Neural Networks and Deep Learning :

Homework 2: Unsupervised Deep Learning

Prepared by : Norhen Abdennadher

Prof : Dr. Alberto Testolin

Academic year 2021-2022

I. Overview:

This homework is all about learning how to implement and test neural network models for solving unsupervised problems. For simplicity and to allow continuity with the kind of data we've seen in the previous homework, it will be based on images of handwritten digits (MNIST). The basic tasks for this homework is to test and analyze the convolutional autoencoder implemented during the Lab practice, by exploring advanced optimizers and regularization methods. The optimization of the model should be done using grid search and the final accuracy should be evaluated using a cross-validation setup.

The homework consists of three main parts:

• Auto-encoder

• Denoising auto-encoder

• Generative adversarial networks

II. Model architecture and hyperparameters:

1. Auto-encoder:

Autoencoder is a type of neural network that can be used to learn a compressed representation of raw data. An autoencoder is composed of an encoder and a decoder sub-model. The encoder compresses the input, and the decoder attempts to recreate the input from the compressed version provided by the encoder.

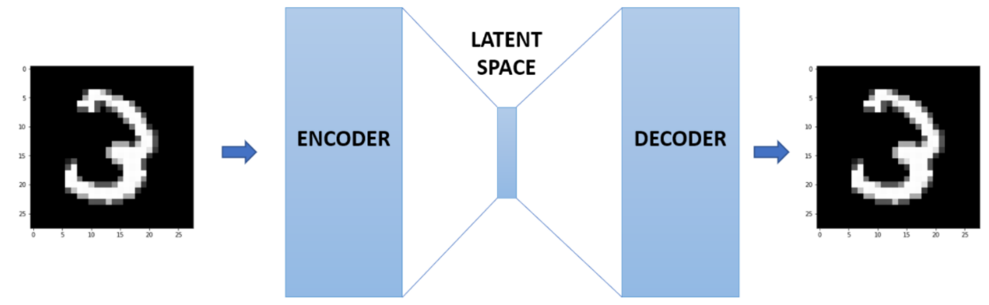


Figure 1: Network architecture

The encoder is a composition of :

• 3 convolutional layers each followed by ReLU activation function.

• Flatten layer: to convert the tensor to a monodimensional array.

• 1 Linear layer with ReLU as activation function.

The Decoder is composed of:

• 2 Linear layers with ReLU as activation function.

•Flatten Layer

• 3 transposed convolutional layers each followed by ReLU activation function and a sigmoid activation function.

Examples of constructed images using the auto-encoder architecture:

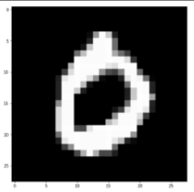
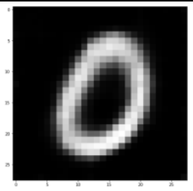
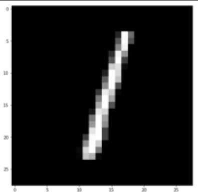
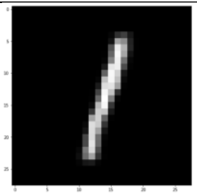
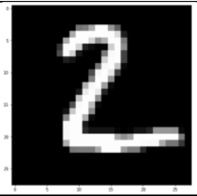
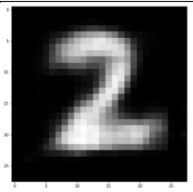
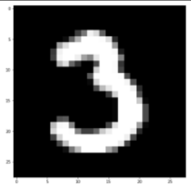
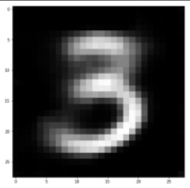
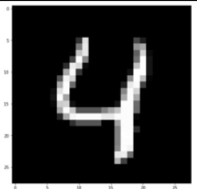
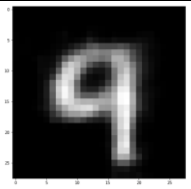
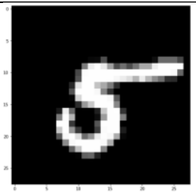
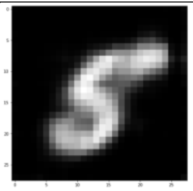
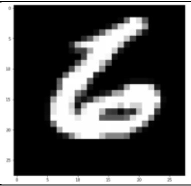
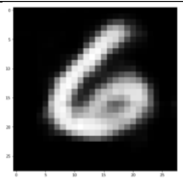
Original	**Constructed**
	
