Ganem and Cunningham, 2019] Loaiza-Ganem, G. and Cunningham, J. P. (2019). The continuous bernoulli: fixing a pervasive error in variational autoencoders.

[Lu et al., 2013] Lu, X., Tsao, Y., Matsuda, S., and Hori, C. (2013). Speech enhancement based on deep denoising autoencoder. In *Interspeech*, volume 2013, pages 436–440.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[McConville et al., 2019] McConville, R., Santos-Rodriguez, R., Piechocki, R. J., and Craddock, I. (2019). N2D: (Not Too) Deep Clustering via Clustering the Local Manifold of an Autoencoded Embedding. *arXiv:1908.05968*.

[McInnes et al., 2017] McInnes, L., Healy, J., and Astels, S. (2017). hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205.

[McInnes et al., 2018] McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

[Meng et al., 2017] Meng, L., Ding, S., and Xue, Y. (2017). Research on denoising sparse autoencoder. *International Journal of Machine Learning and Cybernetics*, 8(5):1719–1729.

[Mika et al., 1999] Mika, S., Schölkopf, B., Smola, A. J., Müller, K.-R., Scholz, M., and Rätsch, G. (1999). Kernel pca and de-noising in feature spaces. In *Advances in neural information processing systems*, pages 536–542.

[Min et al., 2018a] Min, E., Guo, X., Liu, Q., Zhang, G., Cui, J., and Long, J. (2018a). A Survey of Clustering With Deep Learning: From the Perspective of Network Architecture. *IEEE Access*, 6:39501–39514.

[Min et al., 2018b] Min, E., Guo, X., Liu, Q., Zhang, G., Cui, J., and Long, J. (2018b). A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514.

[Misra, 2019] Misra, D. (2019). Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*.

[Mrabah et al., 2020] Mrabah, N., Khan, N. M., Ksantini, R., and Lachiri, Z. (2020). Deep Clustering with a Dynamic Autoencoder: From Reconstruction towards Centroids Construction. *arXiv:1901.07752*.

[Mukherjee et al., 2019] Mukherjee, S., Asnani, H., Lin, E., and Kannan, S. (2019). ClusterGAN: Latent Space Clustering in Generative Adversarial Networks. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4610–4617.

[Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

[Murtagh and Contreras, 2012] Murtagh, F. and Contreras, P. (2012). Algorithms for hierarchical clustering: An overview. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 2:86–97.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*.

[Nasraoui and N'Cir, 2019] Nasraoui, O. and N'Cir, C.-E. B. (2019). *Clustering Methods for Big Data Analytics*. Springer.

[Nene et al., 1996] Nene, S. A., Nayar, S. K., Murase, H., et al. (1996). Columbia object image library (coil-100).

[Oliver et al., 1996] Oliver, J. J., Baxter, R. A., and Wallace, C. S. (1996). Unsupervised learning using mml. In *ICML*, pages 364–372.

[Pappas and Jayant, 1989] Pappas, T. N. and Jayant, N. S. (1989). An adaptive clustering algorithm for image segmentation. In *International Conference on Acoustics, Speech, and Signal Processing,*, pages 1667–1670. IEEE.

[Parikh and Atrey, 2018] Parikh, S. B. and Atrey, P. K. (2018). Media-rich fake news detection: A survey. In *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 436–441. IEEE.

[Pearson, 1901] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[Ramachandran et al., 2017] Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

[Rand, 1971] Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850.

[Rousseeuw, 1987] Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

[Schroff et al., 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

[Shah and Koltun, 2018] Shah, S. A. and Koltun, V. (2018). Deep Continuous Clustering. *arXiv:1803.01449*.

[Shaham et al., 2018] Shaham, U., Stanton, K., Li, H., Nadler, B., Basri, R., and Kluger, Y. (2018). SpectralNet: Spectral Clustering using Deep Neural Networks. In *International Conference on Learning Representations (ICLR)*.

[Shyu et al., 2004] Shyu, M.-L., Chen, S.-C., Chen, M., and Zhang, C. (2004). A unified framework for image database clustering and content-based retrieval. In *Proceedings of the 2nd ACM international workshop on Multimedia databases*, pages 19–27.

[Sleegers et al., 2004] Sleegers, K., Roks, G., Theuns, J., Aulchenko, Y., Rademakers, R., Cruts, M., Van Gool, W., Van Broeckhoven, C., Heutink, P., Oostra, B., et al. (2004). Familial clustering and genetic risk for dementia in a genetically isolated dutch population. *Brain*, 127(7):1641–1649.

