2020 25th International Conference on Pattern Recognition (ICPR)
Milan, Italy, Jan 10-15, 2021

GAN-based Gaussian Mixture Model Responsibility Learning

Wanming Huang *, Richard Yi Da Xu *, Shuai Jiang *, Xuan Liang * and Ian Oppermann * †
*Faculty of Engineering and IT
University of Technology Sydney
81 Broadway, Sydney, Australia
†NSW Data Analytics Centre
2-24 Rawson Pl, Sydney, Australia

Abstract—Mixture model (MM) is a probabilistic framework allows us to define dataset containing K different modes. When each of the modes is associated with a Gaussian distribution, we refer to it as Gaussian MM or GMM. Given a data point x, a GMM may assume the existence of a random index $k \in \{1, \ldots, K\}$ identifying which Gaussian the particular data is associated with. In a traditional GMM paradigm, it is straightforward to compute in closed-form, the conditional likelihood $p(x|k, \theta)$ as well as the responsibility probability $p(k|x, \theta)$ describing the distribution weights for each data. Computing the responsibility allows us to retrieve many important statistics of the overall dataset, including the weights of each of the modes/clusters. Modern large datasets are often containing multiple unlabelled modes, such as paintings dataset may contain several styles; fashion images containing several unlabelled categories. In its raw representation, the Euclidean distances between the data (e.g., images) do not allow them to form mixtures naturally, nor it's feasible to compute responsibility distribution analytically, making GMM unable to apply. In this paper, we utilize the generative adversarial network (GAN) framework to achieve a plausible alternative method to compute these probabilities. The key insight is that we compute them at the data's latent space z instead of x. However, this process of $z \to x$ is irreversible under GAN which renders the computation of responsibility $p(k|x, \theta)$ infeasible. Our paper proposed a novel method to solve it by using a so-called posterior consistency module (PCM). PCM acts like a GAN, except its generator C_{PCM} does not output the data, but instead it outputs a distribution to approximate $p(k|x, \theta)$. The entire network is trained in an "end-to-end" fashion. Trough these techniques, it allows us to model the dataset of very complex structure using GMM and subsequently to discover interesting properties of an unsupervised dataset, including its segments, as well as generating new "out-distribution" data by smooth linear interpolation across any combinations of the modes in a completely unsupervised manner.

I. Introduction

Gaussian mixture model (GMM) is one of the most commonly used probabilistic frameworks for dataset having multiple modes. It assumes that all data points come from a mixture of a finite number of Gaussian distributions. The density function of the GMM is defined below:

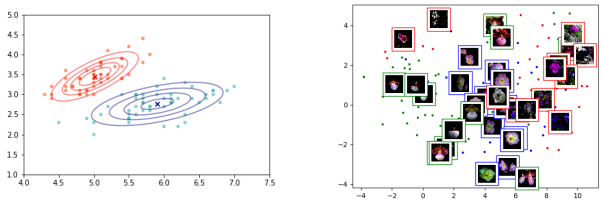
$$p_{\mathcal{X}}(x) = \sum_{k=1}^{K} \alpha_k \mathcal{N}(x; u_k, \Sigma_k), \tag{1}$$

where K is the total number of Gaussians in the mixture, and k^{th} component is characterized by a Gaussian distribution with weight α_k, mean u_k and covariance matrix Σ_k.

Given x is the data and k is the (latent) index of the mixture density, GMM allows us to compute the conditional likelihood $p(x|k, \theta)$ as well as responsibility probability $p(k|x, \theta)$. In the Bayesian paradigm, one may refer $p(k|x, \theta)$ as the posterior density where the prior is $p(k) \equiv (\alpha_1, \ldots, \alpha_k)$.

This will further allow us to retrieve many important statistics and properties about the dataset, for example, (1) the soft clustering membership of the new and existing data; (2) the overall weight of each of the components in the dataset.

When Euclidean distance can be meaningfully defined over the data x in hand (typically when it has a low dimensionality), one can compute both $p(x|k, \theta)$ and $p(k|x, \theta)$ directly, see Figure 1a. However, it will be difficult to do so when dealing with complex and high data dimensional data, such as images, as x in its raw form does not form mixtures naturally, shown in Figure 1b.



(a) (b)

Fig. 1: Illustration of clustering (a) low-dimensional data and (b) high-dimensional data.

An intuitive solution to this problem is to transform the data x into its latent representation z deterministically and to model z using GMM instead. Past works have attempted such as using a variational auto-encoder [1]. On its surface, the method may accomplish this task by transforming x into z first using an encoder so that we can obtain its z, and subsequently to perform both $p(z|k, \theta)$

978-1-7281-8808-9/20/$31.00 ©2020 IEEE

2020 25th International Conference on Pattern Recognition (ICPR) | 978-1-7281-8808-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICPR48806.2021.9412309

and $p(k|z,\theta)$. However, maximizing the likelihood of VAE yields an estimated density that always bleeds probability mass away from the estimated data manifold, results to blur images.