	
	
	
	
	
	

Table 1: Auto-encoder result reconstruction

To optimize the model, we can explore different values of the hyperparameters. Hyperparameters are the variables which determines the network structure(Eg: Number of Hidden Units) and the variables which determine how the network is trained(Eg: Learning Rate).

Hyperparameters are set before training(before optimizing the weights and bias) and it is often required to **search for a set of hyperparameters** that results in the best performance of a model on a dataset. The selection of the best model will be done based on **cross validation** method to make sure that the model isn't overfitting and to make the model selection more accurate.

An optimization procedure involves defining a search space. This can be thought of geometrically as an n-dimensional volume, where each hyperparameter represents a different dimension and the scale of the dimension are the values that the hyperparameter may take on.

Grid search is appropriate for small and quick searches of hyperparameter values that are known to perform well generally.

Search Space is volume to be searched where each dimension represents a hyperparameter and each point represents one model configuration.

A point in the search space is a vector with a specific value for each hyperparameter value. The goal of the optimization procedure is to find a vector that results in the best performance of the model after learning, exactly minimum training and minimum validation error.

For this work we will use grid search technique to tune two hyperparameters which are the leaning rate and the dimension of the latent space. We can also tune other hyperparameters but due to the high complexity and the limited resources I only choose to work on a bidimensional search space:

-Learning_rate=[1e-3, 1e-2, 1e-1]

-Encoded_space_dim=[2,4,6,8,10]

To select the best model, I implemented a cross validation on the training data with 6 folds which corresponds to 10000 samples per fold.

After training the model with cross validation and grid search we selected the best hyperparameters:

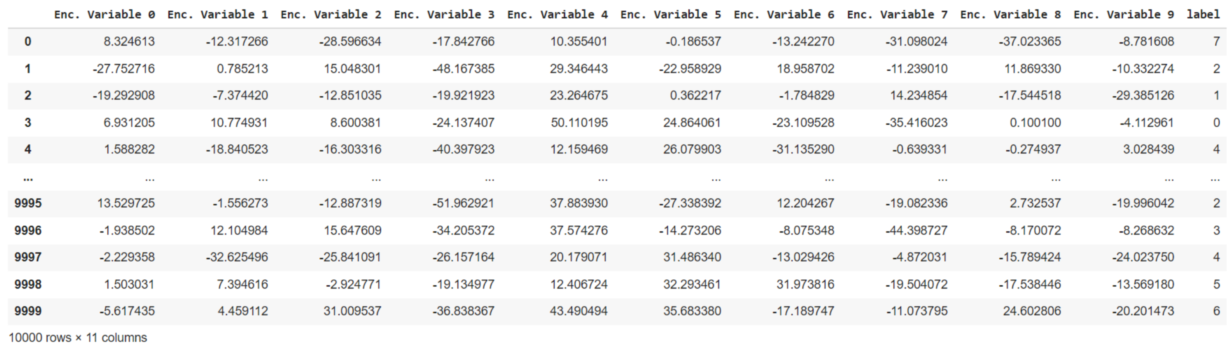
-Best learning rate=1e-3

-Best Encoded_space_dim= 10

Training the model with Encoded_space_dim= 10 and learning rate=1e-3, we got the following results:

Mean Training error=0.014326

Mean Validation error=0.013522



	Enc. Variable 0	Enc. Variable 1	Enc. Variable 2	Enc. Variable 3	Enc. Variable 4	Enc. Variable 5	Enc. Variable 6	Enc. Variable 7	Enc. Variable 8	Enc. Variable 9	label
0	8.324613	-12.317266	-28.596634	-17.842766	10.355401	-0.186537	-13.242270	-31.098024	-37.023365	-8.781608	7
1	-27.752716	0.785213	15.048301	-48.167385	29.346443	-22.958929	18.958702	-11.239010	11.869330	-10.332274	2
2	-19.292908	-7.374420	-12.851035	-19.921923	23.264675	0.362217	-1.784829	14.234854	-17.544518	-29.385126	1
3	6.931205	10.774931	8.600381	-24.137407	50.110195	24.864061	-23.109528	-35.416023	0.100100	-4.112961	0
4	1.588282	-18.840523	-16.303316	-40.397923	12.159469	26.079903	-31.135290	-0.639331	-0.274937	3.028439	4
...	...	...	...	...	...	...	...	...	...	...	...
9995	13.529725	-1.556273	-12.887319	-51.962921	37.883930	-27.338392	12.204267	-19.082336	2.732537	-19.996042	2
9996	-1.938502	12.104984	15.647609	-34.205372	37.574276	-14.273206	-8.075348	-44.398727	-8.170072	-8.268632	3
9997	-2.229358	-32.625496	-25.841091	-26.157164	20.179071	31.486340	-13.029426	-4.872031	-15.789424	-24.023750	4
9998	1.503031	7.394616	-2.924771	-19.134977	12.406724	32.293461	31.973816	-19.504072	-17.538446	-13.569180	5
9999	-5.617435	4.459112	31.009537	-36.838367	43.490494	35.683380	-17.189747	-11.073795	24.602806	-20.201473	6

10000 rows × 11 columns

Figure 2: Latent space table representation

In order to visualize the latent space of dimension 10, we need to apply PCA (principle component analysis) on the encoded representation of the test set and see the repartition of each class.

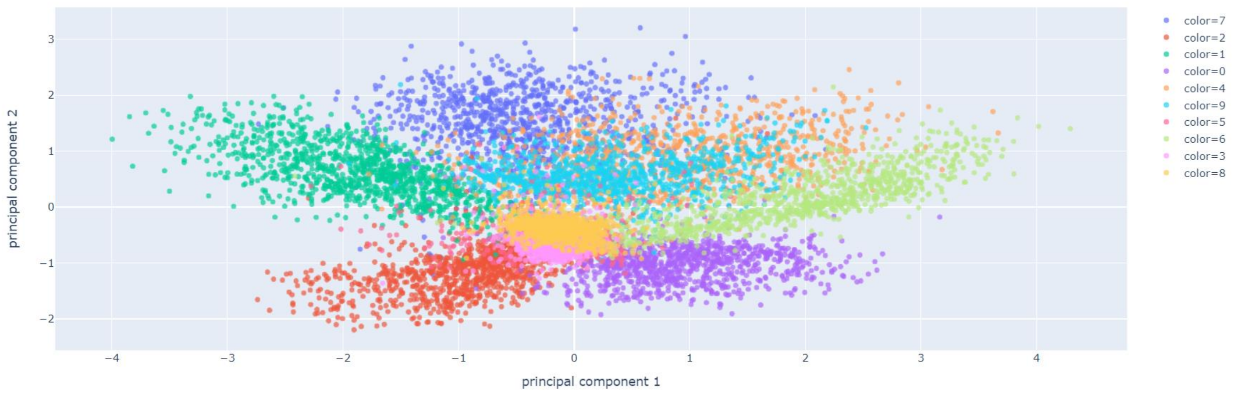


Figure 3: Latent space representation after PCA

As we can see, the classes are clearly separated, and the network has been able to clusterize quite well the different classes. We can notice that number 4 and number 9 are quite similar, and their separation is quite difficult.

2. **Auto-encoder classification task:**

We want to classify the MNIST handwritten digits dataset using the same auto-encoder architecture. To perform this task, there are two different methods:

First classifier can be made up of the half part of the Auto-encoder, more specifically only the encoder part, and then an extra fully connected layer that will be attached to the decoder and will be trained.

