Dauphine | PSL
UNIVERSITÉ PARIS

Generative Adversarial Networks

Report Submitted By:

TEAM: hooliGANs

Alexandre Olech
Lucas Rousselet
Kanupriya Jain

Contents

1 Introduction

Classic Generative Adversarial Networks (GANs) have key limitations, such as mode collapse, where the generator produces limited output varieties, leading to poor generalization and a lack of diversity. This makes GANs unreliable for tasks requiring broad variation. Additionally, classic GANs lack a mechanism to ensure generated samples align with specific data classes, hindering control and interpretation, especially in class-conditional generation tasks. These challenges have driven research into various GAN extensions aimed at overcoming these issues. This report summarizes our work on improving image generation quality and diversity in Generative Adversarial Networks (GANs) through advanced latent space modeling techniques.

We experimented with different variations of Gaussian Mixture GANs (GM-GANs), including static models, static models enhanced with Wasserstein loss, static models with an Expectation Maximization approach where we reconstructed the latent space by inverting the generation process with the real images, and a dynamic model with supervision, which ultimately proved to be the best-performing model. With this last approach, we achieved measurable improvements in image quality and diversity, as evidenced by enhanced Fréchet Inception Distance (FID), precision, and recall metrics. To further investigate the effectiveness of our approach, we also used visualizations of the generated images, confirming better inter-class diversity and quality within each class, highlighting the interpretive benefits of using supervised GM-GANs for class-distinct and high-fidelity samples.

2 Gaussian Mixture Generative Adversarial Networks

2.1 Background

This method investigates the probability distribution used to sample latent vectors as suggested in [Ben-Yosef and Weinshall, 2018].The Gaussian Mixture GAN (GM GAN) framework introduces a probabilistic approach to sampling latent vectors, where a Gaussian mixture distribution is used to enhance the generator's ability to produce diverse and realistic samples. There are mainly two approaches, **"Supervised"** and **"Unsupervised"**. In the supervised setup, the discriminator was designed to output a probability distribution across the real data classes, along with an additional logit specifically dedicated to identifying fake samples. This extra logit, assigned to indicate the "fake" class, is set to 1 for fake samples, signaling the discriminator to recognize them explicitly as synthetic. This approach allowed the discriminator to perform a dual task: classifying real samples correctly across multiple categories while also learning to detect fake samples distinctly. Simultaneously, this modified approach forces the generator to produce samples that correspond to each cluster in the real data space. We explored both the static and dynamic settings for the supervised approach. In the unsupervised approach, the discriminator outputs a probability distribution. However, rather than distinguishing known classes, it learns to identify distinct clusters or modes in the data and a logit for the "fake" class, differentiating synthetic samples from real data.

2.2 Experiments

2.2.1 Static GM-GAN (Supervised)

GANs_static_supervised.ipynb

In this basic GM-GAN model, the parameters of the Gaussian mixture distribution are fixed prior to training and remain constant throughout. More specifically, each of the mean vectors μ_k is uniformly sampled from the multivariate uniform distribution $U[-c, c]^d$, and each of the covariance matrices Σ_k has the form of $\sigma \cdot I_{d\times d}$, where $c \in \mathbb{R}$ and $\sigma \in \mathbb{R}$ are hyperparameters left to be determined by the user.

We used the above-mentioned approach of supervised static GM GAN. For this method, we took $\sigma = 1.0$, $c = 1.0$, $K = 11$, which is the number of components of the Gaussian Mixture Model and latent dimension $d = 200$. We obtained a **FID = 251.592**. These results were not satisfactory. The following images depict the Discriminator and Generator loss and their norms of gradients-

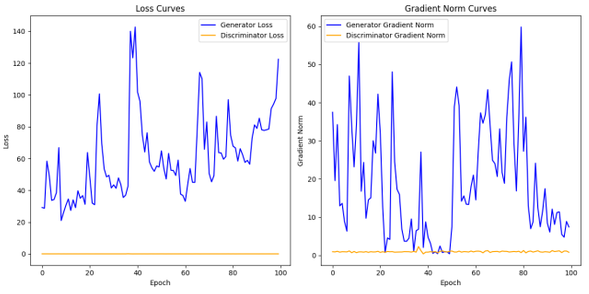

Figure 1: Loss and Gradient Norm of Generator and Discriminator Vs Epochs

We can observe that the Generator loss and gradient are very unstable with very high loss and gradient norm value which resulted in a very high FID.

