DIGITAL IMAGE PROCESSING
ELL715
PROJECT REPORT

15 November 2023

Group Members

1. Bhavik Shankla (2020MT60873)
2. Ritika Soni (2020MT10838)
3. Sai Kiran Gunnala (2020MT60889)
4. Sai Niketh Varanasi (2020MT60895)
5. Yash Pravin Shirke (2020MT60986)

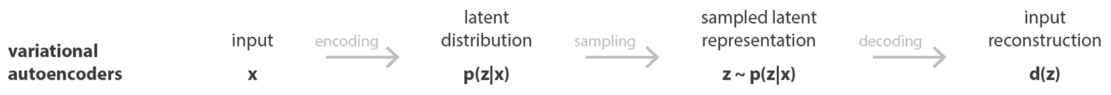
1 Introduction

One key distinction within the field of machine learning lies in the choice between generative and discriminative modeling. While in discriminative modeling one aims to learn a predictor given the observations, in generative modeling one aims to solve the more general problem of learning a joint distribution over all the variables.A generative model tries to generate data using different probability distributions and finding patterns.In the realm of artificial intelligence and computer vision, image generation has become a captivating field with the advent of generative models.For this project, we explore image generation using different generative models like Variational Autoencoders and Generative Adversarial Networks.The fundamental concept behind these generative models is to learn patterns and structures from existing data, enabling them to generate new, often indistinguishable, images.

2 Theory

2.1 Variational Autoencoders

The general idea of autoencoders consists of having an encoder and a decoder as neural networks and to learn the best encoding-decoding scheme using an iterative optimisation process. After each iteration, some input is fed into the model which when compared to the results required is used for the backpropagation of the error through the neural networks.Ensuring the regularity of the latent space is crucial to allow the generative usage of our autoencoder's decoder. Explicit regularization throughout the training phase is one method of achieving this regularity. In order to avoid overfitting and provide favorable qualities in the latent space for generative processes, variational autoencoders are defined as autoencoders with regularization in their training phase. The encoder and decoder in this design are trained to reduce the reconstruction error between the original data and the encoded-decoded output.



2.2 Generative Adversarial Networks

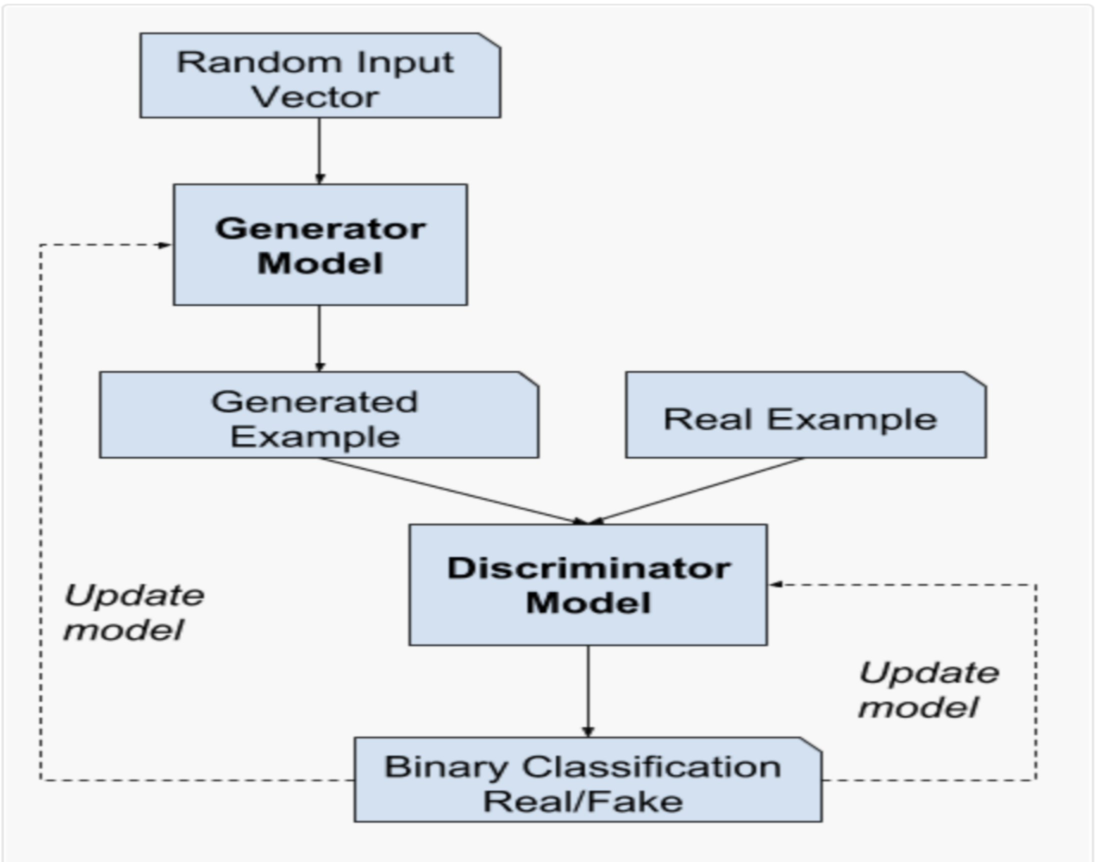
Generative Adversarial Networks (GANs) serve as a fundamental framework for the estimation of generative models through an adversarial procedure. This procedure involves the simultaneous training of two models: a generative model, which aims to capture the distribution of the data, and a discriminative model, which seeks to estimate the likelihood that a given sample originates from the training data.

