

REAL LOOKING NUMBER IMAGE GENERATION USING GAN

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology degree in Information Technology

By

SANTOSH KUMAR (Reg No. 38120076)

ABHISHEK KUMAR (Reg No. 38120002)



**DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNINIVERSITY)**

Accredited with Grade “A” by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI – 600 119

APRIL 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" Grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119.

Phone: 044 - 2450 3150 / 51 / 52 / 54 / 55 Fax: 044 - 2450 2344

www.sathyabama.ac.in



DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **SANTOSH KUMAR (Reg No: 38120076)** and **ABHISHEK KUMAR (Reg No: 38120002)** who carried out the project entitled **“REAL LOOKING NUMBER IMAGE GENERATION USING GAN”** under our supervision from **NOVEMBER 2021** to **APRIL 2022**.

INTERNAL GUIDE

Dr. M.Saravanan M.E.,Ph.D.

HEAD OF THE DEPARTMENT

Dr.R.Subhashini M.E.,Ph.D.

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, SANTOSH KUMAR (Reg No: 38120076) and ABHISHEK KUMAR (Reg No: 38120002) hereby declare that the Project Report entitled as “REAL LOOKING NUMBER IMAGE GENERATION USING GAN” done by us under the guidance of Dr. M. Saravanan M.E.,Ph.D., Assistant Professor Department of Computer Science and Engineering, School of Computing at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology (B.Tech.) degree in Information Technology (IT).

DATE: 22/04/2022

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr. M. Saravanan M.E., Ph.D.,Assistant Professor**, School of Computing and **Dr. R. Subhashini M.E., Ph.D.**,Headof the Department, Department of Information Technology for providing us necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide**Dr. M. Saravanan M.E., Ph.D.,Assistant Professor**, Department of Computer Science and Engineering for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

We wish to express our thanks to all Teaching and Non-teaching staff members of the Department of **Information Technology**who were helpful in many ways for the completion of the project.

ABSTRACT

The “**Real Looking Number Image Generation Using GAN**” is a model which uses Generative Adversarial Network for Generating the images of number which looks real. The technology behind this is Machine Learning Generative Adversarial Networks, which is an Generative model which generates two adverse models. MNIST dataset have been used in this model. This model was developed to produce a prototype for developing a model which can generate a crime scenarios which are not being captured due to the unavailability of CCTVs, especially in rural areas, with the help of activities happened on the spot of crime. The enhancement of this model could solve so many problems specially solving crime cases effectively and quickly. This project is divided into three parts namely, Training Discriminator, Training a Generator, and Generating the Realistic Numbers. Our model has met all the objectives derived and planned. The project is considered successful.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	ix
1	INTRODUCTION	1
1.1	OVERVIEW OF PROJECT	1
1.2	OBJECTIVE OF THE PROJECT	2
2	AIM AND SCOPE OF PRESENT IMPLEMENTATION	3
2.1	AIM AND SCOPE	3
2.2	ARCHITECTURE	3
2.3	TRAINING DISCRIMINATOR	3
2.4	TRAINING GENERATOR	4
3	SYSTEM ANALYSIS	5
3.1	REQUIREMENTS	5
3.2	PROPOSED SYSTEM	5

3.3	ALGORITHM FOR THE PROPOSED SYSTEM	5
3.4	HARDWARE REQUIREMENTS	5
3.5	SOFTWARE REQUIREMENTS	5
4	RESULT AND DISCUSSION PERFORMANCE ANALYSIS	6
4.1	RESULT ANALYSIS	6
5	CONCLUSION AND FUTURE ENHANCEMENT	11
5.1	CONCLUSION	11
5.2	FUTURE SCOPE	11
	REFERENCES	12
	APPENDICES	13
	SOURCE CODE	13
	SCREEN SHOTS	19
	PUBLICATION CERTIFICATE	20

LIST OF FIGURES

FIGURE NO.	NAME OF THE FIGURE	PAGE NO.
1.1	TRAINING OF DISCRIMINATOR	3
1.2	TRAINING OF GENERATOR	4
3.1	IMAGE GENERATION	4
3.2	ARCHITECTURE DIAGRAM	4
3.3	INPUT AND OUTPUT	8

LIST OF ABBREVIATIONS

GAN	Generative Adversarial Network
AI	Artificial Intelligence
ML	Machine Learning
CCTV	Closed Circuit Television
GPU	Graphics Processing Unit
NumPy	Numerical Python
MNIST	Modified National Institute of Standards and Technology

CHAPTER 1

INTRODUCTION

1.1 OUTLINE OF THE PROJECT

Most of the time, crime cases in rural areas remain unsolved due to the lack of availability of CCTVs and it is not possible that a exact crime doer or the crime scene can be spot in the cctvs, so using the behaviour of the crime doer and by creating the crime scenario is very crucial in order to solve the cases. GAN can help in those scenarios by creating the almost real looking scenario. Our model generates MNIST samples that human cannot distinguish from real data. Our model describes the Generation of Real Looking images of numbers using GAN. GAN can also help in creating the sketch of any person just by his or her behaviour. GAN (Generative Adversarial Network) is a ML model in which two neural networks compete with each other to become more accurate in terms of prediction.

1.2 Objective of the Project

The Objective is to produce a base for a system which can be developed to Generate a system which can produce the crime scenarios with the help of activities happened at crime spot. GAN can fulfill this objective if we extend this idea as it generates two adverse networks namely Generator and Discriminator. The generator can be trained by using the existing data and reports of crimes. In our project we have produced a model which generates the number images which looks real.