[Solovieff et al., 2010] Solovieff, N., Hartley, S. W., Baldwin, C. T., Perls, T. T., Steinberg, M. H., and Sebastiani, P. (2010). Clustering by genetic ancestry using genome-wide snp data. *BMC genetics*, 11(1):108.

[Sønderby et al., 2016] Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746.

[Springenberg, 2016] Springenberg, J. T. (2016). Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*.

[Srivastava et al., 1981] Srivastava, R. K., Leone, R. P., and Shocker, A. D. (1981). Market structure analysis: hierarchical clustering of products based on substitution-in-use. *Journal of Marketing*, 45(3):38–48.

[Strehl and Ghosh, 2002] Strehl, A. and Ghosh, J. (2002). Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617.

[Szegedy et al., 2014a] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014a). Going deeper with convolutions.

[Szegedy et al., 2014b] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014b). Going deeper with convolutions. *CoRR*, abs/1409.4842.

[Szegedy et al., 2015] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567.

[Taher and Sammouda, 2011] Taher, F. and Sammouda, R. (2011). Lung cancer detection by using artificial neural network and fuzzy clustering methods. In *2011 IEEE GCC Conference and Exhibition (GCC)*, pages 295–298. IEEE.

[Tolstikhin et al., 2017] Tolstikhin, I., Bousquet, O., Gelly, S., and Schoelkopf, B. (2017). Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*.

[Tukan et al., 2020] Tukan, M., Maalouf, A., and Feldman, D. (2020). Coresets for near-convex functions.

[Tzoreff et al., 2018] Tzoreff, E., Kogan, O., and Choukroun, Y. (2018). Deep Discriminative Latent Space for Clustering. *arXiv:1805.10795*.

[Tzortzis and Likas, 2014] Tzortzis, G. and Likas, A. (2014). The minmax k-means clustering algorithm. *Pattern Recognition*, 47(7):2505 – 2516.

[Uğur et al., 2020] Uğur, Y., Arvanitakis, G., and Zaidi, A. (2020). Variational Information Bottleneck for Unsupervised Clustering: Deep Gaussian Mixture Embedding. *Entropy*, 22(2):213.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

[Xie et al., 2016] Xie, J., Girshick, R., and Farhadi, A. (2016). Unsupervised Deep Embedding for Clustering Analysis. In *International Conference on Machine Learning (ICML)*.

[Yan et al., 2020] Yan, Y., Hao, H., Xu, B., Zhao, J., and Shen, F. (2020). Image Clustering via Deep Embedded Dimensionality Reduction and Probability-Based Triplet Loss. *IEEE Transactions on Image Processing*, 29:5652–5661.

[Yang et al., 2017] Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. (2017). Towards K-means-friendly spaces: simultaneous deep learning and clustering. In *International Conference on Machine Learning (ICML)*, pages 3861–3870.

[Yang et al., 2016] Yang, J., Parikh, D., and Batra, D. (2016). Joint Unsupervised Learning of Deep Representations and Image Clusters. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5147–5156. IEEE.

[Yang et al., 2019a] Yang, L., Cheung, N.-M., Li, J., and Fang, J. (2019a). Deep Clustering by Gaussian Mixture Variational Autoencoders With Graph Embedding. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6439–6448. IEEE.

[Yang et al., 2019b] Yang, X., Deng, C., Zheng, F., Yan, J., and Liu, W. (2019b). Deep Spectral Clustering Using Dual Autoencoder Network. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4061–4070. IEEE.

[Zeng et al., 2018] Zeng, N., Zhang, H., Song, B., Liu, W., Li, Y., and Dobaie, A. M. (2018). Facial expression recognition via learning deep sparse autoencoders. *Neurocomputing*, 273:643–649.

[Zhang et al., 2017] Zhang, D., Sun, Y., Eriksson, B., and Balzano, L. (2017). Deep Unsupervised Clustering Using Mixture of Autoencoders. *arXiv:1712.07788*.

[Zhang et al., 2017] Zhang, Z., Zhau, J., Liu, N., Gu, X., and Zhang, Y. (2017). An improved pairwise comparison scaling method for subjective image quality assessment. In *2017 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–6.

[Zhu et al., 2020] Zhu, Q., Bi, W., Liu, X., Ma, X., Li, X., and Wu, D. (2020). A batch normalized inference network keeps the kl vanishing away. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.