TÉCNICO LISBOA

Generative Deep Clustering with the Hierarchical and Relativistic Wasserstein Autoencoder GAN.

Gustavo Augusto Toscano Morais

Thesis to obtain the Master of Science Degree in

Computer Science

Supervisors: Prof. Pedro Filipe Zeferino Tomás
Prof. Helena Isabel Aidos Lopes

Examination Committee

Chairperson: Prof. José Luis Brinquete Borbinha
Supervisor: Prof. Pedro Filipe Zeferino Tomás
Member of the Committee: Prof. João Carlos Raposo Neves

October 2021

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under projects UIDB/50021/2020 and PCIF/MPG/0051/2018 (ResNetDetect).

Abstract

Clustering is one of the most important tasks in the unsupervised learning field, and traditional methods (like k-means) struggle when dealing with high-dimensional data. For this reason, many researchers have successfully attempted to use deep learning techniques to map the data into a lower dimensional, cluster friendly space. The process usually consists of using an autoencoder-based structure to find a meaningful latent space and then using traditional clustering algorithms on it.

In this dissertation, I developed a novel approach to clustering by using a generative hierarchical autoencoder containing multiple latent groups (the vanilla autoencoder has just one) that are regularized by latent relativistic discriminators in an adversarial manner. These latent groups are then clustered by the density-based clustering algorithm HDBSCAN, which automatically estimates the number of clusters. The resulting model is named Hierarchical and Relativistic Wasserstein Autoencoder GAN (HRWAE-GAN) and this work will show that utilizing multiple latent spaces does **not** offer any advantages when it comes to the task of clustering image datasets.

Keywords

Deep Clustering; Deep Learning; Generative Models; Hierarchical Models; Adversarial models; Relativistic discriminators.

Resumo

O clustering (agrupamento de dados) é uma das áreas mais importantes dentro do mundo da aprendizagem não supervisionada, e os métodos tradicionais (como o k-means) têm dificuldades em lidar com dados de alta dimensionalidade. Por esta razão, muitos investigadores têm utilizado (com sucesso) técnicas de aprendizagem profunda (deep learning) para mapear os dados para um espaço de dimensionalidade reduzida, onde será mais fácil aplicar algoritmos de clustering. Normalmente, o processo consiste em usar um modelo baseado na arquitetura de autoencoders para encontrar um espaço latente de dimensionalidade reduzida e, de seguida, aplicar-lhe um algoritmo tradicional de clustering.

Nesta dissertação criei um método inovador de deep clustering que inclui um autoencoder generativo hierárquico que contém vários grupos latentes (um autoencoder simples tem apenas um) que são regularizados por discriminadores relativistas num cenário de treino adversarial. O algoritmo de densidade HDBSCAN, que estima automaticamente o número de clusters, é depois aplicado em cada um dos espaços latentes. O modelo resultante chama-se Hierarchical and Relativistic Wasserstein Autoencoder GAN (Autoencoder GAN de Wasserstein hierárquico e relativístico), ou HRWAE-GAN, e este trabalho irá mostrar que a utilização de múltiplos espaços latentes não traz vantagens à tarefa de clustering em datasets de imagens.

Palavras Chave

Clustering profundo; Aprendizagem profunda; Modelos generativos; Modelos hierárquicos; Modelos adversariais; Discriminadores relativísticos.

Contents

List of Figures

List of Tables

1

Introduction

Contents

1.1 Motivation

Clustering data is one of the fundamental problems in the unsupervised learning realm. Its goal is to identify data points that are similar to each other in an unsupervised manner (the data is unlabeled) based on some distance/similarity measure. A large number of clustering algorithms have been proposed [Xu and Wunsch, 2005] [Bishop, 2006], but more traditional methods struggle when dealing with high-dimensional data due its underlying complexity [Xu and Wunsch, 2005].

One way to tackle this issue is to reduce the dimensionality of the data, mapping it to a new and more clustering friendly latent space. This is where deep learning techniques (deep clustering, in this case) have come into play. Although some non deep learning methods have been created, such as PCA [Martinez and Kak, 2001] and UMAP [McInnes et al., 2018], deep clustering has proved to greatly improve clustering performance and can even be combined with these aforementioned methods [McConville et al., 2021]. Several deep clustering algorithms have been proposed [Litjens et al., 2017], and they usually consist of two parts: a latent representation finding (often Autoencoder based) and a clustering objective (where traditional clustering is commonly used). Their optimization can either be done jointly or separately.

Following this line of though, I will take on a novel approach to deep clustering using a hierarchical architecture with adversarial training. In deep learning autoencoders, a hierarchical architecture differs from a non-hierarchical one because it can include hierarchical groups between every pair of encoder-decoder hidden layers. In theory, this allows the model to capture both low level and high level details about the data at the different latent groups, thus holding a possible advantage over non-hierarchical models. The adversarial training is included to give the model generative abilities. It utilizes the well known GAN framework [Goodfellow et al., 2014], which turns the training process into a competition between the generator (the model) and a Discriminator, who tries to tell generated samples apart from real samples. This results in the model being able to generate realistic looking samples.

1.2 Goals

The goal of this thesis is to build a generative deep clustering model using state of the art techniques. It's important to mention that this project is a continuation of Pedro Távora Santos' master thesis, who used a mixture of experts (MoE) framework to tackle image clustering problems. On this work, I will develop an alternative framework that, unlike MoE, does not need a pre-training phase. This is discussed in more detail in section 3.

What is most important, however, is to explore an unprecedented path and evaluate how it compares to the current state-of-the-art deep clustering models.

1.3 Outline

This thesis consists of three main sections - Background & related work, Proposed method and Experiments. In section 2 (Background & related work), traditional clustering and deep learning methodologies will be reviewed.Them, we'll move on to explore variational and deep clustering frameworks. In section 3 (Proposed method), the attained model will be thoroughly described and illustrated. In section 4 (Experiments), all of the experiments that were conducted in order to both analyze and optimize the model will be reported.

2

Background & related work

Contents

Clustering is a form of unsupervised learning which attempts to cluster unlabeled data according to some criteria. The general goal is to achieve internal homogeneity (data points in the same cluster should be similar) and external separation (data points in different clusters should **not** be similar). In **hard clustering** algorithms, each data point belongs to only one cluster, while with **soft clustering** they can have a degree of membership to different clusters.

Many clustering techniques have been invented in the last few decades [Xu and Wunsch, 2005] [Bishop, 2006]. In this section, some of them will be briefly described and compared.

2.1 Traditional Clustering

Since clustering is based on the distance between data points, it's important to do a quick overview of distance itself. By definition, a distance $d(x, y)$ meets the following criteria:

- It's positive, so $d(x, y) \geq 0$;
- It's equal to 0 when $y = x$;
- It's symmetric, so $d(x, y) = d(y, x)$
- It meets the triangle inequality $d(x, y) \leq d(x, z) + d(z, y)$

For numeric data, the most common distance metrics are the Euclidean and the Manhattan distances, which are specific cases of the more general Minkowski distance. The latter can be defined as:

$$d(x, y) = \sqrt[q]{|x_1 - y_1|^q + ... + |x_d - y_d|^q}$$

where $x = (x_1, ..., x_d)$ and $y = (y_1, .., y_d)$ represent d-dimensional data points. The Euclidean and Manhattan distances are specific cases where $q = 2$ and $q = 1$, respectively.

For categorical data, one can use the Hamming distance, which returns the number of different feature values between two data points.

Moving on to actual clustering techniques, only the more commonly used approaches will be discussed: Partitioning, Hierarchical, Model-based and density-based approaches.

2.1.1 Partitioning Approaches

Partitioning approaches create partitions which are then iteratively updated. The most well known case of partitioning clustering is k-means, which will be described next.

A – k-means is a very simple squared error based algorithm. It tries to minimize the following metric:

$$J(\mathbf{\Gamma}, \mathbf{M}) = \sum_{i=1}^{K} \sum_{j=1}^{N} \gamma_{ij} \|\mathbf{x}_j - \mathbf{m}_i\|^2$$

where $\gamma = \gamma_{ij}$ is a membership matrix. Note that $\sum_{i=1}^{K} \gamma_{ij} = 1 \forall j$ (each data point belongs only to one cluster); $M = [m_1, ...m_K]$ contains the cluster centroids; m_i is a centroid (centroids are the mean of all data points in a cluster); N_i is the number of data points in cluster i and K is the predefined number of clusters.

The k-means algorithm runs as follows:

1. Partition objects into **K** non-empty subsets and calculate centroids $M = [m_1, ...m_K]$. This can be done randomly or based on prior knowledge;
2. Reassign each object to the cluster with the nearest centroid;
3. Recompute the centroids;
4. Go back to step 2 until reassignments don't change or a given number of iterations is reached.

The k-means algorithm is very simple, but it has quite a few problems: it depends on initialization; it's susceptible to noise and outliers (one way to tackle this problem is by using some other metric different than the mean to compute centroids. k-medoids, for instance, uses the median); K is a parameter that has to be defined in advance; it's biased towards globular shapes and it has no guarantee of a global optimum.

2.1.2 Hierarchical Approaches

Hierarchical Approaches organize data in a hierarchical structure using a distance or similarity matrix. The final result is usually shown in a dendogram. Below, I will describe the two main types of Hierarchical clustering: agglomerative and divisive.

A – Divisive clustering starts out with a lone cluster that includes all the data. Then, it iteratively divides this cluster until each cluster is a single data point. The number of clusters can be chosen a posteriori by selecting a cut off level in the dendogram.

An important issue with this method is the time complexity because, for each cluster with N data points, there are $2^{N-1} - 1$ possible two-subset divisions.

B – Agglomerative clustering goes the other way around (making it less complex, although it still runs in cubic time and uses quadratic memory). In a dataset with N data points, it starts out with N clusters and successively merges them until there in only one cluster. It runs as follows:

1. Initialize N singleton clusters and calculate the distance matrix;
2. Find the minimal distance between all clusters; merge those two clusters C_1 and C_2 into $C_{1,2}$;
3. Update the proximity matrix by removing C_1 and C_2, inserting $C_{1,2}$ and calculating its distance to all other clusters;
4. Go back to step 2 until there is only one cluster.

When it comes to finding the distance between a cluster C_x and a merged cluster $C_{1,2}$, these are the most well known approaches:

- Single link, shortest distance between points. This distance is susceptible to noise and outliers but can find different sized and non-globular clusters.
- Complete link, longest distance between points. This distance has the opposite strengths and weaknesses to single link, it's less susceptible to noise but is biased towards globular clusters.

Other distances, such as centroid link and average link, are also commonly used.

Apart from its lack of robustness, hierarchical clustering techniques have another big issue: lack of dynamism. Once a cluster is formed, there is no going back.

2.1.3 Model based approaches

Model based approaches assume the data is generated by a mixture of probability distributions (typically a Gaussian) where data points that are generated by the same probability distribution belong to the same cluster [Xu and Wunsch, 2005] [Bishop, 2006]. These distributions are often gaussian, but can be of other types.

Assuming that there are K clusters and that $\theta = [\theta_1, ..., \theta_K]$ represents the probability distribution's parameters (each θ_i is a parameter vector), the mixture probability density for the whole data set is

$$p(x|\theta) = \sum_{i=1}^{K} p(x|C_i, \theta_i) P(C_i)$$

where $P(C_i)$ is a prior probability for cluster i and $p(x|C_i, \theta_i)$ is the likelihood of that cluster generating the data. Here, the Maximum Likelihood estimation (finding the model that is most likely to have generated the data) is obtained with the EM algorithm (Expectation Maximization). This algorithm generates a series of θ parameters until convergence like so:

1. initialize θ^0;
2. e-step: assign points to clusters by computing the estimation of the data log-likelihood, i.e., computing the log-likelihood of each data point having been generated by each cluster;
3. m-step: adjust the parameters, maximizing the previous log-likelihood;
4. go back to step 2 until convergence of θ

The main disadvantages of this algorithm are its dependence on this initial guess for θ^0, the need to define K in advance, its slow convergence and the possibility of converging to a local optimum.

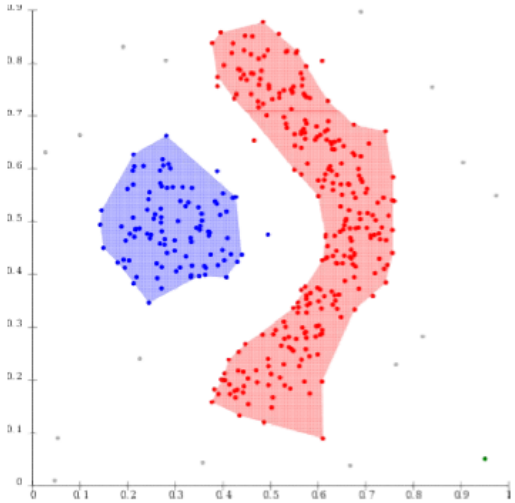


Figure 2.1: DBSCAN example from [Bishop, 2006]. Blue and Red areas identify the clusters.

2.1.4 Density based approaches

Density based approaches see clusters as areas of higher density [Bishop, 2006]. I will describe two of these approaches: DBSCAN and the more recent HDBSCAN.

A – **DBSCAN** [Ester et al., 1996], Density Based Spatial Clustering of Applications with Noise, is a popular density based clustering algorithm. It uses a minimum density level estimation, which is based on a minimum number of neighbors within a radius ϵ threshold (minPts), to identify areas with high density of objects. If a data point satisfies the minPts threshold, it is considered a core point and both him and his neighbors (border points) will belong to a cluster. If one of these neighbors is also found to be a core point, then his neighborhood will be added to the cluster by transitivity, and so on. It's worth noting that data points in more sparse areas (neither core or border points) are considered noise and won't belong to any cluster. Intuitively, this makes sense because these points will have high degrees of uncertainty associated to them.

B – **HDBSCAN** [Campello et al., 2013] [Campello et al., 2015] is a more recent hierarchical approach to DBSCAN. It performs DBSCAN with a range of decreasing ϵ values and integrates the result to find the clusters with the most persistence, i.e., the clusters that take longer to be broken apart (with the constraint that when you select one cluster, you cannot select any of their children). In other words, we are trying to maximize the sum of the lengths of the orange marked branches in figure 2.2.
Its two main advantages over DBSCAN are the fact that it allows to find clusters with different densities and it's less dependent on parameter selection. Although the original HDBSCAN ran in $O(n^2)$, it has since then been tweaked and can now run in $O(nlog(n))$.

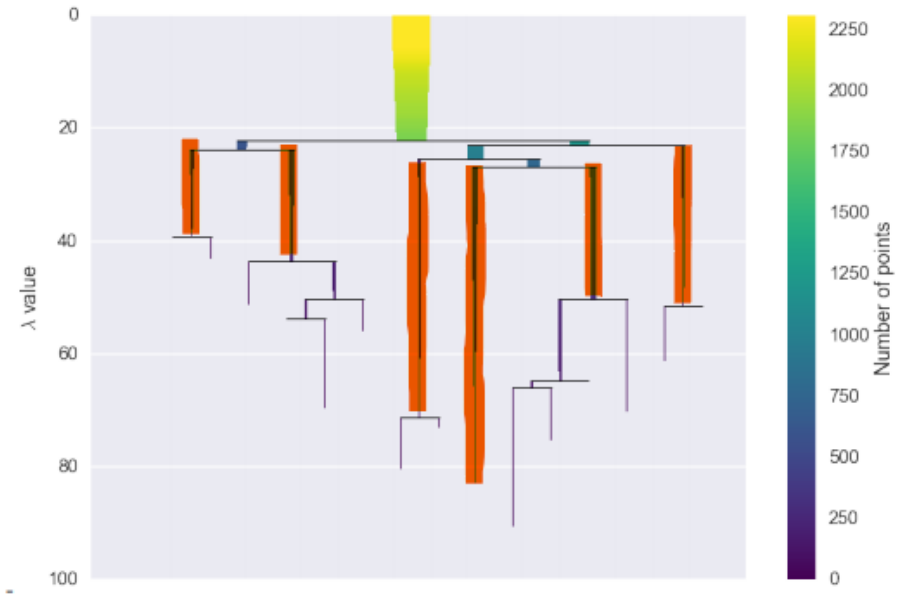


Figure 2.2: HDBSCAN dendogram. Highlighted branches correspond to the desired clusters.

One of the main strengths of density based clustering is the non necessity to specify the number of clusters K in advance. It also has the big advantage of noise identification, not needing to include every data point in a cluster. However, their main weakness is their inability to effectively cluster highly dimensional data. This problem can be addressed with dimensionality reduction, which will be discussed in more detail later on.

2.2 Deep Learning based clustering

As said before, traditional clustering techniques struggle as the dimensionality of the data increases. This struggle is known as the curse of dimensionality. Deep Clustering models solve this issue by mapping some original dataset to a latent space, which usually has a much lower dimensionality, with deep learning networks. This enables classical clustering algorithms (such as density based clustering algorithms) to cluster more effectively. In the next section, we will begin with a quick deep learning overview and then move on to several deep learning models that can be used for deep clustering.