2.2.2 Static GM-GAN with WGAN loss (Unsupervised)

Static_GM_GAN_with_WGAN_loss.ipynb

To address the instability observed in the initial model, we modified the static GM GAN by incorporating a Wasserstein GAN (WGAN)(Milne and Nachman [2021]) framework with Gradient penalty. WGANs replace the traditional GAN loss function with the Wasserstein distance (Earth Mover's Distance). This is an unsupervised framework which does not take into consideration the labels of the images.

The WGAN-GP loss function is defined as follows:

$$L_D = \mathbb{E}_{\tilde{x}\sim P_{\mathcal{X}}(\tilde{x})}\left[D(\tilde{x})\right] - \mathbb{E}_{z\sim P_{\mathcal{Z}}(z)}\left[D(G(z))\right] + \lambda\,\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left(\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2\right). \quad \text{and} \quad L_G = -\mathbb{E}_{z\sim P_{\mathcal{Z}}(z)}[D(G(z))]$$

Under the same above-mentioned set-up for mean and variance but with a different loss function, we experimented with five different values of $\sigma = 0.2, 0.5, 1.0, 1.5, 2.0$. We initialized the mean randomly. We were not able to experiment much due to lack of resources. Each model took around 40-50 minutes to train for 100 epochs. The curve in Figure 4 shows how FID varies with different σ values.

2.2.3 Static GM GAN with Expectation Maximization on the real images latent space (Unsupervised)

EM_7_find_best_ncomponents_with_hpo.py

As described in detail in section 3, we were able to invert the generation process for the real images, and to fit a Gaussian Mixture model to the distribution of the latent space of real images, using the Expectation Maximization algorithm. Resampling from the modeled distributions, with 10 components in the Gaussian mixture, proved successful in generating realistic-looking digits. However, it led to worse results for FID, precision and recall (Table 1) on the DS LAB evaluation platform, compared to the same model with the original Gaussian sampling. Interestingly, the precision diminishes by only 0.02 points, while the recall loses 0.15 points. A possible interpretation is that the remodeled distribution does not cover a sufficient part of the latent space and mainly covers the parts associated with a set of images, which leads to similar images and decent precision, but results in fewer possibilities and thus lower recall and lower diversity. An interesting area of improvement would be to tuning the variance of the different components in the Gaussian mixture, so that the distributions are less sparse, and to see if it leads to a better recall.

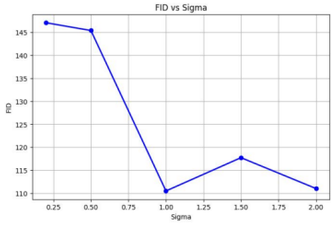

Figure 2: Sigma values Vs FID for static GM-GAN Model with WGAN Loss

Best Results:
$\sigma = 1.0$
FID: 110.47
Further improvements can be done by tuning the other hyper-parameters.

Table 1: Comparison of the DS LAB platform results obtained from two sampling strategies (Gaussian vs. remodeled Gaussian mixture), with the same GAN.

	FID	Precision	Recall
Gaussian Sampling	28.75	0.52	0.26
Reconstructed Latent Space GM sampling	39.58	0.5	0.11

GMGAN/GMtrain.py
GMmodel.py
GMutils.py

2.2.4 Dynamic GM-GAN (Supervised) (BEST MODEL)

In this method, the mean vectors and covariance matrices are initialized as in the static model but are made learnable, allowing them to be optimized during GAN training. This is also the supervised framework because it involves labels. In this setup, we introduce an additional network of learnable parameters that we call Gaussian Mixture model (GM) to better model the latent space and guide the generator to produce diverse samples aligned with real data classes. Below are the modified loss functions for the discriminator, and the combined loss involving the Gaussian mixture model and the generator.