The generator model is trained with the objective of producing generated images, while the discriminator model is tasked with distinguishing between real instances (drawn from the domain) and fake instances (produced by the generator). The two models are jointly trained in a zero-sum game, characterised by adversarial dynamics, until the discriminator model achieves a 50% error rate, indicating that the generator model is successfully producing credible instances.
Discriminator and Generator play the following two-player game with value function V (G, D):

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

The Generator: The generator model takes a fixed-length random vector as input and generates a sample in the domain. For the generation process, we need a seed and thus it is sampled from a gaussian distribution. This model takes a random vector as input and used to generate a sample in the domain. After training, points in this multidimensional vector space will correspond to points in the problem domain, forming a compressed representation of the data distribution which here is images.

The Discriminator: The discriminator which is already trained for the classification task takes the images produced by the generator model as input and performs a binary classification of real or fake(generated).



Example of the Generative Adversarial Network Model Architecture

3 Methodology

3.1 Variational Autoencoders

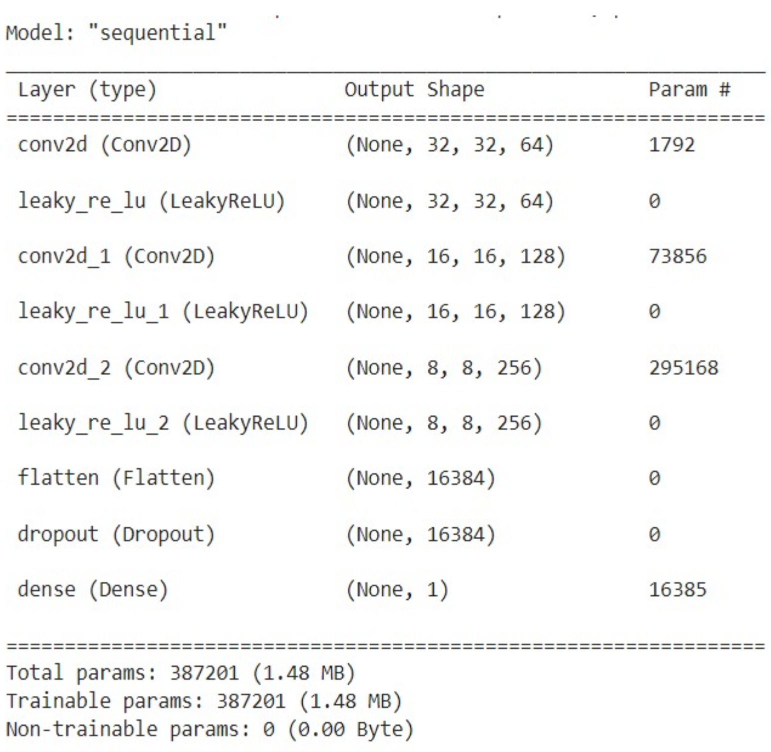
Variational Autoencoders consist of an encoder layer which compresses an image or vector to encoded space also known as latent space, then we use the decoder which takes this compressed image in the latent space as input and tries to recreate the original space of the same dimensions.