At the same time, generative adversarial network (GAN) [2] gives us a way to compute the latent representation z associated with the data x: GAN, introduces a two-player non-cooperative game by a generator G and a discriminator D. The generator produces samples from the random noise vector z. Since the generation of z is independent of x, any arbitrary distribution should theoretically suffice, making a uniform distribution or a standard Gaussian distribution popular choices.

The discriminator differentiates between true samples and fake samples. The objective function of the game is given as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]. \quad (2)$$

Compared with VAEs, GAN results in a much sharper image even if it does not perfectly coincide with the data density, and has become the Generative model of choice for many applications. However, in our setting, if we were to naively modify $p(z)$ from a single-mode Gaussian into a GMM, we can not obtain densities of $p(x|k,\theta)$ and $p(k|x,\theta)$ for the following two reasons:

(1) Firstly, the generation process is one-directional: i.e., $z \to x$, making it infeasible to compute $p(k|x,\theta)$ by finding its corresponding z. A GAN can be viewed somewhat like a VAE with only the Decoder $z \to x$, but without the encoder $x \to z$.

(2) Even if we were able to obtain $x \to z$ under GAN, because of its objective function, some training examples may come close to the generated images but might still have nearly zero probability under the generator, making them unable to achieve our goal, i.e., $p(x|k,\theta)$ is may not be a true representation of its likelihood.

A. Posterior Consistency Module (PCM)

In our proposed work, we are overcoming these shortcomings by devising a posterior consistency module (PCM) as part of end-to-end GAN training. The introduction of PCM has allowed us to efficiently changing $p(z)$ in GAN from single to a mixture of Gaussian densities while still be able to compute the responsibility $p(k|x,\theta)$.

In a nutshell, PCM works just like another GAN. The difference is that its generator C_{PCM} generates distribution instead of data structure like a normal GAN. PCM functions by assuming $p(k|x,\theta)$ to have an equivprobability latent counterpart $p(k|z,\theta)$. It is straightforward to compute $p(k|z,\theta)$ analytically:

$$p(k|z,\theta) = w \equiv (w_1,\ldots,w_K) = \left(\frac{\mathcal{N}(z|\mu_1,\sigma_1)}{\sum_{k=1}^K \mathcal{N}(z|\mu_k,\sigma_k)},\ldots,\frac{\mathcal{N}(z|\mu_K,\sigma_K)}{\sum_{k=1}^K \mathcal{N}(z|\mu_k,\sigma_k)}\right). \quad (3)$$

However, the same cannot be achieved by substituting z with x. Therefore, we applied a neural network $C_\theta(x)$ with softmax output $\hat{w} = (\hat{w}_1,\ldots,\hat{w}_K)$. $\hat{w}$ is then compared with w for consistency. This can be viewed as a pseudo reconstruction loss, except we are not reconstructing the data itself, but to reconstruct the probabilities of the categories the data may belong to, i.e., its responsibility probability.

In addition, PCM also needs to try to make the generated $p(k|\hat{x},\theta)$ (from the generated data $\hat{x}$) to look similar to the distribution of the real data x, i.e., $p(k|x,\theta)$.

In an encoder-decoder paradigm, one may view VAE as to have a balanced decoder and encoder; GAN is to have a powerful decoder in terms of its generator, and a very limited encoder in terms of the discriminator which outputs only a scalar between $(0\ldots1)$. Our proposed model can be seen as to have the same powerful Decoder as GAN, but also having a moderately capable encoder which outputs responsibility distribution instead of a single scalar value.

Of course, the PCM alone does not complete the entire picture, and several other innovations have also been introduced to make our model work. In Section II, we explain how each part of the proposed algorithm works.

B. Closely related work

Below we discuss some recent research that are closely related to our work:

GMMs have been applied in GAN networks so that highly diverse datasets can be better modelled. Probabilistic GAN (PGAN) [3] integrates a Gaussian mixture model (GMM) in the discriminator, so that a GMM likelihood loss function is used to optimize the generator and the discriminator. The discriminator encodes images to feature vectors and estimates the GMM parameters from the encodings of real images. The discriminator is updated in each iteration such that the likelihood of the encoded real images are close to 1 and the likelihood of encoded synthetic images are close to 0.

GM-GAN [4] uses the GMM to model the distribution over the latent space. The entire network of GM-GAN is trained end-to-end, i.e. μ_k and Σ_k for each Gaussian indexed by $\{1,\ldots,K\}$ can be updated by the adversarial loss as in Equation I with proper re-parameterization. Authors claimed that GM-GAN can provide a better fit for a dataset that contains highly diversified samples and allow controllable generation.

As both PGAN and GM-GAN incorporates GMM into GANs, the GMM responsibility is not learned in their frameworks.