The discriminator loss L_D is now given by:

$$L_D = -\mathbb{E}_{x\sim p_X(x)}\left[\log D(x)_{y(x)}\right] - \mathbb{E}_{z\sim p_Z(z)}\left[\log D(G(z))_{k+1}\right]$$

where:

- $\mathbb{E}_{x\sim p_X(x)}\left[\log D(x)_{y(x)}\right]$ is the cross-entropy loss for correctly classifying real samples,
- $\mathbb{E}_{z\sim p_Z(z)}\left[\log D(G(z))_{N+1}\right]$ is the cross-entropy loss for classifying generated samples as fake, with the "fake" class labeled as $k+1$, k being the number of classes.

The generator and the Gaussian Mixture Model (GMM) share the same objective, which is to generate samples that are aligned with the real data clusters. The GMM's parameters, including the means and covariances, are learned simultaneously with the generator to optimize the loss :

$$L_{G+\text{GMM}} = -\mathbb{E}_{z\sim p_Z(z)}\left[\log D(G(z))_{f(y(z))}\right]$$

This loss encourages the generator G to produce samples aligned with the class distributions represented by the GMM, while the GMM's parameters (means and covariances) are adjusted to provide a more accurate latent space representation for generating realistic samples. We obtained the best results with this method. With $K = 10$, $\sigma = 1.4$ and $d = 100$, this yielded a FID Score of **15.67**.

In section 4, we cover experiments aimed at understanding why this approach yields such good results.

3 Latent space modeling with a Gaussian Mixture

3.1 Motivation

This section presents a series of experiments aiming at performing latent space modeling. Within the vanilla GAN framework (by which we mean the initial version of the GAN provided to us for the DS LAB second project), during image generation, the generator takes a variable z as input, which is sampled from a random multivariate Gaussian distribution, and returns an image x. Once such a GAN has been trained, the first idea is to "invert" the generation process on real images, that is, for each real image x, to find the best value of z such that $G(z)$ is very close to x. Then, we obtain a set of values z_{real} that can be seen as the pseudo-inverses $G^-1(x)$ for each real image x.

Then, the second idea is to sample from the distribution of z_{real}, and to see if it improves the diversity and quality of the image generation. For that, we need to build a probability distribution that corresponds to the observed distribution z_{real}. Since this distribution may be more sparse and more complex than a Gaussian, it sounds promising to model it as a Gaussian mixture model, as seen in [Bishop and Bishop, 2023]. As suggested by [Bishop, 2006], this can be done, for example, using the Expectation Maximization (EM) algorithm.

3.2 Inverting the generation process of a basic trained GAN

3.2.1 "Inversion" through gradient descent

EM_2_get_z_real_all.py
EM_3_get_z_real_all_multiprocessing.py

To invert the generative process for each real image x, we defined a simple loss function, which expresses the distance between x and $G(z)$ with the L_2-norm:

$$L(z) = \|x - G(z)\|_2$$

For each real image x_i, we initialized and updated z with 1000 iterations of gradient descent in order to reduce the loss (we made sure that the gradients of the generator were not updated, since the goal was to optimize z for a fixed generator). We then stored the obtained value z_i.

3.2.2 Visualizing the regenerated real images

EM_5_generate.py

The defined loss is likely to not be convex, due to the fact that G is a neural network with feed-forward layers, which usually does not result in a convex function. Thus, reducing the loss might not lead to a global minimizer.

However, the comparison (Figure 3) between real images x_i and the regenerated images $G(z_i)$ shows that, when we randomly pick 20 pairs of images, the regenerated images are very similar to the original, and the classes always seem to match. Thus the optimization succeeds at finding points in the latent space that correspond to a specific class of number with a specific appearance.

