Top-Down Deep Clustering with Multi-generator GANs

Daniel de Mello [1], **Renato Assunção** [2, 1], **Fabricio Murai** [1]

[1] Department of Computer Science, Universidade Federal de Minas Gerais
[2] ESRI Inc.
dani-dmello@hotmail.com, rassuncao@esri.com, murai@dcc.ufmg.br

Abstract

Deep clustering (DC) leverages the representation power of deep architectures to learn embedding spaces that are optimal for cluster analysis. This approach filters out low-level information irrelevant for clustering and has proven remarkably successful for high dimensional data spaces. Some DC methods employ Generative Adversarial Networks (GANs), motivated by the powerful latent representations these models are able to learn implicitly. In this work, we propose HC-MGAN, a new technique based on GANs with multiple generators (MGANs), which have not been explored for clustering. Our method is inspired by the observation that each generator of a MGAN tends to generate data that correlates with a sub-region of the real data distribution. We use this clustered generation to train a classifier for inferring from which generator a given image came from, thus providing a semantically meaningful clustering for the real distribution. Additionally, we design our method so that it is performed in a top-down hierarchical clustering tree, thus proposing the first hierarchical DC method, to the best of our knowledge. We conduct several experiments to evaluate the proposed method against recent DC methods, obtaining competitive results. Last, we perform an exploratory analysis of the hierarchical clustering tree that highlights how accurately it organizes the data in a hierarchy of semantically coherent patterns.

Introduction

Cluster analysis is a fundamental problem in unsupervised learning, with a wide range of applications, especially in computer vision (Shi and Malik 2000; Achanta and Susstrunk 2017; Joulin, Bach, and Ponce 2010; Liu et al. 2018). Its goal is to assign similar points of the data space to the same cluster, while ensuring that dissimilar points are placed in different clusters. One of the main challenges in this approach is to quantify the similarity between objects. For low-dimensional data spaces, similarity might be straightforwardly defined as the minimization of some geometric distance (*e.g.* euclidean distance, squared euclidean distance, Manhattan distance). On the other hand, choosing the distance metric becomes unfeasible for high dimensional data distributions. Images are a clear example of this problem, since any distance metric based on raw pixel spaces is subject to all sorts of low-level noisy disturbances irrelevant for determining the similarity and suffer from a lack of translation or rotation invariance. This motivates the need for some dimensionality reduction technique, by which the fundamental relationships between objects projected onto the resulting embedding space become more consistent with geometrical distances.

In recent years deep clustering techniques have spearheaded the dimensionality reduction approach to clustering by employing highly non-linear latent representations learned by deep learning models (Krizhevsky, Sutskever, and Hinton 2012). Considering the unsupervised nature of cluster analysis, the models that naturally arise as candidates for deep clustering are unsupervised deep generative models, since these must learn highly abstract representations of the data as a requirement for realistic and diverse generated samples. One of such models are the Generative Adversarial Networks (GAN) (Goodfellow et al. 2014), whose extremely realistic results in image generation, semantic interpolation and interpretability in the latent space, are evidence of their capacity of learning a powerful latent representation that captures the essential components of the data distribution. Nonetheless, very few works have proposed GAN architectures designed for clustering. Some of these works are (Mukherjee et al. 2019; Chen et al. 2016), where the authors showed that, by manipulating the generator's architecture in a specific way, it is possible to control the class of the training data to which a generated sample belongs, even when classes labels are not available during the training.

Some recent works employed a GAN architecture with multiple generators (Ghosh et al. 2018; Hoang et al. 2018; Zhang et al. 2018) to achieve greater diversity in image generation, as well as an alternative way of stabilizing the training. In these works, the authors have observed that each generator tends to specialize in generating examples belonging to a specific class of the dataset. This suggests the organic emergence of clusters in the generators' representation of the data and it is one of the main motivations of our work. The rationale is that clustering would be possible by employing a classifier in charge of distinguishing between the generators, and this classifier could later be applied to the real dataset in order to classify real examples without the use of labels. Additionally, the problem of setting the number of generators to be used for representing the classes of the training set is

not addressed in these works, which could be an issue in a real clustering task, where the number of clusters is assumed to be unknown.

In this work, we propose Hierarchical Clustering MGAN (HC-MGAN), a method that leverages the multi-generator GAN for the clustering task. We employ multiple generators, each of them specializing in representing a particular cluster of the training distribution. This should lead to a stronger representation capacity and with more meaningful clusters than what a single generator covering multiple clusters can provide. MGANs have not been used in the previous works exploring GANs for clustering. Additionally, we design our method so that it performs the clustering of the training data in a top-down hierarchical way, creating new generators as divisions of subsequent clusters become necessary, *i.e.*, it permits the user to control different clustering granularity levels according to the task at hand. The main contributions of this work are as follows:

- We propose HC-MGAN, a novel deep clustering method that employs GANs with multiple generators.
- We design HC-MGAN as a top-down hierarchical clustering tree introducing the first hierarchical deep clustering algorithm. Hierarchical clustering allows the user to control different levels of cluster granularity and favors interpretability by taxonomically organizing the clusters.
- We conduct experiments with three image datasets used in other deep clustering works as benchmarks, namely, MNIST, Fashion MNIST and Stanford Online Products. We obtain competitive results against recent deep clustering methods, all of which are horizontal and therefore lack the advantages of the hierarchical approach.
- We explore the clustering pattern obtained throughout the tree, displaying how HC-MGAN is able to organize the data in a semantically coherent hierarchy of clusters.

Related Work

We review some recent work in topics related to Deep Clustering and GANs. Autoencoders (AEs) have been the most prevalent choice in the deep clustering literature (Xie, Girshick, and Farhadi 2016; Yang et al. 2017, 2019; Zhang et al. 2019b), where the clustering objective is usually optimized on the feature representation $\mathbf{Z}$ resultant of the mapping $E : \mathcal{X} \rightarrow \mathcal{Z}$ learned by the encoder component in AEs for a data space $\mathcal{X}$ and a feature space $\mathcal{Z}$. Deep Embedded Clustering (DEC) (Xie, Girshick, and Farhadi 2016) pioneered this approach, obtaining state-of-the art clustering results. It works by pretraining an AE with a standard reconstruction loss function, and then optimizing it with a regularizer based on the clustering assignments modeled by a target student-t distribution, having the cluster centers iteratively updated. DEC's results were surpassed by (Yang et al. 2017), which converted DEC's objective function into an alternated joint optimization with K-Means loss, thus obtaining a clusterization more suited for K-Means.

