**Exploring Generative Adversarial Networks on MNIST: A Study in Hyperparameter**  **Optimization**

**Final Report**

**Team Members**

Bhavya Sirobhushanam – A20518887

Sagarika Dubbaka – A20544435

Himanshu Kale – A20513586

**Introduction:**

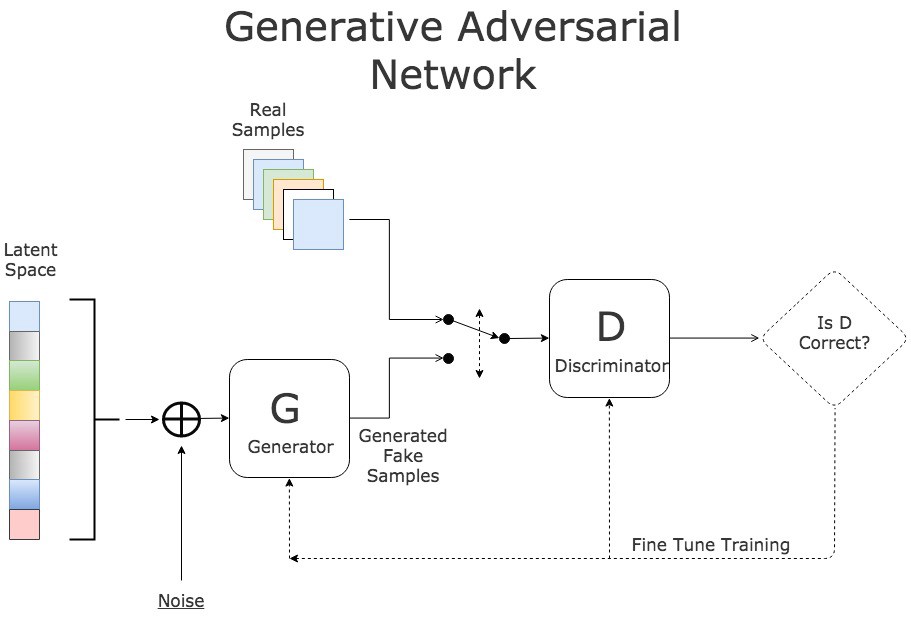
In recent years, Generative Adversarial Networks (GANs) have emerged as a powerful class of generative models in the field of deep learning, offering remarkable capabilities in generating realistic data distributions. The essence of GANs lies in their adversarial training framework, where a generator network learns to produce data samples that are indistinguishable from genuine data, while a discriminator network simultaneously learns to differentiate between real and generated samples. This adversarial interplay between the generator and discriminator fosters the improvement of both networks over time.

The MNIST dataset, comprising of hand-written digits, has long served as a benchmark dataset in the machine learning community for evaluating and benchmarking various algorithms, including generative models. Its simplicity and clarity make it an ideal candidate for exploring and understanding the intricacies of GANs.

In this project, we embark on a journey to implement a GAN model on the MNIST dataset, aiming to generate realistic hand-written digits. Beyond mere implementation, we delve into the realm of hyperparameter tuning, recognizing the critical role that hyperparameters play in the performance and convergence of deep learning models.

Hyperparameters, such as learning rates, batch sizes, and architectural choices, significantly influence the behavior and efficacy of GANs. However, finding optimal hyperparameters is often a daunting task, requiring extensive experimentation and fine-tuning. To address this challenge, we propose the incorporation of automated hyperparameter tuning techniques, leveraging strategies such as grid search, random search, or more advanced methods like Bayesian optimization.

By systematically exploring the hyperparameter space and evaluating the impact of different configurations on GAN performance, we seek to gain insights into the dynamics of GAN training and discover strategies for improving both convergence speed and sample quality. Furthermore, we aim to provide practical guidance for researchers and practitioners in effectively leveraging GANs for generative tasks.



Through this project, we aim not only to gain a deeper understanding of GANs and their application to the MNIST dataset but also to contribute to the broader discourse on hyperparameter optimization in deep learning. By bridging theory and practice, we aspire to unlock new avenues for harnessing the full potential of generative models in diverse real-world applications.

