# A Comparative Study of GAN and VAE Architectural Models

Nikhil Malladi
UFID:5835-3253
nmalladi@ufl.edu

## 1 Abstract

GAN(Generative Adversarial Network) and VAE(Variational Autoencoder) fall in the category of Generative Modelling in modern machine learning. Where a Generative Modelling models the distribution based on each class by calculating the joint probability p(X,y) between the inputs and class labels and then use bayes rule to classify p(y/X). This is counterpart to discriminative modelling in which classification is based on probability estimation. Two of the modern day machine learning Generative Models are GAN and VAE. In this project we have extensively studied the working of both these generative models on a MNIST(Modified National Institute of Standards and Technology) database for handwritten digits. The report consists of a high level introduction of what GAN and VAE are, along with their architectures and then we look at details of MNIST database itself. Then we explain the architectures used in this project and output results for GAN and VAE. Then we briefly explain the problems faced while working with the models and their training. Lastly, we conclude by comparing the models with each other.

## 2 Introduction

In this section we talk briefly a high level overview about the two generative models along with their architectures.

**GAN:-**
GAN stands for Generative Adversarial Networks. It was introduced by Ian Goodfellow in 2014. The main goal of GAN's is to generate data in domains like music, images etc. The new generated data adhere to the statistics and characterstics of the training data. GAN models are being heavily used in recent times especially in Science, Fashion, art and Advertising.

GAN's are comprised of two deep networks. The Generator and the Discriminator. As the name suggests the Generator keeps generating fake plausible data pertinent to the training set. The data is then classified by Discriminator to classify either as a fake data or a real data.

Let us now understand each of these deep networks. Firstly, let us study Generator network. The goal of Generator network is to generate new data that fools discriminator to believe it as a real. So, it is clear that generator requires lot of training and closer integration with discriminator model. Generator training includes (1) Receiving fixed length random vectors input (2) Generator network transforms the input to data instance (3) Discriminator classifies the data and fetches the output to Generator (5) Generator loss. These are the steps that happen in Generator network training.

Discriminator training performs the job of classifying real or fake data generated by generator. Just like there is Generator loss in Generator model, there is Discriminator loss in this model. This is when the model identifies a fake data as real or vice versa. It is used in back propagation to adjust the weights in discriminator network.

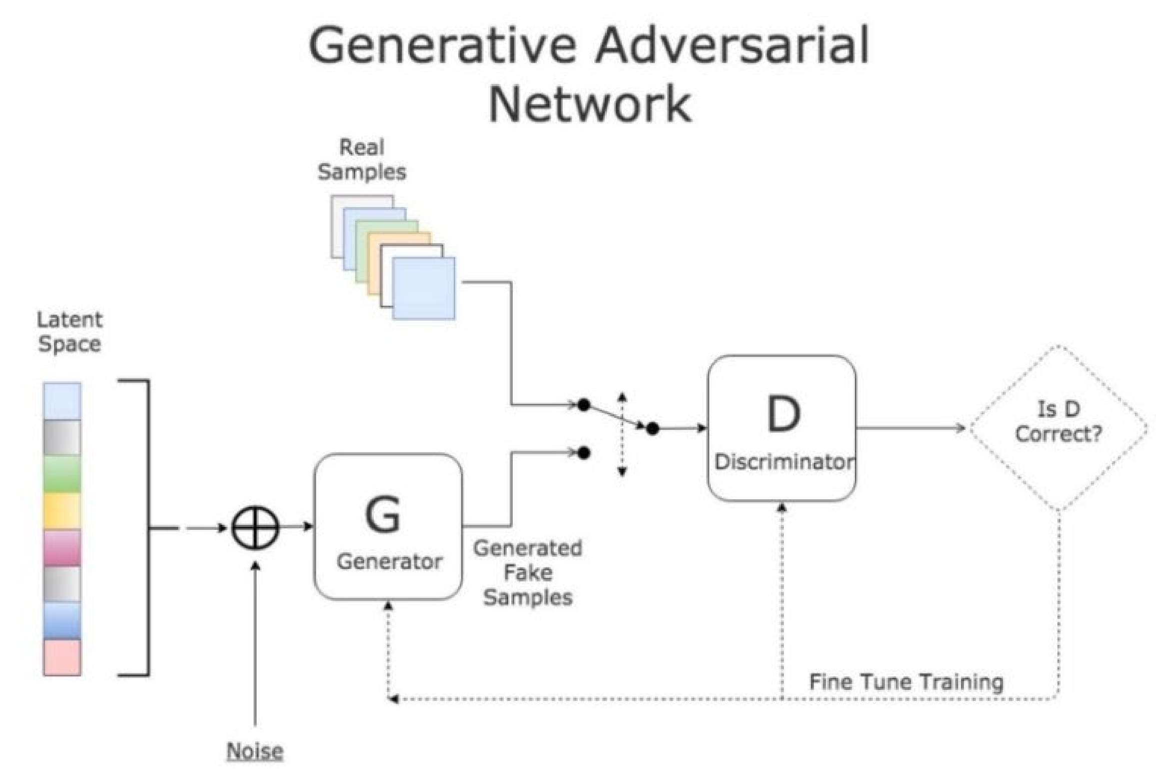


Figure 1: GAN Architectural Diagram showing the Generator Model G and the Discriminator Model D. The Generator model generates fake samples and fetches the output to the Discriminator model D, which then classifies based on the training it received. The result back propagated to Discriminator model D to adjust weights in the network

**VAE:-**
A VAE(Variational Auto Encoder) is an autoencoder whose encodings distribution during the training is regularised to ensure that its latent space has good properties allowing us to generate some new data. It was introduced by The word ”variational” comes from the close relation between regularization and variational inference a concept used in statistics.

VAE consisits of two parts. One is encoder and the other is decoder. VAE takes high dimensional input data and compresses it into smaller data representation. It is called as Dimensionality Reduction which is a term commonly coined in Machine Learning. It refers to reduction in features of describing the input data. This reduction is useful when dealing with low dimensional data. The reduction could by selection or extraction. VAE maps the input data to mean, variance and guassian distribution.