The use of GANs for clustering tasks has been influenced by InfoGAN (Chen et al. 2016), a type of GAN whose latent variable consists of, besides the usual multidimensional variable z, a set c of one-dimensional variables $c_1, c_2 ... c_N$ that are expected to unsupervisedly capture semantic information in a disentangled manner (*i.e.*, with each variable encoding isolated interpretable features of the real data). For obtaining this, the authors of InfoGAN proposed an additional term in the generator's loss function that maximized the mutual information $I(c; G(z, c))$ between a generated image $G(z, c)$ and the latent variable c that originated it. The variables c could be chosen to represent both categorical and continuous features. ClusterGAN (Mukherjee et al. 2019) is an architecture appropriate for clustering tasks based on InfoGAN. The generator learns to generate a certain class of the real distribution correlated with a given one-hot format for the latent variables c. To obtain this, they proposed to use an inference encoder network capable of performing the mapping $E : \mathcal{X} \rightarrow \mathcal{Z}$, which is the inverse of the generator's mapping and similar to an encoder's mapping for an autoencoder architecture. After the training, the encoder can be employed to classify real data samples according to the latent variable to which it is mostly correlated, thus providing the clustering. Two key differences of our work is that (i) we use a separate generator (not a discrete latent variable) to encode a cluster, enabling our method to discover clusters with more representation capacity, and that (ii) we obtain hierarchical clusters, unlike the previous methods.

Kundu et al. (2019) propose the GAN-Tree framework, which slightly resembles our approach, since it also involves a hierarchical structure of independent nodes containing GANs capable of generating samples related to different levels of a similarity hierarchy. There are several differences, however. The most important is that the main motivation for GAN-Tree was a framework capable of addressing the trade-off between quality and diversity when generating samples from multi-modal data. The authors claimed that their approach could be readily adapted for clustering tasks, but no definitive experiments with clustering benchmarks were provided. Other important difference lies in their splitting procedure, which was performed with a latent $\hat{\mathbf{z}}$ inferred by an encoder E for a sample image $\mathbf{x}$, that is, $\hat{\mathbf{z}} = E(\mathbf{x})$. For each node of the tree, they decompose their prior for $\hat{\mathbf{z}}$ into a mixture of two Gaussians with shifted means. They determine the prior Gaussian component to which $\hat{\mathbf{z}}$ is more likely related, and then train the encoder to maximize the likelihood to this prior. For a clustering task, this approach would heavily rely on the assumption that the inference made by E, as well as the cluster encoding with the decomposed Gaussians in $\mathcal{Z}$, will be sufficient to capture semantically meaningful clustering patterns. The split in our approach, on the other hand, is directly embedded into the GAN training, with each generator automatically learning to represent each cluster. Therefore, in our work the clustering semantic quality is directly tied to the GAN's well known representation learning capacity, and, in particular, to the tendency of different generators in MGANs to cover different areas of the training distribution with high semantic discrepancy.

Proposed Method: HC-MGAN

First, we provide a bird's eye view of the proposed method. Then we describe how its two phases – Raw Split and Refinement – work in detail.

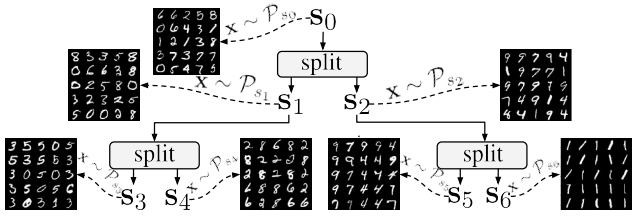

Figure 1: Hierarchical clustering tree overview on MNIST. Nodes are represented by membership vectors $\mathbf{s}_k$. Grids show images sampled from each distribution $\mathcal{P}_k$. Each split divides the probability masses in $\mathbf{s}_k$ into two new nodes.

Algorithm 1: Split

Input: $\mathrm{X}_{\mathrm{Data}}, \mathbf{s}_k$

$\mathbf{s}_l^{(0)}, \mathbf{s}_m^{(0)} \leftarrow \mathrm{raw\_split}(\mathrm{X}_{\mathrm{Data}}, \mathbf{s}_k)$
for $t = 1, \ldots, T$ **do**
  $\mathbf{s}_l^{(t)}, \mathbf{s}_m^{(t)} \leftarrow \mathrm{refinement}(\mathrm{X}_{\mathrm{Data}}, \mathbf{s}_l^{(t-1)}, \mathbf{s}_m^{(t-1)})$
return $\mathbf{s}_l = \mathbf{s}_l^{(T)}, \mathbf{s}_m = \mathbf{s}_m^{(T)}$

Overview of the Hierarchical Scheme

For a given a collection of N training examples $\mathrm{X}_{\mathrm{Data}} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, our method constructs a binary tree of hierarchical clusters, iteratively creating nodes, from top to bottom. The k-th created node is represented by a vector $\mathbf{s}_k \in \mathbb{R}^N$, referred to as membership vector, where each $s_{k,i} = p(Z_i = k \mid \mathbf{x}_i, \boldsymbol{\theta})$ measures the probability of example i belonging to cluster k given $\mathbf{x}_i$ and our model's parameters $\boldsymbol{\theta}$, *i.e.*, a soft clustering approach. The initial $\mathbf{s}_0$ consists of an all-ones vector. Figure 1 depicts the initial development of the tree constructed with the MNIST dataset, for the first 7 nodes. The tree grows via a split mechanism: a soft clustering operation that takes as input a node $\mathbf{s}_k$, and divides its probability masses into two new nodes represented by membership vectors $\mathbf{s}_l$ and $\mathbf{s}_m$, with $\mathbf{s}_l + \mathbf{s}_m = \mathbf{s}_k$. We decide which leaf to split next by taking the node $\mathbf{s}_k$ with the largest total mass, since this roughly measures how many examples are associated with it.

The split mechanism consists of two phases: (i) **raw split** and (ii) **refinement phase**, as shown in Algorithm 1. Before we obtain the final $\mathbf{s}_l$ and $\mathbf{s}_m$ vectors, vector $\mathbf{s}_k$ undergoes a raw split, which outputs the initial membership vectors $\mathbf{s}_l^{(0)}$ and $\mathbf{s}_m^{(0)}$. In most cases, $\mathbf{s}_l^{(0)}$ and $\mathbf{s}_m^{(0)}$ are just rough estimates of how to split $\mathbf{s}_k$ in two clusters. Hence, we use the refinement phase to get them progressively closer to what we expect the ideal soft clustering assignment to be. In Algorithm 1, we can see how the refinement transforms $\mathbf{s}_l^{(0)}$ and $\mathbf{s}_m^{(0)}$ into two new membership vectors $\mathbf{s}_l^{(1)}$ and $\mathbf{s}_m^{(1)}$. This process is repeated for T refinement operations to yield the final result $\mathbf{s}_l = \mathbf{s}_l^{(T)}$ and $\mathbf{s}_m = \mathbf{s}_m^{(T)}$. Note that $\mathbf{s}_m^{(t)} + \mathbf{s}_l^{(t)} = \mathbf{s}_k$ for every t.