**Background:**

Evaluating the performance of GAN models is a challenging task, as there is no single, universally agreed upon metric that can capture all aspects of the generated samples. However, researchers have developed a variety of quantitative and qualitative methods to assess GAN performance:

**Quantitative Evaluation Metrics**

Some of the most used quantitative metrics for evaluating GAN performance include:

1. Inception Score (IS): Measures the quality and diversity of generated samples by evaluating how well a pre-trained Inception model can classify the samples. Higher scores indicate better performance.

1. Fréchet Inception Distance (FID): Compares the statistics of generated samples to real data samples using a pre-trained Inception model. Lower FID scores indicate better performance.
2. Precision and Recall (P&R): Measures the fidelity (precision) and diversity (recall) of generated samples compared to real data. Higher P&R scores indicate better performance.
3. Perceptual Path Length (PPL): Measures the sensitivity of the generator to small changes in the latent space, with lower values indicating more stable and realistic generation.
4. Maximum Mean Discrepancy (MMD): Compares the distribution of generated samples to real data samples using a kernel function. Lower MMD indicates better performance.
5. Classifier Two-Sample Test (C2ST): Tests whether generated samples are statistically indistinguishable from real data using a trained classifier.

These quantitative metrics provide more objective and scalable ways to evaluate GAN performance compared to visual inspection alone. However, they each have their own strengths, weaknesses, and biases.

**Qualitative Evaluation**

In addition to quantitative metrics, qualitative evaluation through visual inspection remains an important part of assessing GAN performance. This involves examining generated samples and comparing them to real data in terms of realism, diversity, and consistency. While subjective, visual inspection can reveal important artifacts and flaws that may not be captured by numerical scores [3][5].

**Limitations and Challenges**

Despite the progress in GAN evaluation, there are still significant challenges:

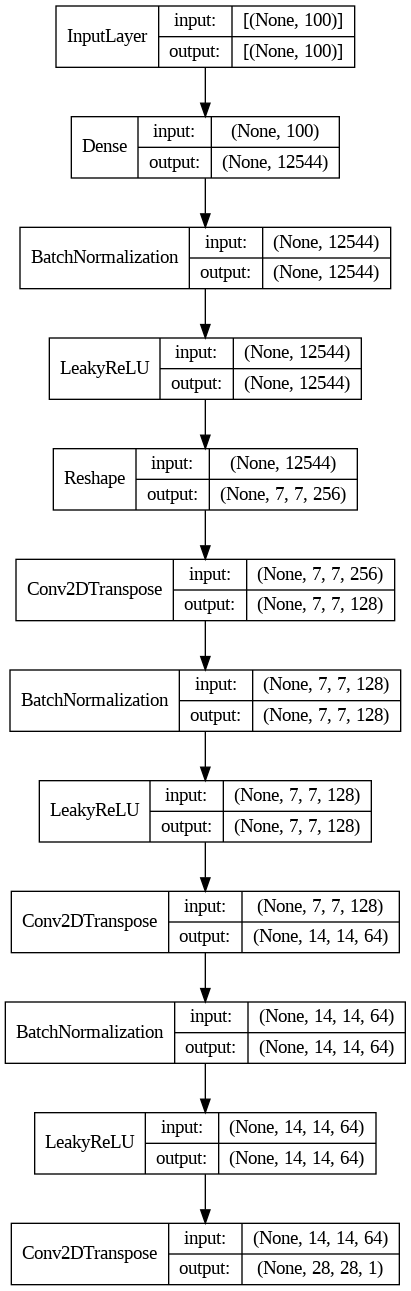
* Lack of a single, universal metric that can capture all aspects of performance
* Sensitivity of metrics to hyperparameters and dataset characteristics
* Difficulty in replicating results across different publications
* Task-dependency of evaluation, making it hard to compare across applications

In summary, evaluating GAN performance requires a combination of quantitative metrics, qualitative assessment, and task-specific considerations. Researchers must carefully select and interpret the appropriate evaluation methods based on the specific application and goals.