CHAPTER 2

AIM AND SCOPE OF THE PROPOSED WORK

2.1 AIM

The aim of the proposed system is to train a model which can generate the images of numbers which looks real. In most of the villages, CCTVs are not yet deployed which causes a lot of difficulties in order to solve any crime case as crime scenes are not captured, most of the time the cases remain unsolved. In this scenarios we need a system which can generates the scenarios of the crime. GAN can be used to produce this required system. Our model can be enhanced to produce this system. This system can be further advanced in several other areas, like Portrait sketching, Natural Language Generation, Medical Imaging, Music Generation, etc.

2.2 Architecure Diagram

GAN is a algorithmic architecture that uses two neural networks, having adverse behaviour in nature in order to generate a new network. Several components of the architecture diagram is shown below.

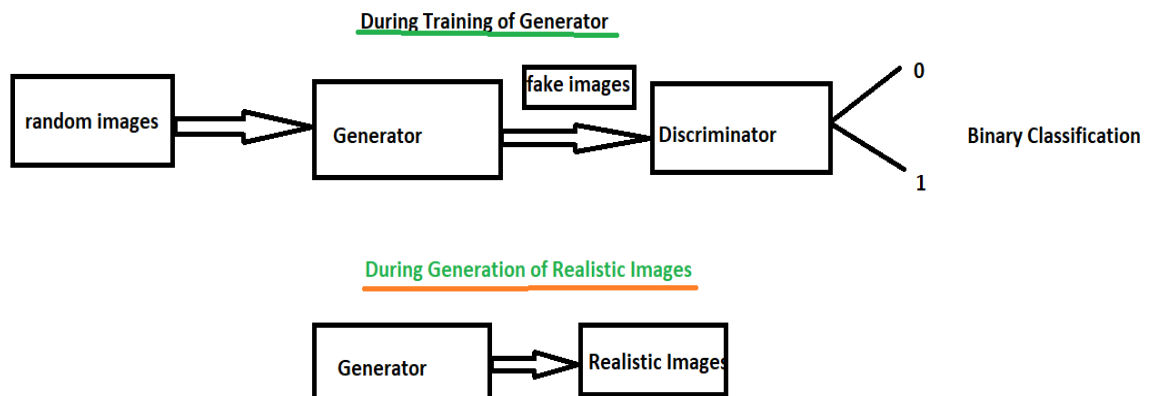
2.3 Training Discriminator

The first and most important part of our implementation is to train the discriminator which discriminates between the real and fake images. In order to train Discriminator, we need to input the dataset images, MNIST dataset has been used in this case. MNIST dataset is consist of 60k images each image having size of 28*28 px., each pixels have value from 0 to 255. We need to flatten the image and make the image pixel value either 0 or 1. The images are given to the discriminator and it classifies the image as real or fake. It is a binary classifier, we label fake image as 0 and real as 1. Once the Discriminator is trained, we need to train the Generator.



2.4 Training Generator

Once the Discriminator is trained, we need to train the Generator. We through random images to the Generator and it goes to Discriminator. Discriminator classifies the images as real or fake. Once Generator is able to generate the images which the Discriminator is not able to identify correctly, then the Generator is trained and now, we can use the generator to generate the required images.



CHAPTER 3

SYSTEM ANALYSIS

3.1 System Requirements

- 64 bit computer
- Gpu (provided by Google Colab)
- Google Colab online Notebook

3.2 PROPOSED SYSTEM

The proposed system “**Real Looking Number Image Generation Using GAN**” is a Machine Learning model which uses GAN (Generative Adversarial Network) for Generating the images of number which looks real. The technology behind this is Machine Learning Generative Adversarial Networks, which is a Generative model which generates two adverse models. MNIST dataset have been used in this model. This model was developed to produce a prototype for developing a model which can generate a crime scenarios which are not being captured due to the unavailability of CCTVs, especially in rural areas, with the help of activities happened on the spot of crime. The enhancement of this model could solve so many problems specially solving crime cases effectively and quickly. This project is divided into three parts namely, Training Discriminator, Training a Generator, and Generating the Realistic Numbers. Our model has met all the objectives derived and planned. The project is considered successful.

3.3 ALGORITHMS USED

- GAN
- Logistic Regression

3.4 HARDWARE REQUIREMENTS

- RAM: 8GB
- Storage: 500GB
- CPU: 2 GHz or faster
- Architecture: 32-bit or 64-bit
- GPU (provided by Google Colab)

3. SOFTWARE REQUIREMENTS

- Python 3.5 in Google Collab is used for data preprocessing, model training and image generation.
- Operating System: windows 10.