Figure 2 shows the progressive improvement resulting from the refinement phase with a grid of 25 MNIST samples, for an $\mathbf{s}_k$ used as a running example. Samples of each membership vector are shown below its label and color-coded (gray scale) according to their probability mass. To visualize how the true class separation changes over iterations, we add a key beside each grid using a font size scale to indicate the amount of probability mass associated with each true class.

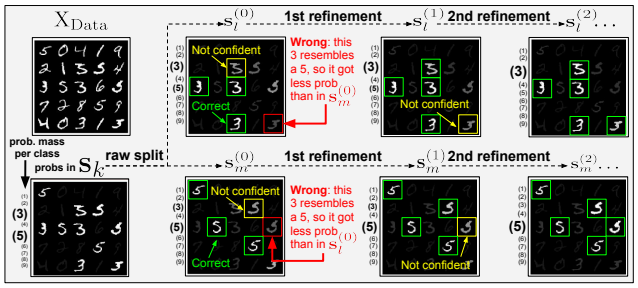

Figure 2: (Best viewed in color) Split mechanism applied to running example $\mathbf{s}_k$ from MNIST. Grids show samples color-coded (gray scale) according to their probability mass in each membership vector (white: 100%). Keys alongside each grid use a font size scale to indicate mass associated with each class. $\mathbf{s}_k$ is mostly associated with 3's and 5's. Raw split provides a rough separation of two classes. Refinement iterations greatly improve the separation quality.

In this example, 3's and 5's are the classes mostly associated with $\mathbf{s}_k$. We expect the final split result ($\mathbf{s}_l$ and $\mathbf{s}_m$) to be a separation of the probability mass of 3's and 5's. After the raw split of $\mathbf{s}_k$, the probability mass of 3's and 5's is roughly divided between $\mathbf{s}_l^{(0)}$ and $\mathbf{s}_m^{(0)}$, but the most ambiguous examples received low probability mass in the membership vector mostly associated to its class. After the first refinement, we observe that $\mathbf{s}_l^{(1)}$ and $\mathbf{s}_m^{(1)}$ provide a better separation of 3's and 5's. After another iteration, most of the probability mass of the 3's (5's) samples is in $\mathbf{s}_l^{(2)}$ ($\mathbf{s}_m^{(2)}$).

Raw Split

For the raw split, we use a two-generator MGAN architecture, which we adapt for binary clustering by leveraging the fact that each generator learns to specialize in generating samples from one sub-region of the real data distribution, typically correlated with a specific set of classes of the data.

Figure 3 (Top) depicts the training of this MGAN for our running example. The MGAN components are: generators $G\alpha_k$, $G\beta_k$, discriminator D_k and classifier C_k. We need each generator to specialize in sub-regions of $\mathbf{s}_k$. Hence, real data samples used for training them must reflect the sample distribution given by $\mathbf{s}_k$ (in the example, these are mostly 3's and 5's). We define distribution $\mathcal{P}_{\mathbf{s}_k}$ over the real examples by (sum-to-one) normalizing $\mathbf{s}_k$, and use it to sample the training batches. We train the MGAN with the usual adversarial game between generators $G\alpha_k$, $G\beta_k$ and the discriminator D_k, while C_k is trained to distinguish between generators. After a few epochs, we observe that one generator is generating mostly 3's while the other, mostly 5's. An important detail is that we also train the generators to minimize C_k's classification loss, increasing the incentive for $G\alpha_k$ and $G\beta_k$ to generate samples from different sub-regions. Moreover, we share some parameters between D_k and C_k, since it is more desirable for C_k to perform its classification in a higher-level feature space, such as the one learned by D_k.

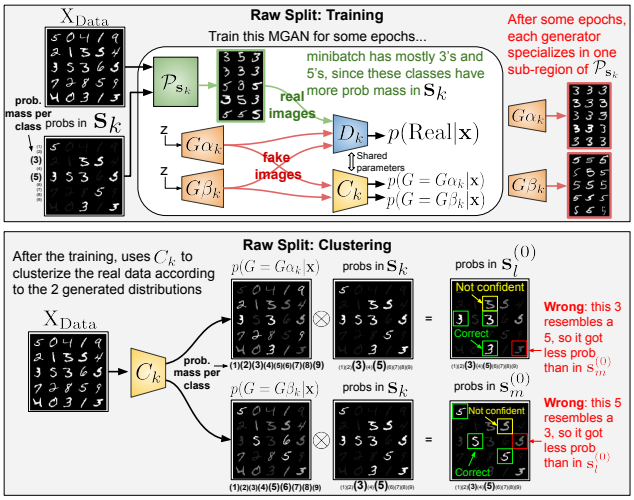

Figure 3: (Top) Training the Raw Split Components: generators $G\alpha_k$, $G\beta_k$, discriminator D_k and classifier C_k. $\mathcal{P}_{\mathbf{s}_k}$ draws real samples in proportion to their mass in $\mathbf{s}_k$. (Bottom) Clustering the dataset with the raw split classifier.

After training the MGAN for enough epochs, we can cluster the data as shown in Figure 3 (Bottom). We use C_k to perform the clustering on the real data according to the two generated distributions it learned to identify, and which we expect to correlate with different classes of the dataset. In this example, the first generated distribution resembled 3's, while the other resembled 5's. Hence, we expect C_k to mostly assign real 3's probability mass to the first soft cluster and real 5's probability mass to the second. As seen in Figure 3 (Bottom), C_k does manage to roughly assign the 3's (resp. 5's) mass to the cluster related to $G\alpha_k$'s (resp. $G\beta_k$).

Note that C_k must be used to perform inference for all examples of every class in the dataset, regardless if they were present in the generated distributions, with $p(G = G\alpha_k|\mathbf{x}) + p(G = G\beta_k|\mathbf{x}) = 1$. While C_k can be seen as classifying examples conditioned on them belonging to node's $\mathbf{s}_k$ subtree, membership vectors actually correspond to estimates for unconditional probabilities. To enforce the condition that, for each example, the sum of its probability masses in the membership vectors of $\mathbf{s}_k$'s children equals vector $\mathbf{s}_k$, we multiply the probabilities of each example predicted by C_k by its probability in $\mathbf{s}_k$.