2.2.1 Deep Learning overview

Supervised Learning methods, generally speaking, take a dataset $D = \{x, y\}_{n=1}^{N}$ where x is a data point and y is its label, and try to find a set of parameters Θ that can accurately predict the data points' labels by minimizing a loss function $L(y, \hat{y})$, where $\hat{y}$ is the label predicted by the model [Litjens et al.,

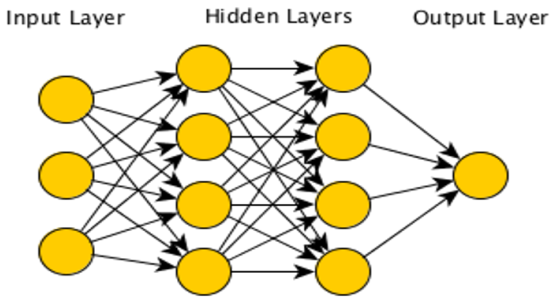


Figure 2.3: MLP architecture https://media.condenast.io/photos/59f25bf95443e85a6c66c865/master/w_768/unnamed0.png

2017].

On this section, we will solely focus on describing **deep learning** methods, which have a greater ability to manipulate the original space before predicting $\hat{y}$.

A – Neural Networks are composed by multiple layers of sets of neurons. Each neuron has a set of parameters $\theta = \{W, b\}$, where W is a weight vector and b is a bias, and an activation function a. The output of a neuron is computed as

$$output = a(W^T x + b)$$

These neurons are combined to produce a final output $\hat{y}$. Multiple layer perceptrons (MLP), the most common neural networks, have several of these multiple neuron layers (hence **deep** learning).

When y is not binary, the probability for each class can be found using a cross-entropy loss, which assesses the difference between the predicted probabilities and a one hot vector with the real label.

Roughly speaking, MLPs minimize the cost function $L(y, \hat{y})$ by computing its gradient relative to the parameters Θ. The parameters of the last layer are updated as such [Bishop, 2006]:

$$W^l = W^l - \eta \frac{\partial L}{\partial W^l}$$

Where l represents the index of the last layer and η is the learning rate. The same update is applied to the Bias. Then, the **chain rule** is used to successively update the parameters of the other layers, going from the last to the first. This backwards directed algorithm is called the Backpropagation algorithm.

B – CNNs have a similar architecture to MLPs but are more appropriate when the dataset consists of image objects.

CNNs, generally speaking, have two key operations: convolutions and poolings. A CNNs has a set of filters, or kernels, per layer (each filter of size k, smaller than the image) which are used to perform the convolutions across the input. An important aspect is that every filter is used across the whole image.

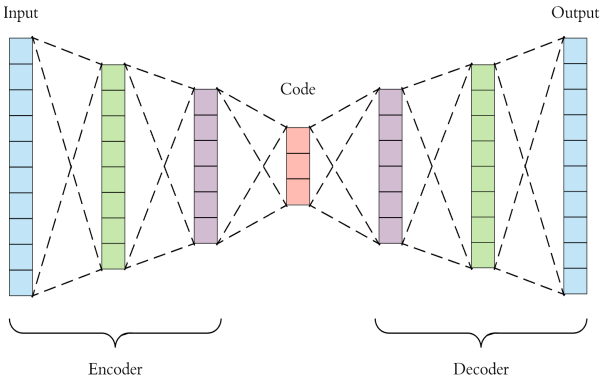


Figure 2.4: AE architecture. https://miro.medium.com/max/3148/1*44eDEuZBEsmG_TCAKRI3Kw@2x.png

This way, the model doesn't need to learn separate detectors for the same object occurring at different positions in the image [Litjens et al., 2017]. It also decreases the number of parameters, since it will not depend on the input size.

At each layer, the result of the convolution operations (plus a bias b) are subjected to an element-wise non linear transform, producing a new feature map X_k. This is when the pooling comes into play. This operation aggregates neighborhood pixel values of X_k, thus reducing its size by some factor (commonly to half its original size). Some of the most common pooling operations include max pooling (result is the max of the neighborhood) and average pooling.

At the end of the network, it's typical to flatten out the current X_k and pass it through fully connected layers to get a prediction of the input image's label.

2.2.2 Models for Deep Clustering

Deep learning models for dimensionality reduction try to learn a latent representation z, with dimensionality d_{latent}, of the input X, with dimensionality $d_{original}$ ($d_{latent} << d_{original}$). Then, they attempt to reconstruct z back to X. The loss function (called the Reconstruction Loss) assesses the difference between X and the reconstruction $\hat{X}$, so, ideally, $\hat{X} = X$. In this section, several variations of dimensionality reduction models will be analyzed. The Autoencoder is the simplest and will be described first.

A – Auntoencorders (AE) are a simple form of unsupervised deep learning techniques [Litjens et al., 2017]. They are formed by two components: The enconder, which is responsible for learning a (usually) dimensionality reduced latent representation z of the original data, and the decoder, which learns to reconstruct z back to its original representation (see figure 2.4).

A key aspect is that the decoder **does not** ever see X - it must be able to reconstruct it only by looking at z.

B – Variational Autoencoders (VAE) can be thought of as probabilistic and generative Autoencoders.

Instead of outputting single point z, the encoder outputs the parameters for a distribution. In the case of a normal distribution, the output is a mean μ and a covariance matrix σ. The Decoder will then sample from the distribution $\mathcal{N}(\mu, \sigma)$, and attempt to reconstruct the input.

The Decoder can be defined as $p(x|z)$, where x is the input and z is the sampled latent variable. This expression describes the distribution of the decoded reconstruction given the encoded variable. Similarly, the Encoder can be defined as $p(z|x)$, which describes the distribution of the latent variable given the input. Note that we're assuming z follows a distribution $p(z)$.

Bayesian theory tells us that we can compute $p(z|x)$ as

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|u)p(u)du}$$

but the computation of $p(x)$ is often intractable, so $p(z|x)$ cannot be computed directly. Instead, we can use **variational inference** to approximate $p(z|x)$. This approximation is denoted $q_x(z)$.

This approximation equates to maximizing the likelihood $p(x|z)$ (1) and minimizing $KL(q_x(z), p(z))$ (2), where the second term denotes the KL divergence between $q_x(z)$ and $p(z)$. In other words, the goal is to minimize

1. The reconstruction loss $RL(x, \hat{x})$.
2. The difference between the distributions of our encoded variable z and the prior $p(z)$. When $p(z)$ is normal distribution (with μ and σ respectively being a 0 vector and an identity matrix), we are trying to force our distribution $q_x(z)$ to be as close to that as possible.

The point of having this probabilistic framework is to regularize our Variational Autoencoder's latent space, enabling it to be generative. The regular Autoencoder's latent space is irregular, since the decoder doesn't have to know how to extract information from all areas, but of specific areas where the encoded variables lie. On the contrary, the variational Decoder has to be able to extract information from larger areas, since the encoded variables are sampled from distributions and are not single points. For this reason, the decoder can sample points from $p(z)$ and generate realistically looking data.

The Beta-VAE [Higgins et al., 2016] uses a hyperparameter β that balances the importance given to the KL term, resulting in the minimization of:

$$L = RL(x, \hat{x}) + \beta KL(q_x(z), p(z))$$

In the experiments section, Beta-VAEs are used instead of the regular VAE.

C – Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] will be introduced as they are relevant for understanding the next type of model.

GANs are game theory based generative models. They use the concept of **adversarial training** to minimize a minimax loss function. As the name indicates, there are two adversaries: the Generator (G) and the Discriminator (D). Succinctly, G picks up a randomly generated variable z that is mapped to the dataset space and tries to generate a realistic data object $G(z)$. Then, both real and generated data objects are fed to the discriminator D, who tries to correctly identify them as real or fake. This game proceeds as G tries to fool D and D tries to avoid that from happening. This can be expressed as:

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

The first half of the equation can be interpreted as the discriminator trying to maximize its ability to identify real data. The second part can be interpreted as the discriminator trying to maximize its ability to classify data generated by G as fake, while G tries to minimize it.

GANs gained a lot of recognition because they are capable of generating very realistic data, more so than VAEs. Their training is, however, much more unstable.

D – Relativistic GANs (RGANs) [Jolicoeur-Martineau, 2018] makes an interesting change in the GAN's discriminator. Instead of trying to determine if a certain input is real or fake, the new relative discriminator takes a pair of real and fake data points (x_r, x_f) and measures the probability p that the real data point is more realistic than the fake one. This way, as the generated data becomes more realistic, p will decrease with the ultimate goal being $p = 0.5$ (fake data is as realistic as real data). RGANs have the interesting properties of providing a more stable training and having the ability to learn with smaller datasets.

E – InfoGANs [Chen et al., 2016] are information theory inspired GANs. The goal of the InfoGAN is to have a generative model that is capable of using additional information to create more specific synthetic data. To do this, the generator takes as input the latent variable z (just like any other generative model) and additional coded information c, so the generator function becomes $G(z, c)$. This information is supposed to have some control of the output (if we are generating numbers, for example, it can be an integer identifying which number we want to generate).

To achieve this, the authors propose to use an auxiliary neural network $Q(G(z, c))$. Q takes as input the generated data and produces $\hat{c}$ - thus its goal is to recover the information c. Theoretically, the idea is to maximize the mutual information between c and $\hat{c}$, but, in practise, the penalty is computed with a cross entropy function. Note that to minimize this penalty, the generator has to produce a disentangled output that reflects c. If not, Q won't be able to recover it.

InfoGAN's optimizing objective can be formulated as:

$$min_G max_D V_I(D,G) = V(D,G) - \lambda I(c; G(z,c))$$

Where $V(D,G)$ is the normal GAN's objective and the second term represent the mutual information goal between c and the generated data $G(z,c)$. Again, this is approximated by:

$$min_{G,Q}max_{D}V_{InfoGAN}(D,G) = V(D,G) - \lambda L_1(G,Q)$$

F – Wasserstein Autoencoders (WAEs) [Tolstikhin et al., 2017] are a variation of VAEs that try to improve the regularization of the latent space, thus allowing a better reconstruction of z . Figure 2.5 illustrates this and is taken from WAE's original paper. While VAEs individually enforce $q_x(z)$ to match $p(z)$ on all points, WAEs enforce a continuous mixture $q := \int q_x(z)dP_x$ to match $p(z)$ across the whole latent space. This allows latent representations of different inputs to stay further away from each other (more **separation** while still enforcing the prior $p(z)$), promoting better reconstruction.

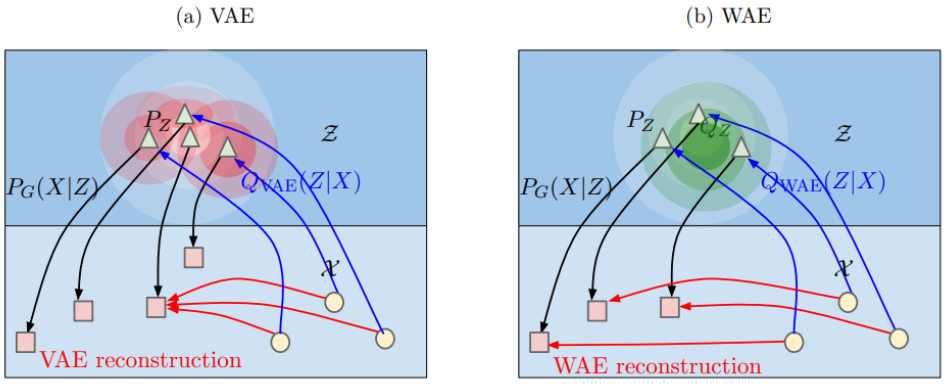


Figure 2.5: Comparison between the VAE's and the WAE's enforcing of the prior $p(z)$ [Tolstikhin et al., 2017]

WAE's paper proposes two different formulations: a Generative Adversarial Network based WAE (WAE-GAN) and a Maximum Mean Discrepancy based WAE (WAE-MMD).

WAE-MMD uses a positive-definite reproducing kernel $k : \mathscr{Z} \times \mathscr{Z} \rightarrow \mathscr{R}$ to calculate the maximum mean discrepancy:

$$MMD_k(P_Z, Q_Z) = || \int_{\mathscr{Z}} k(z,\cdot)dP_Z(z) - \int_{\mathscr{Z}} k(z,\cdot)dQ_Z(z)||_{H(k)}$$

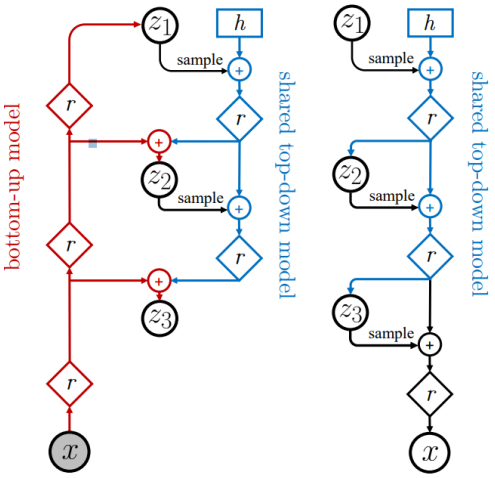
where $H(k)$ is the reproducing kernel Hilbert space real valued function mapping $\mathscr{Z}$ to $\mathscr{R}$. It serves as a regularizer.

WAE-GAN is a combination of a VAE and a GAN. It moves the GAN's discriminator to the VAE's latent space. There, the discriminator is trained to distinguish samples from the prior distribution $p(z)$ and from the encoder's distribution $q_x(z)$.

WAE-GAN is able to achieve better results and has a better time complexity. However, since it is GAN based, its training is not as stable as WAE-MMD's, which is as stable as the original VAE.

G – Nouveau Variational Autoencoder [Vahdat and Kautz, 2020] (NVAE) is, essentially, a VAE with a hierarchically structured latent space. Figure 2.6 is taken from the original paper and shows the architecture of NVAE. Instead of having just one latent representation z, it computes a series of latent representations $\{z_1, ..., z_n\}$. The authors argue that this method allows NVAE to better capture the global structure at the top of the hierarchy (z_n) and local structure at the bottom (z_1). When dealing with pictures of faces, for example, the top embeddings would capture things like skin tone and overall symmetry, while the lower embeddings would capture more low-level details.

Figure 2.6: NVAE's architecture. The r blocks are residual networks and h is a learnable parameter. [Vahdat and Kautz, 2020]



(a) Bidirectional Encoder (b) Generative Model

It could be interesting to perform clustering on the different levels $\{z_1, ..., z_n\}$ and identify which would yield the best results.

Here are some key aspects of the implementation of NVAE:

- Both the encoder and the decoder use deep residual networks (one per hierarchical layer);
- Enlarged kernel sizes and depth wise separable convolutions (more convolutional layers with less filters each);
- Modified batch normalization that tackles the noise caused by regular BN;
- Squeeze and Excitation layers in each residual network.

All of these techniques make small contributions towards improving the performance of NVAE.

2.3 Deep clustering

Deep clustering can be split into several categories. In this section, we'll discuss **Autoencoder**, **VAE** and **GAN** based clustering. Most of the models presented in this section can be found in [Litjens et al., 2017]. Although some work has been done with semi-supervised learning [Lim et al., 2020] (using the known labels to forcibly reorganize the latent space), I will only focus on fully unsupervised approaches.

A – Autoencoder based clustering was first proposed in [Xie et al., 2016], where they formulated the architecture for **Deep Embedding Clustering** (DEC). DEC is widely considered as a deep clustering baseline and has been improved upon by many other researchers.

The main idea of AE based clustering is to perform clustering on the AE's latent space, optimizing

both the network's loss as well as a clustering loss $L = \lambda L_n + (1 - \lambda)L_c$ where $\lambda \in [0, 1]$ is a hyperparameter to balance the two losses.

DEC couples the standard AE loss with a KL divergence based loss that tries to minimize the distance between the soft assignments Q and a target distribution P (that depends on Q). The soft assignments can be defined as follows:

$$q_{ij} = \frac{(1+||z_i-\mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1+||z_i-\mu'_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}$$

where z_i is the embedding of an input x_i, α are the degrees of a Student's t-distribution and q_{ij} is the probability of a data point i belonging to cluster j. In this model, the training process minimizes L_n and L_c in two separate phases. **DCN** [Yang et al., 2017] came shortly after claiming that minimizing the two losses simultaneously would be beneficial.

Deep Multi-Manifold Clustering (DMC) [Chen et al., 2017] is a multi-manifold clustering model where L_n is a locality preserving loss function. DMC considers that each cluster lies on a different manifold. Consequently, the loss function includes both a regular AE reconstruction loss and a locality preserving loss. It's considered to be a state of the art multi-manifold clustering approach.

Deep Embedding Regularized Clustering (DEPICT) [Ghasedi Dizaji et al., 2017] has a similar approach to DEC but makes some improvements. Like DCN, it also performs a joint optimization of L_n and L_c. Instead of utilizing just one regular encoder, DEPICT adds a noisy encoder to the architecture. It also uses two softmax layers, one for each of the encoders' latent spaces. This layer acts as a predictor to perform soft cluster assignment, but only the noisy encoder's layer is used to compute L_c. The regular encoder is used to compute L_n and its weights are tied to the decoder's. According to the authors, this technique improves reconstruction capability and latent space interpretation.

Deep Continuous Clustering (DCC) [Shah and Koltun, 2018] also uses joint optimization and is based on Robust Continuous Clustering, a clustering formulation with a continuous objective and no need to specify the number of clusters a priori. This is a clear benefit when compared to the previous approaches.