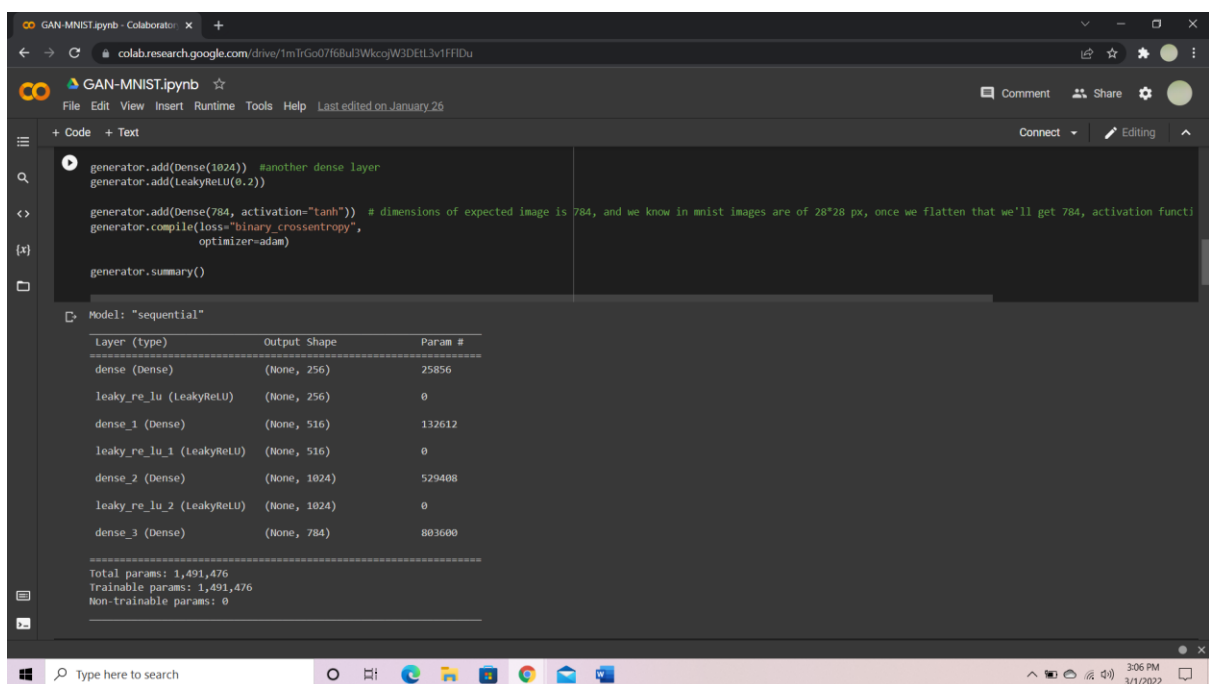
CHAPTER 4

RESULT AND DISCUSSION PERFORMANCE ANALYSIS

4.1 RESULT ANALYSIS

At starting we found that there are images which has different pixel values from 0 to 255.

We flattened the image and converted the values into 0 or 1. Our Generative is sequential model.



The screenshot shows a Google Colab notebook titled 'GAN-MNIST.ipynb'. The code cell contains the following Python code:

```
generator.add(Dense(1024)) #another dense layer
generator.add(LeakyReLU(0.2))

generator.add(Dense(784, activation="tanh")) # dimensions of expected image is 784, and we know in mnist images are of 28*28 px, once we flatten that we'll get 784, activation functi
generator.compile(loss="binary_crossentropy",
                  optimizer=adam)

generator.summary()
```

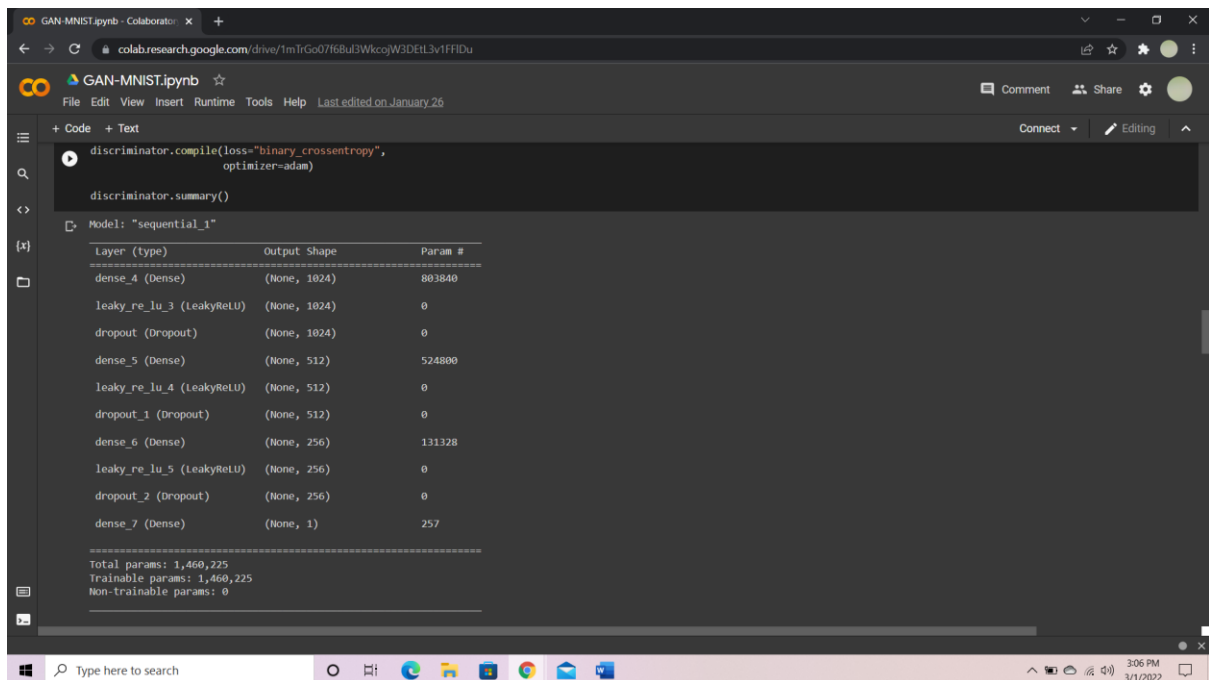
The output of the code is a model summary for a 'sequential' model. It displays a table with columns: Layer (type), Output Shape, and Param #.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	25856
leaky_re_lu (LeakyReLU)	(None, 256)	0
dense_1 (Dense)	(None, 516)	132612
leaky_re_lu_1 (LeakyReLU)	(None, 516)	0
dense_2 (Dense)	(None, 1024)	529408
leaky_re_lu_2 (LeakyReLU)	(None, 1024)	0
dense_3 (Dense)	(None, 784)	883600