Finally, by noting that some samples in the running example were not well separated in $\mathbf{s}_l^{(0)}$ and $\mathbf{s}_m^{(0)}$ by classifier C_k, we conclude that the two distributions obtained from a raw split may not be sufficiently diverse or have enough quality to account for the entire set of 3's and 5's the MGAN had access to. This underlines the need for the refinement phase, which will be covered in the next section.

We now provide a formal definition of the MGAN game occurring in the raw split phase. Dropping the subscript k to avoid clutter, we define the objective function of the two-generator MGAN a sum of two cost functions $\mathcal{L}_{adv}$ and $\mathcal{L}_{cls}$,

$$\min_{G\alpha,G\beta,C} \max_{D} \mathcal{L} = \mathcal{L}_{adv}(G\alpha, G\beta, D) + \lambda \mathcal{L}_{cls}(G\alpha, G\beta, C), \quad (1)$$

Algorithm 2: Raw Split

Input: $\mathrm{X}_{\mathrm{Data}}$, $\mathbf{s}_k$

Creates Components $G\alpha_k$, $G\beta_k$, C_k, D_k
for training_iterations **do**
  sample $\mathbf{x}, \hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta$ from $\mathcal{P}_{\mathbf{s}_k}, \mathcal{P}_{G\alpha_k}, \mathcal{P}_{G\beta_k}$
  Get $\mathcal{L}_{D_k}^{(real)}$ using D_k criterion on $\mathbf{x}$ with real labels
  Get $\mathcal{L}_{D_k}^{(fake)}$ using D_k criterion on $\hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta$ w/ fake labels
  Update $\boldsymbol{\theta}_{D_k}$ with Adam($\nabla_{\boldsymbol{\theta}_{D_k}}(\mathcal{L}_{D_k}^{(real)} + \mathcal{L}_{D_k}^{(fake)})$)
  Get $\mathcal{L}_{C_k}$ using C_k criterion on $\hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta$, with categorical labels for $G\alpha, G\beta$
  Update $\boldsymbol{\theta}_{C_k}$ with Adam $(\nabla_{\boldsymbol{\theta}_{C_k}}(\mathcal{L}_{C_k}))$
  Get $\mathcal{L}_{G_k}^{(disc)}$ using D_k criterion on $\hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta$ with real labels
  Get $\mathcal{L}_{G_k}^{(clasf)}$ using C_k criterion on $\hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta$ with categorical labels for $G\alpha, G\beta$
  Update $\boldsymbol{\theta}_{G\alpha_k, G\beta_k}$ w/ Adam($\nabla_{\boldsymbol{\theta}_{G_k}}(\mathcal{L}_{G_k}^{(disc)} + \lambda \mathcal{L}_{G_k}^{(clasf)})$)
for $\mathbf{x}_i$ in $\mathrm{X}_{\mathrm{Data}}$ **do**
  $s_{l,i} \leftarrow C_k^{(\alpha\_out)}(\mathbf{x}_i) \cdot s_{k,i}$, $s_{m,i} \leftarrow C_k^{(\beta\_out)}(\mathbf{x}_i) \cdot s_{k,i}$
return $\mathbf{s}_l, \mathbf{s}_m$

where $\mathcal{L}_{adv}$ is the cost for the adversarial minimax game between generators and discriminator, given by

$$\mathcal{L}_{adv}(G\alpha, G\beta, D) = \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{\mathbf{s}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{G\alpha}}[\log(1 - D(\mathbf{x}))] + \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{G\beta}}[\log(1 - D(\mathbf{x}))] \quad (2)$$

and $\mathcal{L}_{cls}$ is the classification cost that is minimized by both generators and classifier, given by

$$\mathcal{L}_{cls}(G\alpha, G\beta, C) = \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{G\alpha}}[\log(C(\mathbf{x}))] + \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{G\beta}}[\log(C(\mathbf{x}))]. \quad (3)$$

Note that we multiply $\mathcal{L}_{cls}$ by a regularization parameter λ to weight its impact on the total cost.

Algorithm 2 lists the steps involved in training the MGAN for the raw split and the clustering performed with C_k.

Refinement

The refinement phase comes immediately after the raw split. During this phase, some of the probability mass in the two membership vectors obtained by the raw split, $\mathbf{s}_l^{(0)}$ and $\mathbf{s}_m^{(0)}$, is exchanged so as to iteratively improve the clustering quality (see Figure 2). Each iteration is referred to a refinement sub-block. Without loss of generality, consider the first refinement sub-block, whose components are depicted in Figure 4 (Top), for our running example. The components are divided in two "refinement groups", l and m (each formed by a generator G_i, a discriminator D_i and a classifier C_i, $i \in \{l, m\}$). Each group i takes $\mathbf{s}_i^{(0)}$ as input, and has its own independent GAN game occurring between G_i and D_i.

This scheme with two separated GANs is designed to obtain a more focused generative representation of each sub-region of $\mathbf{s}_k$ than we were able to obtain at the raw split phase using a single MGAN's discriminator to learn to discriminate the entire region described by $\mathbf{s}_k$. *By providing a more focused view of one sub-region to one discriminator, it encounters less variance among the real examples it receives, and thus its discriminative task becomes easier. We expect its adversarial generator's response to be a more diverse and convincing generation of examples associated with that particular sub-region.*

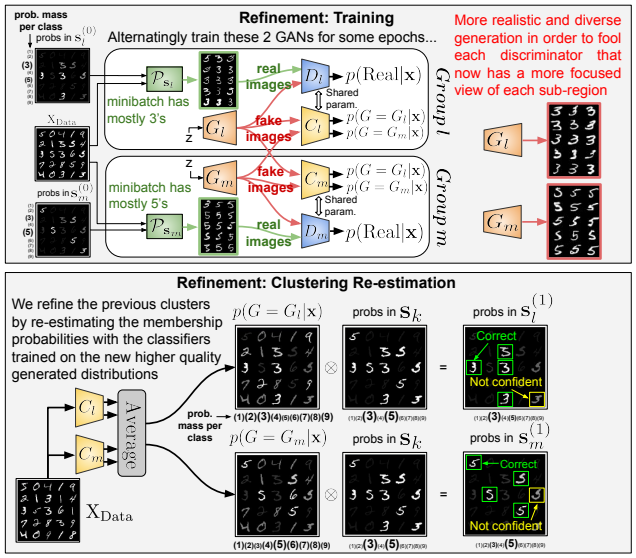

Figure 4: (Top) Refinement Training: generators G_l, G_m, discriminators D_l, D_m and classifiers C_l, C_m. (Bottom) Clustering the dataset with the refinement classifiers.

