A Mixture-of-Experts approach to deep image clustering: Estimating latent sizes and number of clusters

PEDRO SANTOS

***Abstract*—Deep clustering is a field with many widespread applications, ranging from Biology [1] to Marketing [2]. It differs from classical clustering by using a deep algorithms (i.e. autoencoders) to perform representation learning on the raw data. Despite the importance of these deep representation learning algorithms, little to no priority is given to their structure, making most theoretical works fail in real-life applications and most bodies of work do not allow specialization of parts of the network to subsets of data, making it very hard to influence the type of data that is generated. Hence this thesis aims at exploring a novel deep clustering technique based on a mixture of variational autoencoders in which each VAE models a cluster, and a manager network gives, based on the data, a relative importance score to each of the experts. The main contributions of this body of work are fourfold: it is a generative approach to clustering, it has a fully data dependant architecture, removing the need for most hyperparameter selection, the latent dimension finder is a novel approach to determine the optimal number of neurons to have in the bottleneck layer of an autoencoder, and finally by using an algorithm such as HDBSCAN instead of a classical clustering algorithm allows us to have a better algorithm initialization and automatically define the optimal number of clusters for the MoE architecture. The results of the experiments done to evaluate our algorithm's performance were very positive: it surpasses most state-of-the-art deep clustering techniques such as N2D [3], and DynAE [4] and becomes the definitive baseline for Mixture-of-Experts based clustering, far surpassing previous baselines such as DAMIC [5] and MIXAE [6].**

Keywords - Deep image clustering, Image Generation, Mixture of experts model, Pretraining architecture, Automatic determination of the number of clusters, optimization of the latent space dimensionality

I. Introduction

Unsupervised clustering is both one of the fundamental motivators, and one of the most challenging tasks in machine learning. By grouping data in a meaningful manner, we can gain insights and knowledge that is often valuable and not straightforward by regular data analysis. This task gets exponentially harder for traditional methods such as K-means and Gaussian mixture models when we are talking about high dimensional data such as video or images mainly due to the curse of dimensionality [7].

With the advent of deep learning, architectures such as autoencoders somehow alleviated the challenges of higher-dimensional clustering by using what is called Representation Learning. By training a network to reconstruct data using an information bottleneck, it can learn a latent representation of the data that not only has lower dimensionality but also is optimized and deduced from the data itself. This training allows us to, automatically and in an unsupervised manner, extract the essential features underlying the data, making clustering easier. In recent years several architectures have used autoencoders, variational autoencoders (VAEs) [8], adversarial autoencoders (AAEs) [9], and Generative adversarial networks (GANs) [10] as a way to perform representation learning with moderate to great performances [3] [4] [11].

But this is often not enough. Suppose we assume that any given dataset is a union of low dimensional latent representations. In that case, we need to find a way to, given a particular input x_i, not only know from which latent model it belongs to but also to provide ways to generate from and visualize the multiple latent representations. We also have a problem of hyperparameters: for most theoretical works the architecture topology is left at the author's discretion, often being an afterthought and not working or having a bad performance for real-life datasets.

We built an original architecture based on the mixture-of-experts framework. Each expert models and generates a separate cluster of data, and the data distribution is controlled by a manager. We used a fully dynamic architecture in which the depth of the network, the latent size of the experts, and the number of experts used to model the data are all data dependant. Hence, the main contributions of this dissertation are as follows:

- **Generative approach to clustering -** The architecture is generative, i.e., we can generate data from each of the clusters separately. This ability to create per-cluster data presents a new approach for understanding not only how the algorithm groups data, but what prominent features from the data are used when making these decisions.
- **Mixture of experts** - The proposed architecture surpasses state-of-the-art mixture of experts clustering approaches by several percentage points, becoming the *de-facto* model for this type of framework.
- **Automatic clustering estimation** - The proposed model also features an automatic number-of-centroid finder based on the HDBSCAN and the N2D architecture. With this feature in place, there is no need to hardcode the appropriate number of clusters and to perform previous data exploration.
- **Automatic autoencoder latent size finder** - One of the most crucial problems when using autoencoder-like architectures is knowing the number of neurons in the latent space often being left as a hyperparameter. We devised a methodology based on PCA decomposition

that eliminates the need for guesswork. An estimate of the optimal latent size is made based on the explained variance of the PCA decomposition of the data at the autoencoder bottleneck.

- **Comparison of autoencoders in the architecture -** We also compared several autoencoders to determine the best architecture for clustering purposes, always relating each result to the theoretical background of each architecture.