Below the table, the following statistics are shown:

```
=====
Total params: 1,491,476
Trainable params: 1,491,476
Non-trainable params: 0
```

We need to equalize the dimensions of Generator and Discriminator, and Discriminator is also a sequential model and the same is described below



The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

```
discriminator.compile(loss="binary_crossentropy",  
                    optimizer=adam)  
  
discriminator.summary()
```

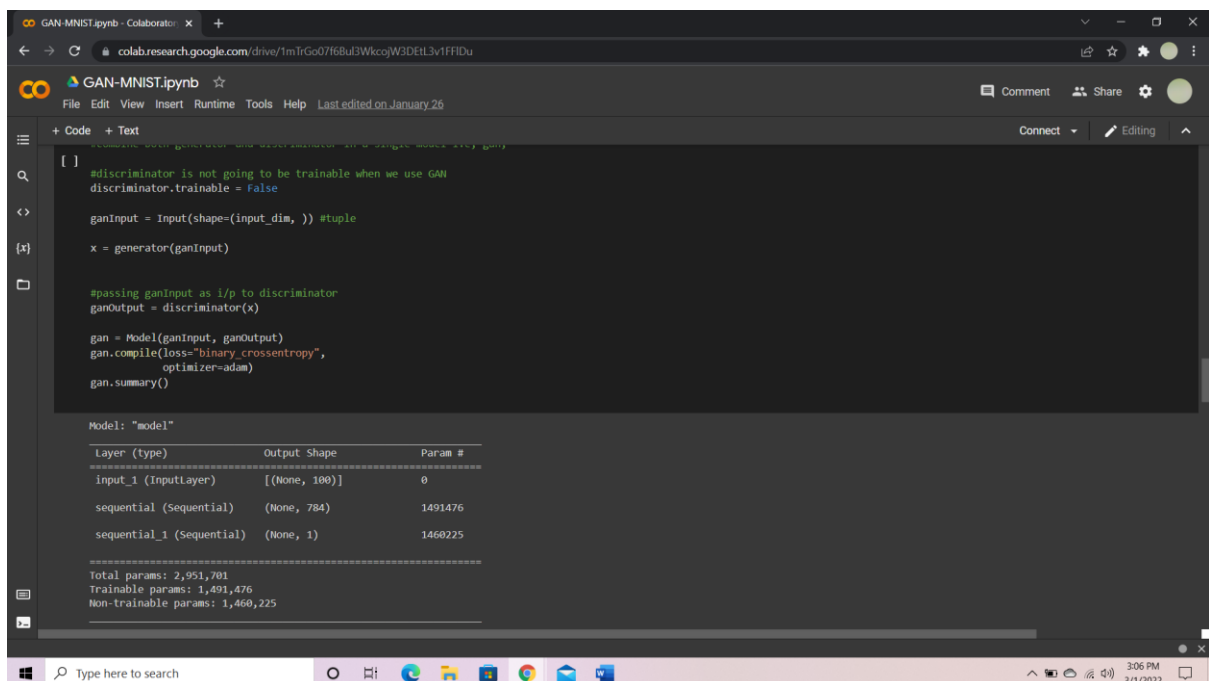
The output of the code is a summary of the model, titled "Model: 'sequential_1'". It displays a table with three columns: Layer (type), Output Shape, and Param #.

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 1024)	803840
leaky_re_lu_3 (LeakyReLU)	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 512)	524800
leaky_re_lu_4 (LeakyReLU)	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 256)	131328
leaky_re_lu_5 (LeakyReLU)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 1)	257

Below the table, the following summary statistics are provided:

```
=====  
Total params: 1,460,225  
Trainable params: 1,460,225  
Non-trainable params: 0
```

After Combining the Generator and Discriminator, the parameters looks as following



The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

```
#combine both generator and discriminator in a single model -> gan  
[ ]  
#discriminator is not going to be trainable when we use GAN  
discriminator.trainable = False  
  
ganInput = Input(shape=(input_dim, )) #tuple  
x = generator(ganInput)  
  
#passing ganInput as i/p to discriminator  
ganOutput = discriminator(x)  
  
gan = Model(ganInput, ganOutput)  
gan.compile(loss="binary_crossentropy",  
            optimizer=adam)  
gan.summary()
```