As shown in Figure 4 (Top), each GAN in groups $i \in \{l, m\}$ draws real samples from its corresponding distribution $\mathcal{P}_{\mathbf{s}_i}$ over $\mathrm{X_{Data}}$, which is equal to the (sum-to-one) normalized vector $\mathbf{s}_i$ (analogously to $\mathcal{P}_{\mathbf{s}_k}$). As a result, the minibatches passed to each group's GAN reflect the probability mass in their respective membership vectors, *e.g.*: in our running example, $\mathcal{P}_{\mathbf{s}_l}$ draws mostly 3's, since 3's have more probability mass in $\mathbf{s}_l^{(0)}$, but it might eventually draw some 5's as well, since there's still some mass for 5's in $\mathbf{s}_l^{(0)}$.

The role of classifiers C_l and C_m in these two separated GAN games is similar to the single classifier C_k used in the raw split phase: learn to distinguish samples from G_l and G_m, thereby providing a way to cluster the real data. However, instead of using a single classifier (as in the raw split), we found that using two separate classifiers which share parameters with the respective discriminators forces the classification to occur in a higher-level feature space, achieving better clustering. Furthermore, it allows us to train G_l and G_m to minimize the classifiers' losses, thus providing incentive for each generator to generate samples more strongly correlated with each sub-region (we did something similar for the 2 generators in the MGAN of the raw split).

After training the two refinement groups alternately for enough epochs, we perform a **clustering re-estimation**, as shown in Figure 4 (Bottom). This is similar to the way C_k was used to cluster the real data in the raw split (Figure 3 (Bottom)). Now, both classifiers estimate the probability that a sample $\mathbf{x}_i \in \mathrm{X_{Data}}$ came from G_l (or its complement, G_m), hereby denoted by $C_i^{(l\_out)}(\mathbf{x}_i)$, for $i \in \{l, m\}$. We take the average probability $(C_l^{(l\_out)}(\mathbf{x}_i) + C_m^{(l\_out)}(\mathbf{x}_i))/2$ as the proportion of the mass associated with $\mathbf{x}_i$ in $\mathbf{s}_k$ that should go into $\mathbf{s}_l^{(1)}$ and. By using this "two-classifier ensemble", we aim to increase the quality of the clustering, since the generators' distributions are expected to be more representative of each sub-region of the dataset than the two generated distributions obtained during the raw split. In the running example, G_l's (G_m's) distributions resembled 3's (5's), so the classifiers assign more of the 3's (5's) probability mass to the cluster associated with G_l (G_m's).

For the subsequent refinement, we expect that using $\mathbf{s}_l^{(1)}$ and $\mathbf{s}_m^{(1)}$ to train new refinement groups l and m can yield generated distributions that are even more representative of the sub-regions encoded by these membership vectors, providing, in turn, even more information for the classifiers to perform the clustering and to obtain improved membership vectors $\mathbf{s}_l^{(2)}$ and $\mathbf{s}_m^{(2)}$ (recall Figure 2). Therefore, by repeating the process over T refinements, it is expected that the initial sub-regions captured by $\mathbf{s}_k$ are increasingly more associated with either of the refinement groups.

We now provide a formal definition for the two simultaneous GAN games occurring for the training of the components of the refinement phase. From the perspective of refinement group l, the training can be defined as an optimization of a sum of two cost functions $\mathcal{L}_{adv}$ and $\mathcal{L}_{cls}$,

$$\min_{G_l, C_l} \max_{D_l} \mathcal{L}(G_l, D_l, C_l) = \mathcal{L}_{adv}(G_l, D_l) + \lambda \mathcal{L}_{cls}(G_l, C_l), \quad (4)$$

where $\mathcal{L}_{adv}$ describes the cost function for the adversarial minimax game between generator G_l and discriminator D_l, that only involves group l components, and is given by

$$\mathcal{L}_{adv}(G_l, D_l) = \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{data}}[\log D_l(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{G_l}}[\log(1 - D_l(\mathbf{x}))] \quad (5)$$

and $\mathcal{L}_{cls}$ is classification cost that is minimized with respect to G_l's parameters and C_l's parameters, but also involves G_m and C_m for computing the cost, given by

$$\mathcal{L}_{cls}(G_l, C_l) = \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{G_l}}[\log C_l(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{G_l}}[\log C_m(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim\mathcal{P}_{G_m}}[\log C_l(\mathbf{x})]. \quad (6)$$

We multiply $\mathcal{L}_{cls}$ by a regularization parameter λ to weight its impact on the total cost. The corresponding equations for refinement group m follow analogously.

Algorithms 3 and 4 respectively list the steps involved in the external training loop that coordinates the alternated training of refinement groups l and m, and in the function called by Algorithm 3 that performs an update for the components of a given refinement group isolatedly.

Experiments

We used the same standard convolutional GAN/MGAN architecture for all the following experiments. Respecting the unsupervised nature of clustering, which does not afford hyperparameter tuning, we only selected slightly different tunings for each dataset, none of which required labeled supervision, merely aiming to stabilize the generators and to avoid overfitting with classifiers while distinguishing between generators. The code for HC-MGAN will be available at https://github.com/dmdmello/HC-MGAN.

Datasets We consider three datasets: MNIST (LeCun et al. 1998), Fashion MNIST (FMNIST) (Xiao, Rasul, and Vollgraf 2017) and Stanford Online Products (SOP) (Oh Song et al. 2016). Following a common practice in DC works,