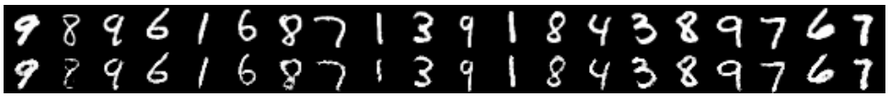

Figure 3: Comparison between real images x_i (top) and regenerated images $G(z_i)$ (bottom) for 20 random (uniformly drawn) pairs. The z_i were obtained by inverting the generation process for each real image x_i.

3.3 Modeling the latent space of real images: Gaussian Mixture with the Expectation Maximization algorithm

EM_4_fit_GM.py
EM_7_find_best_ncomponents_with_hpo.py

As suggested by [Bishop, 2006], fitting a Gaussian Mixture to a set of points can be done with the Expectation Maximization (EM) algorithm.

We implemented this approach with the sklearn *GaussianMixture* class, and we created an MLflow experiment to track the FID results associated to sampling from this Gaussian Mixture.

We then experimented with different values for the number of components in the Gaussian Mixture. The relation with the corresponding FID values can be visualized in Figure 4. The FID globally decreases from 0 to 10, and then starts increasing again, with the exception of the value 17, which gives the lowest FID. Of course, this lowest value of 17 may be due to randomness and sampling more runs would be needed to get a more complete picture of the relation. With a number of components equal to 10, the idea that the distribution has one cluster per class would be a plausible scenario.

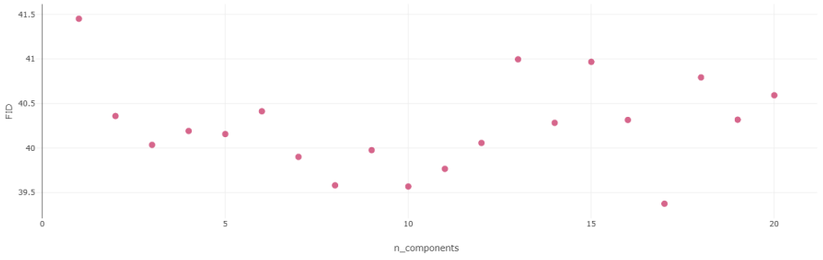

Figure 4: Relation between the number of components in the Gaussian Mixture fitted to the reconstructed latent space and the FID obtained from resampling from this distribution.

4 Interpretative stance on the image generation improvements induced by our best architecture (Dynamic GM GAN with supervision)

4.1 Motivation

The goal of this section is to compare our most promising model (Dynamic GM GAN with supervision) to a baseline, simple model, with the idea of checking the results on the different metrics, and understanding how they translate to the quality and the diversity of the generated images.

4.2 Training a baseline model for the comparison

experiment_latent_state_dim.py

When building complex models, we believe that it is always a good practice, in machine learning, to justify the use of these more complex models by showing that they outperform a tuned version of a simpler model. This is not always done in practice, but we believe it is important, because most of the time, a simpler model is usually preferable to a more complex model when the performance gain is not significant, for a variety of reasons (energy consumption, implementation time, inference time, debugging cost, etc).

Thus we allocated time to tuning some hyper-parameters of the Vanilla GAN to have a tuned baseline. In particular, we created an MLflow experiment to look for the best value of the dimension of the space on which z is sampled before being used as input to the generator. The results of this experiment suggest that the best value for this dimension of the latent space is to be found around 200. We thus used this configuration as our baseline model.

4.3 Improvements on the three metrics (FID, precision, recall)

GMgenerate.py

We compared our most promising model to the tuned baseline, using the values obtained with the DSLab platform for both. The results can be visualized in Table 2. They indicate that our best model leads to significant improvements on all of the three considered metrics: lower FID, higher precision and higher recall.

Table 2: Comparison of our most promising model with a tuned simpler baseline model. Results were obtained with the DS Lab evaluation platform.