We implemented the VAE using torch library in python and trained it for 120 epochs on MNIST dataset for image generation. For image generations we first pass an image from our dataset into the encoder and then add some noise to the encoded image in the latent space, then after decoding this image we get a newly generated image.

We implemented the encoder layer using three fully connected layers with relu activation function After encoding we add noise to the image in the compressed state and hook the new compressed image to the decoder. We then use this to generate new image. The decoder layer consists of three fully connected layers also with Relu activation function. The forward function which was used for training consists of all six fully connected layers with relu activation function and the training is done with batch size = 128 and Mean Absolute error as our loss function.

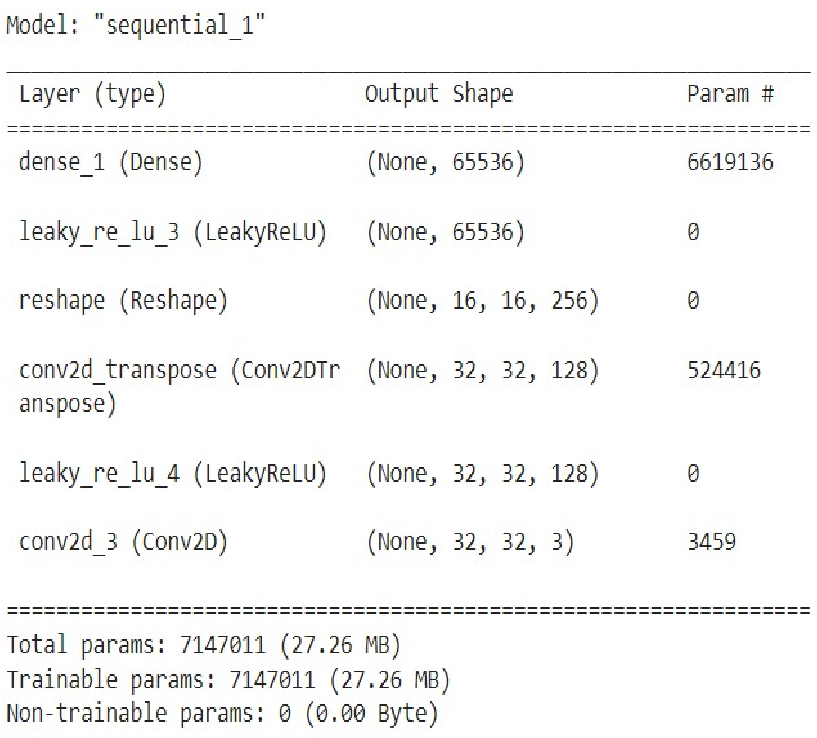
3.2 Deep Convolutional Generative Adversarial Network

In this study, a generative adversarial network (GAN) model was implemented using the Keras library in Python. The generator architecture consists of a fully connected layer followed by two convolutional neural network (CNN) layers, employing Leaky ReLU activation for the first and tanh activation for the second. The discriminator, on the other hand, is a sequential neural network with three CNN layers featuring Leaky ReLU activations, followed by a fully connected layer with a sigmoid activation function. The training process utilizes a binary cross-entropy loss function, with the Adam optimizer and a specified learning rate.