Algorithm 3: Refinement

Input: $\mathrm{X}_{\mathrm{Data}}, \mathbf{s}_l^{(t)} \mathbf{s}_m^{(t)}$

Creates G_l, D_l, C_l for group l and G_m, D_m, C_m for group m
for T iterations **do**
  sample $\mathbf{x}_l, \mathbf{x}_m, \hat{\mathbf{x}}_l, \hat{\mathbf{x}}_m$ from $\mathcal{P}_{\mathbf{s}_l^{(t)}}, \mathcal{P}_{\mathbf{s}_m^{(t)}}, \mathcal{P}_{G_l}, \mathcal{P}_{G_m}$
  TrainRefinGroup($\mathcal{G}_{int} = \{G_l, D_l, C_l, \mathbf{x}_l, \hat{\mathbf{x}}_l\}$, $\mathcal{G}_{ext} = \{C_m, \hat{\mathbf{x}}_m\}$) {trains l with needed external data/-components from m}
  TrainRefinGroup($\mathcal{G}_{int} = \{G_m, D_m, C_m, \mathbf{x}_m, \hat{\mathbf{x}}_m\}$, $\mathcal{G}_{ext} = \{C_l, \hat{\mathbf{x}}_l\}$) {trains m with needed external data/components from l}
for $\mathbf{x}_i$ in $\mathrm{X}_{\mathrm{Data}}$ **do**
  $s_{i,l}^{(t+1)} \leftarrow (C_l^{(l\_out)}(\mathbf{x}_i) + C_m^{(l\_out)}(\mathbf{x}_i)) \cdot (s_{i,l}^{(t)} + s_{i,m}^{(t)})/2$
  $s_{i,m}^{(t+1)} \leftarrow (C_l^{(m\_out)}(\mathbf{x}_i) + C_m^{(m\_out)}(\mathbf{x}_i)) \cdot (s_{i,l}^{(t)} + s_{i,m}^{(t)})/2$
return $\mathbf{s}_l^{(t+1)}, \mathbf{s}_m^{(t+1)}$

Algorithm 4: TrainRefinGroup

Input: $\mathcal{G}_{int} = \{G_{int}, D_{int}, C_{int}, \mathbf{x}, \hat{\mathbf{x}}_{int}\}, \mathcal{G}_{ext} = \{C_{ext}, \hat{\mathbf{x}}_{ext}\}$ #$\mathcal{G}_{int}$ receives internal data/components from the current refinement group being trained, $\mathcal{G}_{ext}$ receives external data/components from the neighbor refin. group needed to train the current group

Get $\mathcal{L}_{D_{int}}^{(real)}$ using D_{int} criterion on $\mathbf{x}$ with real labels
Get $\mathcal{L}_{D_{int}}^{(fake)}$ using D_{int} criterion on $\hat{\mathbf{x}}_{int}$ with fake labels
Updates $\boldsymbol{\theta}_{D_{int}}$ with Adam($\nabla_{\boldsymbol{\theta}_{D_{int}}}(\mathcal{L}_{D_{int}}^{(real)} + \mathcal{L}_{D_{int}}^{(fake)})$)
Get $\mathcal{L}_{C_{int}}$ using C_{int} criterion on $\hat{\mathbf{x}}_{int}, \hat{\mathbf{x}}_{ext}$, with categorical labels for internal and external generated data
Updates $\boldsymbol{\theta}_{C_{int}}$ with Adam($\nabla_{\boldsymbol{\theta}_{C_{int}}}(\mathcal{L}_{C_{int}})$)
Get $\mathcal{L}_{G_{int}}^{(disc)}$ using D_{int} criterion on $\hat{\mathbf{x}}_{int}$ with real labels
Get $\mathcal{L}_{G_{int}}^{(clasf)}$ using C_{int} criterion on $\hat{\mathbf{x}}_{int}, \hat{\mathbf{x}}_{ext}$ with categorical labels for internal and external generated data
Updates $\boldsymbol{\theta}_{G_{int}}$ with Adam($\nabla_{\boldsymbol{\theta}_{G_{int}}}(\mathcal{L}_{G_{int}}^{(disc)} + \lambda \mathcal{L}_{G_{int}}^{(clasf)})$)

we used all available images for each dataset. **MNIST**: This dataset consists of grayscale images of hand-written digits with 28x28 resolution, with 10 classes and approximately 7k images available for each class. **FMNIST**: This dataset consists of gray scale pictures of clothing-related objects with 28x28 resolution, with 10 classes and exactly 7k images available for each class. **SOP**: This dataset consists of color pictures of products with varying resolution sizes, with 12 classes, and a varied number of examples per class, roughly ranging from 6k examples to 13k. SOP, in particular, is designed for supervised tasks, and is very hard for clustering due to high intra-class variance. We follow the practice adopted for SOP in (Zhang et al. 2019b), i.e., we convert the images to grayscale, resize them to 32x32 resolution, and drop the classes "kettle" and "lamp" from it.

Evaluation Metrics We consider two of the most common clustering metrics for evaluating our method's clustering performance on each dataset: *clustering accuracy* (ACC) and *normalized mutual information* (NMI). As usual, these metrics are computed on the results obtained when setting the number of clusters C to the number of classes in the data. For a direct comparison, we let HC-MGAN's tree grow until it reaches C leaves. Additionally, we convert the final soft clustering outputs to hard assignments by attributing each example to the highest probability group, and then compute the evaluation metrics.

Baselines and SOTA methods We present the baselines and state-of-the-art methods used in the comparison. We consider five groups of methods: (i) classical, non Deep Learning (DL)-based based: K-Means (MacQueen 1967), SC (Zelnik-Manor and Perona 2004), AC (Gowda and Krishna 1978), NMF (Cai et al. 2009); (ii) Varied DC: DEC (Xie, Girshick, and Farhadi 2016), DCN (Yang et al. 2017), JULE (Yang, Parikh, and Batra 2016), VaDE (Jiang et al. 2017), DEPICT (Ghasedi Dizaji et al. 2017), SpectralNET, DAC (Chang et al. 2017), (Shaham et al. 2018), DualAE (Yang et al. 2019); (iii) Subspace Clustering (either DL-based or not): NCSC (Zhang et al. 2019b), SSC (Elhamifar and Vidal 2013), LLR (Liu et al. 2013), KSSC (Patel and Vidal 2014), DSC-Net (Ji et al. 2017), k-SCN (Zhang et al. 2019a); (iv) GAN-based DC: ClusterGAN (Mukherjee et al. 2019), InfoGAN (Chen et al. 2016), DLS-Clustering (Ding and Luo 2019);(v) DC w/ data augumentation: (Zhang et al. 2019b), IIC (Ji, Henriques, and Vedaldi 2019), DCCM (Wu et al. 2019) and DCCS (Zhao et al. 2020). *None of the DC methods we found in the literature are hierarchical*. Most results are transcribed from either (Zhao et al. 2020), (Zhang et al. 2019b) or (Mukherjee et al. 2019).

Results Table 1 shows the performance comparison between traditional baselines, state-of-the-art DC methods without and with data augumentation, and our method. In order to check the effectiveness of the refinements, we also display results obtained only with raw splits.

MNIST Our method outperforms the traditional baselines by a large margin. In terms of ACC, it is not among the top 5 presented methods, but it performs reasonably close to them, even outperforming, in either NMI or ACC, some DC methods like ClusterGAN, InfoGAN, DEC, DCN. This result was obtained by employing the same architecture we used for FMNIST, which might be of excessive capacity for MNIST and even harm the clustering result by making it trivial for a single generator to represent all the data, instead of a cluster of similar data points, during the raw split. Of all the datasets, this was the one for which the refinements showed the greatest improvement over raw split experiment.