N2D [McConville et al., 2021] uses a different approach. Instead of a joint optimization, it learns the embedded representations and, separately, performs a shallow clustering technique. Before starting the clustering phase, N2D uses UMAP to obtain a more clustering friendly latent space (keeping the same latent dimensionality). The authors claim that, after having tried different techniques, they found UMAP is able to find the more clusterable manifold. As discussed before, UMAP has the interesting property of providing well separated and compact clusters.

One clear advantage for N2D is the low level of complexity, since there is no joint optimization and UMAP is a fast algorithm.

MIXAE [Zhang et al., 2017], Deep Unsupervised Clustering Using Mixture Of Experts, assumes that each cluster is sampled from a different manifold and consists of two parts: a set of autoencoders (the

experts), where each one learns the representation of a different manifold, and a Mixture Assigning Network (MAN) that takes the latent representations formulated by the experts $[z_1, ..., z_K]$ and outputs a set of probabilities $P = [p_1, ..., p_K]$ that are used for soft clustering assignment, so $p_k = Pr(x \in c_k|z)$ where c_k represents the k^{th} cluster.

MIXAE performs a joint optimization of the experts and the MAN. The goal is for the Ps to be one-hot vectors, with the value 1 corresponding to the expert with minimum reconstruction loss.

In order to achieve one-hot vector like Ps, a sample-wise entropy penalty (which will approximate 0 as P approximates a one-hot vector) is added to the cost function. Additionally, to motivate the equal usage of all experts, the authors also use a batch-wise entropy term that should be maximized.

The downsides of this model are the need to specify the number of experts in advance, a somewhat high complexity (which increases with the number of experts needed) and performance that is not state of the art.

SPICE [Niu and Wang, 2021], a Semantic Pseudo-labeling Framework for Image Clustering, utilizes self-learned pseudo-labels of high confidence in the embedded space to train a deep-clustering network. Initially, a latent space representation is constructed by a convolutional autoencoder SPICE-self. Based on this initial clustering, a set of reliable data points (a reliable data point will be assigned to its cluster with high certainty by some fuzzy clustering algorithm) are used to train a second network SPICE-semi. This network is trained directly with a loss function derived from the self-learned pseudo-labels. Consequently, SPICE is a semi-supervised classification model trained with pseudo-labels obtained from representation learning.

SPICE is a state-of-the-art model and currently ranks number one in unsupervised classification of several colored image datasets such as CIFAR-10, ImageNet-10 and Tiny-ImageNet.

SPC [Mahon and Lukasiewicz, 2021], which stands for Selective Pseudo-label clustering, is another semi-supervised learning deep clustering model. The general idea is similar to SPICE, but the two differ in what they consider to be confident pseudo labels and in the overall training procedure. While SPICE uses a confidence threshold, SPC pre trains a group of K autoencoders and then only uses the agreed points, which are points that received the same label from the clustering algorithm (HDBSCAN or GMM) applied to all K autoencoders' latent space, for classification training. An MLP is then trained with these agreed points for classification. This autoencoder and MLP training cycle is repeated until the number of agreed points stops increasing.

SPC is the current state-of-the-art model for unsupervised classification of the black and white numbers image datasets MNIST and USPS.

DynAE [Mrabah et al., 2020] utilizes the novel concept of a dynamic autoencoder, which is essentially a regular autoencoder with a dynamic loss function, in a self-learned semi-supervised setting. More specifically, this dynamism consists of steadily reducing the importance of the reconstruction objective

in favor of the clustering one. The authors argue that this method increases the model's ability to better and more carefully integrate the uncertain knowledge acquired during training.

Like the two previous works SPICE and SPC, DynAE's training also starts with a pre-training phase. A vanilla autoencoder is trained with augmented data [Guo et al., 2018] (some of the images are rotated and shifted) and adversarially constrained interpolation [Berthelot et al., 2018] (similar to regular adversarial training except the fake data points fed to the discriminator are latent interpolations of two randomly sampled data points x_1 and x_2, which has the effect of regularizing the latent space by incentivizing interpolations to look realistic).

After the pre-training is complete, the dynamic loss function is introduced to optimize cluster and centroid construction.

B – VAE based clustering uses the VAE framework, as the name suggests. For this reason, the models described in this section are **generative**.

Variational Deep Embedding (VaDE) [Jiang et al., 2016] couples a VAE with a gaussian mixture model as a prior (GMM) instead of a single gaussian. The authors argue that, since GMMs assume the data comes from a Mixture of Gaussians (MoG), they are more suitable for clustering tasks.

The GMM also has the advantage that its parameters can be optimized with a EM type algorithm, so this is a more flexible approach when compared to using a single gaussian with fixed parameters.

DGG [Yang et al., 2019], Deep Clustering by Gaussian Mixture Variational Autoencoders with Graph Embedding, uses a VAE with a GMM prior and pairs it with graph embedding. This model combines model-based (GMM) and similarity-based (graph embedding) approaches with the goal of achieving good local (with the GMM) and global (with graph embedding) structure.

Since GMMs are probabilistic and graph embeddings are not, the authors propose a stochastic extension of graph embedding where data points are seen as nodes of a sample similarity graph. The idea behind graph embedding is to build a graph where connected data points should be similar, and thus should have similar latent representations and cluster assignments. The optimal latent features are found by:

$$\{z_n^*\} = argmin_{ZZ^T=I} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} ||z_i - z_j||_2^2$$

Where $Z = [z_1, ..., z_n]$ and w_{ij} is the (i, j) entry of the similarity matrix W. It's required that $\sum_j w_{i,j} = 1$ In this framework, similarity is measured with the Jenson-Shannon divergence.

C – GAN based clustering uses the GAN's adversarial training framework in a few different ways.

Deep Adversarial Clustering (DAC) [Harchaoui et al., 2017] uses an adversarial Autoencoder with a discriminator in the latent space to enforce a GMM prior. DAC has a similar architecture to WAE-GANs, except it is clustering oriented and uses an AE instead of a VAE.

Mixture of Experts Deep Clustering (MoEDC) is the product of Pedro Távora Santos' master thesis. As mentioned before, this project is a continuation of (or an alternative to) Pedro's work. Hence, a brief description of his model follows.

MoEDC has two main components, which are the manager and the experts. The manager is pre-trained during the pre-training stage (see figure 2.7), while the experts are only trained afterwards.

The pre-training stage starts with a PCA based algorithm called the Z-finder finding the optimal latent dimensionality for a given dataset. Then, a VAE is trained with the Z-finder's latent dimension and UMAP is applied to the resulting latent space. In succession, HDBSCAN is used to cluster the manifold outputted by UMAP. Finally, the VAE is retrained without the noisy samples (recall that HDBSCAN identifies noise).

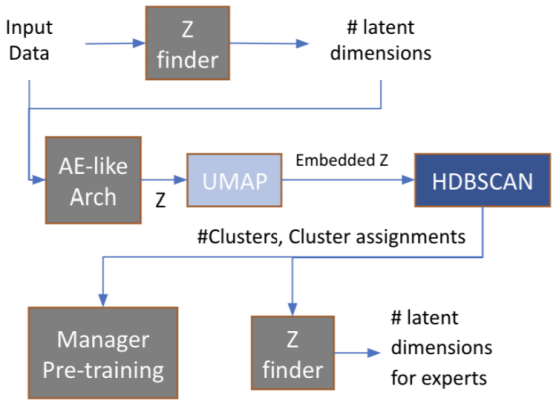


Figure 2.7: Pre-training scheme taken from MoEDC's arcticle.

After the pre-training is completed, the manager is capable of assigning samples to the different experts, who will each specialize in reproducing a more refined latent space of a specific cluster.

2.4 Evaluation Metrics

2.4.1 Clustering Metrics

In this section, I'll provide a brief overview of the most common clustering metrics.

Generally, clustering metrics can be divided into two categories: supervised (or external) and unsupervised (or internal).

Supervised metrics can only be used when the data is labeled, which is often not the case. One disadvantage of these metrics is that our clustering solution may be able to find meaningful clusters and still perform badly because they do not necessarily reflect the data's labeling.

Rand Index (RI) [Rand, 1971] measures the similarity between two assignments. When applied

to clustering, it measures how good the cluster assignment is by comparing the true clusters C_t (data points with the same label are in the same cluster) with the predicted clusters C_p.

$$RI = \frac{TP+TN}{TP+FP+FN+TN}$$

where TP are the true positives (number of pairs of data points that are in the same cluster in C_p and in C_t), TN are the true negatives (number of pairs that are in different clusters in C_p and in C_t), FP are the false positives (number of pairs that are in the same cluster in C_p but in different clusters in C_t), and FN are the false negatives (number of pairs that are in different clusters in C_p but in the same cluster in C_t).

Adjusted Rand Index (ARI) is the corrected-for-chance version of RI, meaning that it takes into account the expected RI for a random model.

Clustering Accuracy (ACC) [Cai et al., 2010] is similar to random index, as it measures the percentage of data points that can be mapped to their correct class by using their cluster assignments.

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

where y_{pred} are the predicted labels, y_{true} are the ground truth labels and T is the function that maximizes the mapping between cluster assignments and ground truth labels.

Purity assesses each cluster's purity by computing what percentage of the cluster is composed by the majority class of that cluster.

$$purity(C) = \frac{1}{N}\sum_{k} max_i|c_k \cap l_i|$$

where l_i is the majority class of cluster c_k. A problem with this measure is that its value is maximized when $K = N$.

Normalized Mutual Information (NMI) [Strehl and Ghosh, 2002] is an information-theoretic measure. It assesses the quality of a clustering solution by computing the mutual information between y_{pred} and y_{real} and then normalizes it with the their individual entropies.

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

Unsupervised metrics do not use the ground truth labels. The big disadvantage with these metrics is that it is very hard to understand if the clusters found have any real meaning. When there is no overall idea of how the algorithm should cluster, it's hard to interpret the results in an intuitive way. Generally speaking, unsupervised metrics are based on two measures: cohesion and separation.

Cohesion assesses how close points in the same cluster are. In a way, it can be thought of as the density of the cluster (in a denser cluster, points are more similar to each other on average).

Separation assesses how different points in different clusters are. In a good clustering solution, different clusters should contain dissimilar data.

Silhouette coefficient combines both cohesion and separation. It's calculated for each data point x_i and is given by:

$$s(x_i) = 1 - \frac{a}{b}$$

where a is the average distance between x_i and the other points in its cluster, and b is the average distance between x_i and the points in all other clusters. If $a > b$, then b is divided by a, but that's not usually the case. Silhouette ranges from -1 to 1 (closer to 1 is better).

2.4.2 Image quality metrics

Image quality metrics can be very useful to assess the quality of image reconstruction and generation, which will be necessary in section 3. There are two main types of image quality metrics: subjective and objective methods [Thung and Raveendran, 2009].

Subjective methods use humans to assess the quality of images by rating them or by trying to distinguish real from fake images [Chen et al., 2005]. Objective methods assess image quality using numerical criteria and can be split into full-reference (FR), reduced-reference (RF), and no-reference (NR), depending on the availability of reference images [Thung and Raveendran, 2009]. The focus will be solely on FR methods because model training will only occur in such conditions. FR metrics assess the similarity between generated images and real target images.

Since subjective methods are not even an option, I will exclusively resort to FR methods. FR options include SSIM [Hore and Ziou, 2010], PSNR [Hore and Ziou, 2010] FID [Heusel et al., 2017], and others [Thung and Raveendran, 2009]. For simplicity's sake, only the more recent FID will be described.

FID [Heusel et al., 2017], or Frechet Inception Distance, evaluates the similarity between generated and real images and it was proposed to assess GAN image generation quality. It uses the previously conceived inception v3 model [Szegedy et al., 2016] to calculate the difference in statistics of computer vision features between a real and a fake dataset. The goal is to achieve a score as close to 0 as possible (the FID score between two identical datasets is 0).

FID can be formulated as:

$$FID = ||\mu_{real} - \mu_{fake}||^2 + Trace(\sigma_{real} + \sigma_{fake} - 2\sqrt{\sigma_{real}\sigma_{fake}})$$

where μ and σ refer to the mean and variance of the real and fake datasets.

2.5 Data Visualization Techniques

Data visualization is extremely important for model interpretation. When creating a latent space, for example, it's useful to visualize it in two dimensions in order to not only understand what the model is doing, but also to visually assess the separation of data points from different classes. Below, three data visualization methods will be construed: PCA [Pearson, 1901], t-SNE [Van der Maaten and Hinton, 2008] and UMAP [McInnes et al., 2018].

A – **PCA** [Pearson, 1901] is a very well know linear Matrix Factorization technique and it tries to find a new coordinate system that describes most of the variation of the data by computing the eigenvectors of the data's covariance matrix [Bishop, 2006]. The eigenvectors with the larger eigenvalues are called the **principle components**, and the idea is to pick only the most important ones. A key fact is that these principle components are always orthogonal, making PCA less flexible than other methods. However, according to [Martinez and Kak, 2001], PCA can more appropriate when dealing with small datasets.

B – **t-SNE** [Van der Maaten and Hinton, 2008] is a non-linear Neighbor Graph technique and is mostly used for 2D data visualization. The goal, of course, is to maintain the structure of the data in the 2D visualization. It operates as follows:

1. Calculate the similarity between all pairs of points. The similarity between points x_i and x_j is proportional to the probability of generating x_j from gaussian centered in x_i. After doing so, we'll end up with a similarity matrix P.
2. Project the points in a lower dimension (with a random initialization) in such a way that the new similarity matrix Q is as similar to P as possible. Here, a t distribution is used instead of a gaussian to avoid clumping the data in a lone cluster.

In more mathematical terms, it tries to minimize a cost function C:

$$C = \sum_{i}^{N} KL(P_i||Q_i) = \sum_{i} \sum_{j} p_{j|i} log \frac{p_{j|i}}{q_{j|i}}$$

where P_i is the conditional probability distribution over all other data points, given x_i of dimensionality $d_{original}$, and Q_i the conditional probability distribution over all other data points, given y_i of dimensionality $d_{projection}$ ($d_{projection} << d_{original}$, usually $d_{projection} = 2$). $p_{j|i}$ and $q_{j|i}$ represent the similarity of points x_i and x_j for both distributions.

This cost function can be interpreted as t-SNE trying to find a lower dimensional distribution Q that is as close as possible to P, based on the similarity between points.

Although it maps data to a two dimensional space, t-SNE has several issues that can make its projection unreliable [Wattenberg et al., 2016]:

- Its highly dependent its perplexity hyperparameter. Perplexity's value reflects how much importance t-SNE gives to local proximities versus the global structure of the data;
- It has high variance, meaning that running the algorithm twice with the same data may yield different results. This is, at least in part, caused by the random initialization of the lower dimension projection;
- It may change the distance between clusters, their sizes, densities and shape;
- It runs in quadratic time, making its use impractical for large datasets.

Basically, the reduced dimensionality visualization may not be so reliable (and it can take a long time to compute). Still, it can be useful to find structure in the data.

C – UMAP [McInnes et al., 2018] is a more recently developed non-linear manifold Neighbor Graph technique. For it to work, UMAP makes the key assumption that the **data is evenly distributed across the manifold**. Because this is not usually the case, it defines a Riemannian metric on the manifold that makes the assumption true. This can be seen as UMAP defining different euclidean scales on different regions of the manifold (so, depending on where you are, euclidean distances are different). Another important assumption is the notion that the **manifold is**, at least, **locally connected** (there are no isolated data points). This is a safer assumption because that's usually the case for real data.

After building the original dimension nearest neighbor fuzzy graph, UMAP derives a lower dimension one by minimizing a cross entropy cost function:

$$\sum_{x \in X} \mu(x) log(\frac{\mu(x)}{\nu(x)}) + (1 - \mu(x)) log(\frac{1 - \mu(x)}{1 - \nu(x)})$$

where μ and ν are membership functions to the fuzzy sets (X, μ) and (X, ν), which correspond to the original and lower dimension fuzzy sets, respectively. The first part of the function enforces the correctness of the clumps of data (local structure), while the second enforces the correct separation between them (global structure).

UMAP has a few hyperparameters. Because the number of nearest neighbors is unknown in the lower dimension, it's passed onto the algorithm. This number will determine UMAP's concern with local structure versus global structure. Another hyperparameter is the minimum distance between points. When small, clusters will be more compact. If it's too small, however, clusters can collapse into single points. UMAP also has a metric hyperparameter, so it can work with many different distance metrics. The final hyperparameter is the number of components, which defines the desired dimensionality.

The advantages of UMAP in comparison to t-SNE are:

- The usage of the Cross Entropy (instead of KL divergence) allows UMAP to capture the global structure a lot better;
- Its scalable, running a lot faster for larger datasets.

2.5.1 Summary

In this section, I presented and discussed both the fundamentals and the state of the art work that has been done in the deep clustering area, as well as some techniques for evaluating and visualizing results. The section follows this logical order:

1. Presenting the existing traditional clustering methods;
2. Discussing why they struggle with more complex datasets (such as image datasets) with high feature dimensionality;
3. Unveiling several types of deep clustering models that attempt to address this issue through deep learning.