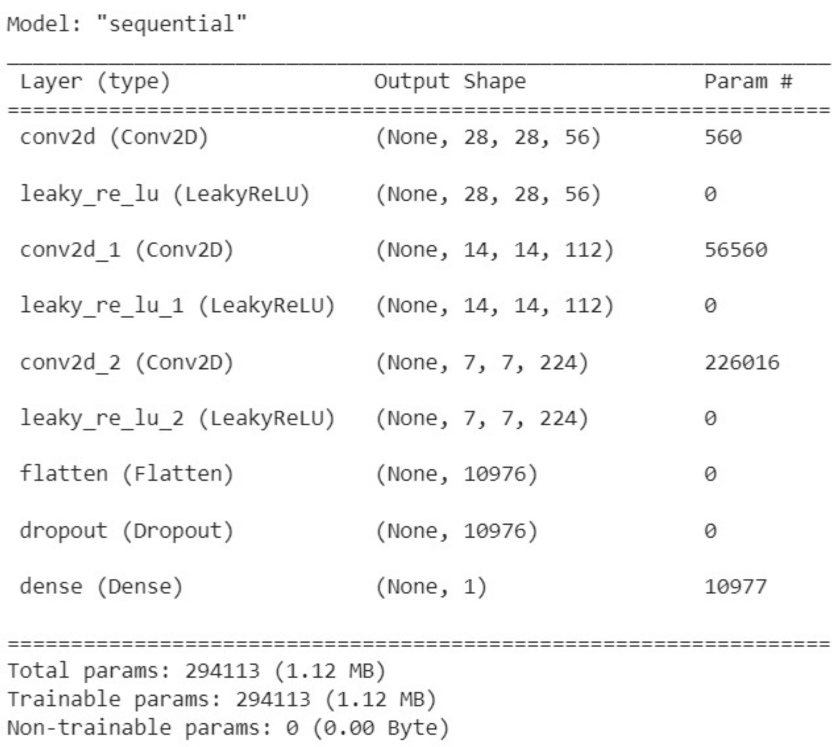
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        1792

 leaky_re_lu (LeakyReLU)     (None, 32, 32, 64)        0

 conv2d_1 (Conv2D)           (None, 16, 16, 128)       73856

 leaky_re_lu_1 (LeakyReLU)   (None, 16, 16, 128)       0

 conv2d_2 (Conv2D)           (None, 8, 8, 256)         295168

 leaky_re_lu_2 (LeakyReLU)   (None, 8, 8, 256)         0

 flatten (Flatten)           (None, 16384)             0

 dropout (Dropout)           (None, 16384)             0

 dense (Dense)               (None, 1)                 16385

=================================================================
Total params: 387201 (1.48 MB)
Trainable params: 387201 (1.48 MB)
Non-trainable params: 0 (0.00 Byte)
```

Discriminator CIFAR



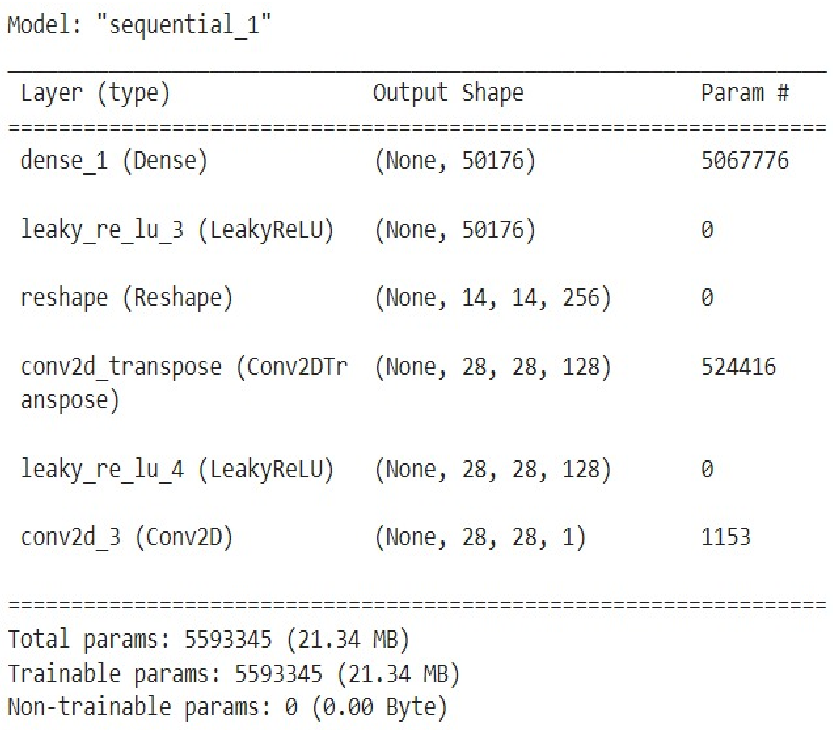
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 65536)             6619136

 leaky_re_lu_3 (LeakyReLU)   (None, 65536)             0

 reshape (Reshape)           (None, 16, 16, 256)       0

 conv2d_transpose (Conv2DTr  (None, 32, 32, 128)       524416
 anspose)

 leaky_re_lu_4 (LeakyReLU)   (None, 32, 32, 128)       0

 conv2d_3 (Conv2D)           (None, 32, 32, 3)         3459

=================================================================
Total params: 7147011 (27.26 MB)
Trainable params: 7147011 (27.26 MB)
Non-trainable params: 0 (0.00 Byte)
```