FMNIST Only DCCS is able to surpass our method's ACC and NMI performance. We emphasize that using data augmentation in DCCS causes a significant improvement, since selecting the right type of augmentations for a specific dataset can reduce much of the intra-class variance. However, augmentation with GANs is challenging (Karras et al. 2020), so we leave this for future work. The refinements still had a positive impact over the raw split only experiment.

SOP The results for existing methods were transcribed from the NCSC work (Zhang et al. 2019b). There the authors state that they handpicked 1k examples per class to create a manageable dataset for clustering, but did not specify how or which examples were selected. Their choice was made in the context of competing subspace clustering methods to which their model was being compared, many of which are not able to scale to larger datasets due to the memory constraints involved in computing a similarity matrix necessary for their

Dataset	MNIST		FMNIST		SOP	
Metrics	ACC	NMI	ACC	NMI	ACC	NMI
K-means[i]	.572	.500	.474	.512	-	-
SC[i]	.696	.663	.508	.575	-	-
AC[i]	.695	.609	.500	.564	-	-
NMF[i]	.545	.608	.434	.425	-	-
DEC[ii]	.843	.772	.590	.601	**.229**	.121
DCN[ii]	.833	.809	.587	.594	.213	.084
JULE[ii]	.964	.913	.563	.608	-	-
VaDE[ii]	.945	.876	.578	.630	-	-
SpectralNet[ii]	**.971**	.924	.533	.552	-	-
DualAE[ii]	**.978**	**.941**	**.662**	.645	-	-
DAC[ii]	.966	**.967**	.615	.632	**.231**	.098
SSC[iii]	.430	.568	.359	.181	.127	.007
LLR[iii]	.552	.665	.345	.254	.224	**.173**
DSC-Net[iii]	.659	.730	.606	.617	**.269**	**.146**
KSSC[iii]	.585	.677	.382	.197	**.268**	**.152**
k-SCN[iii]	.871	.782	.638	.620	**.229**	**.166**
NCSC[iii]	.941	.861	**.721**	**.686**	**.275**	**.138**
ClustGAN[iv]	.950	.890	.630	.640	-	-
InfoGAN[iv]	.890	.860	.610	.590	.198	.082
DLS-Clst[iv]	**.975**	**.936**	**.693**	**.669**	-	-
IIC[v]	**.992**	**.978**	.657	.637	-	-
DCCM[v]	-	-	.657	.637	-	-
DCCS[v]	**.989**	**.970**	**.756**	**.704**	-	-
Ours	.943	.905	**.721**	**.691**	**.229**	.072
Ours (raw)	.877	.856	.704	**.690**	.204	.063

Table 1: Clustering performance results on 3 datasets w.r.t. ACC and NMI (top 5 in bold). "Ours" indicates HC-MGAN, with (raw) indicating no refinement operations performed. [i]: non-deep. [ii]: varied DC methods. [iii]: Subspace Clustering. [iv]: GAN-based DC. [v]: DC w/ data augmentation.

methods. We tried without success to contact the authors to obtain the same subset of the data. Therefore, we decided to evaluate our results on the entire data. Due to the class imbalance, we compute both the mean ACC over classes (reported in the table) and the overall ACC achieved by HC-MGAN, which are, respectively, 0.229 and 0.221. NMI is invariant to class imbalance. On this dataset, our model does not perform as well as the subspace clustering group, especially NCSC, KSSC and DSC-Net, even though it was on par with NCSC and greatly surpassed the KSSC and DSC-Net on other datasets. But for other DC methods reported from (Zhang et al. 2019b), our method performs closely, even outperforming DCN and InfoGAN in accuracy.

Qualitative Analysis Figure 5 shows a visualization of the clustering tree for FMNIST. For each node k in the tree, it displays a grid of 25 real examples sampled in proportion to the weights in membership vector $\mathbf{s}_k$. The prevalence of each class in $\mathbf{s}_k$ is represented in a font size scale by the class names inside parenthesis beside each grid. Specifically, the probability mass of a class A in $\mathbf{s}_k$ is given by $\sum_i^N s_{k,i} \cdot \mathbf{1}(c_i = A)$, where $\mathbf{1}$ is an indicator function and c_i is the class of the i-th example. Through a quick inspection, we observe that the 1st split ($\mathbf{s}_0$) occurred with almost perfect precision, with examples of the same class having their probability mass nearly entirely allocated to either $\mathbf{s}_1$ or $\mathbf{s}_2$. We begin to notice some imprecision at the 2nd split ($\mathbf{s}_2$), with a small portion of the probability mass of classes coat and t-shirt being assigned to $\mathbf{s}_3$ while the largest portion went to $\mathbf{s}_4$, and at the 3rd split ($\mathbf{s}_1$), with a small portion of the sneakers' mass being assigned to $\mathbf{s}_5$ and the largest portion going to $\mathbf{s}_6$. The most imprecise splits occurred during the 4th, 7th and 9th splits (i.e., $\mathbf{s}_4$, $\mathbf{s}_6$ and $\mathbf{s}_7$) but the classes involved in these splits (t-shirt, pullover, coat and shirt) are the most visually similar in the dataset, thus being the hardest to accurately separate into clusters. The other classes are relatively well separated. One fact that can explain why our method performed very well w.r.t. NMI on FMNIST is that most of the probability mass of each class for which our method exhibited low accuracy was assigned to at most two clusters. For the NMI metric, having the wrongfully assigned classes concentrated on fewer clusters provides better mutual information than having them spread throughout more clusters.

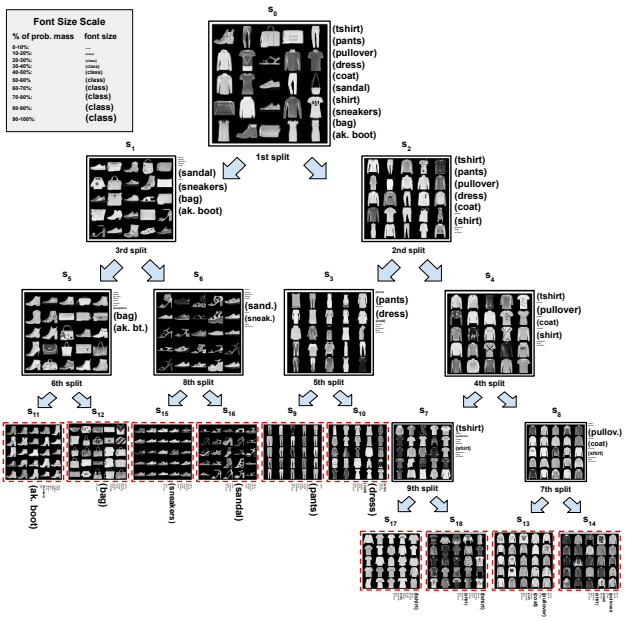

Figure 5: Tree for the FMNIST dataset. Grids show sampled images; keys beside grid represent probability mass of each class in font size scale. Dashed red line indicate leaf nodes.