II. Related Work

A. Clustering

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

Classical clustering: Classical clustering algorithms involve all methodologies that do not use deep learning. These were the original methodologies to perform clustering and often are the final step in deep learning clustering approaches. We can sub-divide these methods into four main groups: Connectivity-based clustering, Centroid-based clustering, Density-based clustering and Distribution-based clustering. Connectivity-based clustering defines a distance function between objects and a maximum distance by which a connection between objects constitutes a cluster, and hierarchically clusters them, using a dendrogram. In centroid-based clustering, we optimize the cluster centres instead of the distance between individual data points. Optimization, in this case, is done by attempting to minimize the intra-cluster distance while maximizing the inter-cluster reach, leaving the borders of the proposed clusters unoptimized. In distribution-based clustering, we assume that we can express the whole dataset as a set of probability distributions, where points on the same cluster are likely to have been generated from the same probability distribution. The final type of clustering methodology is Density-based clustering. Here clusters are defined by regions with higher data point density than the rest of the space, and areas with lower point density represent noise/outliers.

Deep clustering: In recent years, advances in deep learning architectures over a multitude of fields, the availability of vast amounts of data, and the increase of processing power of computers have galvanized the use of artificial neural networks for computational tasks. By leveraging a neural network, we can perform feature selection and clustering in an end to end manner that surpasses most classical clustering approaches. Most deep learning algorithms use a deep learning architecture to perform representation learning and then a shallow clustering algorithm on the learned manifold to get the labels for the data. Depending on the deep learning architecture, we can subdivide the algorithms into four groups:

- **A**utoencoder-based clustering Autoencoders [12] are a type of Neural Network used for efficient data compression and dimensionality reduction in an unsupervised manner. It is composed of 3 main blocks: an encoder, a latent dimension, and a decoder. The encoder aims to learn a low dimensional representation of the input data, by discarding input noise and attempting to learn relevant features and patterns of the data. On the opposite side, we have a decoder that does the opposite: from the low dimensional code, it attempts to reconstruct the data as close as possible to the input data. The latent dimension is the reduced code that we get by encoding the data. The main idea of autoencoder based clustering is to use an autoencoder to reduce the dimensionality of the data while using a clustering loss to group the low dimensional space. Most of these approaches use a pretraining scheme based on the reconstruction loss before applying the clustering loss. Popular methodologies include but are not limited to DEC [13], Deep embedded regularized clustering [14], N2D [3] and DynAE [4].
- **Generative model-based clustering** As the name suggests, generative model-based clustering uses generative models like VAEs in an architecture. These types of models can sample the representation space to generate new samples from the underlying data distribution. Despite the VAE having autoencoder in its name, the two should not be confused. The autoencoder maps the input data into a latent variable space z, often compressing the data, and then proceeds to decode the latent variable back into the input data. In contrast, the variational autoencoder tries to predict the probability distribution of the input variable, and from this distribution, get the input data back. So we have a dataset that has an associated probability density function $P(X)$, a latent vector z with an associated probability density function $P(z)$ and a set of deterministic functions $f(z;\theta)$ that map the latent variable back to the dataset X, where θ is the optimization parameter. More precisely, we are attempting to maximize the probability of each X_i in the dataset via the following expression:

$$P(X) = \int_z P(X|z;\theta)P(z)dz \tag{1}$$

where $f(z;\theta) = P(X|z;\theta)$ due to the maximum likelihood principle. In VAEs, the choice of output distribution is often made Gaussian with mean $f(z;\theta)$ and covariance matrix equal to the identity multiplied by a scalar σ (hyperparameter). The choice of this function is because, with this distribution, we can easily perform gradient descent and train the model to make adjustments to θ to make the model approximate $P(X)$. Despite this theoretical background being solely focused on standard Gaussians, several authors have also already shown how to optimize this type of networks to several other probability distributions [15], [16]. Examples of these types of algorithms are VaDE [17], InfoGAN [18] and ClusterGAN [19]

- **Direct cluster optimization based clustering** Cluster optimization algorithms only use clustering loss to optimize the latent space, skipping over the reconstruction loss. Algorithms include JULE [20] and DAC [21].
- **Mixture of experts clustering** MoE clustering can be considered a subset of autoencoder based clustering, and in some cases, can also be perceived as a generative approach. However, due to the main focus on this paper being an architecture of this kind, it required a separate section. MoE clustering is based on a set of independent neural networks called experts, and a balancing system called a manager (Figure 1). The experts' objective is to specialize in one subset of the data while the manager aims to ensure that the correct expert is chosen for each data point. The main contributions in this field are DAMIC [5], MIXAE [6] and MoeSim-VAE [22]