Table 2.1 shows both the accuracy and the Normalized Mutual Information (NMI) on the MNIST dataset for many of the deep clustering models that have been developed in recent years. In this table we can see that there is a considerable heterogeneity when it comes to the type of architecture used, which I believe shows how the problem of deep clustering is quite open. It is also important to notice that many papers don't report exactly how they obtained their results (whether it's an average, median, or the best result obtained). This is certainly unfortunate as it makes it much harder to compare results. Even so, and although what is considered state of the art currently depends on the dataset in question, the best performing models (at the time of the writing of this dissertation) are SPICE [Niu and Wang, 2021], SPC [Mahon and Lukasiewicz, 2021], ASPC-DA [Guo et al., 2019] and DynAE [Mrabah et al., 2020].

Table 2.1: Comparison of state-of-the-art methods based on results achieved on the MNIST dataset. 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, 2015]	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%	-	-
GMVAE [Dilokthanakul et al., 2016]	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%	0.81	-
VaDE [Jiang et al., 2016]	VAE	94.5%	-	best 10 trials
DEPICT [Ghasedi Dizaji et al., 2017]	CAE	96.5%	0.92	avg 5 trials
IDEC [Guo et al., 2017a]	AE	88.1%	0.87	-
DAC [Chang et al., 2017]	CNN	97.8%	0.94	-
DCEC [Guo et al., 2017b]	CAE	89%	0.89	-
MIXAE [Zhang et al., 2017]	AE	85.6%	-	-
LRAE [Chen et al., 2018]	AE	60.7%	0.62	-
CCNN [Hsu and Lin, 2017]	CNN	-	0.88	-
DCC [Shah and Koltun, 2018]	AE	97.4%	-	-
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., 2018a]	CAE	96.4%	0.92	-
ClusterGAN [Mukherjee et al., 2019]	GAN	95%	0.89	best 5 trials
ASPC-DA [Guo et al., 2019]	AE	98.8%	0.94	avg 5 trials
LTVAE [Li et al., 2018b]	VAE	86.3%	0.83	best 10 trials
ClusterGAN [Ghasedi et al., 2019]	VAE	96.4%	0.92	avg 5 trials
[Yang et al., 2019]	AE	97.8%	0.94	avg 10 trials
BAE [Chen and Huang, 2019]	CAE,SAE,AAE	83.7%	0.81	-
N2D [McConville et al., 2021]	AE	97.9%	0.94	-
DAMIC [Chazan et al., 2019]	AE	89%	0.87	avg 5 trials
IIC [Ji et al., 2019]	CNN	98.7%	-	-
DGG [Yang et al., 2019]	VAE	97.6%	-	-
DynAE [Mrabah et al., 2020]	AE	98.7%	0.96	-
MoE-SIM-VAE [Kopf et al., 2019]	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
SPC [Mahon and Lukasiewicz, 2021]	AE	**99%**	**0.97**	avg 5 trials

3

Proposed Method

Contents

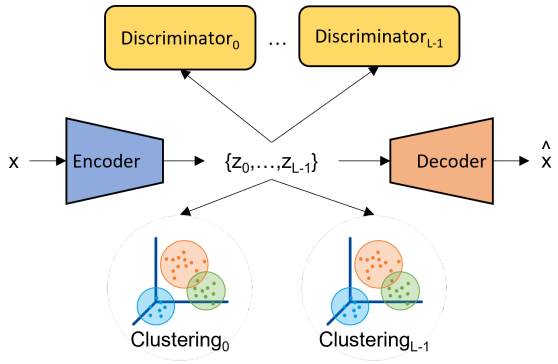


Figure 3.1: Simplified architecture of HRWAE-GAN (see Figure 3.2 for more details). The Encoder produces L latent variables, which are each judged by the discriminator during training and clustered when training is complete.

The Hierarchical and Relativistic Wasserstein Autoencoder GAN is comprised of two essential components: the Hierarchical and Relativistic Wasserstein GAN Autoencoder (HRWAE-GAN) and the clustering box (HDBSCAN paired with UMAP) applied to the latent space. These components will be described in the following sub sections.

3.1 Autoencoder structure

This work can be divided in two big chunks: (1) building a generative framework, where the WAE-GAN [Tolstikhin et al., 2017] was adapted to make its training more stable, and (2) incorporating a hierarchical structure into the model. These chunks will be respectively explained in sections 3.1.1 and 3.1.2.

3.1.1 Relativistic Wasserstein GAN

When comparing the Beta-VAE [Higgins et al., 2016] and WAE-GAN [Tolstikhin et al., 2017] models, the latter has a better reconstruction ability (see section 2.2 and figure 2.5) but a more unstable training (Beta-VAE's training is completely stable). Reconstruction quality is of great importance as it allows us to infer how much information is retained in the latent space (in a fully unsupervised setting, at least), and so it correlates to its quality. The goal thus became to enable the usage of WAE-GAN by making its training more stable.

The training instability of GANs is a well known issue which has been addressed in many different works like [Salimans et al., 2016], [Jolicoeur-Martineau, 2018], [Gulrajani et al., 2017] and [Arjovsky and Bottou, 2017]. While the label smoothing technique [Salimans et al., 2016] is helpful but not completely effective, the Relativistic GAN's modified loss function [Jolicoeur-Martineau, 2018] proved to completely

solve training instability (with regards to the adversarial training) with minimal changes to the model and was therefore adopted (it is only required to make a small change in the adversarial loss function).

First, let us recall Generative Adversarial Networks (GANs) as they were explained in section 2.2: GANs are game theory based generative models. They use the concept of adversarial training to minimize a minimax loss function. As the name indicates, there are two adversaries: the Generator (G) and the Discriminator (D). Succinctly, G picks up a randomly generated variable z that is mapped to the dataset space and tries to generate a realistic data object G(z). Then, both real and generated data objects are fed to the discriminator D, who tries to correctly identify them as real or fake. This game proceeds as G tries to fool D and D tries to avoid that from happening. This can be expressed as:

$$min_G max_D L(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))] \tag{3.1}$$

The first half of the equation can be interpreted as the discriminator trying to maximize its ability to identify real data. The second part can be interpreted as the discriminator trying to maximize its ability to classify data generated by G as fake, while G tries to minimize it.

Wasserstein GANs (WAE-GANs) move the discriminator to the Autoencoder's latent space, where it will be tasked with distinguishing data points sampled from the prior distribution p_z (real points) and from the encoder's approximation $q_x(z)$ (fake points). Consequently, in WAE-GANs, the encoder serves as the GAN's generator. Equation 3.1 can be adapted to this use case as:

$$min_G max_D L(D,G) = E_{z\sim p_z}[logD(z)] + E_{\hat{z}\sim q_x(z)}[log(1-D(\hat{z}))] \tag{3.2}$$

where D is the latent discriminator and G is the encoder.

In [Jolicoeur-Martineau, 2018], the author argues that standard GANs are missing a key property: the probability of real data being real should decrease as the probability of fake data being real increases. This makes use of the *a priori* knowledge that half of the dataset is fake. To fix this issue, a relativistic discriminator (RD) is used. The RD, instead of trying to classify single data points as being real or fake, takes pairs of points (z_{real}, z_{fake}) and outputs the probability of z_{real} being more realistic than z_{fake}. As the game progresses and G becomes more competent, D might even consider fake points to be more realistic than real ones, allowing G to generate samples of higher quality than before.

The loss functions for the RGAN's Discriminator and Generator are, respectively:

$$L_D = E_{(z,\hat{z})\sim(p_z,q_x(z))}[-log(f(D(z)-D(\hat{z})))] \tag{3.3}$$

$$L_G = E_{(z,\hat{z})\sim(p_z,q_x(z))}[-log(f(D(\hat{z})-D(z)))] \tag{3.4}$$

where f is generally the $sigmoid$ function. In equation 3.3, the Discriminator tries to maximize the

difference in realism between real and fake images; in equation 3.4, the Generator tries to maximize the opposite difference, thus trying to generate realistically looking fake samples.

Notice that while in equation 3.2 G only has influence in the rightmost part, in equations 3.3 and 3.4 it has full influence.

Experimentally, it was found that the RD greatly improves the stability of adversarial training. Although it may require some tuning of the Discriminator's VC dimension, stable adversarial training is easily achievable.

3.1.2 Hierarchical structure

One of the main challenges with Autoencoder models is fully capturing both the global structure of data (such as general shape and color) as well as local structure (finer details such as lettering on an image of a shirt). The hierarchical latent structure presented in NVAE [Vahdat and Kautz, 2020] was used to tackle this challenge to great effect. Instead of a single latent variable z, with a hierarchical Autoencoder there is a set of latent variables $\{z_0, ..., z_{L-1}\}$, where z_0 corresponds to the latent variable at the top of the hierarchy, z_{L-1} is positioned at the bottom and L is the number of latent groups. If we consider N to be the number of layers in the model, note that $L \in [1, N]$, so $L \leq N$. In this setting, the latent groups at the top of the hierarchy are supposedly able to capture global structure whilst the bottom ones capture local structure.

Considering this hierarchical structure, the total Discriminator and Generator losses become:

$$L_D = \alpha_0 E_{(z,\hat{z})\sim(p_z,q_x(z))}[-log(f(D(z) - D(\hat{z})))] + \sum_{l=1}^{L-1} \alpha_l E_{(z,\hat{z})\sim(p_{(z_l|z_{<l})},(q_x(z|z_{<l}))}[-log(f(D(z) - D(\hat{z})))] \tag{3.5}$$

$$L_G = \alpha_0 E_{(z,\hat{z})\sim(p_z,q_x(z))}[-log(f(D(\hat{z}) - D(z)))] + \sum_{l=1}^{L-1} \alpha_l E_{(z,\hat{z})\sim(p_{(z_l|z_{<l})},(q_x(z|z_{\leq l}))}[-log(f(D(\hat{z}) - D(z)))] \tag{3.6}$$

where $\{\alpha_0, ..., \alpha_{L-1}\}$ are balancing terms with decreasing value and $q_x(z|z_{<l}) = \prod_{i=0}^{l-1} q_x(z_i|z_{<i})$ are the approximate posteriors the l^{th} group.

Regarding the priors, the top layer prior $p(z_0)$ corresponds to a simple normal distribution $\mathcal{N}(\mu, \sigma)$ with μ being a zero vector and σ being the identity covariance matrix with whatever dimensionality d_{latent} the top latent space may have. For the remaining priors $p(z_l|z_{<l})$ of the lower latent groups, they correspond to normal distributions as well but their parameters are computed by trainable parameter cells.

3.1.3 Dynamic architecture

The generic architecture of HRWAE-GAN can be visualized in Figure 3.2, where each cell is composed of convolutional and deconvolutional layers for the encoder and decoder structures, respectively. However, because images of larger dimensions are usually more complex, there is a need for the model depth to be data dependent. To account for this, the number of encoder cells N is determined by:

$$N = log_2(d) - 1 \tag{3.7}$$

where $d = \sqrt{D}$ with D being the dimensionality of each data point from some dataset (disregarding the number of channels C). This way, for 32x32xC images (of dimensionality $D = 32x32$), the model will form $log_2(32) - 1 = 4$ layers and output a feature map with dimensions 2x2xF (as it will for any other dimension whose square root is a square of 2), where F is the number of filters of the final layer. In order to fully reconstruct the input, the decoder must also have N cells.

3.1.4 Additional details

The implementation of HRWAE-GAN was additionally inspired by the work of [Radford et al., 2015], where the authors propose several techniques that improve the performance of deep convolutional GANs. Hence, the following strategy was employed:

- Replacement of pooling layers with strided convolutions because while pooling layers force the down-sizing artificially, strided convolutions allow the generator to learn it more naturally.
- Usage of batch normalization in both the generator and the discriminator. This method prevents mode collapse, which happens when the generator's outputs collapse to a single point that reliably fools the discriminator.

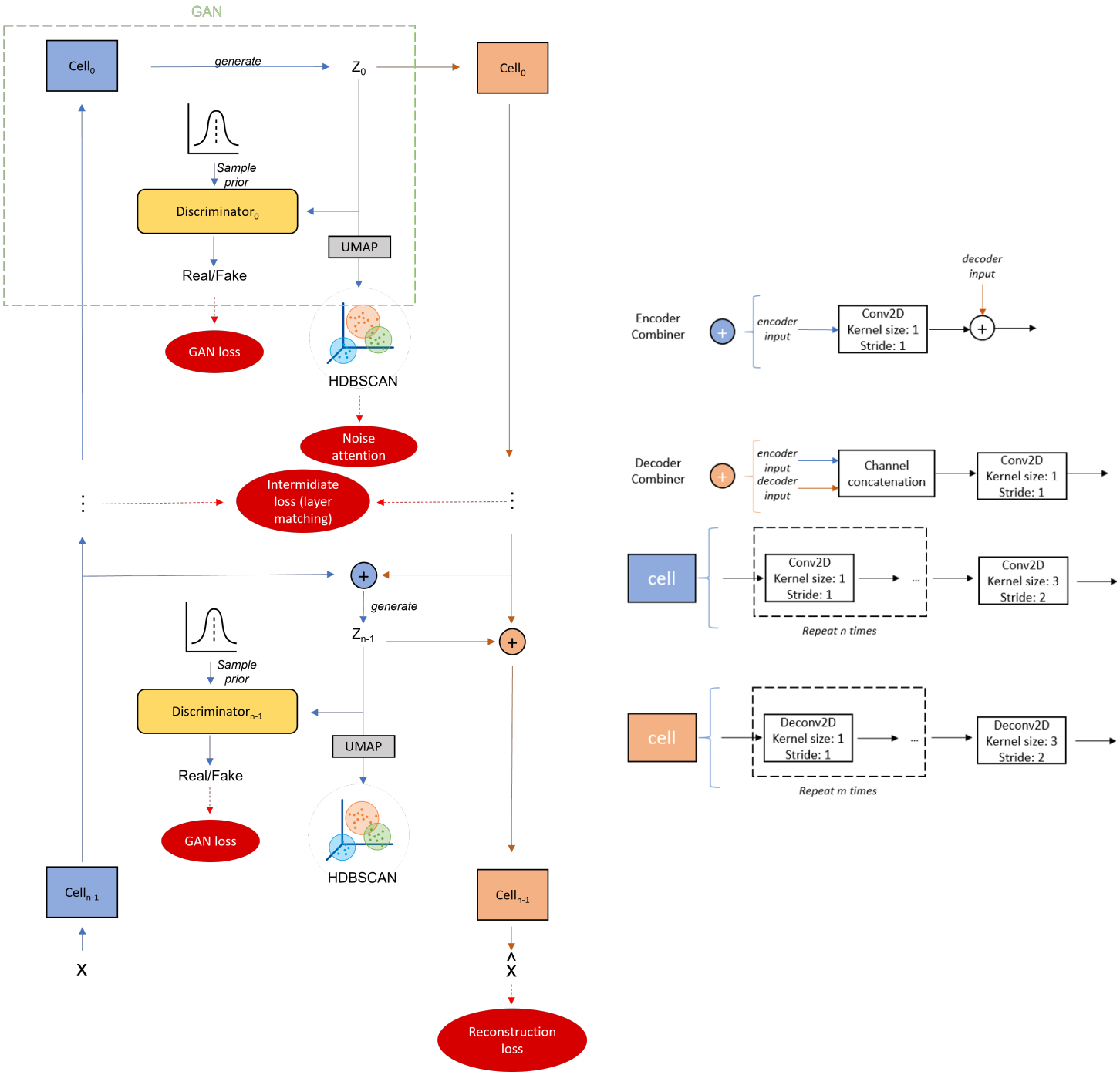


Figure 3.2: Detailed architecture of HRWAE-GAN. The input x is passed through $N = log_2(d) - 1$ convolutional cells, where $d = \sqrt{D}$ and D is the dimensionality of x (disregarding the number of channels C. The final convolutional cell will consequently and necessarily output a feature map of dimensions 2x2xF, where F is the number of filters of this last cell. Then, the latent variable z_0 is judged by the latent relativistic discriminator $Discriminator_0$, which will compute how realistic z_0 is compared to the prior p_z. The latent variable is then passed through the N deconvolutional cells and combined with the intermediate feature maps of the encoder. The encoder combiner cells output the remaining latent variables $\{z_1, ..., z_L\}$, which are judged by their respective discriminators $\{Disciminator_1, ..., Discriminator_L\}$. These remaining latent variables are reintegrated in the Decoder with the Decoder combiners. Caption continues in the next page.

In each convolutional and deconvolutional cell, the feature maps may be passed through n and m non-reducing layers, which, as the name suggests, do not reduce the dimensions. Although this should increase the model's ability to understand the input and overall expressiveness in theory, it showed to be unpractical because it increases the instability beyond what allows training to be completed. For this reason, both hyperparameters were set to zero. During training, and as a refinement measure, the latent space at the top latent group (Z_0) is transformed by UMAP and then clustered by HDBSCAN, which identifies noisy points. These noisy points will then receive increased attention during the following epochs. After training, each latent group $\{Z_0, ..., Z_L\}$ can be independently clustered by some clustering algorithm (usually with HDBSCAN or a GMM).