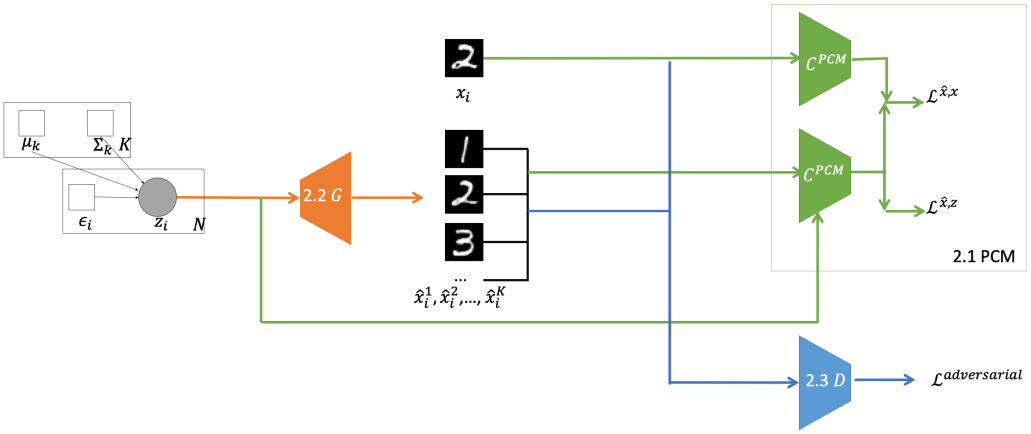


Fig. 2: The overall architecture. The feed-forward logic of the classifier C, the generator G and the discriminator D are marked with different colours.

There are previously other algorithms that aim to perform unsupervised clustering with deep generative models and GMM. GMVAE [1] and VaDE [5] uses GMM to model the latent space of VAE. Therefore, the class assignment $p(k|x)$ can be estimated with $p(k|z)$ with a properly trained model. AAE [6] adds an adversarial network on top of the latent of VAE to guide the posterior distribution to be closer to the prior.

Different from previous algorithms, our work is by far the first work that tries to study the GMM responsibility of complex data through pure GAN structure.

ClutsterGAN [7] is a GAN based network that the latent variable in ClusterGAN is a combination of a one-hot encoded variable and a continuous latent variable. Our work differs from ClusterGAN as the latent variables are sampled from a GMM of which parameters are updated in the training. This also allows us to perform smooth linear interpolation between categories of images, which is not possible in ClusterGAN which captures the categorical information in the one-hot variable.

C. Paper Organization

In Section II, we describe how each component of the proposed mechanism works. In Section III, we demonstrate the performance of the proposed method by comparing it against several baseline models. Section IV concludes the paper.

II. Architecture

The proposed architecture consists of three networks: First, we describe PCM, as well as the loss function that matches it with $p(k|z,\theta)$. Second, we have a generator G which produces synthetic samples from GMM random vectors. Third, a discriminator D which encodes samples to feature vectors and discriminates between real and synthetic samples. The overall architecture design is shown below in Figure 2. In the following sections, we explain the details of the three networks.

A. PCM

The PCM is comprised of a few components. In its core, it is to be comprised of a classifier/ generator C_{PCM} which outputs $p(k|x,\theta)$ given x. When a synthetic data $\hat{x}$ is used as its input, the output approximates $p(k|\hat{x},\theta)$ and it needs to be matched against:

1) $p(k|z,\theta)$, this is to ensure that the responsibility probability condition on its latent variable z (from using Equation 3) is similar to the responsibility distribution dependent on x (from using the neural network). For obvious reason, the corresponding loss is named as:

$$\begin{aligned}\mathcal{L}^{\hat{x},z} =& \mathbb{E}_{\{z_1\sim\mathcal{N}(\mu_1,\Sigma_1),\cdots,z_K\sim\mathcal{N}(\mu_K,\Sigma_K)\}} \\ & \Big[\frac{1}{K}\sum_{k=1}^{K} I(p(k|z_k,\theta), p(k|\hat{x}^k,\theta))\Big] \\ =& \mathbb{E}_{\{z_1\sim\mathcal{N}(\mu_1,\Sigma_1),\cdots,z_K\sim\mathcal{N}(\mu_K,\Sigma_K)\}} \\ & \Big[\frac{1}{K}\sum_{k=1}^{K} I(p(k|z_k,\theta), C^{\theta}_{\text{PCM}}(\hat{x}^k))\Big]. \end{aligned} \tag{4}$$

2) $p(k|x,\theta)$, this is to ensure that the distribution generated is similar to those generated by the real data x. The corresponding loss is named as:

$$\begin{aligned}\mathcal{L}^{\hat{x},x} &= \mathbb{E}_{x_i\sim p_{data}} \sum_{k=1}^{K} I\big(p(k|x_i,\theta), p(k|\hat{x},\theta)\big) \\ &= \mathbb{E}_{x_i\sim p_{data}} \Big[\sum_{k=1}^{K} I\big(C^{\theta}_{\text{PCM}}(x_i), C^{\theta}_{\text{PCM}}(\hat{x})\big)\Big],\end{aligned}$$

where

$$I(X;Y) = \sum_{y\in\mathcal{Y}}\sum_{x\in\mathcal{X}} p_{(X,Y)}(x,y)\log\left(\frac{p_{(X,Y)}(x,y)}{p_X(x)\,p_Y(y)}\right). \tag{5}$$

TABLE I: The overall network structure.