Generator CIFAR



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 56)        560

 leaky_re_lu (LeakyReLU)     (None, 28, 28, 56)        0

 conv2d_1 (Conv2D)           (None, 14, 14, 112)       56560

 leaky_re_lu_1 (LeakyReLU)   (None, 14, 14, 112)       0

 conv2d_2 (Conv2D)           (None, 7, 7, 224)         226016

 leaky_re_lu_2 (LeakyReLU)   (None, 7, 7, 224)         0

 flatten (Flatten)           (None, 10976)             0

 dropout (Dropout)           (None, 10976)             0

 dense (Dense)               (None, 1)                 10977

=================================================================
Total params: 294113 (1.12 MB)
Trainable params: 294113 (1.12 MB)
Non-trainable params: 0 (0.00 Byte)
```

Discriminator MNIST



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 50176)             5067776

 leaky_re_lu_3 (LeakyReLU)   (None, 50176)             0

 reshape (Reshape)           (None, 14, 14, 256)       0

 conv2d_transpose (Conv2DTr  (None, 28, 28, 128)       524416
 anspose)

 leaky_re_lu_4 (LeakyReLU)   (None, 28, 28, 128)       0

 conv2d_3 (Conv2D)           (None, 28, 28, 1)         1153

=================================================================
Total params: 5593345 (21.34 MB)
Trainable params: 5593345 (21.34 MB)
Non-trainable params: 0 (0.00 Byte)
```

Generator MNIST

4 Discussions

The initial approach involved utilising a Variational Autoencoder (VAE) as the image generation network. The VAE was trained for a total of 5000 epochs using the MNIST dataset. However, the training process proved to be computationally intensive and the generated images exhibited suboptimal quality, characterised by blurriness. This issue was attributed to the objective function employed by the VAE.

Consequently, an alternative approach was adopted, involving the use of a Deep Convolutional Generative Adversarial Network (DCGGAN). The DC GAN model shown promising performance after only 10 epochs when trained on the MNIST dataset. Consequently, we decided to evaluate its capabilities further by employing the more complex CIFAR 10 dataset. During the experimentation with the CIFAR 10 dataset, promising outcomes were observed around epoch 70. However, achieving these results required extensive hyperparameter tuning due to frequent model crashes caused by convergence to local minima.

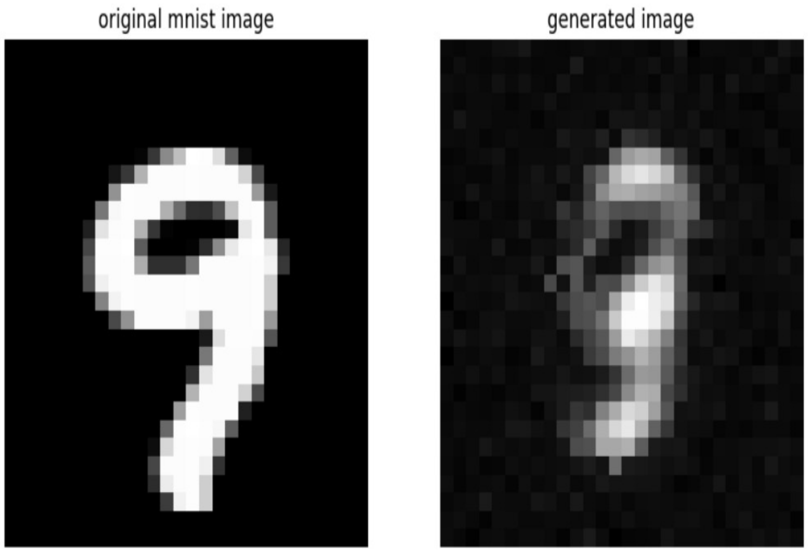
Nevertheless, after careful hyperparameter tuning, the model demonstrated effective recognition of both foreground and background elements, yielding satisfactory results. An additional enhancement that could be considered is the utilisation of the presgan model, which is designed to mitigate the issue of mode collapse.