3.2 Clustering methodology

Considering the task of assigning N data points $X = [x^0, ..., x^{N-1}]$ to K clusters, we start by deriving the embeddings of each latent group $Z = \{[z_0^0, ..., z_0^{N-1}], ..., [z_{L-1}^0, ..., z_{L-1}^{N-1}]\}$, where L is the number of latent groups. Then, we may apply some clustering function to each group in Z.

In a fully unsupervised setting it's ideal to let the model identify the number of clusters on its own. For this reason, density based approaches can be extremely useful in such settings as they do not require the number of clusters K as a hyperparameter. In this work, HDBSCAN [Campello et al., 2013,Campello et al., 2015] was used as the main clustering algorithm to cluster on the latent space. Because it is density based, HDBSCAN also has the useful feature of noise identification. Any point x_1 that does not have a neighbor x_2 with $d_{1,2} < \epsilon$, where $d_{1,2}$ is the distance between two points and ϵ is some distance determined by HDBSCAN, is considered as noise. The hierarchical aspect of HDBSCAN allows this distance ϵ to vary between clusters, thus allowing for clusters of different densities.

However, density based approaches are known to struggle with high feature dimensionality. For some image dataset X with moderate dimensions, such as 32x32x1, the original dimensionality D is too large for effective clustering. This is why it is key to derive a lower dimensionality latent space Z from X where HDBSCAN is capable of operating effectively.

3.2.1 UMAP cluster separator and compactor

UMAP [McInnes et al., 2018] is a manifold neighbor graph technique that can be used to visualize data in lower dimensions, just like PCA [Martinez and Kak, 2001] and t-SNE [Van der Maaten and Hinton, 2008], but has the interesting properties of maintaining both global and local structures while providing more compact sub spaces. In [McConville et al., 2021], the authors took advantage of UMAP by applying it to the Autoencoder's latent space. As a result, they achieved a better clustering performance.

In this work, UMAP improved the clustering performance as well. In figure 3.3, we can see that the latent space becomes better separated and with more compact clusters after applying a UMAP transform.

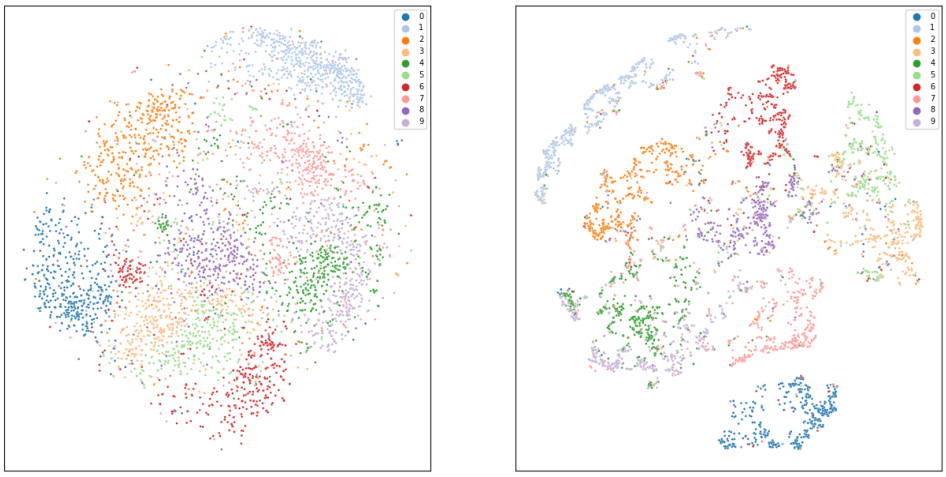


Figure 3.3: Visualizing MNIST's latent space with TSNE: raw (left) and after UMAP transform (right). It is visually clear that UMAP increases the separation and compactness of the clusters.

As previously mentioned, UMAP is highly dependent on its number of neighbors hyperparameter. This hyperparameter controls how much attention is given to local versus global structure. For low values, it will be more focused on capturing local structure; for high values it will be focusing on global structure. Although this is the general idea behind this hyperparameter, what is considered as a high or low value is dependent on the dataset's size. For example, if we consider a dataset with only 100 samples per cluster, then 100 neighbors will provide a global view of the dataset. However, if the dataset contains 1000 samples per cluster, 100 neighbors will provide a more a more localized view.

To deal with this issue, Pedro Távora Santos decided it made sense to have a data dependent number of neighbors and created the following expression:

$$n\_neighbors = min\{\lfloor \frac{dataset\_size}{300} \rfloor, 100\} \tag{3.8}$$

The 300 in the denominator was determined empirically to deliver the best results, but the optimal value is often dependent on the dataset as well. The number of neighbors is not allowed to exceed 100 due to memory constraints.

It is still important to note that while UMAP does boost clustering performance, there may be some variance in its transforms. This variance may be more or less noticeable depending on the dataset and its properties. For the large image datasets used in this work, the variance is not problematic.

3.3 Loss function

The resulting loss function has 4 components and can be written as:

$$L = L_R + \alpha L_G + \kappa L_I + \beta L_N \tag{3.9}$$

where L_R is the reconstruction loss, L_G is the GAN loss, L_I is the intermediate loss, L_N is the noise attention and α, β and κ are balancing hyperparemeters. L_G has already been discussed in section 3.1.1; the remaining three losses will be described in the following sub sections. The objective is optimized using the reparameterization trick [Kingma and Welling, 2013] [Rezende et al., 2014].

3.3.1 Reconstruction Loss

Recalling section 2.2, an Autoencoder is composed by an encoder, which takes some data point $x \in R^{d_x}$ and generates an embedding $z \in R^{d_z}$ with $d_z << d_x$, and a decoder, which reconstructs z to an approximation of x. The similarity between x and the reconstruction $\hat{x}$ is of great importance because it is a way of assessing how meaningful the latent embedding z is (if it's not meaningful, then the decoder cannot reconstruct effectively).

To guarantee this similarity, the reconstruction loss L_R is introduced. This loss measures the dissimilarity between x and $\hat{x}$ and is dependent on the dataset. For black and white images the binary cross-entropy can be used because the pixel values are approximately binary, while for more complex colored images the Mean Squared Error (MSE) is preferred.

3.3.2 Intermediate Loss

Experimentally, I found that cluster separation was only being achieved at some of the latent groups in Z. This indicates that the model does not naturally make use of all layers when the number of latent groups L is greater than some amount. I empirically verified that this effect is noticeable when $L > 2$. As discussed in [Gulrajani et al., 2017], this issue was solved through layer matching. Layer matching consists of adding a restraint to ensure that the mid-level feature maps produced by the decoder cells match the ones produced by the encoder cells of their respective level. Formally, this can be written as:

$$L_I = \sum_{l=1}^{L-1} D(i^l_{encoder}, i^{l-1}_{decoder}) \tag{3.10}$$

where l denotes some latent group (the top group, L is the total number of latent groups, D is some error function such as the Cosine Dissimilarity, and $i_{encoder}$ and $i_{decoder}$ are the intermediate feature maps.

The effect of this constraint will be demonstrated in the experiments section.

3.3.3 Noise attention

HDBSCAN is a density based clustering algorithm with the particularity of noise identification (it does not classify points which it considers as noise). Noisy points can arguably be considered as points that are not correctly embedded, as they are not within reasonable distance from other points from the same class (recall that density clustering algorithms consider two data points x_1 and x_2 to be from the same if $d_{1,2} < \epsilon$, where $d_{1,2}$ is the distance between the two points and ϵ is some minimum distance). For this reason, it makes sense to identify these noisy points and increase the model's attention towards them. This results in a diminishing of noise in the latent spaces, as will be demonstrated in the experiments section.

This noise attention can be formalized as:

$$L_N = \sum_{n=0}^{N-1} \begin{cases} 1 & if \quad x_n \in Noise \\ 0 & otherwise \end{cases} \tag{3.11}$$

where $Noise$ is the set of points that are classified as noise by HDBSCAN. The percentage of increased attention is determined by $\kappa - 1$, where κ is the hyperparameter used in equation (3.9).

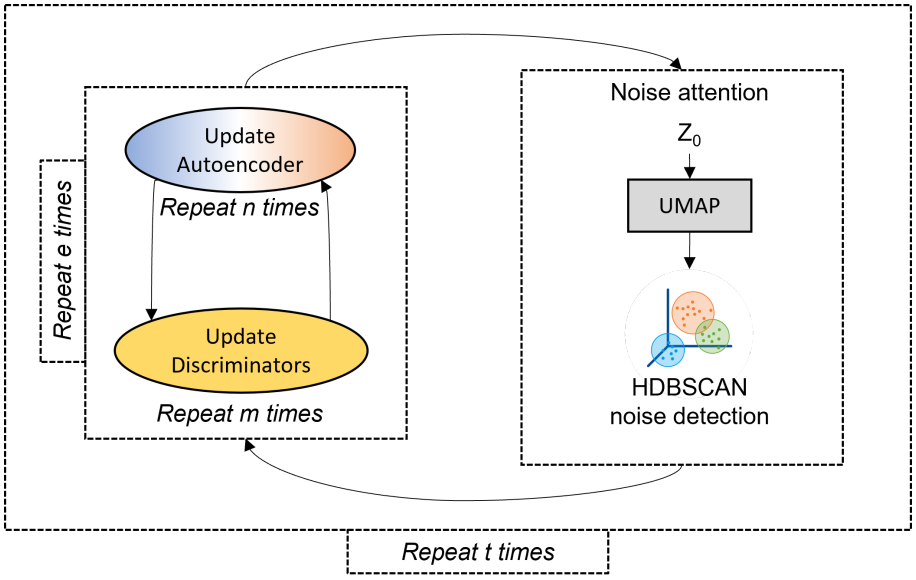


Figure 3.4: HRWAE-GAN's training cycle. The autoencoder and the discriminators are consecutively updated n and m times, respectively. When $n, m > 1$, the adversarial training becomes more stable because both the autoencoder and the discriminators have more time to adapt to each other. This repetitive training was crucial to stabilize training. After e iterations of this training, the latent group Z_0 is tested to find potential noise, as was described in the previous section. Training then resumes with special attention given to noisy data points. This entire process is repeated t times, so the total number of epochs is given by et.

3.4 Summary

In this section, HRWAE-GAN's architecture was thoroughly explained. To summarize, this model is composed of two main blocks: the Hierarchical Wasserstein Autoencoder GAN, which is essentially a generative autoencoder with a hierarchical latent structure that is regularized with adversarial training; and the clustering box, which applies a traditional clustering algorithm (usually HDBSCAN [Campello et al., 2013, Campello et al., 2015]) to the obtained latent groups after a UMAP transform [McInnes et al., 2018].

The model also has quite the complex loss function, which is described in section 3.3. Apart from the reconstruction loss, it utilizes a relativistic adversarial loss [Jolicoeur-Martineau, 2018], which stabilizes the adversarial training; a layer matching loss, which enforces the model to make use of all latent groups; and a noise attention loss, which attempts to refine clustering performance by increasing attention to noisy data points.

Finally, HRWAE-GAN also uses a custom training cycle that can be visualized in figure 3.4.

4

Experiments

Contents

4.1 Datasets

The experiments in this section were conducted on three distinct datasets: the MNIST dataset [LeCun et al., 2010], the Fashion MNIST (FMNIST) dataset [Xiao et al., 2017] and the CIFAR-10 dataset [Krizhevsky, 2009].

The MNIST dataset consists of 70,000 28x28 fully labeled black and white images of single handwritten digits from zero to nine, which is usually divided into a training set of 60,000 samples and a test set of 10,000 samples. The digits in the dataset were handwritten by 500 different writers, so the data is varied and real world based.

The Fashion MNIST dataset has the exact same specifications as the MNIST dataset but consists of images of clothing articles. More specifically, it contains images of t-shirts/tops, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags and ankle boots.

The CIFAR-10 dataset contains 60,000 32x32 fully labeled colored images of ten different types of objects and animals, which are split into a 50,000 samples training set and a 10,000 samples training set. The dataset's classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.

4.2 Choice of the autoencoder

Initial experiments were focused on choosing the best type of generative Autoencoder. The two variants considered were the Beta Variational Autoencoder [Higgins et al., 2016] (which will be referred to as VAE, for short) and the Wasserstein Autoencoder GAN (WAE-GAN). To make this comparison, I considered their capabilities in image generation, image reconstruction, and clustering friendliness.

Training setup: to compare the two models, a 10-fold cross validation was used to train them on the MNIST and FMNIST datasets. Both the average results and the results of each model's best run will then be compared. Based on previous work, a latent dimension of 23 was used for the models. This will also make the comparison of performance easier. The β parameters are set to 30. Both models also have the exact same structure (number of layers, hidden neurons per layer and kernel size).

In figure 4.3, where we are comparing the reconstruction ability of each model's best run, it's clearly visible that the WAE-GAN is capable of reconstructing images with overall better quality and that are more faithful to the original form. In the second column (with the number 0), for example, we can see the WAE-GAN's reconstruction picks up on the original image's extra bit of girth on the left side. On the other hand, the VAE's reconstruction just adds some undesired blurriness to that zone. On the FMNIST dataset, it is even more evident that WAE-GAN can reconstruct images with much better quality. It can't, however, pick up on certain particularities of the clothing articles (like the letters on the second column).

Furthermore, the FID score was evaluated for both models and for each dataset: VAE achieved an average FID score of 11.84 and WAE-GAN outperformed it with an average of 5.63. Once again, this

Figure 4.1: Reconstructions on MNIST

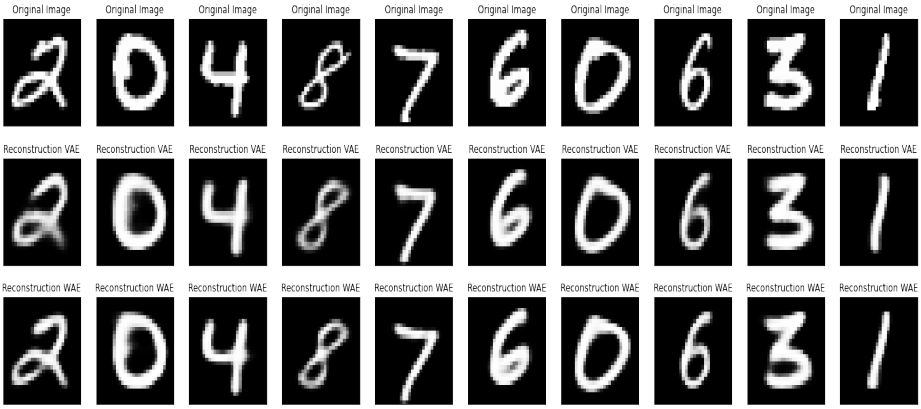


Figure 4.2: Reconstructions on FMNIST

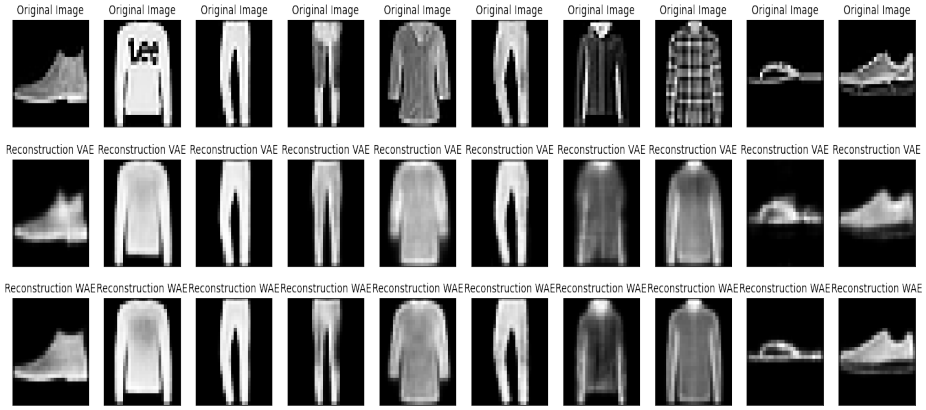
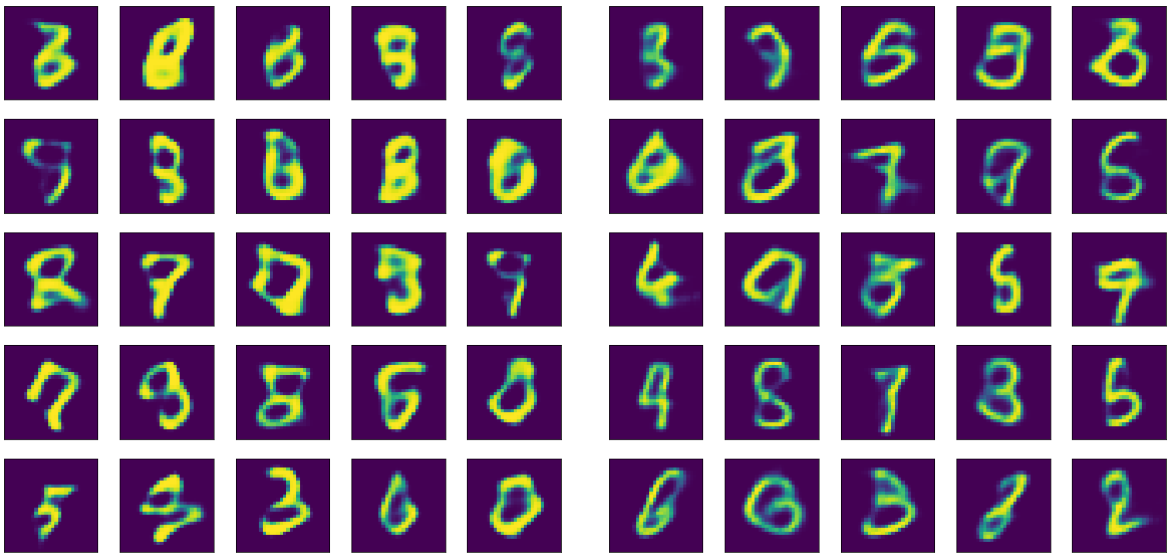


Figure 4.3: Comparison between original images (top), VAE's reconstruction (middle) and WAE-GAN's reconstruction (bottom) on MNIST's and FMNIST's test datasets.