	FID	Precision	Recall
Tuned baseline (Vanilla GAN with d=200)	27.73	0.52	0.2
Supervised Dynamic GM GAN	15.67	0.55	0.29

4.4 Improvements on image generation

We have shown in the previous section that our best model outperforms the baseline in all of the three considered metrics. Now, we ask the following questions: how does this translate to image generation ? On what aspect are the generated images better, in terms of quality and diversity ? How can we explain such qualitative improvements by considering the architecture of the GM Dynamic Supervised GAN ?

4.4.1 Training a ResNet classifier on MNIST to infer the labels of generated samples

train_resnet.py

To interpret the improvements induced by our best model, we wanted to have labels of the generated digits at our disposal. Since the generated images do not come with a label (or at least for not all of the models we wanted to compare), we had to infer the classes with a classifier trained on the MNIST dataset. We thus trained a ResNet model, which is known to perform very well for classifying the MNIST digits.

Of course, this approach has its limits because it relies on the quality of the generated images and on the performance of the classifier. If the images are of low quality, or if the classifier struggles too much in classifying images, we will usually not get meaningful labels for the generated images, and it might be difficult to infer the quality

of image generation from such an approach in such context. However, for the MNIST dataset, the data is of reasonably good quality (it is relatively well annotated and most numbers are easily distinguishable for humans) and state of the art classifiers are known to easily achieve 0.99 accuracy nowadays. Thus we thought that such an approach could help us understand the labels of our generated images, which have reasonable quality.

4.4.2 Visualization with the inferred labels

With these inferred labels, we were able to implement three types of visualizations that compare our baseline with our best model in terms of generated images . Figure 5 displays samples with the inferred labels. Figure 6 displays the distribution of (inferred) classes. Figure 7 displays t-SNE representations of samples.

We will use these visualizations in the analyses of the two next sub-subsections.

plot_classes_distribution_and_100_img_with_labels_and_get_frame.py

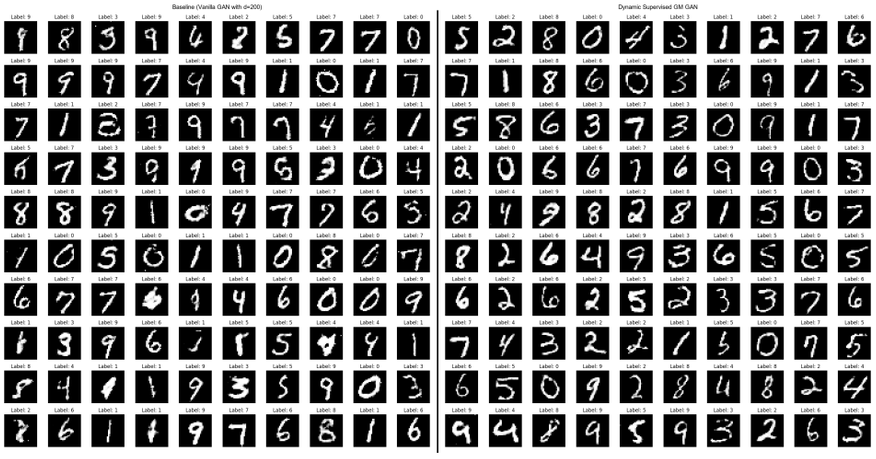

Figure 5: Generated digits with the baseline model (left) and our best model (right).

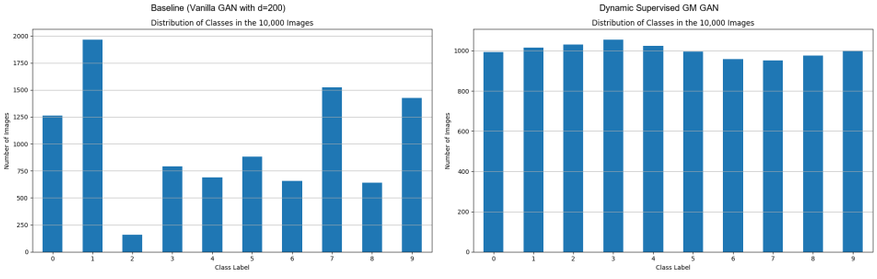

Figure 6: Distribution of (inferred) classes in the digits obtained with the baseline model (left) and with our best model (right).