Below is the following procedure of how training is performed on VAE dataset In every iteration, the image is sent to encoder and is mapped to set of mean and log-variance parameters of the approximate posterior q(z/x) The sample q(z/x) is applied with reparametrization trick Lastly the reparameterized samples are passed to the decoder to obtain the logits of the generative distribution p(x/z)

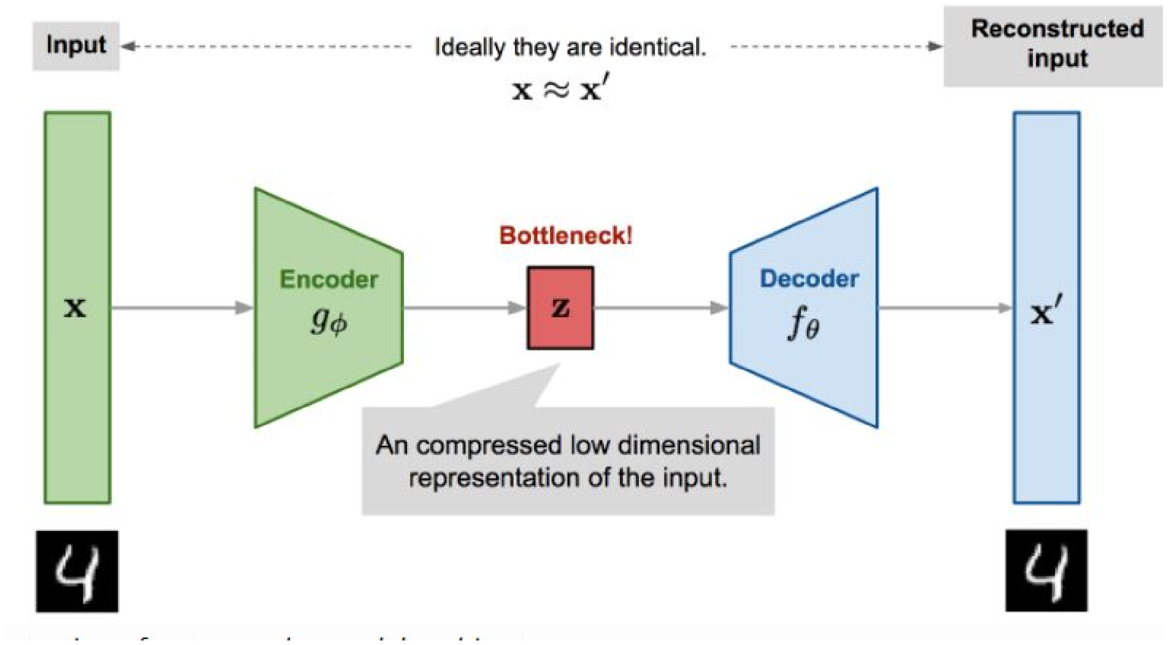

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
$g_\phi$
Bottleneck!
$\mathbf{z}$
Decoder
$f_\theta$
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Figure 2: VAE Architectural Diagram showing the Encoder and the Decoder. We can also see the original input image after going through various layers of VAE has successfully reconstructed an exact matching image. Also the input is low dimensional data(dimensionality reduction)

## 3 Dataset

The database used for this project is MNIST database of handwritten digits. It is procured from the website "http://yann.lecun.com/exdb/mnist/". The training set has 60,000 examples and test set of 10,000 examples. This database has been choosen because it is well suitable for learning techniques by spending minimal efforts on data preprocessing and formatting.

## 4 Architecture of the models

### 4.1 Generative Adversarial Network

Sequential model is used to build the model both Generator model and discriminator model. The generator and discriminator have different optimizers as they are trained separately Adam(1e-4)

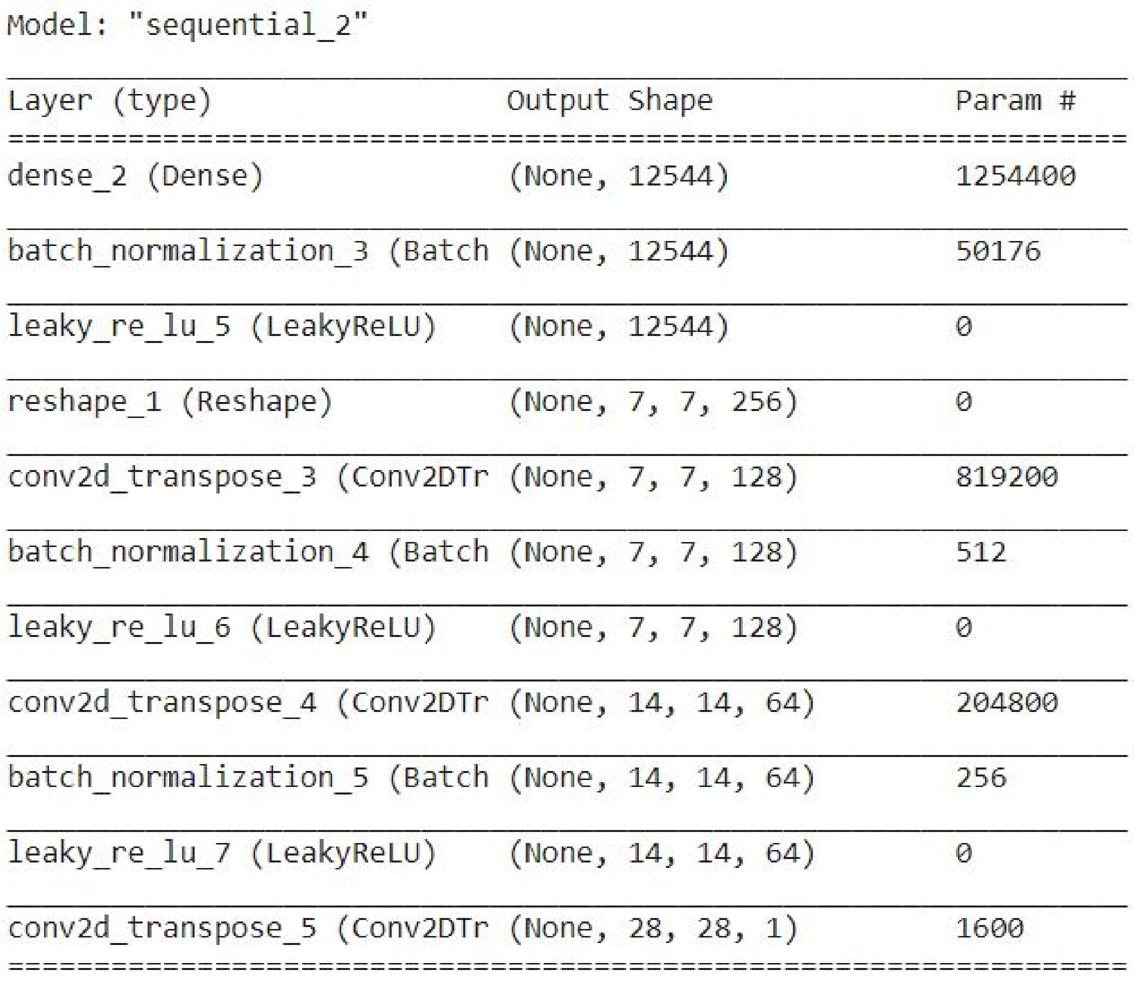
1. ***Generator model***: The number of layers taken here is 4 layers. The model here taken is Sequential. The layers used here are:
   (a) The generator model uses Dense and Conv2DTranspose layers to produce an image.
   (b) The filter size used is 7 * 7 in dense layer and number of nodes is 256
   (c) The Conv2DTransport layer is two layered with 128 nodes and 64nodes.
   (d) LeakyReLU activation is applied on above two layers
   (e) Output is a desired image of 28 * 28 * 1 size pixel output which uses the activation function "tanh"
   (f) Nodes in Dense are fully connected and in Conv2DTranspose are convoluted.
2. ***Discriminator model***: The model taken here is the Sequential model. The generative model's output is taken as an input here. The number of layers used is 3. The layers used are:
   (a) The output from Generator model is passed as an input to discriminator model