Stage	Sub-stage	Name	Input Tensors	Output Tensors
idrule C	Encoding Network (if shared)	Conv (kernel=5, stride=2) + LeakyReLU	$D_{\text{img}} \times D_{\text{img}} \times D_h$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$
		Conv (kernel=4, stride=2) + Batch norm + LeakyReLU + Flatten	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$
	Encoding Network (if not shared)	Conv (kernel=5, stride=1) + ReLU	$D_{\text{img}} \times D_{\text{img}} \times D_h$	$D_{\text{img}} \times D_{\text{img}} \times 32$
		MaxPool (pool_size=2, stride=2)	$D_{\text{img}} \times D_{\text{img}} \times D_h$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 32$
		Conv (kernel=5, stride=2) + ReLU + Flatten	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 32$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 16$
		Linear + ReLU	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 16$	1024
	Classification Network (if shared)	Linear	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$	K
	Classification network (if not shared)	Linear	1024	K
G	Linear + Batch norm + LeakyReLU + Reshape	$1 \times D_z$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 256$	
	Transposed Conv (kernel=5, stride=1) + Batch norm + LeakyReLU	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 256$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$	
	Transposed Conv (kernel=5, stride=2) + Batch norm + LeakyReLU	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$	
	Transposed Conv (kernel=5, stride=2) + Tanh	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$	$D_{\text{img}} \times D_{\text{img}} \times D_h$	
D	Encoding Network	Conv (kernel=5, stride=2) + LeakyReLU	$D_{\text{img}} \times D_{\text{img}} \times D_h$	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$
		Conv (kernel=4, stride=2) + Batch norm + LeakyReLU + Flatten	$D_{\text{img}}/2 \times D_{\text{img}}/2 \times 64$	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$
	Discriminator Network	Linear	$D_{\text{img}}/4 \times D_{\text{img}}/4 \times 128$	1

In reality, all we need is a measure function to compute the distance between two distributions, so KL or DS divergence can also be used. However, in our experiment, mutual information loss provides the best result.

In terms of the neural network design, the classifier has two plausible alternatives: the first is to share the feature encoding layers with the discriminator. The shared feature encoding network encodes each image to a feature vector, and the classifier will be a simple standard linear softmax classifier built on top of the features. The second is to build a stand-alone network that contains multiple CNN layers to classify images.

Details of both network designs can be found below in Table I. In Table I, we use acronyms for operations in the table: "Conv" is the convolution operation, of which the kernel and stride size are in the bracket; "Batch norm" is short for batch normalization; "Flatten" refers to the operation that flattens a tensor to 1D array. D_{img}, D_h and D_z are related to the datasets we use, the exact value of each are reported in Section III.

In Section III-C, we report the results of both classifier designs in terms of generation performance and computation costs.

Parameters of the classifier are optimized by both the adversarial loss and the mutual information loss, the details of how the update is performed are introduced in Section II-D.

The C_{PCM} is not required for generating new images after the model is fully trained. During testing, a random vector is sampled directly from the GMM model for the generation. The C_{PCM} can then used to assign an unseen image to Gaussian and subsequently perform segmentation on the testing set.

B. The Generator

The ideal output of the trained generator is that all random vectors sampled from the same Gaussian of the GMM should generate similar images when fed through the generator. However, as our training is label independent, one cannot control the correspondence between which Gaussian is selected and the category of the generated sample during training. Therefore, instead of sampling a random vector from one Gaussian during training, we sample one vector from all Gaussians during training and we use the classification output to weigh the loss values. The details are explained below.

We sample one random vector from each Gaussian as $z_i = [z_i^1, \ldots, z_i^K]$. These vectors are used to generate K synthetic images $[\hat{x}_i^1, \ldots, \hat{x}_i^K]$. The classification output is used to weigh the adversarial loss calculated from each pair of a generated sample $\hat{x}_i^k$ and the real image x_i.

In addition, as our design expects the training to be performed in an "end-to-end" fashion, the reparameterization trick is applied to the z_i sampling process so that the back-propagation can be used to update the parameters μ_k and Σ_k of each Gaussian. Instead of sampling $z_i^k \sim \mathcal{N}(\mu_k, \Sigma_k)$, we define $z_i^k = \Sigma_k \epsilon + \mu_k \ \ \forall k \in [1, K]$, where ϵ is sampled as $\epsilon \sim \mathcal{N}(0, I)$.

The generator structure used in our experiments is in Table I. In the table, "LeakyReLU" which is short for "leaky rectified linear unit" and "Tanh" are activation functions.

C. The Discriminator

The design of the discriminator has two options: whether or not the image encoding layers are shared with the classifier. The image encoding network is followed by a standard linear logistic regression to identify the given image to be real or fake.