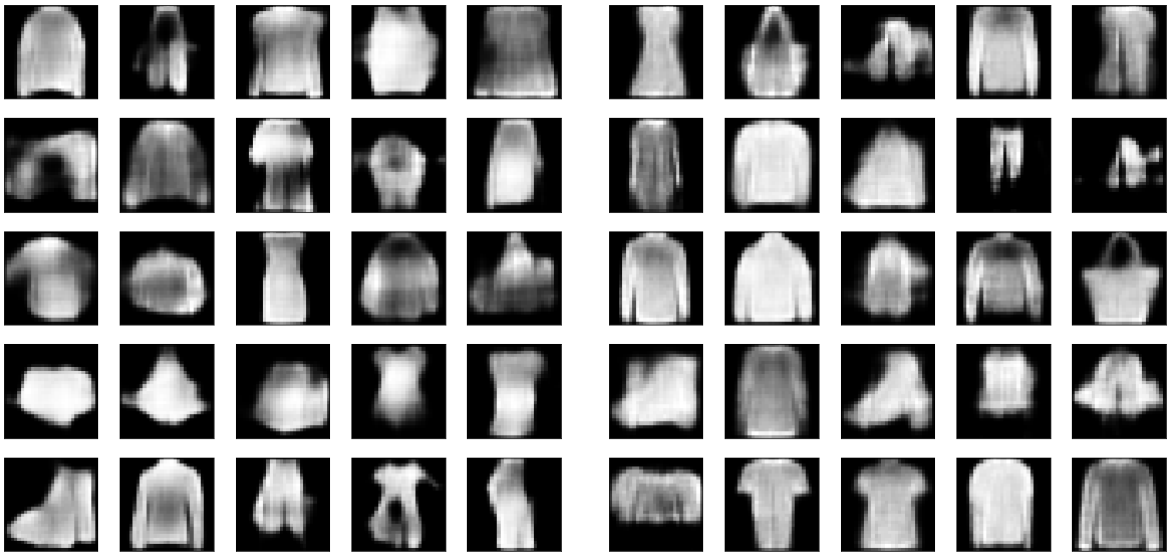
reinforces WAE-GAN's superiority regarding reconstruction ability.

Regarding image generation, the two models behaved similarly and obtained almost the exact same FID scores. This can be verified in Figure 4.4. Poor image generation proved to be WAE-GAN's main weakness, as the images it generates are easily distinguishable from real ones.



Images generated by WAE-GAN on MNIST.
FID = 94.027

Images generated by VAE on MNIST.
FID = 94.0266.



Images generated by WAE-GAN on FMNIST.
FID = 41.88.

Images generated by VAE on FMNIST.
FID = 42.02.

Figure 4.4: Generated images

As for clustering friendliness, the WAE-GAN is able to produce a better separated latent space (see Figure 4.5) which allows for better clustering results.

Table 4.1: Clustering results with HDBSCAN on the MNIST and FMNIST datasets. In the columns displaying average results, both the average (left) and the standard deviation (right) are being shown.

		#clusters	% labeled (avg)	acc labeled (avg)	total acc (avg)	% labeled (best)	acc labeled (best)	total acc (best)
MNIST	WAE-GAN	10	88.4% +/- 6.42%	92.43% +/- 4.5%	85.41% +/- 6.57%	92%	96.53%	90.51%
	VAE	11	77.7% +/- 6.42%	90.6% +/- 6.11%	76.63% +/- 6.11%	81%	96.05%	84.67%
FMNIST	WAE-GAN	10	77.9% +/- 6.16%	66.09% +/- 2.37%	56.79% +/- 1.47%	83%	66.45%	57.95%
	VAE	11	64.3% +/- 5.66%	63.89% +/- 1.95%	45.28% +/- 2.44%	84%	61.37%	50.87%

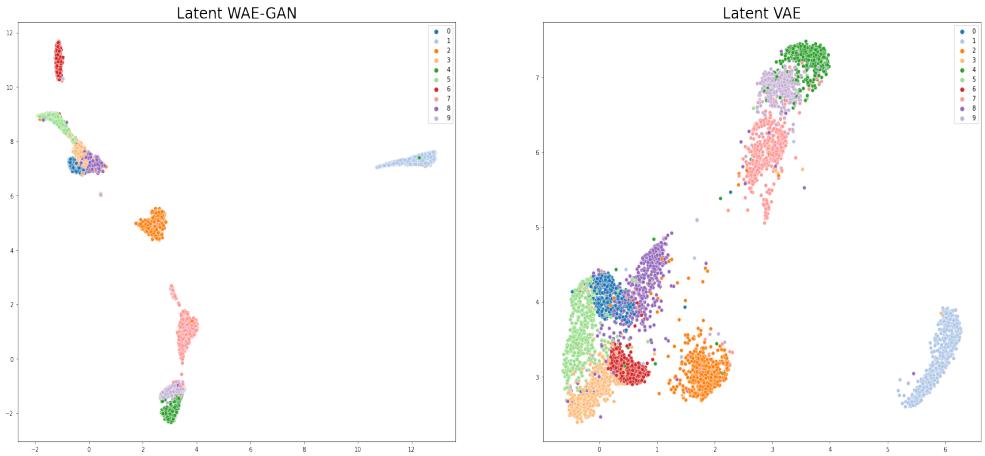


Figure 4.5: The latent spaces of WAE-GAN (left) and VAE (right) after a UMAP transform for the MNIST dataset. It's visible that WAE-GAN's latent space is more separated and compact than VAE's.

4.3 Improving training

As previously mentioned, one of the main issues with adversarial training is training instability. To address this issue, some different training techniques were experimented with. These techniques will be described in the following subsections.

4.3.1 Repeated network updates

The most simple way to train a GAN is to make interleaved updates to the generator (G) and discriminator (D) networks. However, this method can lead to instability as one of the two participants may have a hard time adapting to the rapid changes. One way to combat this issue is by letting G and D make n and m times in a row, respectively. Experimentally, it was apparent that setting both n and m to values greater than 1 was extremely beneficial towards improving stability. Going even further, if we consider r_{update} as the ratio of between m and n, then finding some ideal value for this ratio is helpful in increasing training stability.

4.3.2 Two phase adversarial training

Some works have found that, in adversarial training, it may be helpful to have two stages of training. In the first stage, $r_{update} >> 1$ (the Discriminator receives more updates); in the second stage r_{update} is some value closer to 1. The reasoning behind this method is to first allow D to learn what fake and real data points look like so that in the second stage it is capable of providing helpful feedback to G. If G begins its training without a well trained D, then its learning could result unfruitful.

Initial experiments split the training into two phases. On the first phase, the discriminator received more updates per step ($m > n$); on the second phase, the generator received more updates per step ($n > m$). The first problem encountered was with the hierarchical model quickly diverging when entering the second phase. It was assumed that this issue could have been being caused by the optimizer being used, Adam [Kingma and Ba, 2017]. Adam utilizes the concept of momentum, first proposed in [Qian, 1999], to achieve a faster yet controlled convergence to the objective function's minimum. Momentum is an intuitive concept that adds the concept of time to parameter updates. Apart from utilizing the gradient of time step t, it adds a momentum term that corresponds to the gradient of the time step $t-1$ and multiplies it by some value γ, which is commonly set to 0.9 or similar. The effect of this technique is that the updates gain momentum with time, moving faster in the direction of the previous updates (a common analogy is to imagine a ball zig zagging down the interior of an upside down cone. As it rolls down, it loses lateral movement and gains vertical momentum, moving faster and faster in the direction of the tip of the cone). This is formally described in equation 4.1.

$$W_t = W_t - \eta \nabla L(W_t) + \gamma v_t \tag{4.1}$$

$$v_t = \eta \nabla L(W_{t-1}) + v_{t-1} \tag{4.2}$$

Adam, which stands for Adaptive Moment Estimation, improves upon this concept by utilizing adap-

tive learning rates. Simply put, adaptive learning rates consist in starting off with big steps that change their magnitude as training progresses. This is accomplished by dividing the learning rate η by a term that changes with time. In Adam's case, the learning rate is divided by the exponentially decaying average of past squared gradients. Using this exponentially decaying average allows Adam to better control the updates, slowing them down when recent gradients are very large (when variance increases).

So, Adam's momentum is usually good as it allows for faster convergence times. However, in this specific case where the loss landscape is dynamic (changes drastically as we move from phase 1 to phase 2), it can prevent the model from learning. For this reason, for experiments where the values of n and m change very drastically, the alternative optimizer RMSprop was used. RMSprop is similar to Adam, except it does not use the first moment of momentum (corresponding to the average, described in equations 4.1 and 4.2). Instead, it only uses the second moment (which corresponds to the variance). Consequently, it has an easier time readjusting during the second phase.

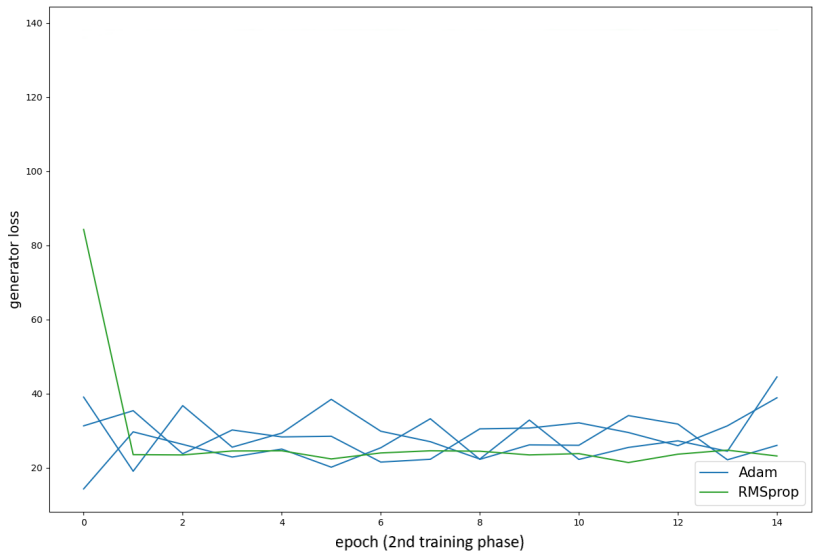


Figure 4.6: Comparison between the generator's loss function when using Adam (blue) and RMSprop (green). While RMSprop is capable of a better recovery than Adam, which is unable to converge, it causes the first phase to often diverge and it's convergence time is comparably slower.

Although this technique is theoretically interesting, it showed to be impractical when combined with a hierarchical autoencoder due to the latter's highly irregular loss landscape. Because this technique adds instability, it considerably contributes towards divergence during training time.

4.3.3 Training with warm up

The final technique that was experimented with was to add a warm-up phase at the beginning of training. Recall HRWAE-GAN's loss function (equation 3.9), where the hyperparameter α is used to weight the importance of the GAN loss. In this warm-up phase, α starts with a very low value (close to 0) and slowly increases step wise (increases on every step of each epoch). Experimentally, I found this technique to be of extreme use regarding the hierarchical stability as it allows the model to slowly incorporate the adversarial training. For this reason, this technique was used for the remainder of the experiments and it contributed towards the final results.

4.4 Choice of the clustering algorithm

Three clustering algorithms were considered for this work: HDBSCAN, Gaussian Mixture Model (GMM) and the Agglomerative clustering algorithm.

HDBSCAN is a hierarchical density based clustering algorithm. While its antecedent DBSCAN requires the minimum neighbor distance ϵ as a hyperparameter, HDBSCAN tries increasingly smaller distances and keeps the more persistent clusters. Unlike what happens with DBSCAN, this may result in clusters of varying density. However, HDBSCAN still requires the minimum cluster size as a hyperparameter. To properly assess its performance, initial experiments were focused on finding the most appropriate value for this hyperparameter. Since the size of different datasets can vary, it does not make sense to actually define a minimum cluster size. Instead, the hyperparameter becomes the denominator that divides the number of samples N of some dataset. This is formally described in equation 4.3.

Table 4.2 shows the percentage of labeled samples (although it is not expected of HDBSCAN to label every sample, since it is safe to assume every large enough dataset has some noisy data points, too low of a value is undesirable as well), the accuracy among the labeled samples (according to the real labels), the total accuracy and the silhouette. Since we are trying to simulate a fully unsupervised environment (and because the accuracy is among the top values as well), it was decided to pick the denominator with the highest silhouette score - 50. These results were obtained on the test set.

$$min\_cluster\_size = \frac{N}{denominator} \tag{4.3}$$

Table 4.2: Assessment of HDBSCAN with different denominator hyperparameters on the Fashion MNIST test set.

Denominator	%labeled	accuracy labeled	accuracy total	silhouette
20	64.06%	62.54%	48.6%	0.2
25	66.94%	62.7%	50.41%	0.24
30	67.58%	62.84%	50.92%	0.25
35	68.2%	62.74%	51.31%	0.25
40	71.92%	63.33%	49.19%	0.26
45	70.79%	63.55%	48.76%	0.25
50	71.35%	63.43%	48.91%	0.27
55	69.79%	60.42%	45.97%	0.23

As for the remaining algorithms, the default parameters were used and the number of clusters was set to the number of clusters found by HDBSCAN. By letting HDBSCAN pick this hyperparameter for each dataset, we are better simulating a truly unsupervised setting.

4.5 Building the loss function

4.5.1 Layer matching

To analyse the impact of the layer matching constriction (see section 3.3.2), which encourages the decoder to output intermediate feature maps that are similar the ones outputted by the encoder at their respective level, an array of values was tested and evaluated through HDBSCAN's noise identification. The idea is that a more clusterable latent space should have a lower percentage of points being considered as noise since, realistically, only a small percentage of points actually are noise. In figure 4.7, when can see that when $L = 2$ (when there are two latent groups), layer matching does not play a huge factor. The variance of the model is also evident, with one of the runs failing completely (without having diverged) and outputting completely unclusterable latent spaces.

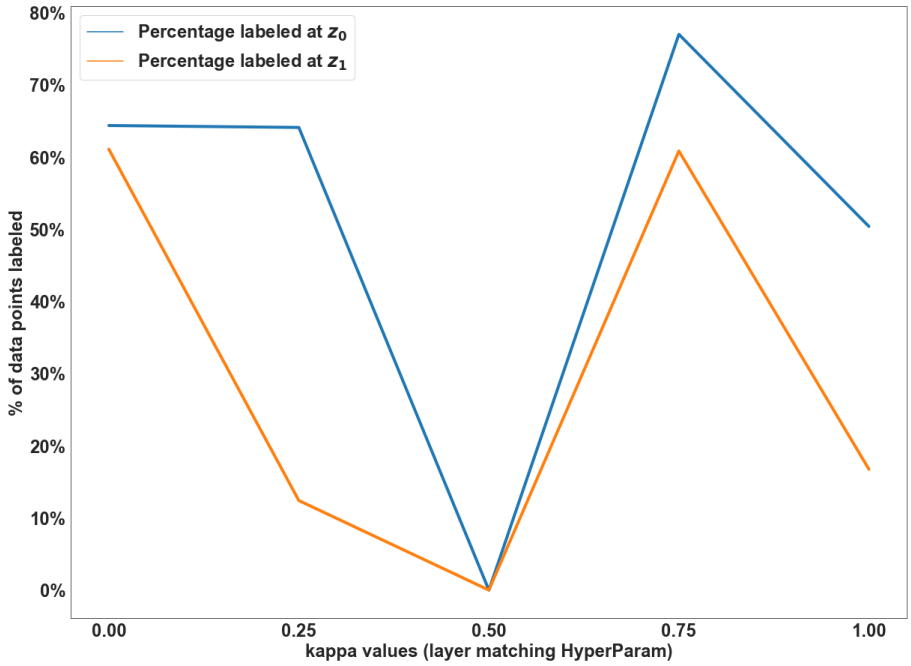


Figure 4.7: This plot shows how the percentage of labeled data points. i.e. data points that are not considered as noise, varies in each latent group from Z as we increase the importance of the layer matching constraint when the number of latent groups L equals two.

However, when $L = 3$, the effect becomes more apparent. In figure 4.8 we can see that without this constraint, some of the latent groups have absolutely no separation between classes. After introducing layer matching, the model fixes this issue and thus makes use of every latent group.

Be that as it may, this is not very relevant in practise because the model's training instability is quite problematic whenever $L > 2$. It consequently becomes difficult to evaluate layer matching in those circumstances, so its effects are hard to prove empirically.