(b) The Discriminator model uses Conv2D and Dense layers to produce an image

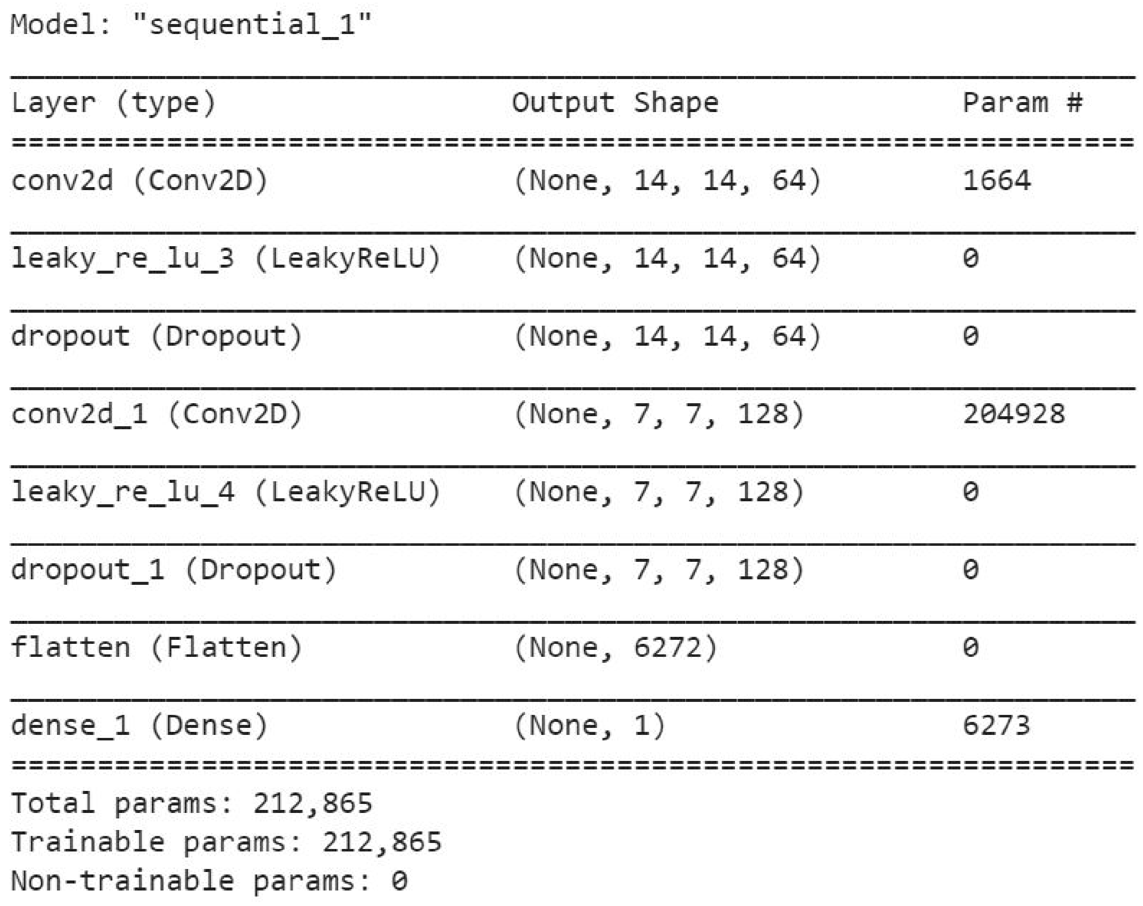
(c) The Conv2D is two layered with 64 and 128 nodes.

(d) LeakyReLU activation is applied just like in Generator model

(e) The model returns positive values for real images and negative values for fake images.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 12544)             1254400
_________________________________________________________________
batch_normalization_3 (Batch (None, 12544)             50176
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 12544)             0
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 7, 7, 128)         819200
_________________________________________________________________
batch_normalization_4 (Batch (None, 7, 7, 128)         512
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 14, 14, 64)        204800
_________________________________________________________________
batch_normalization_5 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 28, 28, 1)         1600
=================================================================
```



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 64)        1664
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 128)         204928
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 6273
=================================================================
Total params: 212,865
Trainable params: 212,865
Non-trainable params: 0
```