plot_2d_rpz_of_samples_with_classes.py
plot_2d_rpz_of_real_samples_with_classes.py

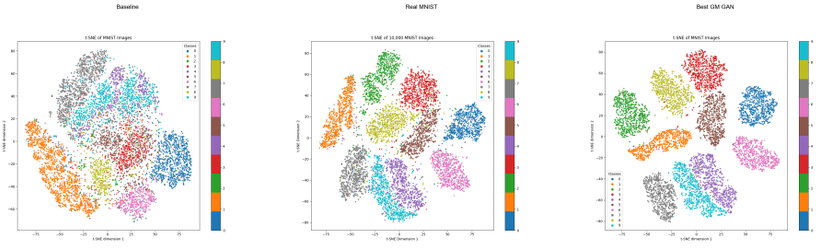

Figure 7: t-SNE representations of the digits colored with the classes (for real MNIST data) or the inferred classes (for the generated digits of the baseline and of our best model). The vanilla model corresponds to the left plot, the real MNIST images to the second plot and our best GM model corresponds to the right plot.

4.4.3 Improved inter-class diversity

As can be seen in Figure 6, our baseline model has a strong bias towards specific classes in the generated digits. It has a very low probability of generating the digit 2, for instance. When we look at the t-SNE representation of the baseline digits, using Figure 7, we find that there is no visible cluster for this digit, whereas others, such as the digit 0, have a well-separated cluster and a more important probability of being generated. The number 1 has the biggest probability of being generated, and it also has the biggest cluster in the t-SNE plot, compared to other digits. A very plausible explanation is that the baseline model has become bad at generating the digit 2, but became decent at generating other digits, such as the digit 1, for instance, and thus focuses on such digits to better fool the discriminator. Since the loss of the baseline model only incites the generator to fool the discriminator, independently of the class, the generator may completely ignore one class, and prefer to favor other classes, which explains the situation we observe here.

For the GM model, looking at the distribution of classes in Figure 6, we see that it is not biased towards a specific class. The explanation likely comes from the fact that this supervised dynamic GM GAN has a different loss, which enforces that the generator improves at generating real-looking digits of a specific class, for each class. Thus it cannot adopt strategies that consist in focusing on specific digits and ignoring others. As a result, it is forced to search for a fitting sub-space for each number.

This property of the GM GAN results in more inter-class diversity in the generated images.

4.4.4 Improved quality for each class of images

The baseline model has a tendency to generate hybrid numbers, as can be seen in Figure 5. For instance, the digits in position (2,5) and (5,6) on the left side of the figure look both like 4 and like 9. Furthermore, when we look at the points of classes 4 and 9 in Figure 7, we see that these two digits share a common region together. In practice, such images are likely to be good for fooling the discriminator, because they take advantage of looking both like 4 and like 9, thus potentially adding up the realistic-ness associated to the two. But in practice, such hybrids are low quality, since humans usually do not write 4 that look like 9 and vice versa. This is particularly true for the images where it looks like a small part of the 9 was removed in the top when writing the number, but this does not correspond to an actual writing pattern in terms of writing movements.

In the GM model, the class clusters in the t-SNE are well separated and we do not observe such hybrid numbers. This is due to the fact that it explicitly models the multimodal distribution of the data, where each class is represented by a single mode. As a result, this enhances class-specific features in the generated digits, which leads to clearer separation of classes and thus to an improvement in quality.