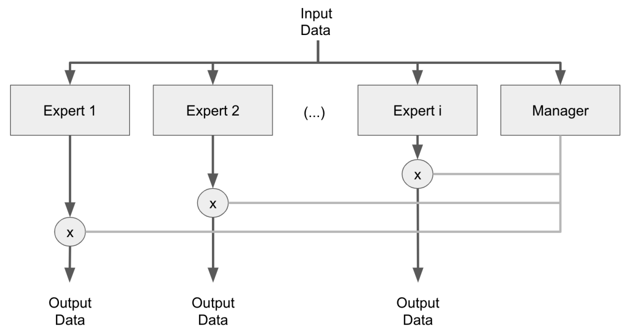

Fig. 1: MoE general architecture

In Table 1, we provide an overview of the performance of deep clustering algorithms on the MNIST dataset. To collect this data, we consulted several surveys, such as [23], [24] [25] and [26]. As seen from the table, the type of result by which the performance is inferred varies greatly from model to model. It can be an average of multiple runs, the best of a subset of runs or even unspecified. This discrepancy of ways to get final results leaves a lot of margin for error. It also reduces the credibility of algorithms, since there isn't a real base standard for evaluating clustering performance. Still, at the time of this dissertation's writing, the best algorithms for clustering are N2D, DynAE and ASPC-DA, all autoencoder-based methods. The MoE based methods are far from the state-of-the-art (by MNIST standards), with very few works using this framework to perform clustering.

III. The architecture

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

TABLE I: Comparison of state-of-the-art methods based on network architecture and MNIST results reported in the original papers. DNN - Deep Neural Network; AE - Autoencoder; VAE - Variational Autoencoder; GAN - Generative Adversarial Network; MoE - Mixture of Experts

Type of Arch.	Method	Pre-train?	MNIST		
			Acc	NMI	Type of result
DNN	JULE [20]		–	0.91	avg 3 trials
	IMSAT [27]		98.4%	–	avg 12 trials
	DAC [28]		97.8%	0.94	–
	CCNN [29]	yes	–	0.88	–
	SpectralNet [11]		97.1%	0.92	–
	IIC [30]		98.4%	–	–
AE	DEC [13]	yes	84.3%	–	best 20 trials
	DMC [31]	yes	–	0.86	avg
	DCN [32]	yes	83.0%	0.81	–
	IDEC [33]	yes	88.1%	0.87	–
	DEPICT [14]	yes?	96.5%	0.92	avg 5 trials
	DCEC [34]	yes	89.0%	0.89	–
	DCC [35]	yes	96.2%	0.91	–
	Tzoreff et al. [36]	yes	97.4%	–	–
	DEC-DA [37]	yes	98.5%	0.96	avg 5 trials
	DBC [38]	yes	96.4%	0.92	–
	ASPC-DA [39]	yes	98.8%	0.97	avg 5 trials
	Yang et al. [40]	yes	97.8%	0.94	avg 10 trials
	BAE [41]	yes	83.7%	0.81	best 5 trials
	N2D [3]		97.9%	0.94	–
	DynAE [4]	yes	98.7%	0.96	–
	DERC [42]	yes	97.5%	0.93	–
VAE	GMVAE [43]		96.9%	–	best
	VaDE [17]	yes	94.5%	–	best 10 trials
	Figueroa et al. [44]		85.8%	0.82	best
	LTVAE [45]	yes	86.3%	0.83	best 10 trials
	DGG [46]	yes	97.6%	–	–
	VIB-GMM [47]	yes	96.1%	–	best 10 trials
	S3VDC [48]	yes?	93.6%	–	avg 5 trials
GAN	CatGAN [49]		95.7%	–	–
	InfoGAN [18]		95.0%	–	–
	DAC [21]		94.1%	–	median 10 trials
	ClusterGAN [19]		95.0%	0.89	best 5 trials
	ClusterGAN [50]		96.4%	0.92	avg 5 trials
MoE	MIXAE [6]		85.6%	–	–
	DAMIC [5]	yes	89.0%	0.87	avg 5 trials
	MoE-Sim-VAE [22]		97.5%	0.94	–

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

$$f(x_i) = \sum_{k=1}^{K} \pi_k(x_i) f_k(x_i) \tag{2}$$

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

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

A. *Training*

The training architecture is represented in Figure 2.