The second method is to train the encoder such that the latent dimension space should be of dimension=10 which is the number of classes in the dataset, so we can consider each neuron value as the possibility for the input to be the index digit. Using this method, it only took few epochs to reach almost the same results we got in the first homework .

The results : Average test loss : 0.079229

Test accuracy: 98.369%

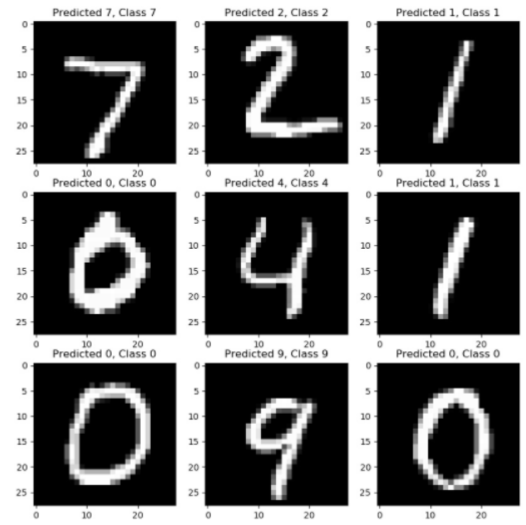


Figure4: Predicted and original class

3. Denoising Auto-encoder:

Denoising autoencoders are an extension of the basic autoencoder, and represent a stochastic version of it. Denoising autoencoders attempt to address identity-function risk by randomly corrupting input (i.e. introducing noise) that the autoencoder must then reconstruct, or denoise.

The architecture is the same as before. The main difference of the denoising auto-encoder is that the model takes MNIST images with added noise as input and tries to reconstruct or to extract clean images as output from it. In the figure below, we can see some samples from the MNIST dataset denoised with gaussian noise and a random rotation.

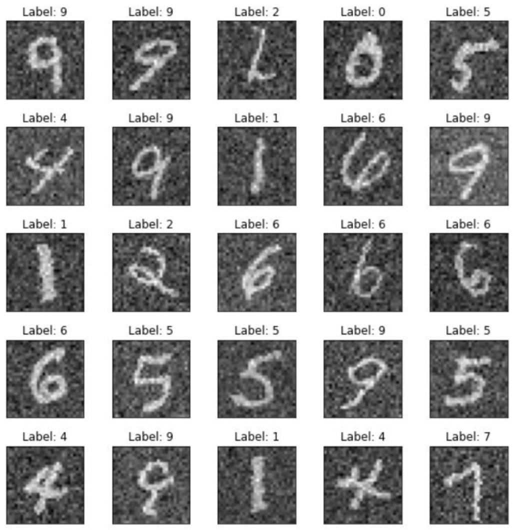


Figure5: MNIST samples with gaussian noise

After training the same network on the new noised dataset, we can notice that our denoising auto-encoder is able to denoise the images very well.

The reached results:

Train loss: 0.058499

```
VALIDATION - EPOCH 40/40 - loss: 0.058079
```

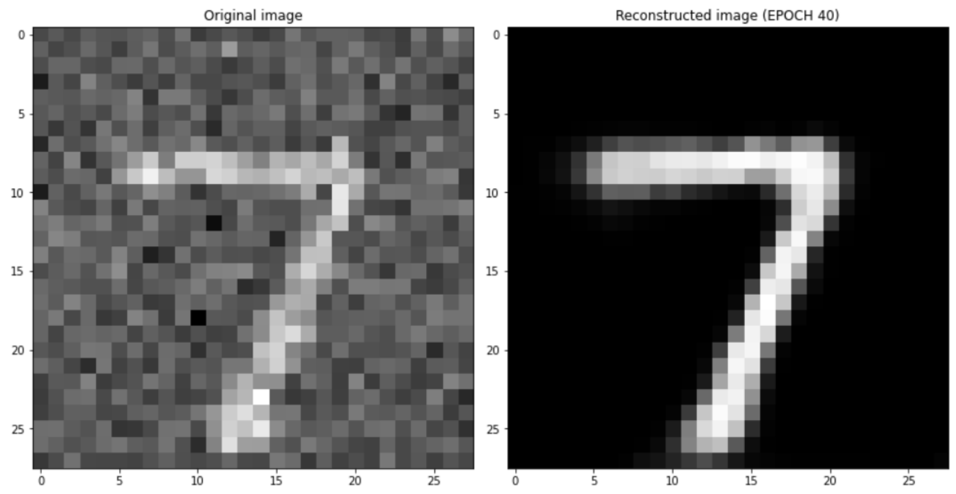


Figure6: Denoised reconstructed image at last epoch

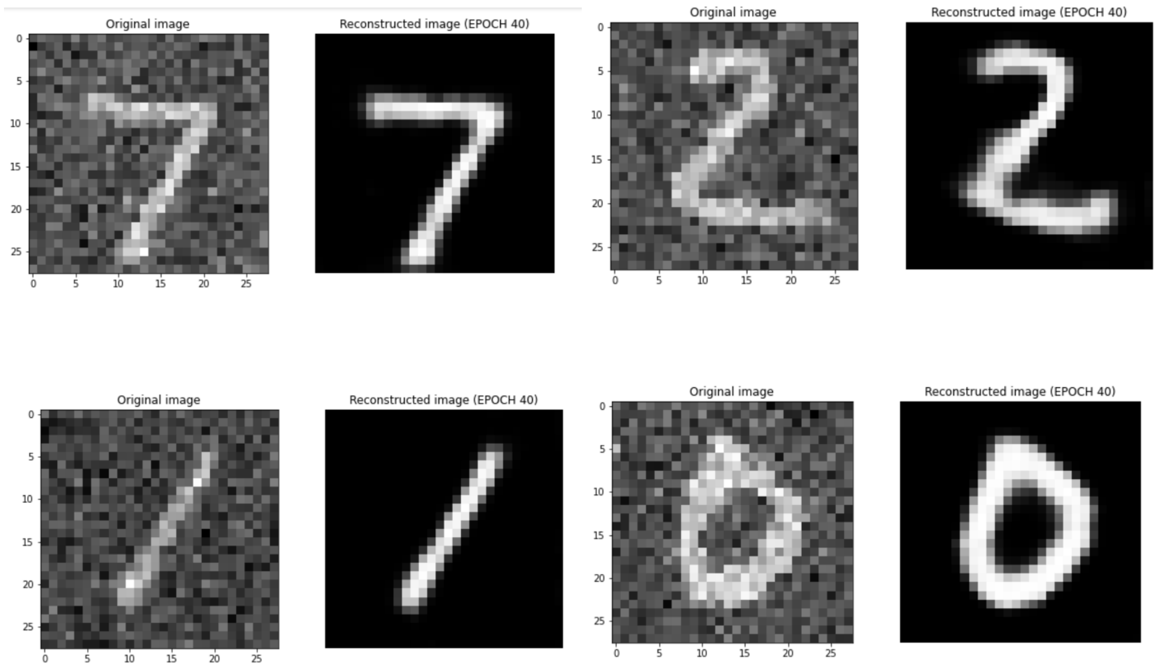


Figure7: Reconstructed samples

4. **GAN:**

Generative Adversarial Networks, or GANs for short, are types of Neural Networks used for Unsupervised learning.

Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models: the generator model that we train to generate new examples, and the discriminator model that tries to classify examples as either real (from the domain) or fake (generated). The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible examples. We have a Discriminative network to evaluate generated data.This model consists of :

A generator:

-2 hidden linear layers with LeakyReLU as activation function and dropout layer with probability equals 0.5.

-An output layer with Tanh activation function.

A discriminator:

-2 hidden linear layers with LeakyReLU as activation function and dropout layer with probability equals 0.5.

-An output layer with sigmoid activation function.