5 Results and Observations

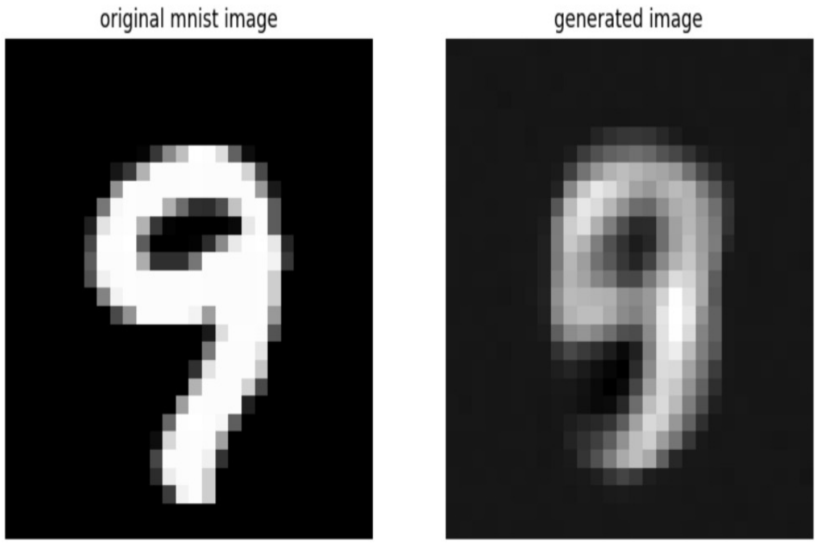
5.1 VAE on MNIST

Right after the first round of training with VAE, the images looked a bit blurry, and it was hard to tell which numbers were in them. This blurriness is something VAEs often have because they try to understand the overall data pattern, even if it means losing some sharpness in individual pictures.

As the VAE kept training and got to the 10,000th epoch, the pictures got clearer, and it became easier to identify the numbers in the generated images.



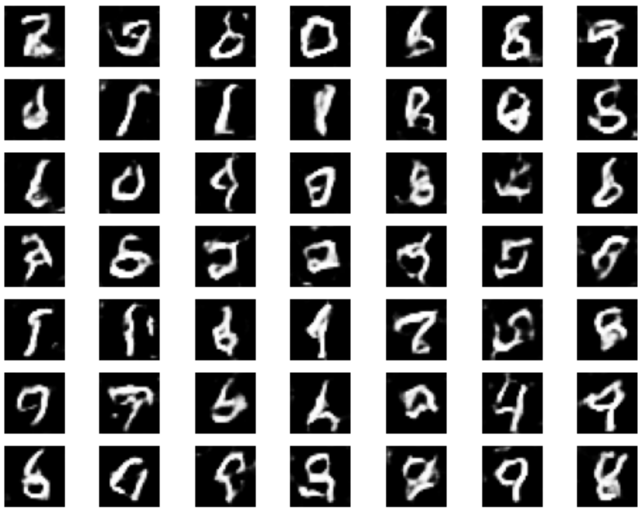
VAE on MNIST after initial epochs



VAE on MNIST after final epochs

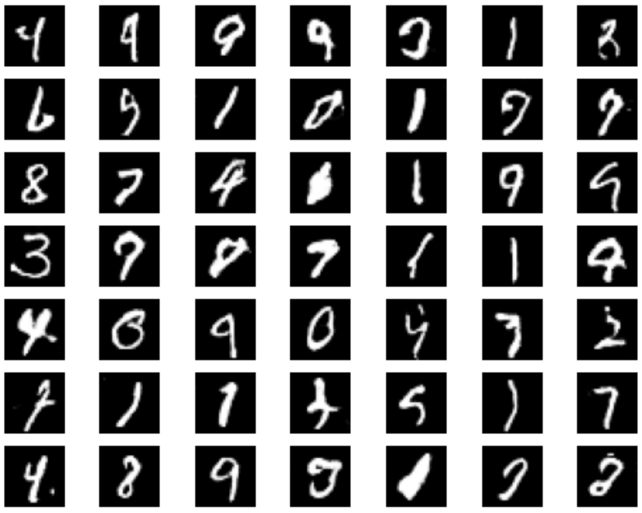
5.2 DCGAN on MNIST

Notable findings were noted about the generated images during the early phases of DC-GAN training on the MNIST dataset. The model started to produce composite images with 49 digits per image with the low visual clarity and distinguishability which made it difficult to identify individual digits in these early images.