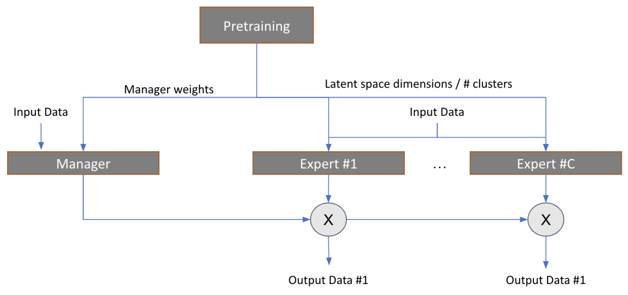

Fig. 2: Diagram of the full architecture

It consists onset of K experts controlled by a manager that, depending on the input data x_i, gives an importance value $\pi_k(x_i) \in [0;1]$ to each expert on the model, as per the DAMIC architecture [5]. However, this approach has a central problem: if left as-is we have cluster collapse: the experts compete instead of cooperating, leading to one expert specializing in the whole dataset, reducing clustering performance. To attenuate this effect we took a page from the MIXAE architecture [6] and changed the vanilla loss function by adding batch entropy and sample entropy.

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

$$R(\theta) = \frac{1}{\eta} \sum_{i=0}^{\eta} \sum_{k=0}^{K} p_k^i * d(x_i, \hat{x}_k^i) \tag{3}$$

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

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

$$S(\theta) = -\frac{1}{\eta} \sum_{i=0}^{\eta} \sum_{k=0}^{K} p_k^i * log(p_k^i) \tag{4}$$

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

$$B(\theta) = \sum_{k=0}^{K} \hat{p}_k * log(\hat{p}_k) \tag{5}$$

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

So the overall loss function will be as follows:

$$L(\theta) = R(\theta) + \alpha S(\theta) + \beta B(\theta) \tag{6}$$

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

B. *Pretraining*

The pretraining architecture (Figure 3) is where most of the innovation of this architecture is established.

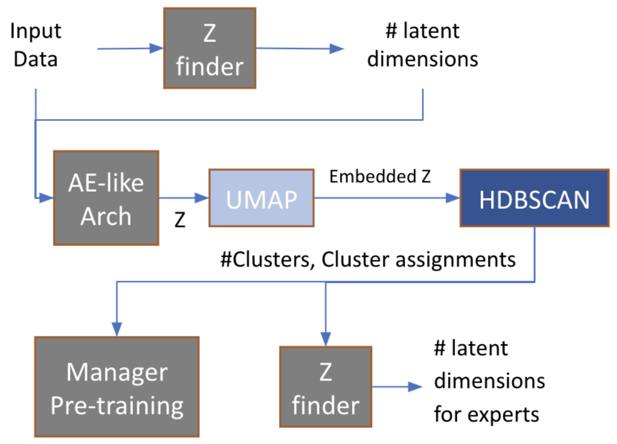

Fig. 3: Diagram of the pretraining architecture

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

1) Train an autoencoder-like architecture with a predetermined large latent dimension size (e.g. 100) on the full dataset.
2) Apply a latent dimension finder on the latent space of the VAE architecture and find the optimal latent dimension size
3) Retrain the VAE architecture with this new latent size on the input dataset.
4) Apply UMAP on the latent manifold of the trained VAE architecture.
5) Apply HDBSCAN on the manifold outputted from the UMAP block, getting a vector of cluster assignments for each datapoint and the number of clusters.

6) Excluding the noisy samples, train the manager using the provided labels and original data.
7) Use the latent dimension finder on each subset of data (each cluster found in the HDBSCAN block) find the optimal latent dimension for each expert
8) Using the found number of clusters and optimal latent dimensions construct the MoE architecture.
9) Run the MoE architecture with the pretrained manager.

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

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

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

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

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

As shown above, the optimal number of dimensions does not change: for a large enough dimension, it uses the minimum number of components to encode necessary features (high variance), allowing us to find the optimal latent size for autoencoders. In practice, we attempt to retain around 90% of the explained variance for clustering and 95% for data reconstruction. The latent dimension finder block is represented in figure 4

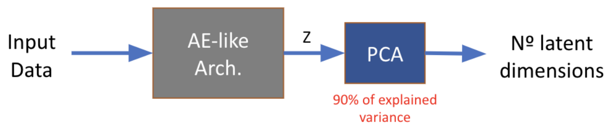

Fig. 4: Diagram of the Dynamic latent space finder

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

UMAP: After determining the optimal latent space, we retrain the VAE architecture with the optimal latent dimension. We use a UMAP transform on the latent dimension of the encoder to make the low dimensional space more clusterable and overall improve clustering results, as seen in [3].