The adversarial loss, which is calculated over K pairs of real and synthetic samples as:

$$\mathcal{L}^{\text{adversarial}} = \mathbb{E}_{x_i \sim p_{data}}\Big(\frac{1}{K}\sum_{k=1}^{K} p(k|x_i, \theta) \times (\log D(x_i) + \log(1 - D(\hat{x}^k)))\Big). \tag{6}$$

D. Training parameters for the prior distribution

In our setting, the prior distribution is a GMM model, the two trainable variables are means for K Gaussians $\mu \in \mathbb{R}^{K \times D_z}$ and standard deviations for K Gaussians $\sigma \in \mathbb{R}^{K \times D_z \times D_z}$. Both variables are updated by the adversarial loss in addition to the mutual information loss as the training is performed "end-to-end".

Below we give the pseudocode about how the updates are performed on each network in one iteration.

TABLE II: Statistics of the different datasets used in the empirical evaluation.

Dataset	Number of Classes	Data Dimension	Train Samples	Validation Samples	Test Samples	Number of Epochs	Learning Rate γ	D_{img}	D_h
idrule MNIST	10	$28 \times 28 \times 1$	60,000	-	10,000	200	0.0002	28	1
Fashion-MNIST	10	$28 \times 28 \times 1$	60,000	-	10,000	200	0.0002	28	1
Oxford-102 Flower	102	$64 \times 64 \times 3$	1,020	1,020	6,149	10,000	0.0002	64	3

Algorithm 1 Training the proposed model for 1 iteration

Require: $\mathrm{X} = [x_1, x_2, \ldots, x_M]$ - M training images in one batch

for $i = 1 \ldots M$ do
  Classify x_i into K Gaussians
  for $k = 1 \ldots K$ do
    $\epsilon \sim \mathcal{N}(0, I)$
    $z_k = \Sigma_k \epsilon + \mu_k$
    $\hat{x}_i^k \leftarrow G(z_k)$
    Classify $\hat{x}_k$ into K Gaussians
  end for
  Calculate $\mathcal{L}^{\text{adversarial}}$ from x_i and $[\hat{x}_i^1 \ldots \hat{x}_i^K]$ as in Equation 6
  Calculate $\mathcal{L}^{\hat{x},z}$ as in Equation 4
  Calculate $\mathcal{L}^{\hat{x},x}$ as in Equation 5
  Update the discriminator with $\mathcal{L}^{\text{adversarial}}$
  Update both the generator and the C_{PCM} with $\mathcal{L}^{\text{adversarial}}$
  Update the classifier with $\mathcal{L}^{\hat{x},x}$
  Update the classifier, parameters for the latent distribution with $\mathcal{L}^{\hat{x},z}$
end for

III. Experiments

In this section, we evaluate the performance of the proposed method by comparing it with several baselines.

A. Experiment setup

The datasets we use are the MNIST [8], Fashion-MNIST [9] and Oxford-102 Flower [10] datasets. The details are listed below in Table II. In particular, we only select a subset of Oxford-102 to perform the training, which is the images that belong to the first 10 classes. For experiments performed on each dataset, we used different hyper-parameters, the details are listed in Table II.

B. Linear Interpolation across Gaussian

Below in Figure 4, we show samples generated by the proposed model trained on several datasets in a completely unsupervised manner. We set the number of Gaussians equal to the total number of classes of the dataset in all experiments. When we are performing the linear interpolation as in the right panels, the random vector z for each image generation is calculated as $z = \Sigma_k \epsilon + \mu_k$ $\forall k \in \{1, \ldots, K\}$, where ϵ is sampled as $\epsilon \sim \mathcal{N}(0, I)$. ϵ is kept the same for all images for each dataset.

We can draw two conclusions from the results in Figure 4. First, a fully trained proposed method can learn to "allocate" each class of image to a Gaussian. Second, the trained model can be used to perform smooth linear interpolation between Gaussians and even among more than two Gaussians. In Figure 3, we demonstrate the linear interpolation performed over three categories. The proportion of Gaussians of the synthetic images can be set manually.

C. Image Generation Quality

The generation performance is measured with two commonly used metrics: Inception score [11] and Fréchet Inception Distance (FID) score [12]. Inception score is calculated as $I = \exp(\mathbb{E}_x D_{\text{KL}}(p(y|x)||p(y)))$, and a higher value generally indicates a better performance. where x is a generated image and y is the label predicted by the Inception model [13]. FID score is another metrics that measures the image generation quality. A lower value shows a better image quality and diversity. It calculates the difference between real images x and generated images g as $\text{FID}(x, g) = ||\mu_x - \mu_g||_2^2 + \text{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{\frac{1}{2}})$.

Note that limitations of both Inception and FID score have been pointed out in several previous literature [14], [15], and there is currently no "perfect" metrics at this moment. These two metrics are used as an indication rather than a hard measure.

This evaluation is performed on the Oxford-102 dataset. As the Inception score is suggested to be evaluated on a large enough number of samples, we generate 5K synthetic samples to calculate both values. Figure 5 plots Inception scores and FID scores calculated over the training epochs. As the figure shows, the proposed framework, whether or not the encoding layers are shared, constantly achieves higher inception scores and lower FID scores from the start. Applying shared encoding layers allows an even superior performance.