The Generative network learns to map from a latent space to a data distribution of interest, while the Discriminative network distinguishes candidates produced by the generator from the true data. The generative network's training objective is to increase the error rate of the discriminative network.

After Training the network we got the following results :

Gen/Dis Loss: 1.3827/0.4937

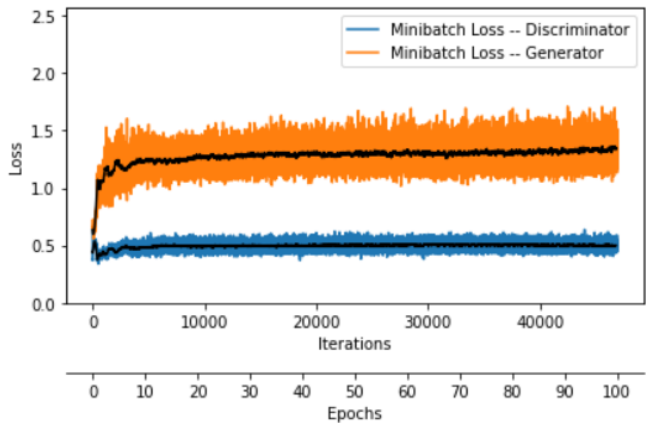


Figure 8:Training loss for discriminator and generator networks

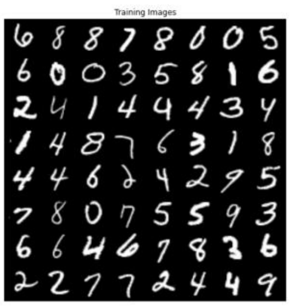
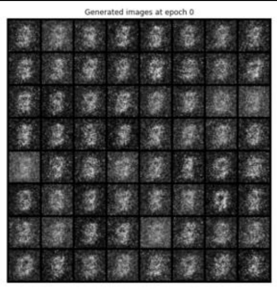
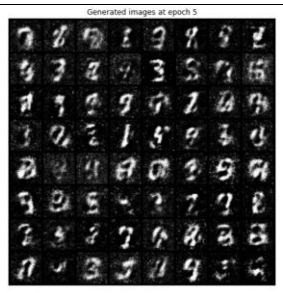
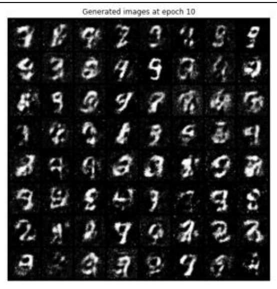
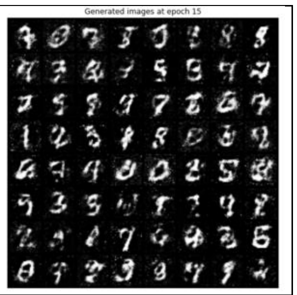
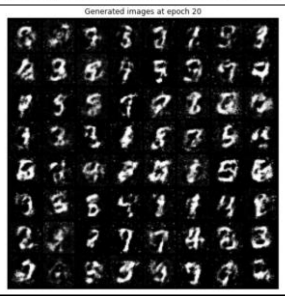
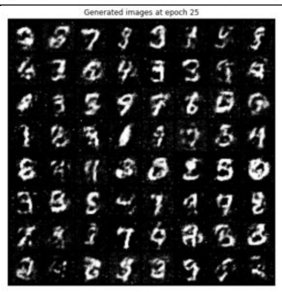
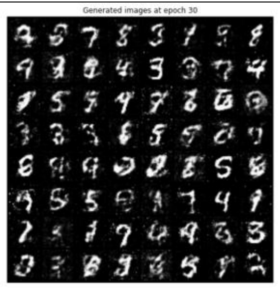
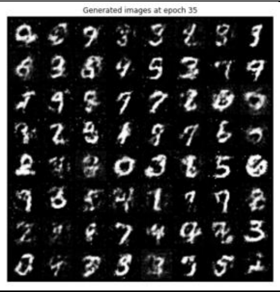