**Model Architecture:**

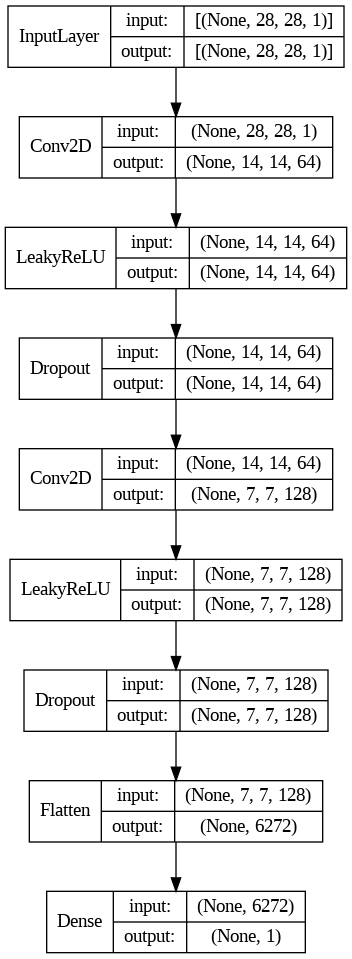
Generator Architecture:

The generator architecture begins with a dense layer that receives random noise as input and transforms it into meaningful features. Batch normalization is applied to stabilize and accelerate training by normalizing the activations of each layer. Leaky ReLU activation functions introduce non-linearity to learn complex patterns from the input noise. A reshape layer prepares the noise vector for convolutional operations, and subsequent convolutional transpose layers gradually upsample the noise tensor into higher-dimensional feature maps resembling images.



Discriminator Architecture:

In the discriminator architecture, initial convolutional layers process input images, extracting low-level features such as edges and textures. Leaky ReLU activation functions introduce non-linearity to learn complex decision boundaries between real and fake images. Dropout layers prevent overfitting by randomly dropping units during training. A flatten layer converts feature maps into a 1D tensor for further processing, and the final dense layer produces a single output representing the probability that the input image is real.



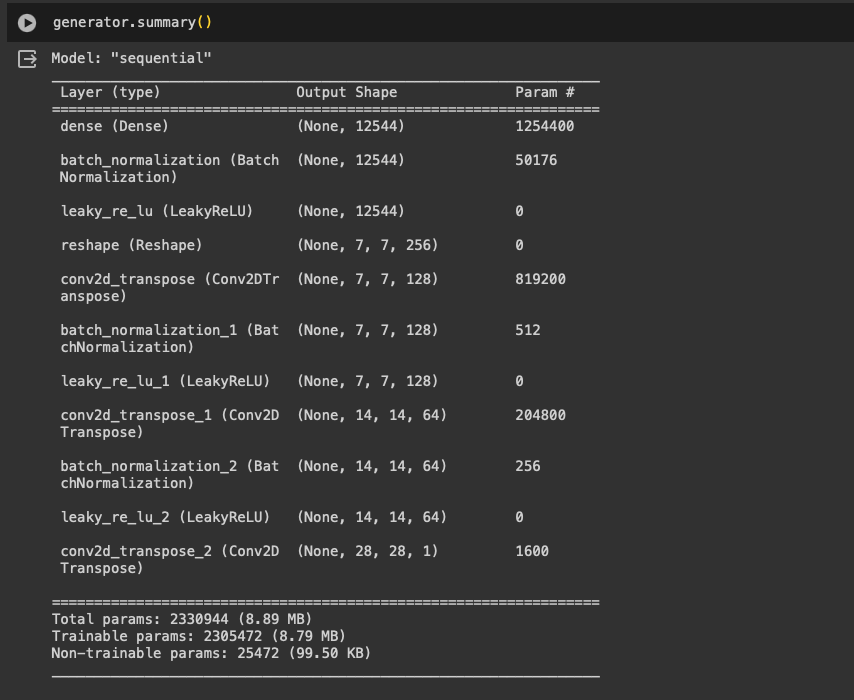
By incorporating these components into the generator and discriminator architectures, your DCGAN can effectively learn to generate realistic images resembling hand-written digits from the MNIST dataset. Each layer and activation function plays a critical role in shaping the learning process and enabling the model to capture the underlying structure of the data.