In Table III we report the number of trainable parameters, the best Inception and FID score achieved for each algorithm. The proposed framework, with separate feature encoding layers, results in 10.81% higher Inception score and 3.48% lower FID score compared with GM-GAN. The shared encoding layers provide 5.74% higher Inception score and 10.85% lower FID score compared with the one without the shared layers.

TABLE III: Number of parameters, Inception scores and FID scores of the proposed method and the baselines.

	number of parameters	Inception Score ↑	FID score ↓
Proposed (encoding not shared)	13,005,411	2.9664 ± 0.2188	231.0577 ± 7.5371
Proposed (encoding shared)	8,794,835	3.1368 ± 0.1596	205.9776 ± 7.8587
GM-GAN	8,467,145	2.6770 ± 0.1079	239.3936 ± 6.7672
Vanilla GAN	8,366,145	2.4882 ± 0.1065	247.0610 ± 7.2361

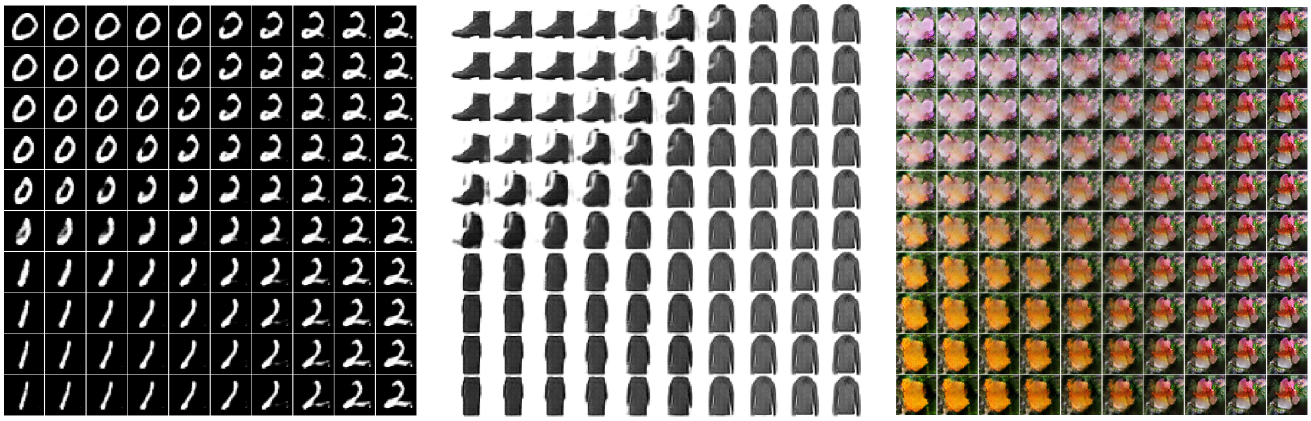


Fig. 3: Linear interpolation over 3 Gaussians on the MNIST, Fashion-MNIST and Oxford-102 dataset.