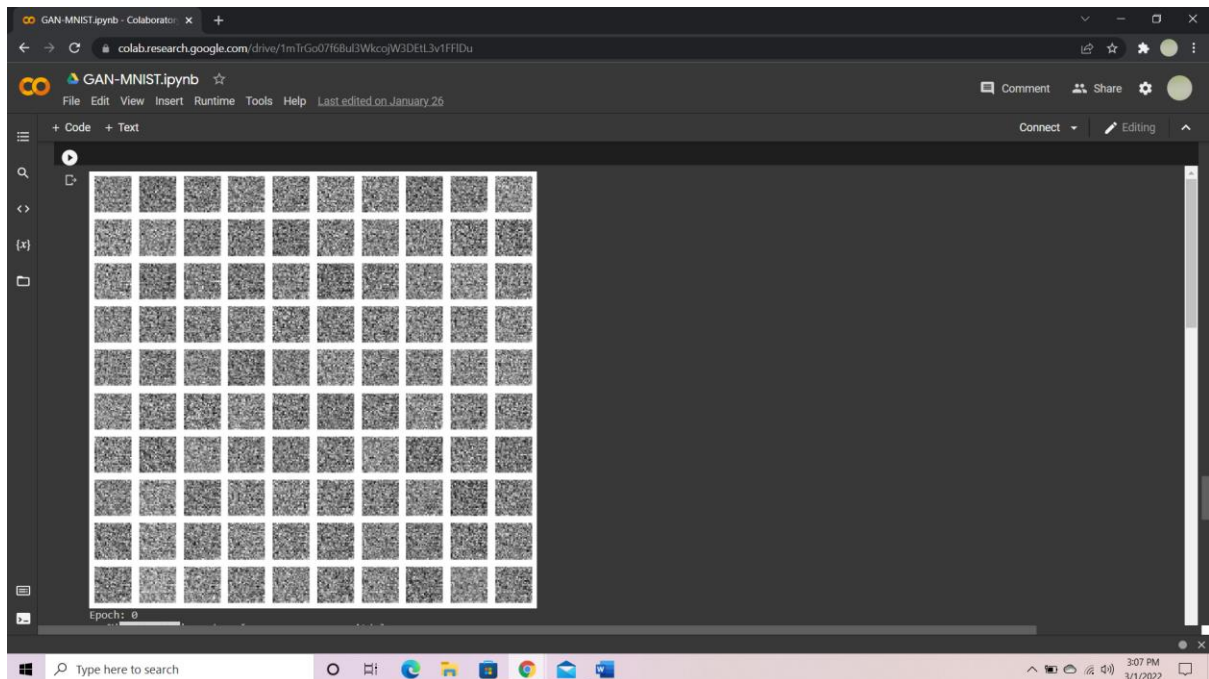
The output of the code is a summary of the model, titled "Model: 'model'". It displays a table with three columns: Layer (type), Output Shape, and Param #.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 100)]	0
sequential (Sequential)	(None, 784)	1491476
sequential_1 (Sequential)	(None, 1)	1460225

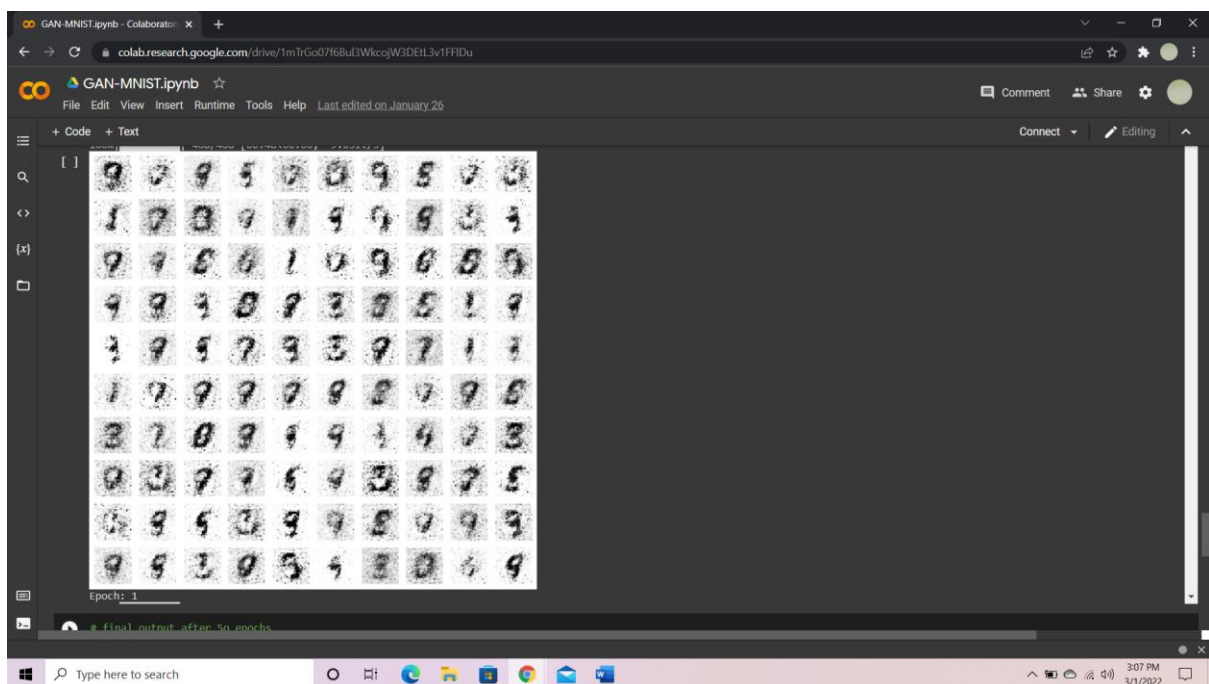
Below the table, the following summary statistics are provided:

```
=====  
Total params: 2,951,701  
Trainable params: 1,491,476  
Non-trainable params: 1,460,225
```