5 Future experimentations

5.1 Sub-class GM GAN

To enhance our GMGAN, we would like to reach sub-clusters within the primary classes of MNIST (e.g., variations in how each digit is written). Our goal is for the generator to capture subtle intra-class variations, making generated samples more diverse and realistic by learning the nuances in each digit's appearance. Implementing sub-clusters in GM-GAN presents several technical challenges:

- Lack of Sub-Cluster Labels: MNIST only provides class labels for the digits (0–9) without further information on sub-categories or writing styles. This makes it difficult for the GAN to learn intra-class variations without explicit supervision.
- Training Stability with Multiple Gaussians: Expanding the Gaussian mixture to represent multiple sub-clusters within each class requires a carefully balanced training process. With more components in the mixture model, the GAN is more prone to instability, as the generator and discriminator have to learn to recognize and generate samples across a much larger number of modes.
- Unknown and Potentially Unbalanced Sub-Class Distribution: The underlying distribution of sub-classes within each digit class is unknown and may be unbalanced, as some sub-styles of a digit (e.g., "4" with an open top versus closed top) could be more frequent than others. This lack of balance and knowledge about sub-class distributions can make it difficult for the GAN to accurately capture the diversity within each class, as it may overfit to more frequent sub-styles or fail to represent rarer ones effectively.

We tried to use contrastive learning with SimCLR (Chen et al. [2020]), and then perform clustering via k-NN, but we did not manage to obtain satisfactory results at this time.

6 Conclusion

We explored the use of Gaussian Mixture Generative Adversarial Networks (GM-GAN) to improve the diversity and quality of generated samples, especially in complex, multi-modal datasets. By modifying the latent space to incorporate Gaussian mixtures, we were able to encourage the generator to learn more distinct modes in the data, effectively addressing challenges like mode collapse and enhancing the representation of real class distributions.

The summary of our results can be found in Table 3. They indicate that out of the four approaches we tried, only the dynamic supervised GM GAN outperformed our baseline model in FID, precision and recall.

Table 3: Summary of results. All results come from the LAB evaluation platform except for the static GM GAN.

	FID	**Precision**	**Recall**
Baseline	27.73	0.52	0.2
Static GM GAN	251.59	-	-
Static GM WGAN	110.47	-	-
EM Static GM GAN	39.58	0.5	0.11
Dynamic Supervised GM GAN	15.67	0.55	0.29

While the other approaches did not outperform the baseline model, for the static GAN as well as for the EM static GAN, our experiments showed significant improvements when we tuned the hyper-parameters, which opens the possibility that they might become more efficient with a lot more of tuning.

As can be seen in our experiments section, the reason why the supervised dynamic GM GAN was so successful, compared to the other approaches, is likely due to the fact that, since the parameters of the Gaussian mixture are learned through the neural network, it requires relatively low hyper-parameter tuning, compared to the static variants. In a context where each run takes 45 minutes, this difference is especially relevant because tuning over a grid of three hyper-parameters might be impossible in a short amount of time.

As can be seen in our interpretative section, the reason why the supervised dynamic GM GAN outperforms the baseline model is due to its different loss, that uses class supervision and to the fact that it effectively models the multi-modal distribution of the latent space.

We can summarize these observations and say that the success of the dynamic GM GAN lies in three key aspects:

- A different loss that allows for supervision with the classes of images.
- An ability to model complex, multi-modal distributions.
- A low need for hyper-parameter tuning since the parameters are learned through the network.

Additionally, we discussed possible future directions, including the implementation of self-supervised sub-clustering to capture even finer intra-class variations, though this approach presents new challenges, such as unbalanced sub-class distributions and training instability.

Overall, the GM-GAN framework offers a robust pathway for enhancing GAN performance on diverse datasets, providing a foundation for future work on capturing more complex data structures and intra-class subtleties.

References

Matan Ben-Yosef and Daphna Weinshall. Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images. *CoRR*, abs/1808.10356, 2018. URL http://arxiv.org/abs/1808.10356.

C. Bishop. Pattern recognition and machine learning. *Springer*, pages 423–455, 2006.

C. Bishop and H. Bishop. Deep learning: Foundations and concepts. *Springer Nature*, pages 70–85, 2023.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 1597–1607. PMLR, 2020.

Tristan Milne and Adrian I. Nachman. Wasserstein gans with gradient penalty compute congested transport. *CoRR*, abs/2109.00528, 2021. URL https://arxiv.org/abs/2109.00528.