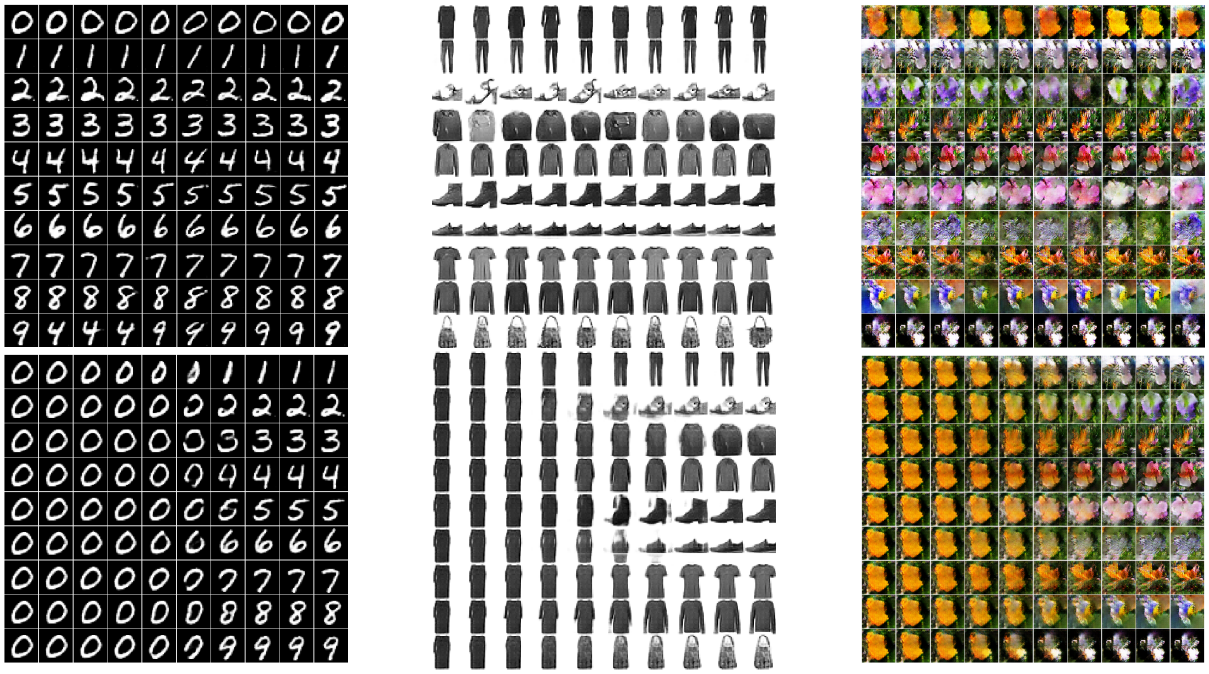


Fig. 4: Samples generated by the proposed models trained on the MNIST (left column), Fashion-MNIST (middle column) and CIFAR-10 (right column) datasets. The top row contains images generated using random vectors sampled from a different Gaussian. The bottom row shows the linear interpolation result from one Gaussian to another. Note that the index of Gaussian does not necessarily correspond to the actual digit, generated images are reordered for demonstration purpose.

From Figure 5 and Table III, it is clear to see that the proposed method is able to out-perform previous baseline models in terms of image generation quality. The shared feature encoding layers would further improve the performance and reduce the size of the network.

D. Network Compression

In this section, we demonstrate that applying Gaussian mixture to GANs can significantly reduce the number of parameters while achieving close performance compared with the vanilla GAN. Figure 6 plots Inception and FID scores over training epochs of the proposed GMM-based GAN with 1/2, 1/4, and 1/8 the size of the original. The original size of the network is demonstrated in Table I, 1/2 means that the number of parameters in each layer is reduced by half, same for 1/4 and 1/8. These results are compared with the vanilla GAN, which uses the original number of parameters.

Figure 6 shows that the proposed approach, achieves a higher Inception score and lower FID score on the CUB dataset compared with the vanilla GAN, while using only 1/2 of the number of parameters.

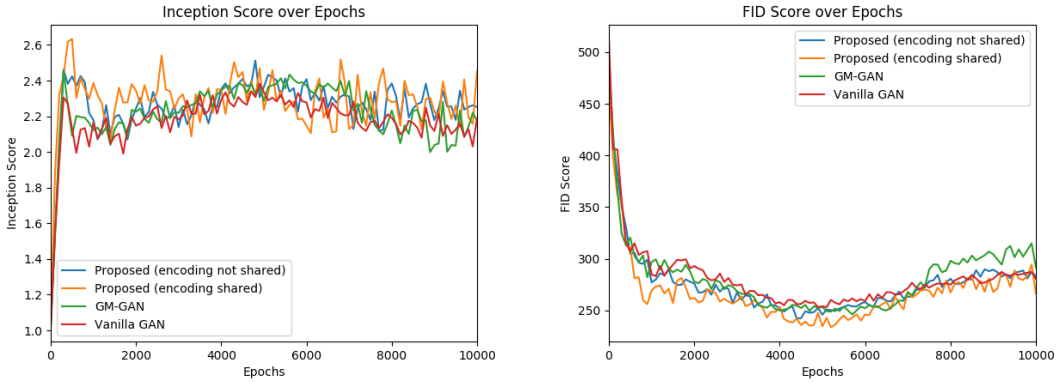
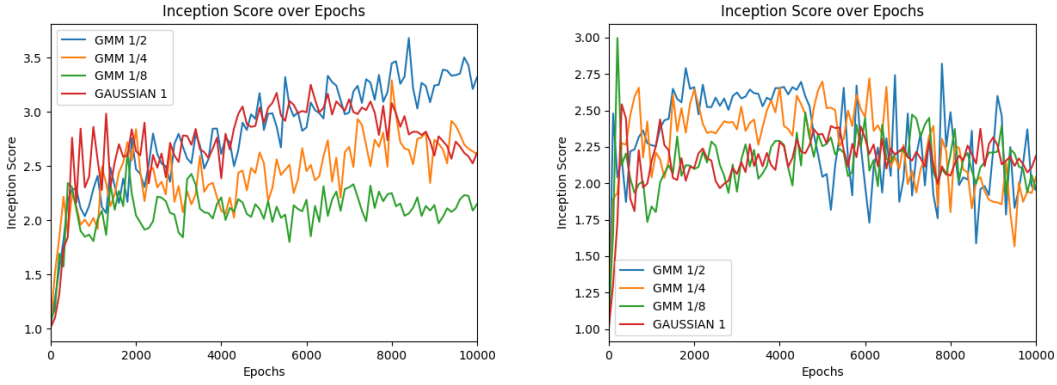


Fig. 5: Inception score and FID score over training epochs on the Oxford dataset.



(a) Inception scores on the CUB dataset.

(b) Inception scores on the Oxford-102 dataset.

Fig. 6: Inception scores over epochs for the proposed GMM-based GAN with various size of the network and the vanilla GAN.