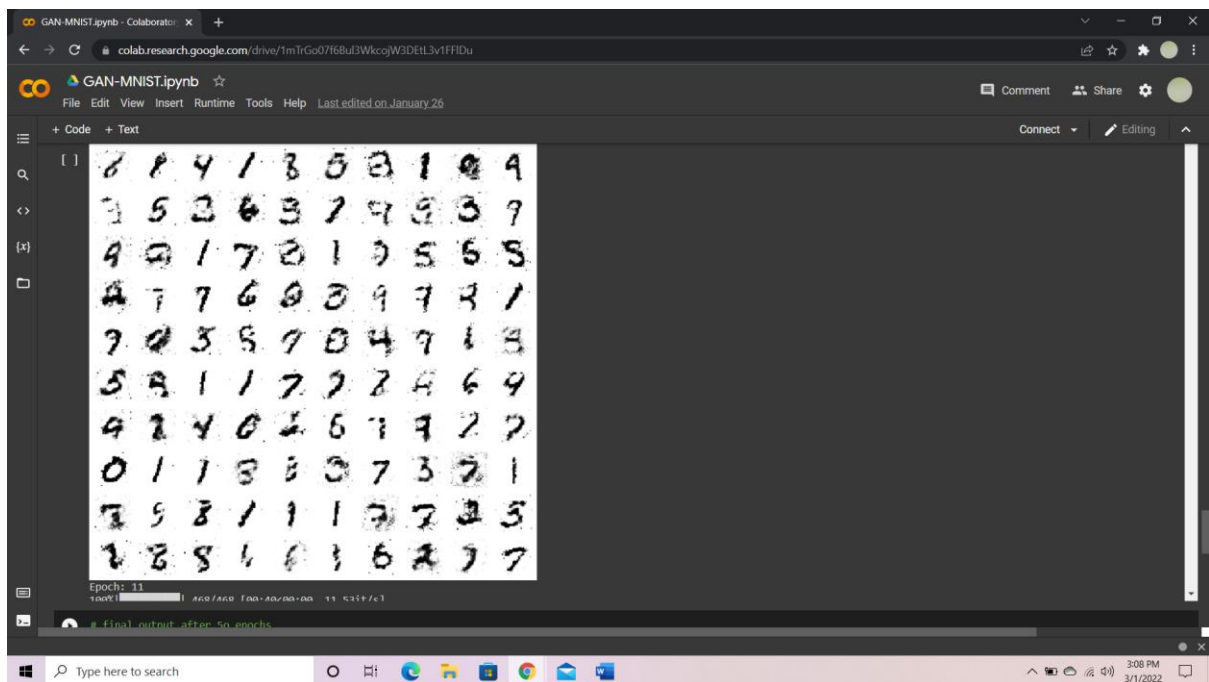
During the training of Generator, the output is depicted below. We can clearly observe that, nothing is clear.



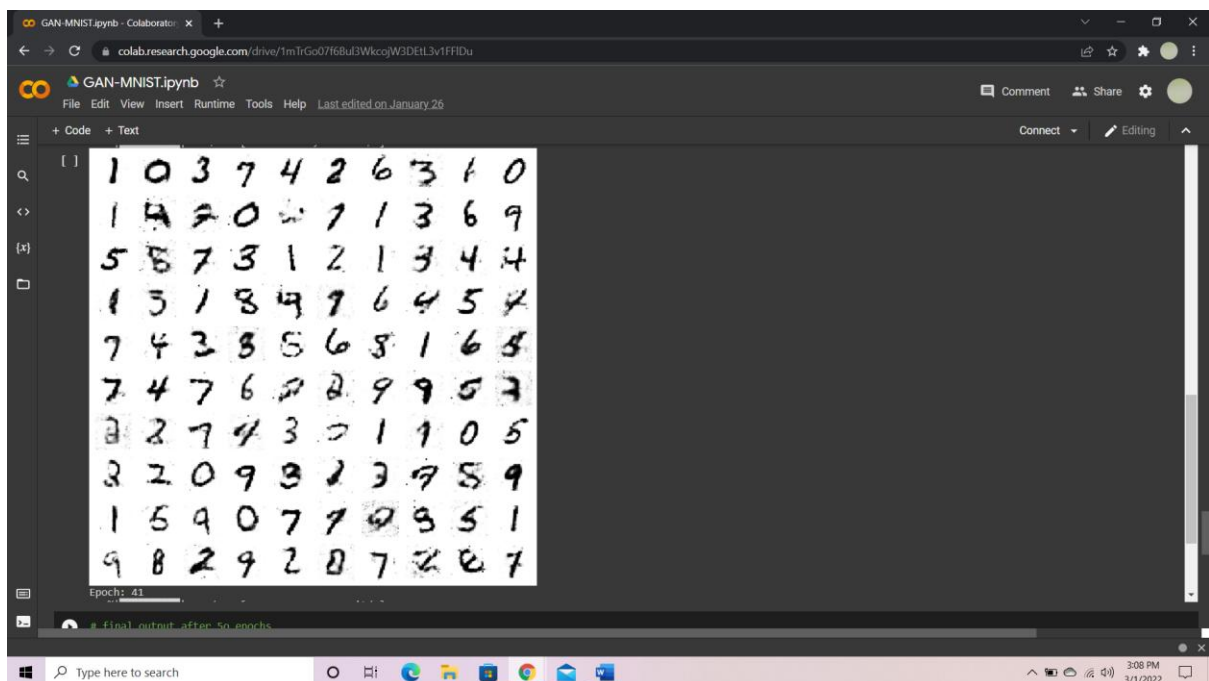
After the first epochs, the images get somewhat cleared, it starts producing the images