In N2D [3], it was shown that the use of UMAP on the latent space of a dense autoencoder, without dimensionality reduction, improves classification, achieving the state of the art results in most image datasets. The reason for this behaviour is that UMAP can cluster the data taking into account the global structure of the data, providing better inter-cluster separability and intra-cluster compactness.

HDBSCAN for automatic cluster and outlier detection: Most works in deep clustering, be it MoE based architectures such as DAMIC [6] or otherwise, such DynAE [4] use a shallow clustering algorithm at the end of the pretraining stage to get initial clustering assignments. The two most widespread methodologies are the K-Means and the GMM. Still, both of them have several problems such as no resistance to outliers, points between clusters being mislabelled. They also do not support highly irregular cluster shapes and the fact that the number of clusters needs to be fed to the algorithm. The usage of HDBSCAN as the final clustering algorithm of the pretraining stage presents two distinct advantages when compared to a regular GMM or KMeans approach. On the one hand, it automatically detects the appropriate number of clusters that best fit the data, removing the need to know a priori how many groups the data has, or to find the best amount of clusters using silhouette coefficients/gap statistics. On the other hand, it presents a noise vector: points which the algorithm considers outliers are put in a separate cluster that is not labelled. This outlier detection mode can be useful if we do not pretrain the network in noisy points: it allows the experts to initialize their latent spaces better, and the conflicting data can be introduced during training.

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

At the end of the pretraining phase, we have three different elements: the weights of the manager, the latent space dimension of the experts and the number of experts in the training architecture. We pass these elements onto the training architecture as seen in figure 2, solving the problems brought up in the MoE architecture.

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

C. Dynamic architecture features

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

Convolutional architecture: It has been shown [51] [52] that by mixing different kernel sizes we can have more dynamic architectures that better model image features, so we created two distinct blocks: general and specific. General blocks reconstruct general/significant features of the image, and specific blocks reconstruct particular features. These different blocks are important since different kernel sizes/number of filters can map different features of an image. Smaller kernel sizes map smaller image details whereas larger kernel sizes map overall image structure, improving latent space separability and image reconstruction. The two most widespread choices for filters are 3x3 or 5x5 due to memory and simplicity sake. We chose to use two 3x3 blocks instead of 5x5 since both have the same receptive field, but the two 3x3 blocks have less mathematical operations, leading to lower training times as found in [53]. We also use Batch Normalization [54] before the activation function, according to [55]. The to the two primary blocks of our autoencoder-like networks can be seen in figure 5.

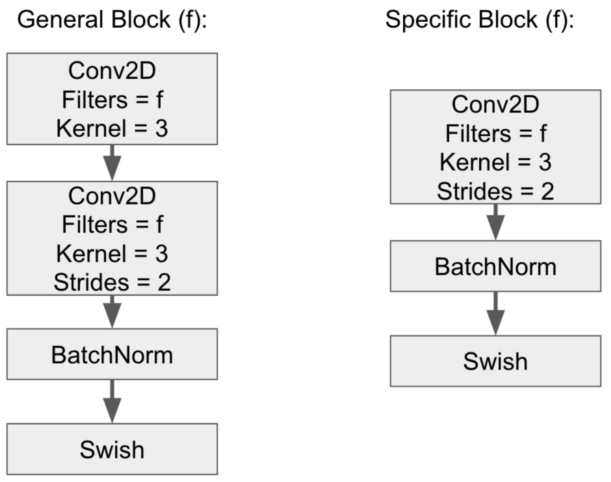
General Block (f):
Conv2D Filters = f Kernel = 3
Conv2D Filters = f Kernel = 3 Strides = 2
BatchNorm
Swish
Specific Block (f):
Conv2D Filters = f Kernel = 3 Strides = 2
BatchNorm
Swish

Fig. 5: Diagram of the Convolutional blocks

Depth of the architecture: A network that works well for a 16x16 image dataset such as USPS may not work as well for a dataset such as COIL20 that has 128x128 sized images. To tackle this, we made the depth of the network, and the filters depend on the image size. Assuming that an image has dimensions DxD, we can calculate the number of blocks in the encoder like:

$$N_{Blocks} = log_2(D) - 1 \tag{7}$$

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

$$N_{general} = \lceil N_{Blocks}/2 \rceil \tag{8}$$

$$N_{specific} = N_{Blocks} - N_{general} \tag{9}$$

The number of filters is given by multiplying D by two every for each block of the encoder. The autoencoder-like architecture can be summarized like so:

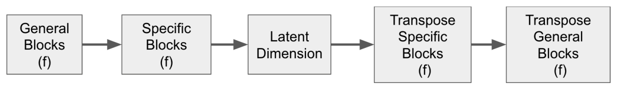

Fig. 6: Diagram of the Autoencoder-like architecture

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

IV. Experimental Results

A. Datasets

We compare our architecture with an array of various clustering algorithms in 5 baseline datasets: MNIST [56], FMNIST [57], USPS [58], CIFAR10 [59] and COIL20 [60].

- **MNIST**: A dataset of 70000 images of handwritten digits separated into ten classes. Each sample is a 28x28 grayscale image.
- **FMNIST**: A dataset of 70000 images of fashion items separated into ten classes. Each sample is a 28x28 grayscale image.
- **USPS**: A dataset of 9298 images of handwritten digits separated into ten classes. Each sample is a 16x16 grayscale image.
- **CIFAR10**: A dataset of 60000 images of handwritten digits separated into ten classes. Each sample is a 32x32x3 RGB image.
- **COIL20**: A dataset of 1440 images of handwritten digits separated into ten classes. Each sample is a 128x128 grayscale image.

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

B. Pretraining Results

At each step of the architecture, as mentioned earlier, we used a VAE with data augmentation. Data augmentation consists of random rotations, shifts, and crops on an image. We use this method in supervised learning as a form of regularization. With data augmentation, we have more data and variability between samples, which allows for better feature generalization cluster distinction. The main idea is that when we perform augmentation, the augmented data shares the same manifold/probability distribution of the input data. This augmentation allows the data manifold/distribution to become smoother, especially in datasets with few samples. We used a rotation range of 10 degrees for all datasets, a zoom range of 0.1 and width and height shift ranges of 0.1. For the FMNIST, COIL20, and CIFAR10 dataset, we also use horizontal flips.

Each instance of VAE training consisted of a learning rate of 5^{-4}, a batch size of 100 and training for 1000 epochs with the loss being the standard VAE loss. We used UMAP with a dynamic number of neighbours that according to the following formula

$$n\_neighbours = max\{int\Big(\frac{dataset\_size}{300}\Big), 100\} \qquad (10)$$

and the HDBSCAN had default parameters. At the end of the pretraining stage the results for the MNIST and FMNIST datasets were as follows (Figure II)

TABLE II: Results clustering the pretrained data

	MNIST	FMNIST
N° experts	10	11
% of dataset labeled	0.96	0.48
Accuracy on labeled dataset	0.978	0.673
GMM accuracy for the dataset	0.956	0.583

Just this clustering alone already gives outstanding clustering results: by integrating the manifold learning techniques of UMAP and the density-based cluster estimation of HDBSCAN we could not only find a number of clusters that is very close to the theoretical optimum, but also increase the clustering accuracy of the labelled area when compared to a GMM. Afterwards, we used the labels provided by this step to train the manager network.

C. Manager training Results

The manager network was trained with a learning rate of 10^{-3}, batch size of 100 and for 1000 epochs with the loss being the kullback-Leiber distance between the soft assignments produced by the manager and the soft cluster labels produced by the shallow clustering methods. This loss function choice was not random. We wanted to train the manager with a soft probability output to make samples that are close to two clusters (for example a four that looks like a 9 in the MNIST dataset) remain in between two clusters. This noise vector reduces sample misclassification and allows for better expert initialization in the training phase. The results for each dataset are present in Table III.

As we can see, training the manager using only the HDBSCAN labels makes the manager's accuracy improve on most datasets. This performance improvement is mainly because the HDBSCAN has far fewer mislabelled samples, reducing possible errors in the manager training and improving clustering performance on the whole dataset.