E. Performance on Highly Imbalanced Dataset

Training GAN networks often suffers from mode collapse, therefore we also demonstrate the performance on the highly imbalanced dataset as in Figure 7.

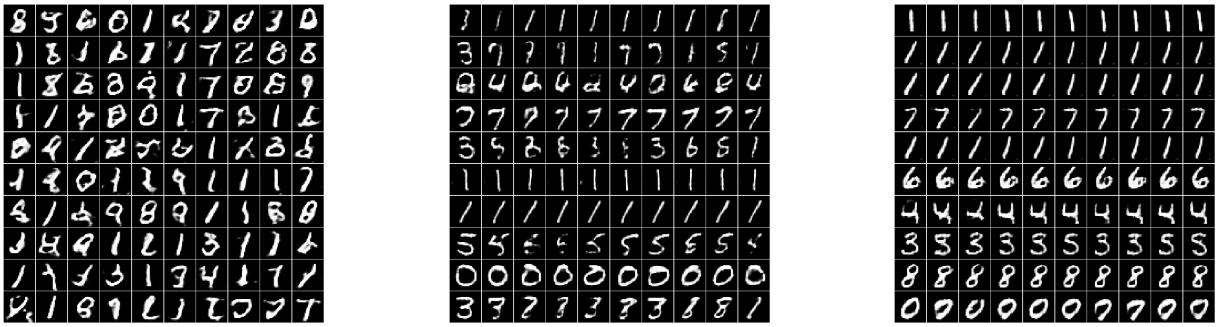
The imbalanced dataset we used is a subset of the original MNIST dataset. From digit 0 to digit 9, we randomly choose digit d as the dominant category. $1,000$ samples are randomly selected from all other categories and the category d is kept the same. This becomes the imbalanced training dataset. Figure 7 shows the synthetic samples generated by the three algorithms when the digit 1 becomes the dominant category in the imbalanced dataset. While using both the standard Gaussian and GMM result in poor generation quality and collapsed output, the proposed framework is able to deliver digits with clear shape and categorization.

IV. Conclusion

In this paper, we proposed a novel framework to better capture the latent structure of a complex dataset. Under this framework, GAN generator's simple distribution was replaced by GMM. Subsequently, many technical innovations were proposed to make this framework able to be trained in an "end-to-end" fashion. Most noticeably, a Posterior Consistency Module was innovated to help the model to better approximate GMM's responsibility distribution given the data. In addition to GMM modelling, we also demonstrate the multitude of benefits in our approach: (1) We demonstrated through experiments that our proposed method retains GAN's ability in sharper image generation compared with other GMM GAN methods. (2) the proposed approach can significantly save the computation cost as only half number of parameters is required to achieve the same image generation quality compared to classic DCGAN. (3) Thanks to the mixed densities, the proposed method surpasses previous baselines when dealing with highly imbalanced dataset.

References

[1] 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," CoRR, vol. abs/1611.02648, 2016. [Online]. Available: http://arxiv.org/abs/1611.02648



(a) Gaussian as the latent distribution (b) GMM as the latent distribution (c) The proposed framework

Fig. 7: Performance on high imbalanced dataset.

[2] 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 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[3] H. Eghbal-zadeh and G. Widmer, "Probabilistic generative adversarial networks," CoRR, vol. abs/1708.01886, 2017. [Online]. Available: http://arxiv.org/abs/1708.01886

[4] M. Ben-Yosef and D. Weinshall, "Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images," CoRR, vol. abs/1808.10356, 2018. [Online]. Available: http://arxiv.org/abs/1808.10356

[5] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, "Variational deep embedding: A generative approach to clustering," CoRR, vol. abs/1611.05148, 2016. [Online]. Available: http://arxiv.org/abs/1611.05148

[6] A. Makhzani, J. Shlens, N. Jaitly, and I. J. Goodfellow, "Adversarial autoencoders," CoRR, vol. abs/1511.05644, 2015. [Online]. Available: http://arxiv.org/abs/1511.05644

[7] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, "Clustergan : Latent space clustering in generative adversarial networks," CoRR, vol. abs/1809.03627, 2018. [Online]. Available: http://arxiv.org/abs/1809.03627

[8] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist, vol. 2, 2010.

[9] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," CoRR, vol. abs/1708.07747, 2017. [Online]. Available: http://arxiv.org/abs/1708.07747

[10] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training gans," in Advances in Neural Information Processing Systems 29, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2234–2242. [Online]. Available: http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf

[12] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a nash equilibrium," CoRR, vol. abs/1706.08500, 2017. [Online]. Available: http://arxiv.org/abs/1706.08500

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," CoRR, vol. abs/1512.00567, 2015. [Online]. Available: http://arxiv.org/abs/1512.00567

[14] S. Barratt and R. Sharma, "A note on the inception score," arXiv preprint arXiv:1801.01973, 2018.

[15] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," in Advances in Neural Information Processing Systems 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 700–709. [Online]. Available: http://papers.nips.cc/paper/7350-are-gans-created-equal-a-large-scale-study.pdf