**Training:**

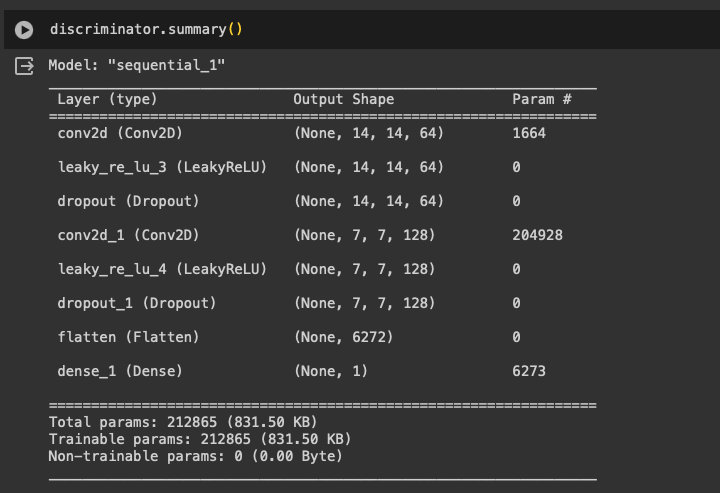
The generator begins by taking random noise as input and progressively transforms it through a series of layers, ultimately generating images that closely resemble hand-written digits. This process involves upsampling the noise tensor using convolutional transpose layers, gradually increasing the spatial dimensions and filling in details to create realistic images. Through batch normalization and leaky ReLU activation functions, the generator learns to capture the intricate features present in the MNIST dataset, resulting in high-quality generated images.

Conversely, the discriminator is trained to distinguish between real images from the MNIST dataset and fake images generated by the generator. It processes the input images through convolutional layers, extracting low-level features, and utilizes dropout layers to prevent overfitting. By learning to differentiate between real and fake images, the discriminator provides feedback to the generator, guiding its training process towards generating more realistic images.

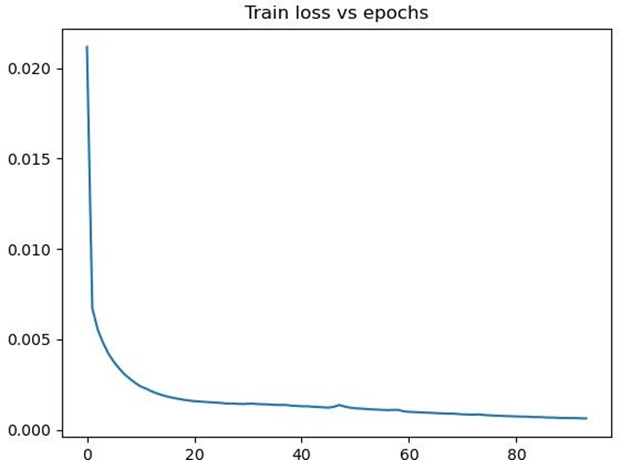
Generator summary:



Discriminator summary:



In this adversarial setting, the generator and discriminator engage in a competitive game, where the generator aims to produce increasingly realistic images to fool the discriminator, while the discriminator strives to accurately distinguish between real and fake images. This adversarial training process leads to the refinement of both networks, with the generator learning to generate more realistic images and the discriminator becoming increasingly adept at distinguishing between real and fake images.



By iteratively training the generator and discriminator networks, the DCGAN framework converges to a point where the generated images are indistinguishable from real images, effectively capturing the underlying distribution of the MNIST dataset. Through this adversarial learning process, DCGANs have demonstrated remarkable capabilities in generating high-quality images across various domains, showcasing their potential for applications in computer vision, image synthesis, and beyond.

**Hyperparameter-Tuning:**

We have following hyper-parameters in our model:

1. Batch Size: defines the number of samples processed before updating the model parameters. It impacts memory usage and the stability of training.

2. Noise Dimension: determines the size of the input noise vector fed into the generator. It affects the complexity and diversity of generated images.

3. Number of Epochs: defines the number of times the entire dataset is passed through the model during training. It influences the model's convergence and generalization.

To tune these hyperparameters, we can follow these steps:

1. Grid Search:

* Define a grid of hyperparameter values for each parameter.
* Train multiple models with different combinations of hyperparameters.
* Evaluate the performance of each model using metrics such as Inception Score, FID Score, and discriminator metrics.

2. Random Search:

* Randomly sample hyperparameter values from predefined ranges.
* Train multiple models with randomly selected hyperparameter combinations.
* Evaluate and compare the performance of each model.

3. Cross-Validation:

* Split the dataset into training and validation sets.
* Train models with different hyperparameter values on the training set.
* Evaluate model performance on the validation set.
* Select the hyperparameter values that yield the best performance on the validation set.