Conclusion

In this work, we proposed a method for hierarchical clustering that leverages the deep representation capability of GANs and MGANs. To the best of our knowledge, this is the first hierarchical DC method. It constructs a tree of clusters from top to bottom, where each leaf node represents a cluster. Each cluster is divided in binary splits, which are performed in two phases: a raw split and a refinement phase. We have shown how well our method compares to other deep clustering techniques on clustering datasets, obtaining competitive results against other DC methods that lack advantages of the hierarchical approach.

Acknowledgments

The authors thank the partial support from Brazilian research supporting agencies: CNPq (grant PQ 313582/2018-1), FAPEMIG (grant CEX - PPM-00598-17, grant APQ-02337-21), and CAPES (grant 88887.506739/2020-00).

References

Achanta, R.; and Susstrunk, S. 2017. Superpixels and Polygons Using Simple Non-Iterative Clustering. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Cai, D.; He, X.; Wang, X.; Bao, H.; and Han, J. 2009. Locality Preserving Nonnegative Matrix Factorization. In *International Jont Conference on Artifical Intelligence (IJCAI)*.

Chang, J.; Wang, L.; Meng, G.; Xiang, S.; and Pan, C. 2017. Deep Adaptive Image Clustering. In *IEEE International Conference on Computer Vision (ICCV)*.

Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; and Abbeel, P. 2016. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ding, F.; and Luo, F. 2019. Clustering by Directly Disentangling Latent Space. arXiv:1911.05210.

Elhamifar, E.; and Vidal, R. 2013. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

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 *IEEE International Conference on Computer Vision (ICCV)*.

Ghosh, A.; Kulharia, V.; Namboodiri, V.; Torr, P. H. S.; and Dokania, P. K. 2018. Multi-agent Diverse Generative Adversarial Networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Net. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Gowda, K. C.; and Krishna, G. 1978. Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern Recognition*.

Hoang, Q.; Nguyen, T.; Le, T.; and Phung, D. 2018. MGAN: training generative adversarial nets with multiple generators. In *International Conference on Learning Representations (ICLR)*.

Ji, P.; Zhang, T.; Li, H.; Salzmann, M.; and Reid, I. 2017. Deep Subspace Clustering Networks. In *International Conference on Neural Information Processing Systems (NeurIPS)*.

Ji, X.; Henriques, J. F.; and Vedaldi, A. 2019. Invariant Information Clustering for Unsupervised Image Classification and Segmentation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*.

Jiang, Z.; Zheng, Y.; Tan, H.; Tang, B.; and Zhou, H. 2017. Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Joulin, A.; Bach, F.; and Ponce, J. 2010. Discriminative clustering for image co-segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Karras, T.; Aittala, M.; Hellsten, J.; Laine, S.; Lehtinen, J.; and Aila, T. 2020. Training Generative Adversarial Networks with Limited Data. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Kundu, J. N.; Gor, M.; Agrawal, D.; and Babu, R. V. 2019. GAN-Tree: An Incrementally Learned Hierarchical Generative Framework for Multi-Modal Data Distributions. In *IEEE/CVF International Conference on Computer Vision (ICCV)*.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Liu, G.; Lin, Z.; Yan, S.; Sun, J.; Yu, Y.; and Ma, Y. 2013. Robust Recovery of Subspace Structures by Low-Rank Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

Liu, Z.; Lin, G.; Yang, S.; Feng, J.; Lin, W.; and Goh, W. L. 2018. Learning Markov Clustering Networks for Scene Text Detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

MacQueen, J. 1967. Some methods for classification and analysis of multivariate observations. In *Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*.

Mukherjee, S.; Asnani, H.; Lin, E.; and Kannan, S. 2019. ClusterGAN: Latent Space Clustering in Generative Adversarial Networks. *AAAI Conference on Artificial Intelligence (AAAI)*.

Oh Song, H.; Xiang, Y.; Jegelka, S.; and Savarese, S. 2016. Deep Metric Learning via Lifted Structured Feature Embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Patel, V. M.; and Vidal, R. 2014. Kernel sparse subspace clustering. In *IEEE International Conference on Image Processing (ICIP)*.

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

Shi, J.; and Malik, J. 2000. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence (PAMI)*.

Wu, J.; Long, K.; Wang, F.; Qian, C.; Li, C.; Lin, Z.; and Zha, H. 2019. Deep Comprehensive Correlation Mining for Image Clustering. In *IEEE/CVF International Conference on Computer Vision (ICCV)*.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

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

Yang, B.; Fu, X.; Sidiropoulos, N. D.; and Hong, M. 2017. Towards K-Means-Friendly Spaces: Simultaneous Deep Learning and Clustering. In *International Conference on Machine Learning (ICML)*.

Yang, J.; Parikh, D.; and Batra, D. 2016. Joint Unsupervised Learning of Deep Representations and Image Clusters. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yang, X.; Deng, C.; Zheng, F.; Yan, J.; and Liu, W. 2019. Deep Spectral Clustering Using Dual Autoencoder Network. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zelnik-Manor, L.; and Perona, P. 2004. Self-Tuning Spectral Clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhang, H.; Xu, S.; Jiao, J.; Xie, P.; Salakhutdinov, R.; and Xing, E. P. 2018. Stackelberg GAN: Towards Provable Minimax Equilibrium via Multi-Generator Architectures. arXiv:1811.08010.

Zhang, T.; Ji, P.; Harandi, M.; Hartley, R.; and Reid, I. 2019a. Scalable Deep k-Subspace Clustering. In *Computer Vision – ACCV 2018*.

Zhang, T.; Ji, P.; Harandi, M.; Huang, W.; and Li, H. 2019b. Neural Collaborative Subspace Clustering. In *International Conference on Machine Learning (ICML)*.

Zhao, J.; Lu, D.; Ma, K.; Zhang, Y.; and Zheng, Y. 2020. Deep Image Clustering with Category-Style Representation. In *European Conference on Computer Vision (ECCV)*.