Figure9: Training images

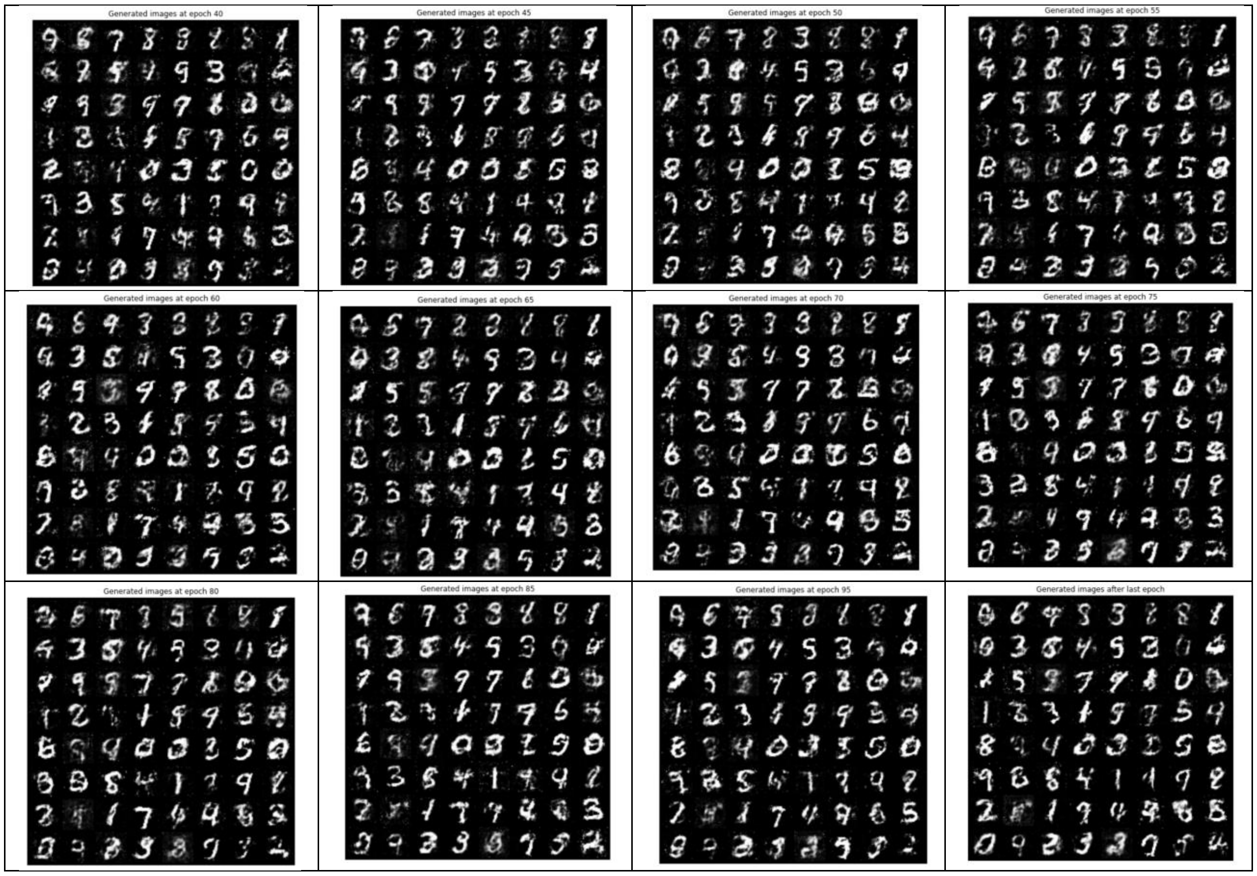

Generated images at epoch 40
Generated images at epoch 45
Generated images at epoch 50
Generated images at epoch 55
Generated images at epoch 60
Generated images at epoch 65
Generated images at epoch 70
Generated images at epoch 75
Generated images at epoch 80
Generated images at epoch 85
Generated images at epoch 95
Generated images after last epoch

Figure 10:Generated images at different epochs from training process