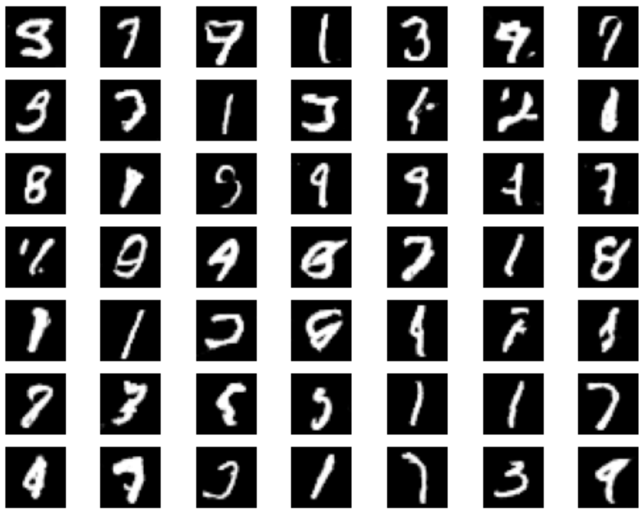
DCGAN on MNIST after 1st epoch

Improvements started to show when the model iterated through 25 epochs of training. Despite the difficulties, some numbers, including 1, 0, 7, and 3, were easier to recognise in the generated images.



DCGAN after MNIST 25th epoch

The images that were produced at 50th epoch exhibited a notable level of clarity, facilitating a more distinct identification of several digits, such as 4, 8, and 9. We can see that the model is generating better images as train for more iterations.