D. Training Results

In this last step, we use the entropy losses mixed with the standard VAE loss to progressively harden assignments and achieve one-hot labels for each data point. The full training stage is composed of 100 epochs, and the architecture was trained with a learning rate of 10^{-4} and a batch size of 100. The hyperparameters were set to $\alpha = 100$ and $\beta = 1$. The average of five runs' final accuracy, NMI and ARI results for the whole architecture and MNIST and FMNIST is present in table IV

TABLE III: Results of manager training on each of the datasets

	MNIST			FMNIST		
	ACC	NMI	ARI	ACC	NMI	ARI
Manager	0.967	0.922	0.927	0.623	0.685	0.528

TABLE IV: Accuracy for all mixture of experts models

	MNIST	FMNIST
DEC	0.89	0.518
N2D	0.97	0.672
DynAE	0.98	0.591
DAMIC	0.89	0.60
MIXAE	0.85	-
MoE	0.97	0.68

As we can see, our model far surpasses these two MoE architectures in the two datasets for all metrics. These results provide a way for our algorithm to be a definitive baseline in MoE based clustering: all of our design decisions are justified, and we have results to back them up. For the autoencoder based clustering architectures, we once again see that our algorithm presents the state of the art accuracy results on all given datasets from these results. It far surpasses DEC and is on par with the DynAE approach. N2D only outperforms it in the pretraining stage on the FMNIST dataset, which seems strange because our pretraining stage is essentially an N2D with a few modifications. Since there is no information on how the results on N2D were achieved (e.g. whether the results are an average across runs or a best case) and we couldn't reproduce these results with the authors' proposed architecture, we assumed that it is a best of multiple runs. Still, our pretraining method surpasses or equals the N2D approach for the rest of the datasets.

E. Expertise of the experts

Expertise of reconstruction: To determine how each expert specializes in the final architecture, we reconstructed an input image where every pixel value is equal to one for the USPS dataset (Similar to MNIST). The results are present in Figure 7 where the leftmost image is the original image, and the rest are the reconstructions made by each expert.

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

Expertise of data generation: By feeding random multivariate Gaussian vectors with the same length as the expert's latent dimension, we can generate new samples from the

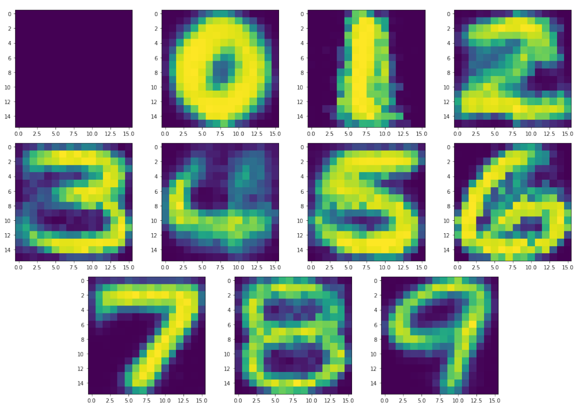

Fig. 7: Reconstructed images

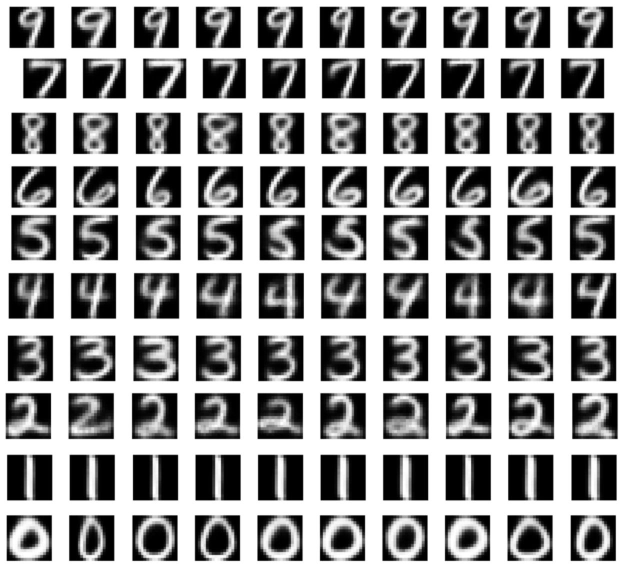

Fig. 8: Images generated from the experts trained on the USPS dataset

inherent distribution of the network. By applying this process to the experts trained in the USPS dataset, we obtained the samples in Figure 8, where each line represents the generated images from 10 random vectors taken from a multivariate Gaussian with zero mean and diagonal unitary covariance matrix.

As seen from the figure, every single expert once again specializes in one specific digit/class and provides high-quality image generation. One advantage of the latent space of the experts is that not only is it continuous, but by sampling it along its dimensions makes the data generated gradually change. For example, suppose we perform a walk between -1 and 1 on the first dimension of the variational autoencoder trained on the USPS dataset. In that case, we have the following images (Figures 9 and 10).