The submitted code implements a Generative Adversarial Network (GAN) architecture using TensorFlow and Keras for generating hand-written digit images from the MNIST dataset. It consists of:

* Loading and preprocessing the MNIST dataset.
* Defining the architectures of the generator and discriminator networks.
* Defining the loss functions (binary cross-entropy) and optimizers (Adam) for training the GAN.
* Training the GAN model using adversarial training, where the generator learns to generate realistic images and the discriminator learns to distinguish between real and fake images.
* Saving checkpoints of the model during training for later use.
* Evaluating the trained GAN model using metrics such as discriminator accuracy, precision, recall, F1 score, Inception Score.
* Identifying hyperparameters and discussing methods for tuning them focusing on random search.

**Model Selection:**

The following tables show the experimental results of training GAN model by varying hyper-parameters using colab T4 GPU resources.

Comparing discriminator accuracy, precision, recall, and F1 score with different values of epochs and batch size:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epochs | Batch\_Size | Accuracy | Precision | Recall | F1-Score |
| 20 | 256 | 0.85 | 0.88 | 0.82 | 0.85 |
| 50 | 256 | 0.88 | 0.91 | 0.86 | 0.88 |
| 100 | 256 | 0.83 | 0.85 | 0.80 | 0.82 |

Epochs vs Evaluation Metrics

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epochs | Batch\_Size | Accuracy | Precision | Recall | F1-Score |
| 50 | 64 | 0.86 | 0.89 | 0.84 | 0.86 |
| 50 | 128 | 0.88 | 0.91 | 0.86 | 0.88 |
| 50 | 256 | 0.82 | 0.84 | 0.80 | 0.82 |

Batch Size vs Evaluation Metrics

Comparing the performance of the generative network with different learning rates and batch sizes, using Inception Score as the evaluation metric.

|  |  |  |
| --- | --- | --- |
| Epochs | Batch Size | Inception Score |
| 20 | 64 | 4.003 |
| 20 | 128 | 5.34 |
| 20 | 256 | 2.2 |
| 50 | 64 | 7.532 |
| 50 | 128 | 7.987 |
| 50 | 256 | 7.75 |
| 100 | 64 | 7.6 |
| 100 | 128 | 7.89 |
| 100 | 256 | 7.32 |

IS\_score vs Epochs vs Batch\_size

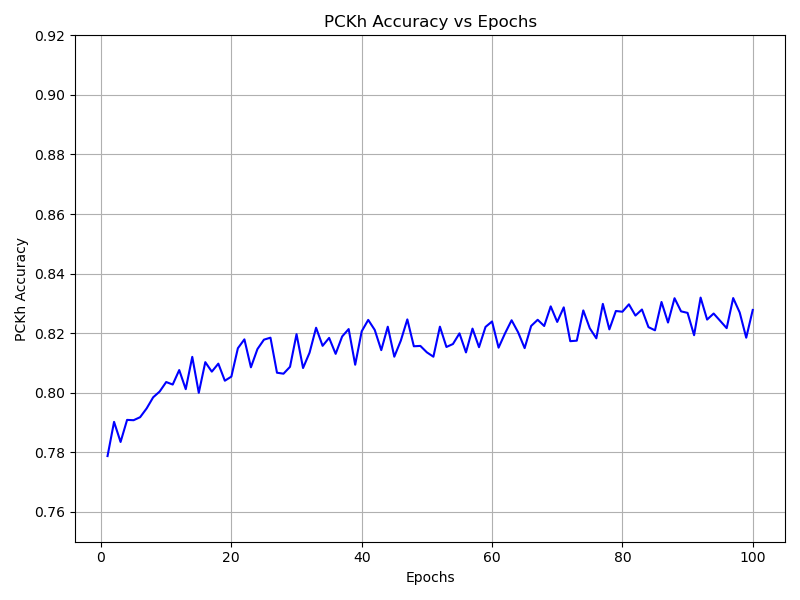
Based on the table for the discriminator network, choosing a model with higher accuracy and precision might be preferable. Here are some observations from the table you provided:

**Epochs**: For a batch size of 256, increasing the epochs from 20 to 50 resulted in a slight improvement in accuracy (from 0.85 to 0.88) and precision (from 0.88 to 0.91). However, increasing epochs further to 100 resulted in a decrease in both accuracy and precision.