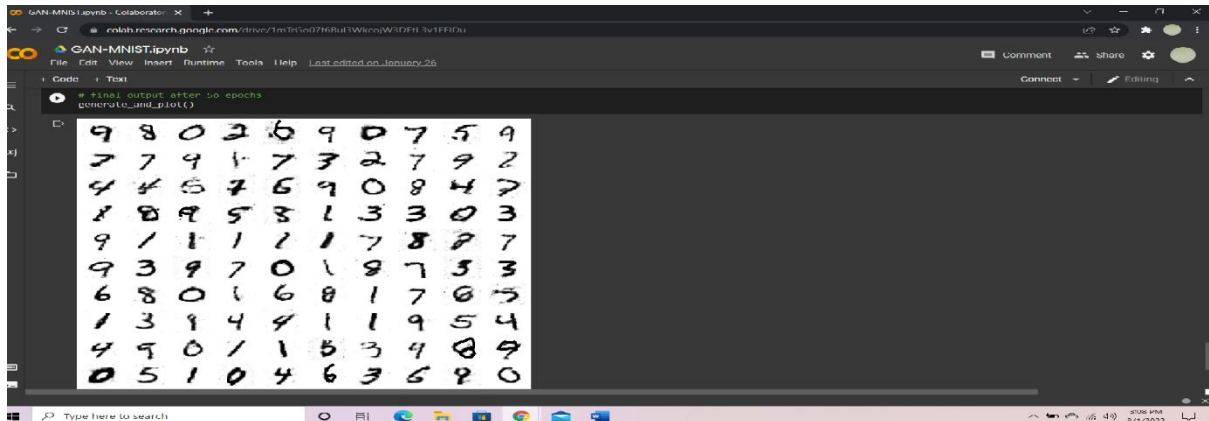
After 11 epochs, the images got more dense and become much clearer than earlier



The following is the output after 40 epochs, and the changes are easily observed.



After the 100 epochs, the images are very much clear, and it is able produce the output correctly.



From above results, we can clearly observe that, the images get clearer with the epochs. Initially, it was faded and nothing could be concluded from the initial image. As the number of epochs increases, the images get clearer and we get the required results.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

The lack of availability of CCTVs in rural area forces us to produce a system which can generate a crime scenarios on the basis of the activities happened in order to solve the case with ease and effectively. So, this system can work as the base for producing those systems as proposed system generates the images of number by training a model. We got the desired output and the work is completed successfully.

6.2 FUTURE ENHANCEMENT

In the future part of work, it is recommended to deploy the implementation so that other can also get the insights from this model to solve other problems as well. This model can be used in other industries as well. It can be applied in Music Generation, Medical Intelligence, Natural Language Generation, etc.

REFERENCES

- [1] A. Mino, and S. Gerasimos "LoGAN: Generating logos with a generative adversarial neural network conditioned on color." In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 965-970, Netherland, 23 Oct 2018.

- [2] J. Patel, S. Pandya, V. Shah "Review on generative adversarial networks" International Journal of Technical Innovation in Modern Engineering. vol . 4, no. 7. pp. 1230-1235, 2018.

- [3] P. Isola, Z. Jun-Yan, Z. Tinghui, and F. Alexei "Image-toimage translation with conditional adversarial networks." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 11251134. 2017.

- [4] I. Goodfellow, P. Jean, M. Mehdi, X. Bing, W. David, O. Sherjil, C. Aaron, and B. Yoshua, "Generative adversarial nets." In Advances in neural information processing systems, pp. 2672-2680. 2014.

- [5] V. Premachandran and A.L. Yuille"Unsupervised Learning using Generative advasarial training and Clustring." In Proceedings of the ICLR,2017.

APPENDICES

SOURCE CODE

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#tqdm is used for showing progress bar in the long run
from tqdm import tqdm

#import backend from keras which resets all all state generated by keras
from keras import backend as K

#import mnist dataset from keras
from keras.datasets import mnist

#random number that are generated is going to be consistent across the
multiple rounds of this program
#this step is optional
np.random.seed(1000)

# load mnist dataset and save in 4 parts namely x_train, y_train, x_test,
y_test
(x_train, y_train), (x_test, y_test) = mnist.load_data()

#Normalization: images have each pixel have value from 0 to 255, in order
to normalize in the range -1 to +1, do the following

#o/p at this stage which will be input at later stage is sotred in X_train
X_train = (x_train.astype(np.float32) - 127.5) / 127.5

# print(X_train.shape) #this shap is 60k images, each have 28*28 single
channel images

#flattening the 28*28 images into 784
X_train = X_train.reshape(60000, 784)
print(X_train.shape)
```

```

#creating optimizer

from tensorflow.keras.optimizers import Adam

#optimizer used is Adam
adam = Adam(lr=0.0002, beta_1=0.5)

#starting creating generator first and then discriminator
from keras.models import Model, Sequential

#dropout to prevent overfitting, LeakyReLU is non linear activation function
from keras.layers import Input, Dense, Dropout, Flatten, LeakyReLU

#creating generator which is going to be sequential model
#random dimension
input_dim = 10*10
generator = Sequential()