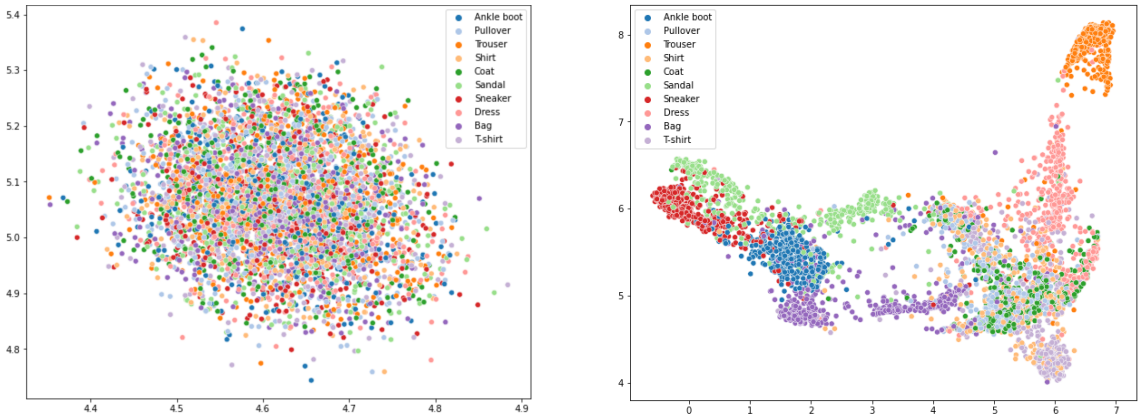


Figure 4.8: HRWAE-GAN's latent space on the top layer without (left) and with (with) layer matching when when the number of latent groups L equals three.

4.5.2 Noise attention with HDBSCAN

As discussed in section 3.3.3, HDBSCAN can be used to identify noisy points (data points that don't belong to any cluster). By using this feature, we can increase the attention the model pays to these points with the goal of refining their encoding.

In figure 4.9 we can see that increasing the attention up to a certain point causes the percentage of noise to decrease in latent group z_1. This is especially interesting because noisy points are identified at z_0 but the model somehow utilizes this information to refine z_1, leaving z_0 basically unchanged.

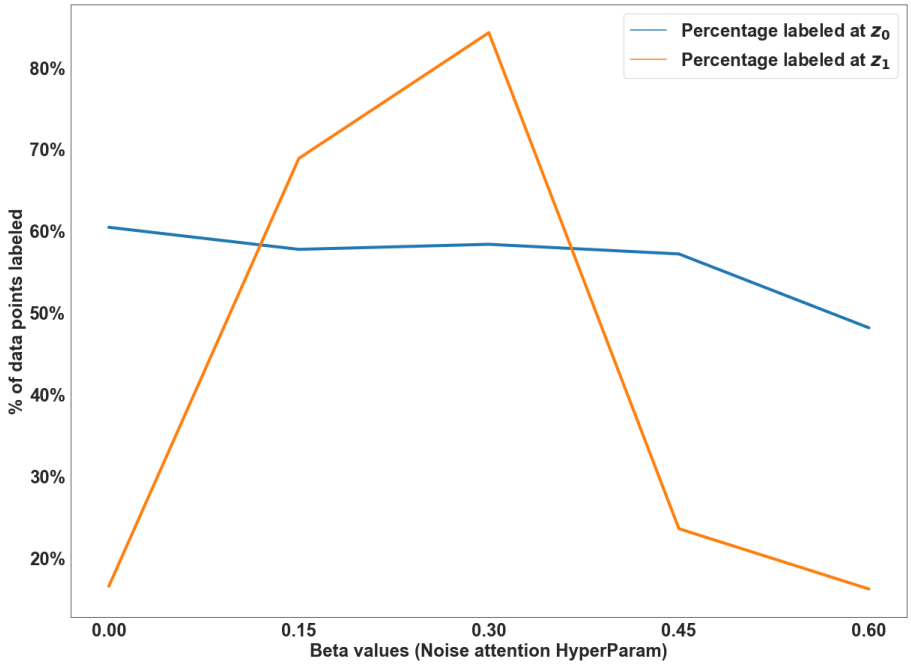


Figure 4.9: This plot shows how the percentage of labeled data points. i.e. data points that are not considered as noise, varies in each latent group from Z as we increase the extra attention paid to noisy data points when the number of latent groups L equals two. Example: when beta equals 0.15, noisy data points receive an extra 15% attention.

4.6 Results & Discussion

4.6.1 Number of latent groups

This subsection will be used to analyse the effects of varying the number of latent groups. This analysis will be based on three factors: clustering performance, quality of image reconstruction and quality of image generation. Note that when the number of latent groups L equals 1, the model drops its hierarchical structure and becomes a regular Wasserstein Autoencoder GAN.

In table 4.3 we can see how adding an extra latent group affects clustering performance. In latent group 0, clustering is slightly worse than the non-hierarchical model but without significant increase in

Table 4.3: Clustering results on the Fashion MNIST dataset for different amounts of latent groups (first column) and for each latent group (second column). The columns tagged with "(best)" correspond to the values of the model with highest total clustering accuracy.

#latent groups	latent group	#clusters (best)	%labeled (avg)	acc labeled (avg)	totall acc (avg)	%labeled (best)	acc labeled (best)	total acc (best)
1	0	10	68.42% +/- 4.6%	62.1% +/- 2.1%	47.84% +/- 5%	68.9%	63.4%	51.71%
2	0	10	56.4% +/- 4.26%	69% +/- 5.6%	44.2% +/- 2.98%	60.4%	72.8%	49.6%
	1	10	41.86% +/-28.9%	40% +/- 21%	29.38% +/- 20.1%	84.3%	65.1%	56.8%

variance. In latent group 1, however, results are a bit more interesting. When it comes to the average results, they are significantly worse and with very large standard deviation. This shows how the hierarchical architecture can have very high variance between different instances of the model. Even though the best accuracy result was achieved in the second latent group, the high variability makes it unusable in a fully unsupervised environment since there is no way of knowing how reliable an instance of the model is.

So, at least for this particular implementation, **using a hierarchical architecture did not present any advantages when it comes to clustering tasks**.

Nonetheless, in figure 4.12 it's visible that the quality of image reconstruction increases significantly with additional latent groups. Unfortunately, since the only use of reconstruction quality is to have an idea of how trustworthy the latent representations are and we've determined that hierarchical models do not increase clustering performance, this positive aspect is basically insignificant.

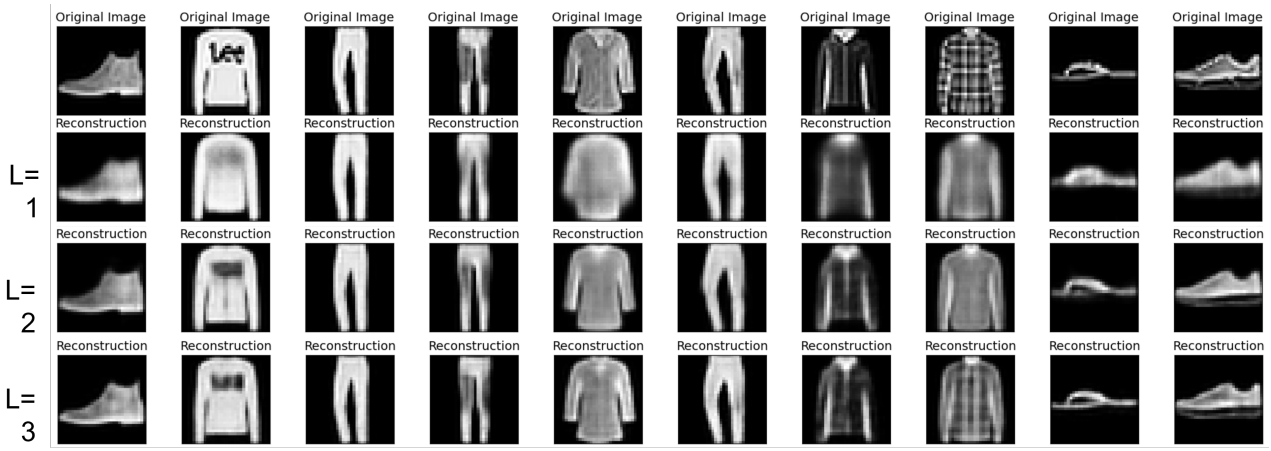
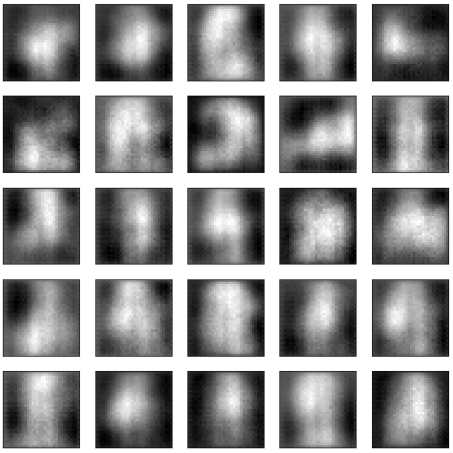


Figure 4.10: In this image we can see how reconstruction quality improves with the number of latent groups L.

Images generated by HRWAE-GAN with a single latent group ($L = 1$).
Average FID = 32.95 +/- 0.71



Images generated by HRWAE-GAN with two latent groups ($L = 2$).
Average FID = 30.76 +/- 2.24

Figure 4.11: Images generated by HRWAE-GAN. FID scores were obtained by averaging 5 runs (each run corresponds to a different instance of the model).

Finally, we'll take a look at the effect of multiple hierarchical groups on the generative capabilities of HRWAE-GAN. In figure 4.11, it's apparent that adding a latent group blurred the generated images, even though the FID score somehow improves. This was surprising because the authors of NVAE [Vahdat and Kautz, 2020], who used a hierarchical VAE to generate images, obtained opposite results. Since the biggest difference between HRWAE-GAN and NVAE is the regularization of the latent spaces (HRWAE-GAN utilizes a GAN framework), I believe this could possibly mean that **mixing GANs with a hierarchical architecture is either not beneficial or requires a lot of fine tuning**.

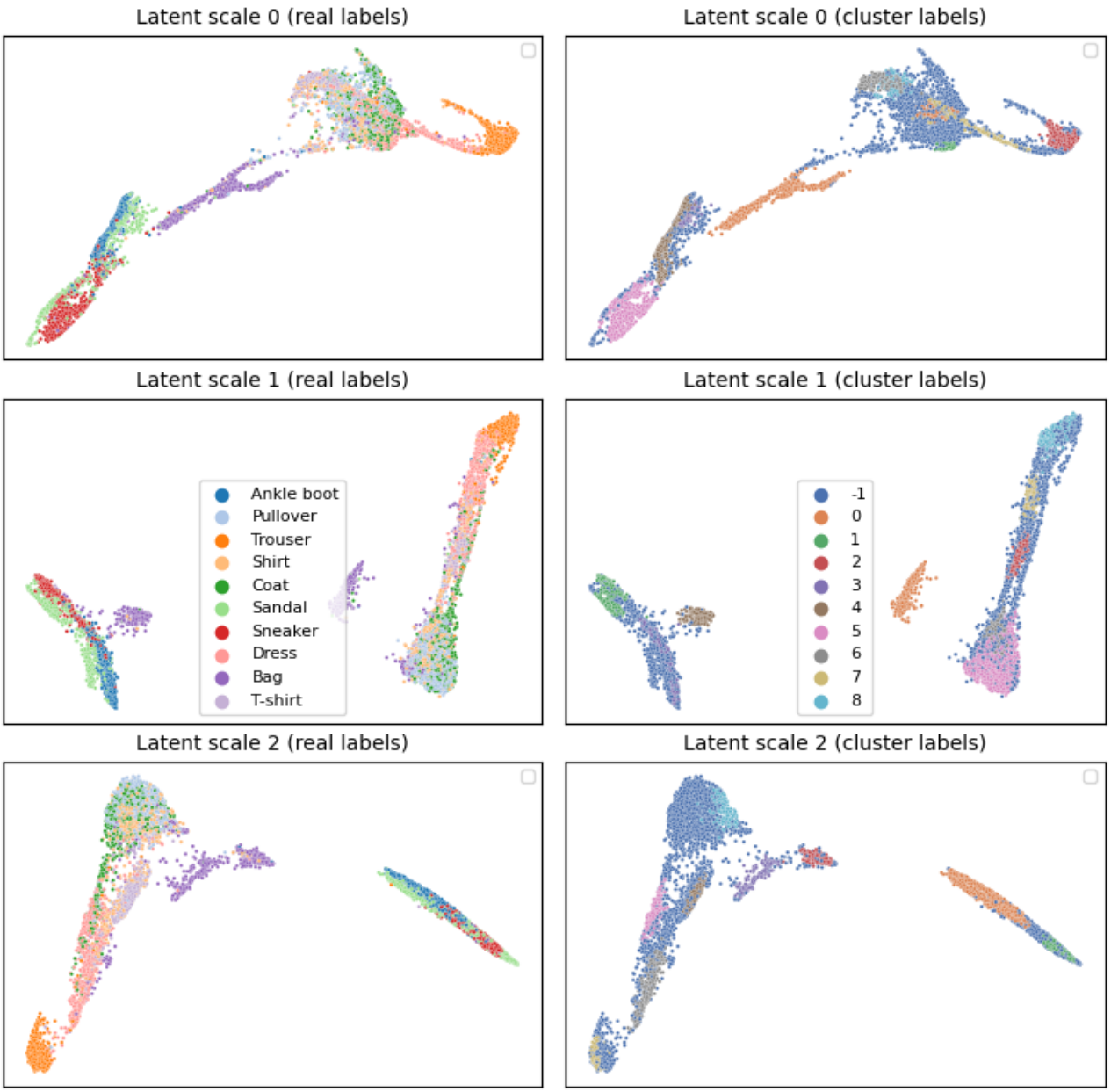


Figure 4.12: Latent groups of HRWAE-GAN when the number of latent groups L equals three. The left column shows the real labels while the right column shows HDBSCAN's labels. Data points labeled with -1 are considered as noise by HDBSCAN.

4.6.2 Comparison of clustering performance with other state-of-the-art models

Table 4.4 shows how HRWAE-GAN compares to both MoE and the current best performing model for each one of the three experimented on datasets. It's visible that the attained model is outperformed by a significant margin on all cases, but especially of the CIFAR-10 dataset.

	#clusters	%labeled (avg)	acc labeled (avg)	totall acc (avg)	%labeled (best)	acc labeled (best)	total acc (best)
HRWAE-GAN GMM (L=1)	-	-	-	54.3%	-	-	60.4%
HRWAE-GAN HDBSCAN (L=1)	10	68.4% +/- 4.57%	62.1% +/- 2.1%	47.8% +/- 5%	68.87%	63.35%	51.7%
HRWAE-GAN GMM (L=2)	-	-	-	60% +/- 3.4%	-	-	63.9%
HRWAE-GAN HDBSCAN (L=2)	10	56.4% +/- 4.6%	69% +/- 5.6%	44.2% +/- 2.98%	84.3%	65.1%	56.8%

Table 4.4: Clustering results comparison between a two latent group HRWAE-GAN, the Mixture of Experts and the best performing model for each datataset - SPC [Mahon and Lukasiewicz, 2021], AE + SNNL [Agarap and Azcarraga, 2020] and SPICE [Niu and Wang, 2021].

		#clusters	%labeled (avg)	acc labeled (avg)	totall acc (avg)	%labeled (best)	acc labeled (best)	total acc (best)
MNIST	HRWAE-GAN GMM (L=2)	-	-	-	70.37% +/- 4.5%	-	-	77%
	HRWAE-GAN HDBSCAN (L=2)	10	55.2% +/- 10.3%	85.6% +/- 7%	54.9% +/- 6.6%	78.86%	94.14%	78.8%
	MoE	10	-	-	97.4%	-	-	-
	SPC	-	-	-	99.03% +/- 0.1%	-	-	-
FMNIST	HRWAE-GAN GMM (L=2)	-	-	-	60% +/- 3.4%	-	-	63.9%
	HRWAE-GAN HDBSCAN (L=2)	10	56.4% +/- 4.6%	69% +/- 5.6%	44.2% +/- 2.98%	84.3%	65.1%	56.8%
	MoE	10	-	-	68%	-	-	-
	AE + SNNL	-	-	-	84.4%	-	-	-
CIFAR10	HRWAE-GAN GMM (L=2)	-	-	-	14.7% +/-2.2%	-	-	16.48%
	HRWAE-GAN HDBSCAN (L=2)	2	2.9% +/- 18.3%	22% +/- 11.5%	15.7% +/- 2.8%	58%	21%	17.6%
	MoE	3	-	-	32.8%	-	-	-
	SPICE	-	-	-	91.7%			

4.6.3 Discussion

All things considered, HRWAE-GAN's performance fell below expectations. It's fair to say that the added complexity of multiple latent groups is not rewarding. Apart from its results not being up to par with the current state-of-the-art, it also suffers from three main issues:

1. As the number of latent groups L increases, training becomes considerably more unstable and often diverges before completion. This is why my experiments were mostly done with $L = 2$. This is unfortunate because it would have been interesting to see what would happen with larger Ls.