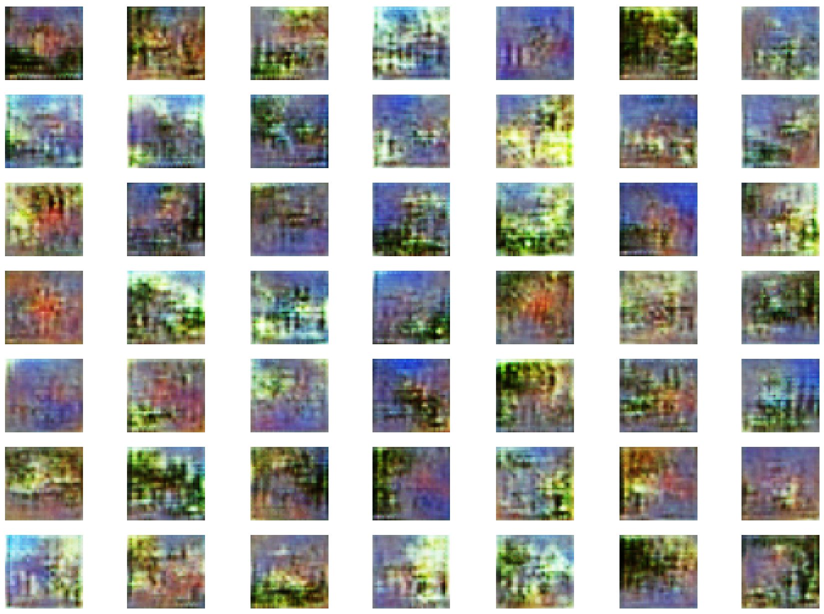

DCGAN after MNIST 50th epoch

5.3 DCGAN on CIFAR-10

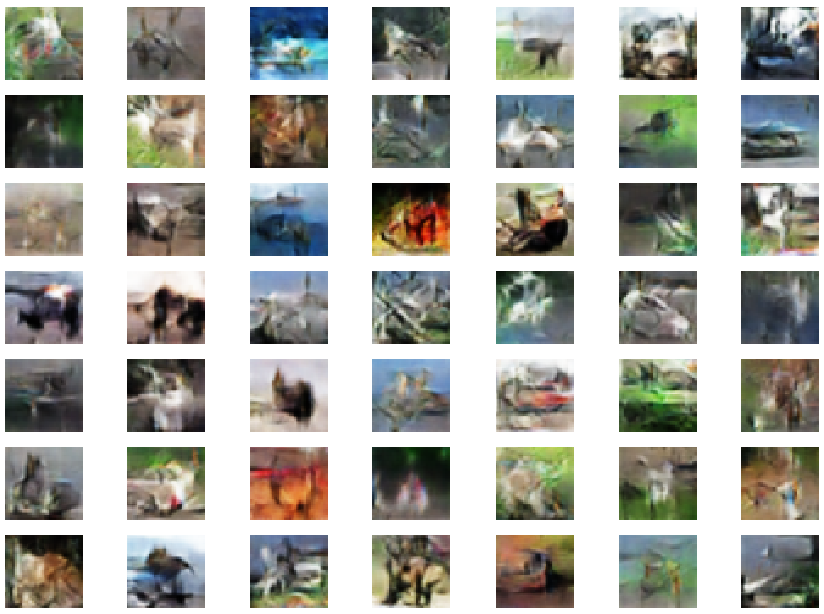
Early on in the model's training, more precisely in the first epoch, the visuals produced in this initial phase were primarily blue in colour, with yellow and orange spots scattered throughout. This output can be explained by the intrinsic properties of the CIFAR-10 dataset, which show that some classes or categories may have a preference for particular colour schemes.

Generated images got better after 25th epoch at showing different features and using more colors. In the initial epochs, we could only see colors, and there were no clear features. Even though we still couldn't tell exactly what objects were in the pictures, they started to show more details.

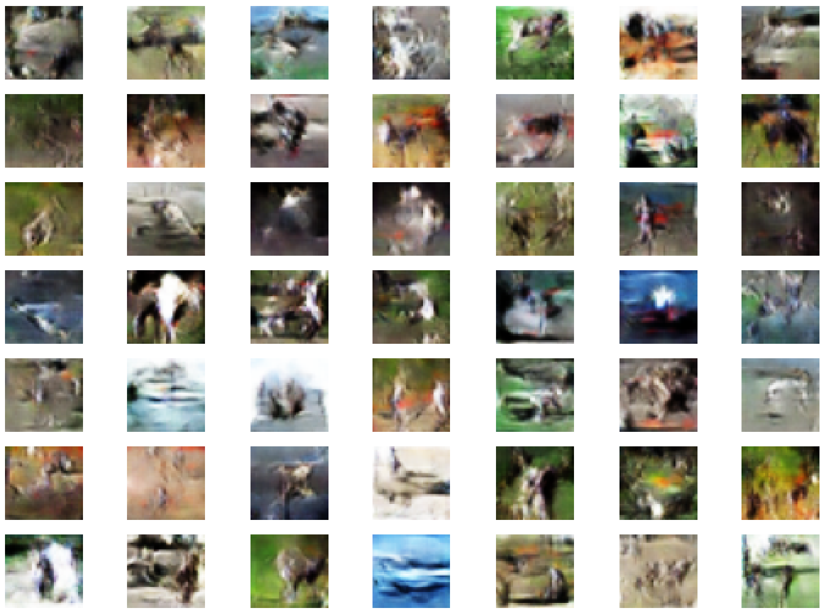
By the time the model reached the 50th epoch, it started producing images with more features. Now, every created image displayed distinct and different colour schemes with a distinction of background and foreground. When images were looked closely, patterns were apparent, exposing animal resemblances.



DCGAN on CIFAR-10 after 1st epoch



DCGAN on CIFAR-10 after 25th epoch



DCGAN on CIFAR-10 after 50th epoch

6 Conclusion

- In case of VAE, we observed that the images produced were blurry. The reason being that VAE prioritises learning the distributions over ensuring the clear and sharp images. We have used a simple linear model for VAE and there was a need to use some other model for better image generation.
- The addition of extra layers in the DCGAN resulted in network collapse due to its entrapment in a local minimum.
- When we added more layers in DCGAN it lead to collapse of network because it got stuck on a local minima. To avoid this we need can add dropout or use more advanced models like PresGAN and adjust the LR optimizer parameters.

7 Github

This is the github link for our project : $https : //github.com/Niketh09/ELL715\_Project$

8 References

- https://arxiv.org/pdf/2006.05218v2.pdf
- https://arxiv.org/pdf/1910.04302v1.pdf
- https://github.com/adjidieng/PresGANs
- https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/
- https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939