### 4.2 Variational Auto-Encoder

Model being built is Sequential

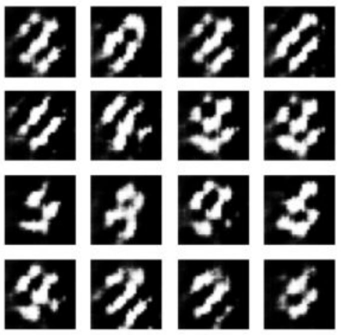
1. ***Encoder***:
   (a) The layers used are 2 convolutional and one dense layer.
   (b) The input layer takes input shape of 28 * 28 * 1
   (c) Two convolutional layers take 32 and 64 nodes with equal number of filters
   (d) There is no activation applied on any of the layers
2. ***Decoder***:
   (a) The layers used in the decoder are dense and three Conv2dTranspose layers.
   (b) The dense filter uses 7 * 7 size
   (c) The Conv2DTranspose uses 64, 32 and 1 size filters
   (d) The activation function used is RELU. Strides taken are two (2,2) and one (1,1)

## 5 Results

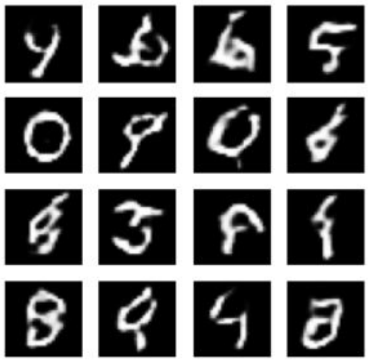
Now, let us discuss the output results we got after running the GAN model and VAE models with above mentioned specifications. The GAN model is run at increasing epoch values until 2000. Similarly VAE model is run upto epoch 1000. Below are the generated images

**GAN:-**

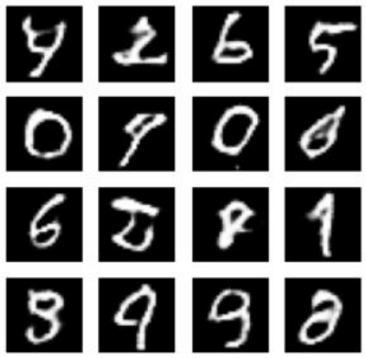
1. **EPOCH NUMBER 05**:



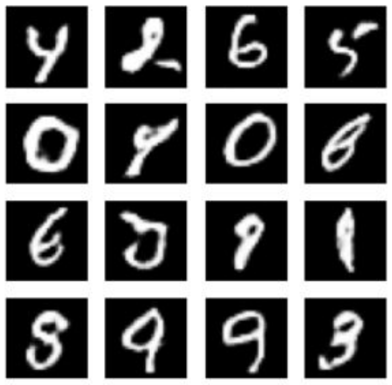
2. **EPOCH NUMBER 50**:



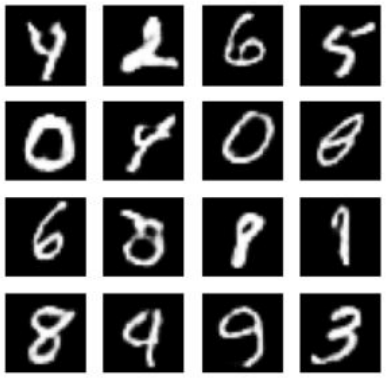
3. **EPOCH NUMBER 100**:



4. **EPOCH NUMBER 250**:



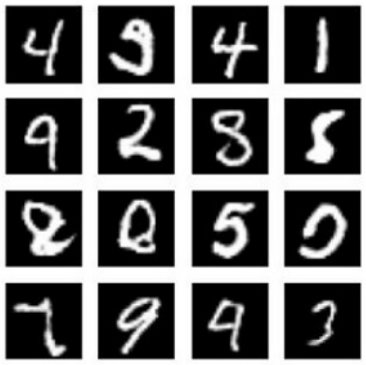
5. **EPOCH NUMBER 500**:



6. **EPOCH NUMBER 1000**:

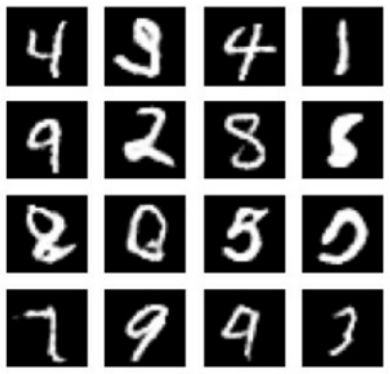
When Ran on Epoch Number 1250 to 2500 the model showed almost no improvement (Please note the values in cells are different to above cells. The code was re-ran)

7. **EPOCH NUMBER 1250**:

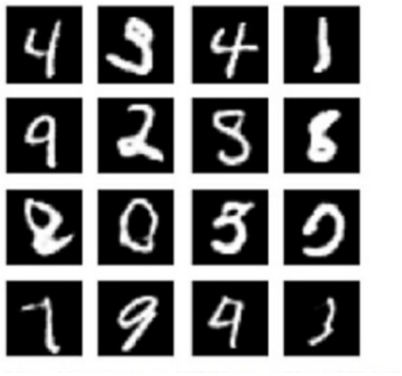


8. **EPOCH NUMBER 1500**:

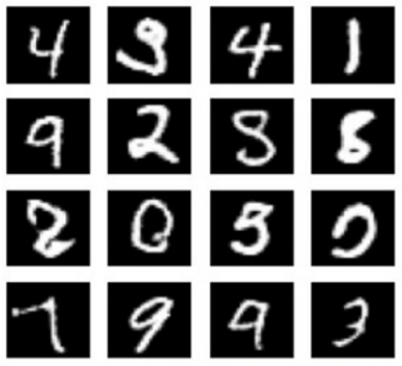
9. **EPOCH NUMBER 1750**:



10. **EPOCH NUMBER 2000**:

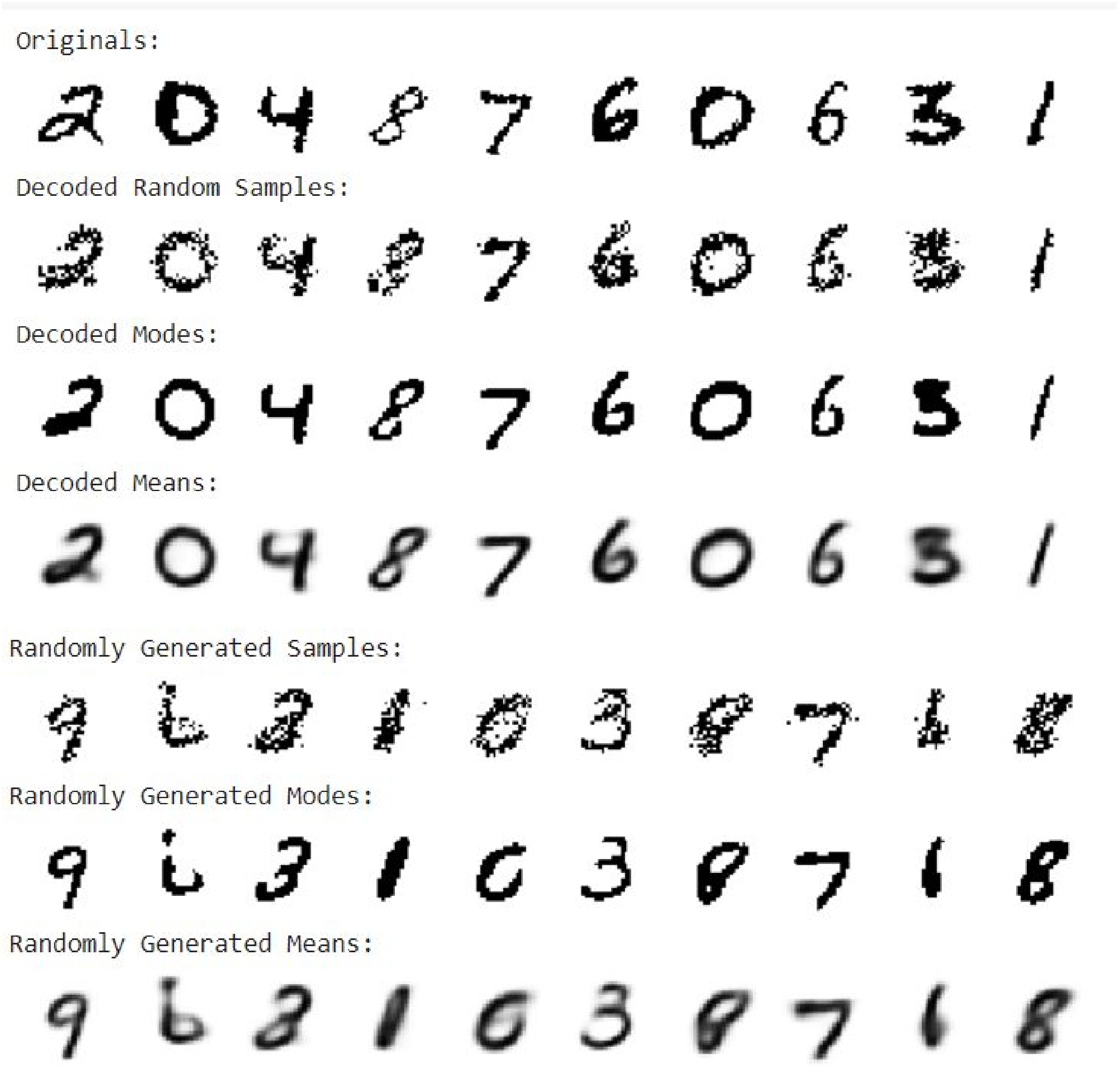


11. **EPOCH NUMBER 2500**:

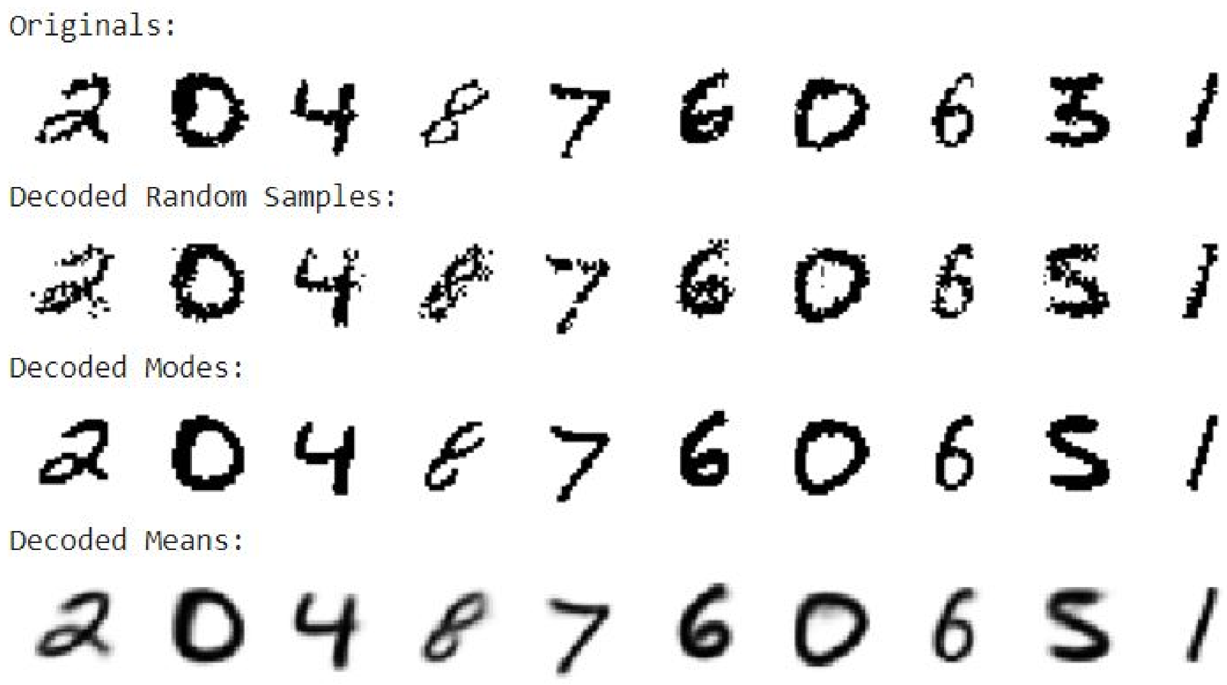


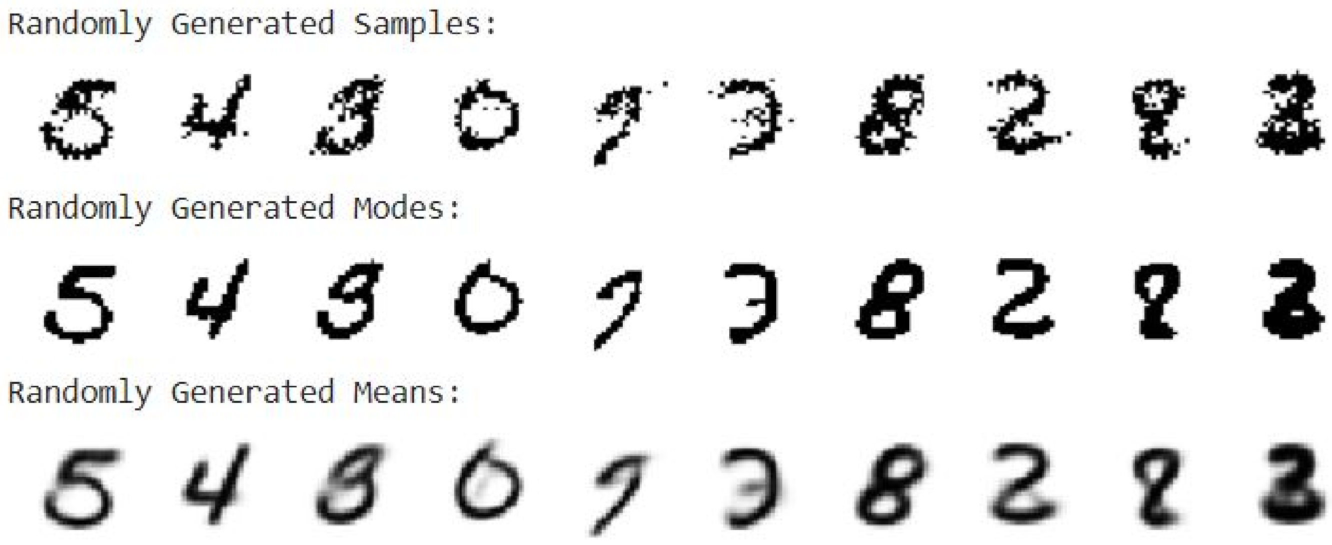
**VAE:-**

1. **EPOCH NUMBER 05**:

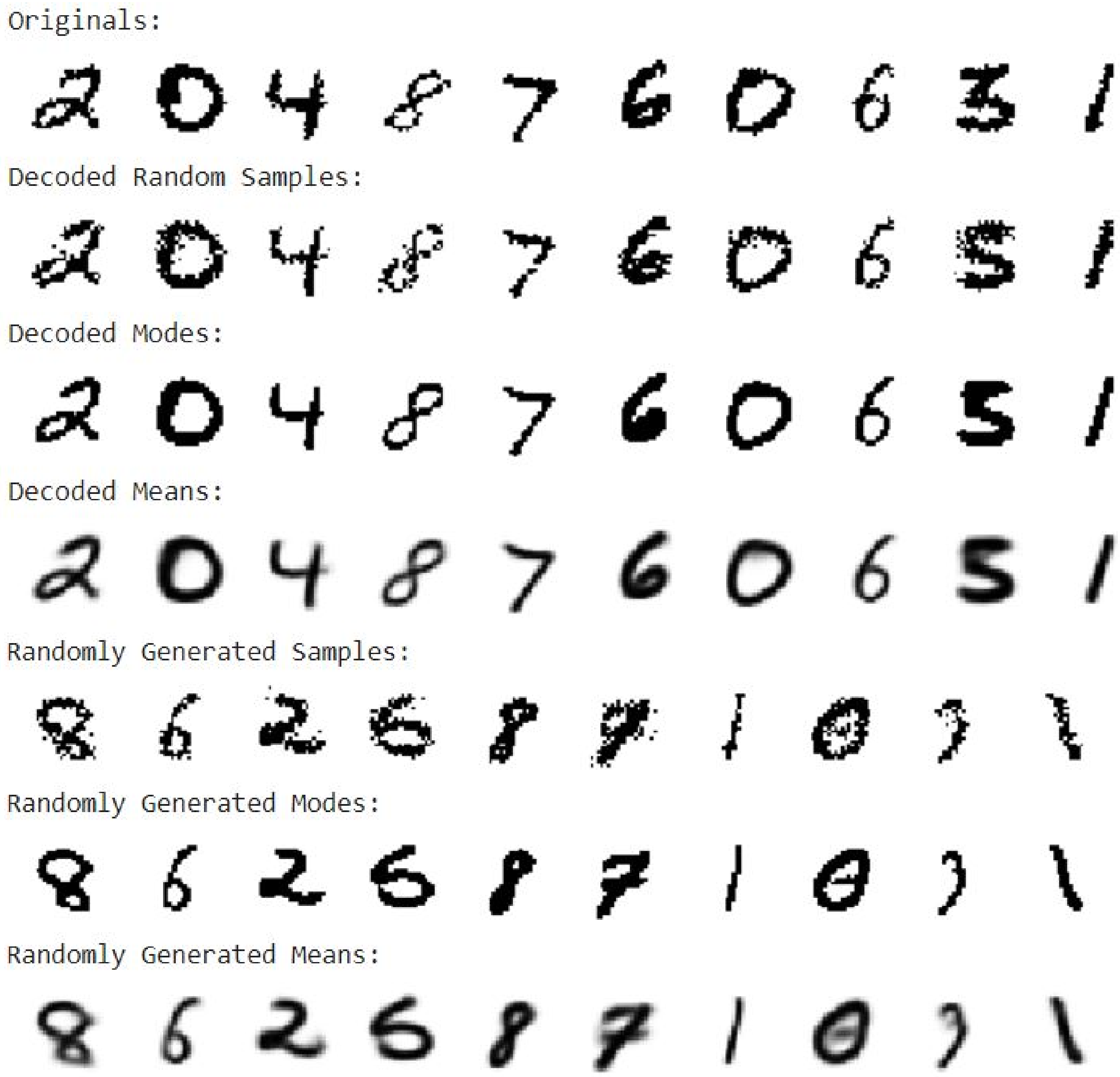

Originals:
Decoded Random Samples:
Decoded Modes:
Decoded Means:
Randomly Generated Samples:
Randomly Generated Modes:
Randomly Generated Means:

2. **EPOCH NUMBER 50**:



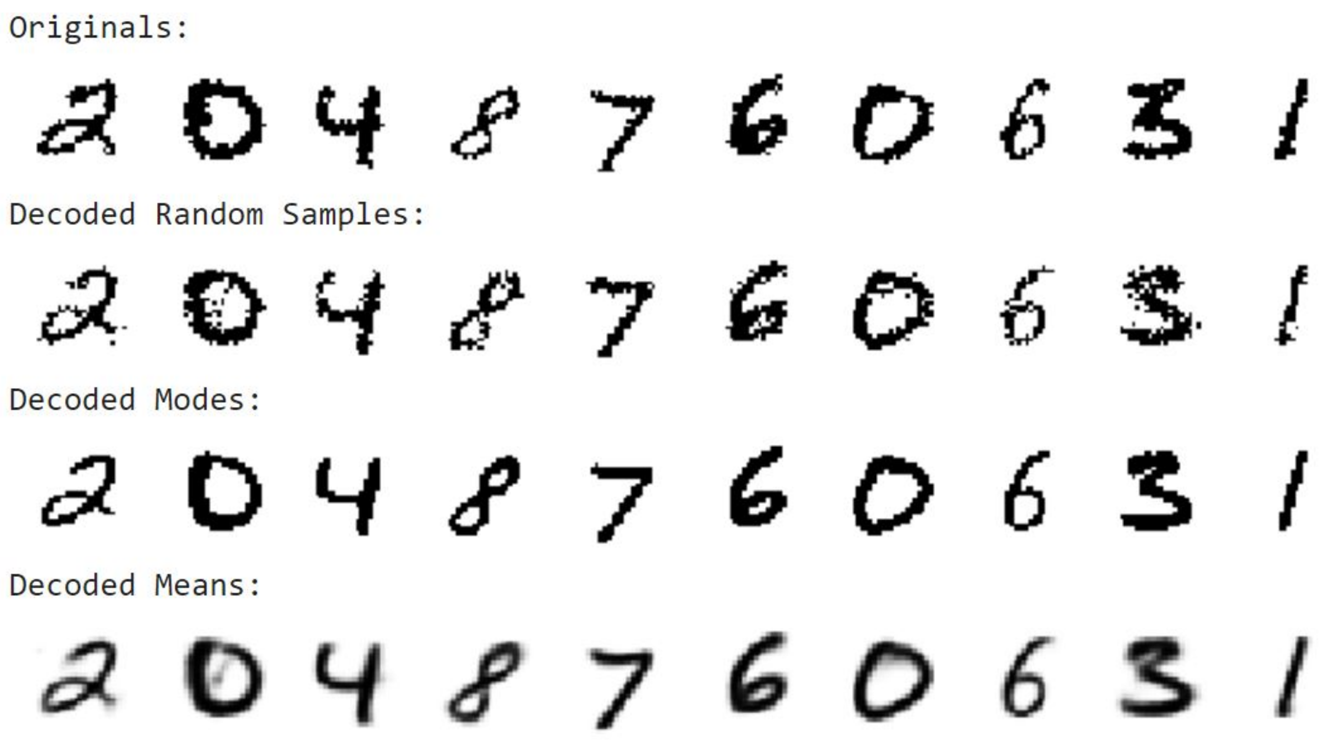

Randomly Generated Samples:
Randomly Generated Modes:
Randomly Generated Means:

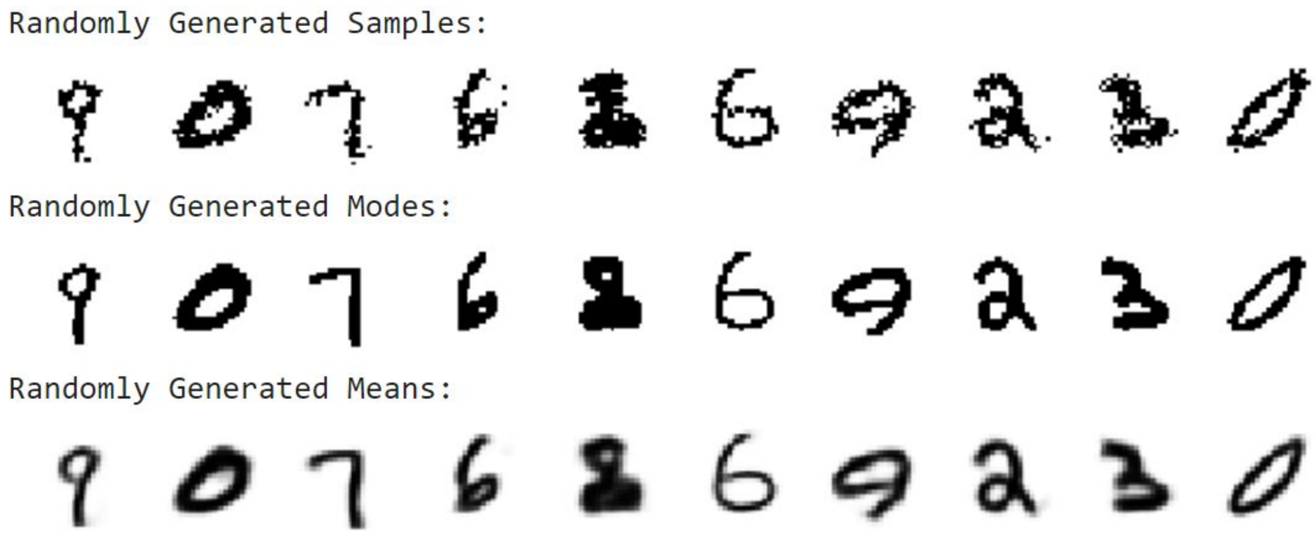
3. **EPOCH NUMBER 100**:



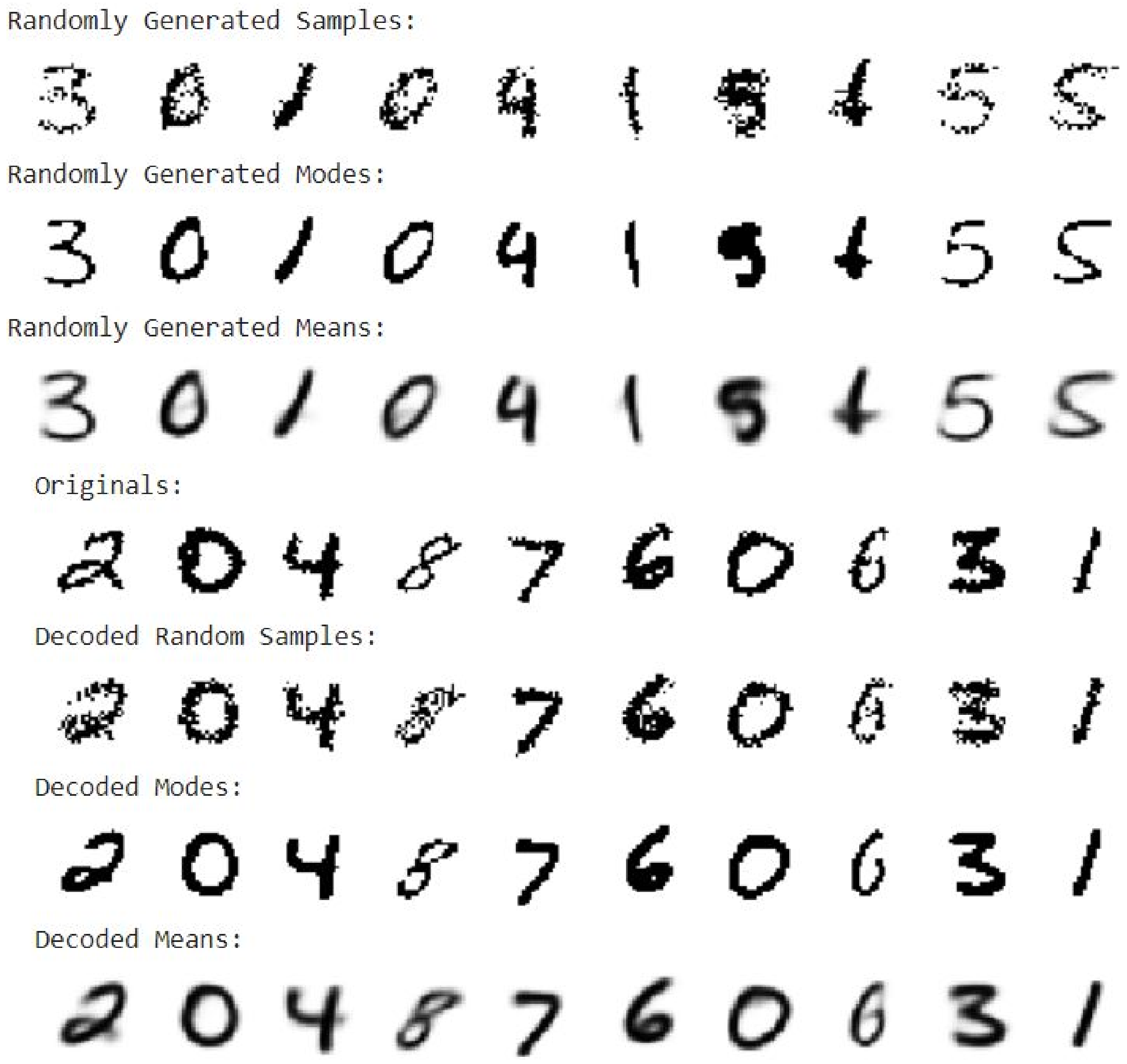
```
Originals:

Decoded Random Samples:

Decoded Modes:

Decoded Means:

Randomly Generated Samples:

Randomly Generated Modes:

Randomly Generated Means:
```

4. **EPOCH NUMBER 250**:




Randomly Generated Samples:
Randomly Generated Modes:
Randomly Generated Means:

5. **EPOCH NUMBER 1000**:


Randomly Generated Samples:
Randomly Generated Modes:
Randomly Generated Means:
Originals:
Decoded Random Samples:
Decoded Modes:
Decoded Means:

## 6 Inferences From Results

From the above results it is clearly evident that as the epoch value increases the data generated by GAN and VAE models will improve. This is because the models get better trained on the MNIST dataset. VAE has generated better images with better clarity when compared to GAN. Also, the time taken on an average to generate an image in an epoch is 10.5 seconds for GAN and 13 seconds for VAE. For the developed model, the epoch number 100 to 500 has seen most of the improvement in GAN models. This could be due to the reason the layers and weights are adjusted well in this phase of model training. After 500 epoch value there is no much significance observed in GAN models. Similarly for VAE after epoch value 100 the model showed significant learning curve. For the epoch value 1000 the VAE data showed the best quality images.

## 7 Problems/Challenges Faced

Some of the problems faced while developing this project are

1. For GAN and VAE, it was challenging to decide on number of layers and type of layers to be used for all four models i.e discriminator model, Generator model, Encoder and Decoder model. It was successfully achieved with better study and understanding.

2. Attempts were made to get better GAN results like better image quality, better images in lower epochs etc. However, there is only partial success in this challenge.

## 8 Conclusions

Both GAN and VAE are very well suitable for generative modelling of handwritten dataset. Also, our study suggested that they are rapidly growing in use for example in Music and Advertisements. This growth suggests that these models are capable to meet some of the big challenges the industry poses. When it comes to performance of GAN vs VAE for our developed model, GAN epoch is slightly shorter than VAE epoch time. However, VAE models are able to generate more accurate models in lesser epoch time. This shows some edge in performance for VAE models. Also, from what we have discussed we have seen briefly that GAN use back propagation to fine tune themselves in generating new models. VAE models on the other hand use probability metrics like mean, variance and guassian distribution for generation. Also, we have discussed the models we have used with their architectures and then drew some important inferences from the Results we have generated from these models. To evaluate the system more precisely the models can be tried testing on any other database that is close to MNIST. The models are currently working well with MNIST database. I will be taking this as a future research work to learn more about the GAN and VAE models.

## References

[1] Link to GitHub Repository - https://github.com/nmalladi6m/CAP6610ProjectWork

[2] https://www.tensorflow.org/tutorials/keras/classification

[3] https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

[4] https://towardsdatascience.com/gans-vs-autoencoders-comparison-of-deep-generative-models-985cf15936ea

[5] https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

[6] https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

[7] https://developers.google.com/machine-learning/gan/generator