2. The model suffers from training variance, meaning that two models trained with the same hyper-

parameters will output differently organized latent spaces and generate images of varying quality. This makes it very hard to interpret how hyperparameters and different added constraints effectively change the outcome of training.

3. The latent space regularization through adversarial training is also a hard task. While the relativistic discriminator heavily increases training stability, the loss evolution can be hard to interpret because it is not a simple minimization problem. This makes it harder to know what to change when results are sub par.

According to recent state-of-the-art works like SPICE [Niu and Wang, 2021] and SPC [Mahon and Lukasiewicz, 2021], semi-supervised approaches easily outperform hierarchical ones such as this one and are easier to implement and experiment with (the implementation of this model was very challenging).

I believe the most important observation I have come to realize is that a meaningful latent space does not necessarily equate to a clusterable latent space. A priori, I would expect the reconstruction loss (which relates to the meaningfulness of the latent space) to be highly correlated to clusterability. Obviously, it is to some extent. If the data points are just randomly mapped to the latent space, clustering performance will be underwhelming. However, I have found that the relationship is not linear. This can be visualized in figure 4.13, where quite a few badly reconstructed points are clustered correctly (red circles) while some well reconstructed ones are clustered incorrectly (green crosses).

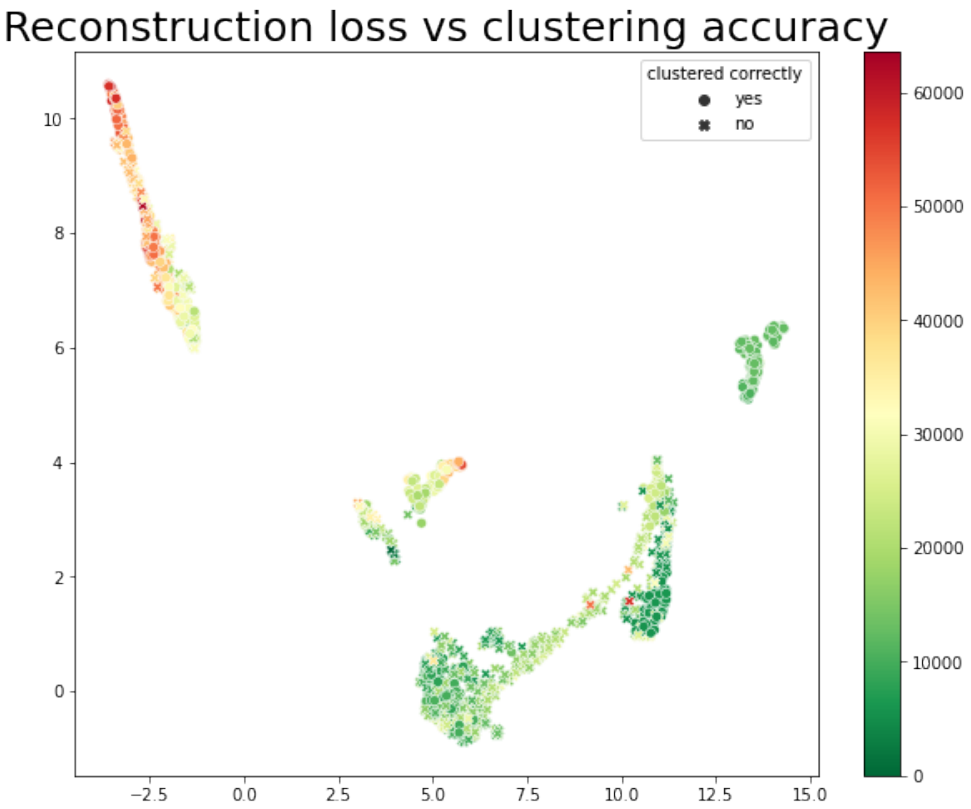


Figure 4.13: Visualizing how reconstruction quality relates to clustering accuracy. Greener points have better reconstruction quality. Clearly, there is not a linear relationship between reconstruction quality and clustering accuracy.

Another aspect that is important to consider is how some data points from different classes x_1 and x_2 can look more similar to each other than to other data points in their same classes. Consequently, in an unsupervised setting, x_1 and x_2 may end up closer to each other than to other in the latent space than to other points from their respective classes. This is further illustrated and discussed in figure 4.14.

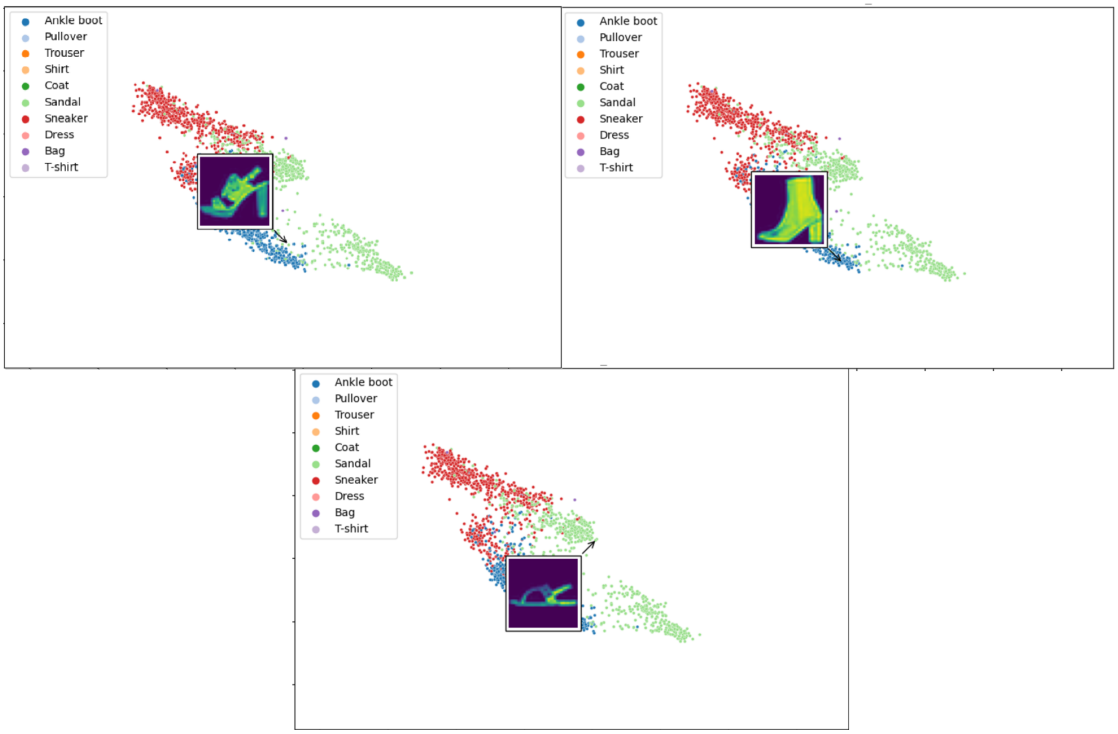


Figure 4.14: Details of the "shoes" cluster in the latent space. Notice that each figure has an arrow pointing to its respective data point. Lets refer to each picture, from left to right and top to bottom, as $sandal_1$, $ankle\_boot_1$ and $sandal_2$. Visually we can see that $sandal_1$ is more similar to $ankle\_boot_1$ than to $sandal_2$. For this reason, the encoder maps $sandal_1$ to be closer to $ankle\_boot_1$ than to $sandal_2$. From a clustering perspective, however, the accuracy measure would penalize this decision since we have two points from different classes in the same cluster. This suggests that clustering accuracy may not always be a fully reliable way of assessing the quality of an embedding.

4.6.4 Summary

In this section, several experiments were conducted in order to both guide the development of the model and to evaluate it. I started by comparing two types of generative frameworks: the Beta VAE [Higgins et al., 2016] and the WAE-GAN [Tolstikhin et al., 2017], which lead me to choose the latter as the base for HRWAE-GAN. Then, I experimented with some training techniques with hopes of achieving greater training stability, obtaining mixed results depending on the techniques. Having achieved a somewhat stable training, I experimented with various clustering algorithms and optimized and analyzed the loss function hyperparameters. Finally, I concluded the section by presenting and discussing the obtained results, which revealed the model fell under the initial expectations.

5

Conclusion

Contents

5.1 Revision of the work done

In this Dissertation, I had the goal of understanding the current state-of-the-art deep clustering techniques and subsequently following a novel approach to tackle this difficult task. The result was HRWAE-GAN, a generative autoencoder with a hierarchical latent space (it contains a set of latent spaces $Z = \{z_0, ..., z_{L-1}\}$, instead of just one) that is regularized by a set of latent relativistic discriminators in an adversarial training setup.

Overall, this model is very complex and challenging to implement without the reward of state-of-the-art performance. Although the hierarchical latent structure has shown to be beneficial for data generation [Vahdat and Kautz, 2020], this did not seem to be the case for clustering. Initially, there was hope that having several latent spaces could possibly be used in some advantageous way. This could happen if the model was consistent in its way of organizing each latent space (i.e, it always utilized the same logic in z_0, z_1, etc. to organize the data points), but that was not the case. It's also important to mention that the added complexity of the hierarchical architecture makes training significantly more unstable, even when using the regularization techniques recommended in NVAE [Vahdat and Kautz, 2020].

Given that much better results have been achieved by recent works like SPC [Mahon and Lukasiewicz, 2021] with much simpler and more intuitive architectures, the path followed in this thesis does not seem to be recommendable.

5.2 Future Work

Although the hierarchical approach fell under initial expectations regarding clustering performance, its image generation qualities are still of potential interest, as was shown in [Vahdat and Kautz, 2020]. To this effect, I believe the most important aspect that is missing is a mechanism to control the images that are generated.

In InfoGAN [Chen et al., 2016], the authors suggest that feeding some extra information c to the generator G, who's function becomes $G(z, c)$ where z is the latent code, and having an extra neural network Q reconstructing c based on G's output can provide this desired functionality of controlling the images generated (see subsection 2.2, paragraph E for more detailed information about InfoGAN). With this in mind, it would make sense to incorporate this extra component to generative models such as HRWAE-GAN.

To improve clustering performance, the recent state-of-the-art work like SPICE [Niu and Wang, 2021] and SPC [Mahon and Lukasiewicz, 2021] suggests adopting a semi-supervised approach based on the models' pseudo-labels can highly increase clustering accuracy. Still, it's worth noting that generating pseudo-labels comes with the cost of increased training time.

Bibliography

[Agarap and Azcarraga, 2020] Agarap, A. F. and Azcarraga, A. P. (2020). Improving k-means clustering performance with disentangled internal representations. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[Arjovsky and Bottou, 2017] Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*.

[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*.

[Bishop, 2006] Bishop, C. M. (2006). Pattern recognition. *Machine learning*, 128(9).

[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.

[Campello et al., 2013] Campello, R. J., Moulavi, D., and Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer.

[Campello et al., 2015] Campello, R. J., Moulavi, D., Zimek, A., and Sander, J. (2015). Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(1):1–51.

[Cao et al., 2020] Cao, L., Asadi, S., Zhu, W., Schmidli, C., and Sjöberg, M. (2020). Simple, scalable, and stable variational deep clustering. *arXiv preprint arXiv:2005.08047*.

[Chang et al., 2017] Chang, J., Wang, L., Meng, G., Xiang, S., and Pan, C. (2017). Deep adaptive image clustering. In *Proceedings of the IEEE international conference on computer vision*, pages 5879–5887.

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

[Chen et al., 2017] Chen, D., Lv, J., and Zhang, Y. (2017). Unsupervised multi-manifold clustering by learning deep representation. In *Workshops at the thirty-first AAAI conference on artificial intelligence*.

[Chen et al., 2005] Chen, L., Singer, B., Guirao, A., Porter, J., and Williams, D. R. (2005). Image metrics for predicting subjective image quality. *Optometry and Vision Science*, 82(5):358–369.

[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 *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2180–2188.

[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.

[Dilokthanakul et al., 2016] Dilokthanakul, N., Mediano, P. A., Garnelo, M., Lee, M. C., Salimbeni, H., Arulkumaran, K., and Shanahan, M. (2016). Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*.

[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.

[Ghasedi et al., 2019] Ghasedi, K., Wang, X., Deng, C., and Huang, H. (2019). Balanced self-paced learning for generative adversarial clustering network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4391–4400.

[Ghasedi Dizaji et al., 2017] Ghasedi Dizaji, K., Herandi, A., Deng, C., Cai, W., and Huang, H. (2017). Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *Proceedings of the IEEE international conference on computer vision*, pages 5736–5745.

[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. *Advances in neural information processing systems*, 27.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.

[Guo et al., 2017a] Guo, X., Gao, L., Liu, X., and Yin, J. (2017a). Improved deep embedded clustering with local structure preservation. In *Ijcai*, pages 1753–1759.

[Guo et al., 2017b] Guo, X., Liu, X., Zhu, E., and Yin, J. (2017b). Deep clustering with convolutional autoencoders. In *International conference on neural information processing*, pages 373–382. Springer.

[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*, 32(9):1680–1693.

[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*, pages 550–565. PMLR.

[Harchaoui et al., 2017] Harchaoui, W., Mattei, P.-A., and Bouveyron, C. (2017). Deep adversarial gaussian mixture auto-encoder for clustering.

[Heusel et al., 2017] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.

[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.

[Hore and Ziou, 2010] Hore, A. and Ziou, D. (2010). Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE.

[Hsu and Lin, 2017] Hsu, C.-C. and Lin, C.-W. (2017). 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*, pages 1558–1567. PMLR.

[Ji et al., 2019] Ji, X., Henriques, J. F., and Vedaldi, A. (2019). Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9865–9874.

[Jiang et al., 2016] Jiang, Z., Zheng, Y., Tan, H., Tang, B., and Zhou, H. (2016). Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*.

[Jolicoeur-Martineau, 2018] Jolicoeur-Martineau, A. (2018). The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

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

[Kopf et al., 2019] Kopf, A., Fortuin, V., Somnath, V. R., and Claassen, M. (2019). Mixture-of-experts variational autoencoder for clustering and generating from similarity-based representations.

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

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

[Li et al., 2018b] Li, X., Chen, Z., Poon, L. K., and Zhang, N. L. (2018b). Learning latent superstructures in variational autoencoders for deep multidimensional clustering. *arXiv preprint arXiv:1803.05206*.

[Lim et al., 2020] Lim, K.-L., Jiang, X., and Yi, C. (2020). Deep clustering with variational autoencoder. *IEEE Signal Processing Letters*, 27:231–235.

[Litjens et al., 2017] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., Van Der Laak, J. A., Van Ginneken, B., and Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88.

[Mahon and Lukasiewicz, 2021] Mahon, L. and Lukasiewicz, T. (2021). Selective pseudo-label clustering.

[Martinez and Kak, 2001] Martinez, A. M. and Kak, A. C. (2001). Pca versus lda. *IEEE transactions on pattern analysis and machine intelligence*, 23(2):228–233.

[McConville et al., 2021] McConville, R., Santos-Rodriguez, R., Piechocki, R. J., and Craddock, I. (2021). N2d:(not too) deep clustering via clustering the local manifold of an autoencoded embedding. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 5145–5152. IEEE.

[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*.

[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. *Neural Networks*, 130:206–228.

[Mukherjee et al., 2019] Mukherjee, S., Asnani, H., Lin, E., and Kannan, S. (2019). Clustergan: Latent space clustering in generative adversarial networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4610–4617.

[Niu and Wang, 2021] Niu, C. and Wang, G. (2021). Spice: Semantic pseudo-labeling for image clustering.

[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.

[Qian, 1999] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151.

[Radford et al., 2015] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

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

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR.

[Salimans et al., 2016] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242.

[Shah and Koltun, 2018] Shah, S. A. and Koltun, V. (2018). Deep continuous clustering. *arXiv preprint 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. *arXiv preprint arXiv:1801.01587*.

[Springenberg, 2015] Springenberg, J. T. (2015). Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*.

[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., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[Thung and Raveendran, 2009] Thung, K.-H. and Raveendran, P. (2009). A survey of image quality measures. In *2009 international conference for technical postgraduates (TECHPOS)*, pages 1–4. IEEE.

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

[Tzoreff et al., 2018] Tzoreff, E., Kogan, O., and Choukroun, Y. (2018). Deep discriminative latent space for clustering. *arXiv preprint arXiv:1805.10795*.

[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.

[Vahdat and Kautz, 2020] Vahdat, A. and Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. *arXiv preprint arXiv:2007.03898*.

[Van der Maaten and Hinton, 2008] Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

[Wattenberg et al., 2016] Wattenberg, M., Viégas, F., and Johnson, I. (2016). How to use t-sne effectively. *Distill*, 1(10):e2.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/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*, pages 478–487. PMLR.

[Xu and Wunsch, 2005] Xu, R. and Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678.

[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*, pages 3861–3870. PMLR.

[Yang et al., 2016] Yang, J., Parikh, D., and Batra, D. (2016). Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5147–5156.

[Yang et al., 2019] Yang, L., Cheung, N.-M., Li, J., and Fang, J. (2019). Deep clustering by gaussian mixture variational autoencoders with graph embedding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6440–6449.

[Zhang et al., 2017] Zhang, D., Sun, Y., Eriksson, B., and Balzano, L. (2017). Deep unsupervised clustering using mixture of autoencoders. *arXiv preprint arXiv:1712.07788*.