#adding dense layer with 256 neurons and i/p dimension is some random dimension
generator.add(Dense(256, input_dim=input_dim))

#applying non linear activation function, leak for -ve is 20% of magnitude
generator.add(LeakyReLU(0.2))

generator.add(Dense(516)) #another dense layer
generator.add(LeakyReLU(0.2))

generator.add(Dense(1024)) #another dense layer
generator.add(LeakyReLU(0.2))

# dimensions of expected image is 784, and we know in mnist images are of 28*28 px, once we flatten that we'll get 784, activation function is tanh(o/p range is -1to+1),

generator.add(Dense(784, activation="tanh"))
generator.compile(loss="binary_crossentropy",
                  optimizer=adam)

generator.summary()

```

```

#building discriminator which is going to be sequential model

discriminator = Sequential()

#dim should be same from generator, i.e, 784
discriminator.add(Dense(1024, input_dim=784))
discriminator.add(LeakyReLU(0.2))

#to avoid overfitting

discriminator.add(Dropout(0.3))

discriminator.add(Dense(512))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))

discriminator.add(Dense(256))
discriminator.add(LeakyReLU(0.2))
discriminator.add(Dropout(0.3))

#o/p layer, 1 neuron helps in classification
discriminator.add(Dense(1, activation="sigmoid"))
discriminator.compile(loss="binary_crossentropy",
                      optimizer=adam)

discriminator.summary()

discriminator.trainable = False

ganInput = Input(shape=(input_dim, )) #tuple

x = generator(ganInput)

ganOutput = discriminator(x)

gan = Model(ganInput, ganOutput)
gan.compile(loss="binary_crossentropy",
            optimizer=adam)
gan.summary()

```

```

#combine both generator and discriminator in a single model i.e, gan,

#discriminator is not going to be trainable when we use GAN
discriminator.trainable = False

ganInput = Input(shape=(input_dim, )) #tuple

x = generator(ganInput)


#passing ganInput as i/p to discriminator
ganOutput = discriminator(x)

gan = Model(ganInput, ganOutput)
gan.compile(loss="binary_crossentropy",
            optimizer=adam)
gan.summary()

def generate_and_plot():
    num_examples = 100
    noise = np.random.normal(0, 1, size=[num_examples, input_dim])
    generated_images = generator.predict(noise)
    generated_images = generated_images.reshape(num_examples, 28, 28)

    plt.figure(figsize=(10, 10))
    for i in range(num_examples):
        plt.subplot(10, 10, i+1)
        plt.imshow(generated_images[i],
                  cmap="gray_r",
                  interpolation="nearest")
        plt.axis("off") #hiding the x and y axis
    plt.show()

def train(epochs=1, batch_size=128):
    m = X_train.shape[0]
    batch_count = m // batch_size

    generate_and_plot() #noisy images
    for e in range(epochs):
        print(f"Epoch: {e}")
        for _ in tqdm(range(batch_count)):

```

```

    #generate some noisy image
    noise = np.random.normal(0, 1, size=[batch_size, input_dim])
    generated_images = generator.predict(noise)
    real_images = X_train[np.random.randint(0,m, size=batch_size)]
    X = np.concatenate([real_images, generated_images])
    y_dis = np.zeros(2*batch_size)
    y_dis[:batch_size] = 0.9
    discriminator.trainable = True
    d_loss = discriminator.train_on_batch(X, y_dis)
    discriminator.trainable = False
    noise = np.random.normal(0, 1, size=[batch_size, input_dim])
    y_gan = np.ones(batch_size)
    gan.train_on_batch(noise, y_gan)

if e%10==0:
    generate_and_plot()

#defining the generate and plot function

def generate_and_plot():
    num_examples = 100
    noise = np.random.normal(0, 1, size=[num_examples, input_dim])
    generated_images = generator.predict(noise)
    generated_images = generated_images.reshape(num_examples, 28, 28)

    plt.figure(figsize=(10, 10))
    for i in range(num_examples):
        plt.subplot(10, 10, i+1)
        plt.imshow(generated_images[i],
                    cmap="gray_r",
                    interpolation="nearest")
        plt.axis("off") #hiding the x and y axis
    plt.show()

# function to start training the GAN
def train(epochs=1, batch_size=128):
    m = X_train.shape[0]
    batch_count = m // batch_size

    generate_and_plot() #noisy images
    for e in range(epochs):
        print(f"Epoch: {e}")

```

```

for _ in tqdm(range(batch_count)):
    #generate some noisy image
    noise = np.random.normal(0, 1, size=[batch_size, input_dim])
    generated_images = generator.predict(noise)
    real_images = X_train[np.random.randint(0,m, size=batch_size)]
#generates 128 random image from X_train

    #combining generate and real images
    X = np.concatenate([real_images, generated_images])
    y_dis = np.zeros(2*batch_size)
    y_dis[:batch_size] = 0.9
#one sided label smoothing, this also in preventing over fitting
    discriminator.trainable = True
    d_loss = discriminator.train_on_batch(X, y_dis)
    discriminator.trainable = False

    #training entire GAN which will just train the generator
    noise = np.random.normal(0, 1, size=[batch_size, input_dim])
    y_gan = np.ones(batch_size)
    gan.train_on_batch(noise, y_gan)

    if e%10==0: #after every 10 epochs, call generate_and_plot()
        generate_and_plot()

#train the GAN
train(50)

# final output after 50 epochs
generate_and_plot()

```

SCREENSHOTS