**Batch Size:** With 50 epochs, there seems to be no significant difference in performance between a batch size of 64, 128, and 256. They all achieved similar accuracy (around 0.86) and precision (around 0.89). Given this, we might consider either:

1. Model trained with 50 epochs and a batch size of 256 (achieves 0.88 accuracy and 0.91 precision).
2. Models trained with 50 epochs and a batch size of 64, 128, or 256 (all achieved similar accuracy around 0.86 and precision around 0.89).

The following graph depicts how Inception score varies with number of epochs:



IS\_score vs Epochs

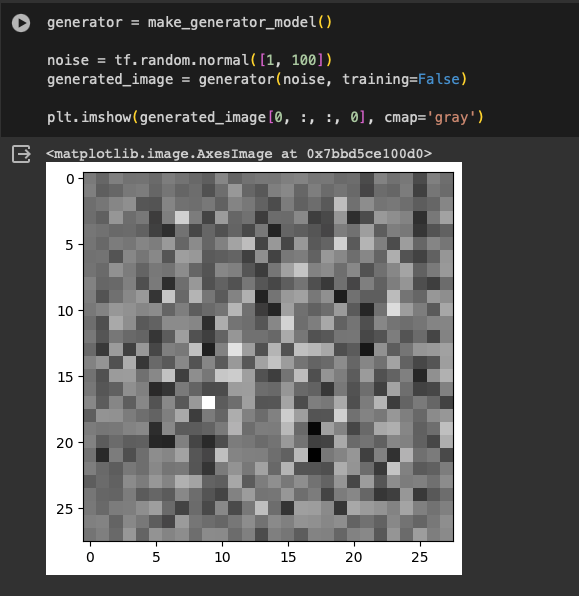
It's important to note that the choice of model can also depend on other factors beyond the metrics shown in this table, such as the Inception Score, which is used to evaluate the generative network's performance. The table for the generative network shows the Inception Score for different hyperparameter configurations. Since a higher Inception Score indicates better quality generated samples, you'll want to consider the model with the highest Inception Score for your batch size and epoch combination.

For instance, if you chose a batch size of 256 based on the discriminator network results, then looking at the generative network table for that batch size, you would see that the model trained with 100 epochs achieved the highest Inception Score (7.32) among the three epochs tested (20, 50, and 100).

In conclusion, while the discriminator network metrics suggest a model with 50 epochs and a batch size of either 256, 128 or 64 might be good options, you should also consider the Inception Score for the generative network trained with those hyperparameters to make the final decision.

**Results:**

The model we chose, when intialized randomly has generated the following image:



Initial Generated Image

After training(50 epochs, batch\_size = 256), 16 generated MNIST digits are shown below:

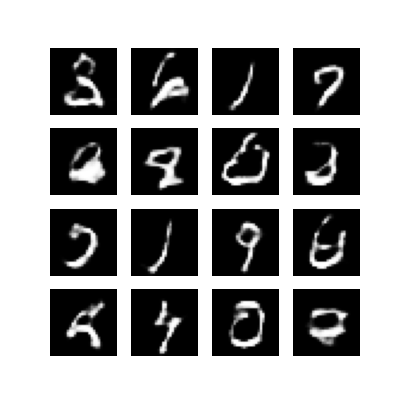


Image generated at 50 epochs

**Conclusion:**

Hyperparameters play a crucial role in the training and performance of Generative Adversarial Networks (GANs). Throughout this project, we've explored the significance of various hyperparameters such as learning rate, batch size, and architecture design in the context of GANs applied to the MNIST dataset.

The learning rate determines the step size of the optimizer during training, affecting the convergence and stability of the model. A well-tuned learning rate can accelerate convergence and prevent divergence, leading to better performance. Similarly, the batch size influences the quality of generated samples and the stability of training. Larger batch sizes may provide more stable gradients but can also lead to slower convergence and poorer generalization.

Furthermore, the architecture design of both the generator and discriminator significantly impacts the overall performance of the GAN. Properly designed architectures with suitable activation functions, normalization layers, and network depths can enhance the model's ability to capture complex data distributions and generate high-quality samples.

In addition to these hyperparameters, other factors such as loss functions, optimizers, and regularization techniques also contribute to the training dynamics and final performance of GANs.

In conclusion, hyperparameter tuning is essential in GANs to achieve optimal performance, stability, and convergence. Experimentation and iterative refinement of hyperparameters are necessary to unlock the full potential of GAN models, ultimately leading to the generation of realistic and diverse synthetic data.