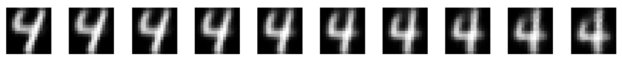

Fig. 9: Images generated from the expert specialized in fours

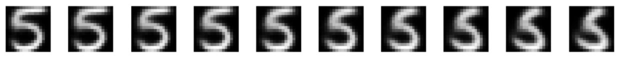

Fig. 10: Images generated from the expert specialized in fives

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

V. Conclusions

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

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

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

References

[1] K. de Queiroz and D. A. Good, "Phenetic clustering in biology: a critique," *The Quarterly Review of Biology*, vol. 72, no. 1, pp. 3–30, 1997.

[2] K. R. Harrigan, "An application of clustering for strategic group analysis," *Strategic Management Journal*, vol. 6, no. 1, pp. 55–73, 1985.

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

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

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

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

[7] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *Database Theory — ICDT 2001*, J. Van den Bussche and V. Vianu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 420–434.

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

[9] A. Makhzani, J. Shlens, N. Jaitly, and I. Goodfellow, "Adversarial Autoencoders," in *International Conference on Learning Representations (ICLR) - Workshop track*, 2016.

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

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

[12] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

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

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

[15] Y. Jin, L. Du, L. Gao, Y. Xiang, Y. Li, and R. Xu, "Variational autoencoder based bayesian poisson tensor factorization for sparse and imbalanced count data," 2019.

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

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

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

[19] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, "ClusterGAN: Latent Space Clustering in Generative Adversarial Networks," in *AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4610–4617.

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

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

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

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

[24] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long, "A Survey of Clustering With Deep Learning: From the Perspective of Network Architecture," *IEEE Access*, vol. 6, pp. 39 501–39 514, 2018.

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

[26] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long, "A survey of clustering with deep learning: From the perspective of network architecture," *IEEE Access*, vol. 6, pp. 39 501–39 514, 2018.

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

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

[29] C.-C. Hsu and C.-W. Lin, "CNN-Based Joint Clustering and Representation Learning with Feature Drift Compensation for Large-Scale Image Data," *IEEE Transactions on Multimedia*, vol. 20, no. 2, pp. 421–429, 2018.

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

[31] D. Chen, J. Lv, and Z. Yi, "Unsupervised Multi-Manifold Clustering by Learning Deep Representation," in *AAAI Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017, pp. 385–391.

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

[33] X. Guo, L. Gao, X. Liu, and J. Yin, "Improved Deep Embedded Clustering with Local Structure Preservation," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017, pp. 1753–1759.

[34] X. Guo, X. Liu, E. Zhu, and J. Yin, "Deep Clustering with Convolutional Autoencoders," in *Neural Information Processing, ICONIP*, 2017, vol. 10635, pp. 373–382.

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

[36] E. Tzoreff, O. Kogan, and Y. Choukroun, "Deep Discriminative Latent Space for Clustering," *arXiv:1805.10795*, 2018.

[37] X. Guo, E. Zhu, X. Liu, and J. Yin, "Deep Embedded Clustering with Data Augmentation," in *Asian Conference on Machine Learning (ACML)*, 2018, pp. 550–565.

[38] F. Li, H. Qiao, and B. Zhang, "Discriminatively boosted image clustering with fully convolutional auto-encoders," *Pattern Recognition*, vol. 83, pp. 161–173, 2018.

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

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

[41] P.-Y. Chen and J.-J. Huang, "A Hybrid Autoencoder Network for Unsupervised Image Clustering," *Algorithms*, vol. 12, no. 6, p. 122, 2019.

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

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

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

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

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

[47] Y. Uğur, G. Arvanitakis, and A. Zaidi, "Variational Information Bottleneck for Unsupervised Clustering: Deep Gaussian Mixture Embedding," *Entropy*, vol. 22, no. 2, p. 213, 2020.

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

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

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

[51] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385
[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” 2014.
[53] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842
[54] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
[55] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther, “Ladder variational autoencoders,” in *Advances in neural information processing systems*, 2016, pp. 3738–3746.
[56] Y. LeCun, C. Cortes, and C. Burges, “Mnist handwritten digit database,” *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.
[57] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.
[58] J. J. Hull, “A database for handwritten text recognition research,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, 1994.
[59] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” 2009.
[60] S. A. Nene, S. K. Nayar, H. Murase *et al.*, “Columbia object image library (coil-100),” 1996.