Deep Learning Assignment

Name: Moturu Praveen Bhargav

Ph: +91 7396382592

Email id: moturu.praveenbhargav@gmail.com

OBJECTIVES

1. INTRODUCTION

What is GAN's ?

GAN stands for Generative Adversarial Network, which is a type of deep learning architecture made up of two neural networks, the generator and the discriminator. The generator produces new data samples, while the discriminator evaluates the authenticity of these generated samples.

The generator tries to create realistic samples that can deceive the discriminator, while the discriminator tries to correctly classify whether a sample is real or generated. This adversarial process continues until the generator produces samples that are indistinguishable from the real samples, and the discriminator can no longer differentiate between the two.

GANs have been successfully applied in a variety of domains, including image synthesis, text generation, and music composition, and have the potential to revolutionize many industries by generating novel and diverse content.

Applications of GAN’s:

- Image Synthesis
- Video Generation
- Style Transfer
- Text Generation
- Music Generation

The motto of the project to perform any one of the application such as “Image Synthesis”. Image Synthesis: GANs can generate realistic images by training on a dataset of real images, and then using the generator network to produce new images. This can be used in a variety of applications, such as generating new artwork, creating 3D models from 2D images, and even generating realistic images of non-existent objects.

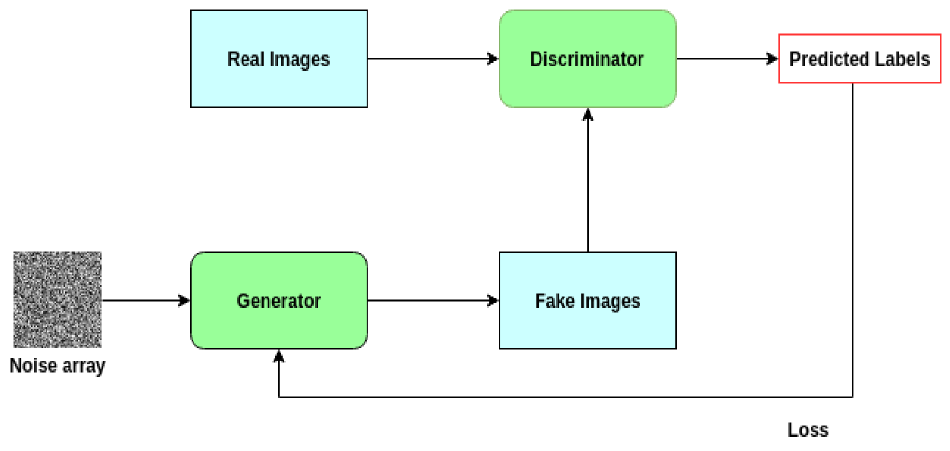


Fig. 1.s GAN working Flow-Chart

2. DATASET

CIFAR-10 dataset can be used as a training dataset (50,000) of image shape 32x32x3 for Generative Adversarial Networks (GANs) to generate realistic images in the 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck) that the dataset contains. In the context of GANs, the goal is to train a generator network that can produce synthetic images that are similar to the real images in the dataset.

In the case of CIFAR-10, the generator network is trained on a set of random noise vectors and tries to generate realistic-looking images that resemble the 10 classes of objects in the CIFAR-10 dataset. The discriminator network is trained to distinguish between the real CIFAR-10 images and the generated images from the generator network.

The generator network is trained to minimize the difference between the generated images and the real images in the CIFAR-10 dataset, while the discriminator network is trained to maximize the difference between the real and generated images. The training process continues until the generator network can generate synthetic images that are indistinguishable from real images by the discriminator network.

Once the GAN model is trained on the CIFAR-10 dataset, it can be used to generate new images in the 10 classes present in the dataset. These generated images can be used for various applications, including data augmentation, image synthesis, and image manipulation.

In summary, the CIFAR-10 dataset can be used as a training dataset for GANs to generate realistic images in the 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck) that the dataset contains, and this trained GAN model can be used to generate new images in these classes.

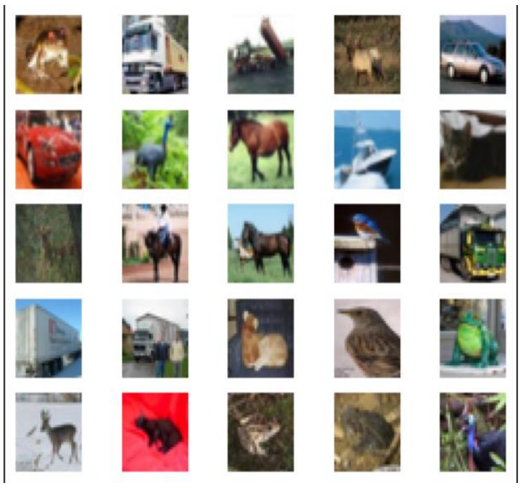


Fig. 2 Overview of Dataset

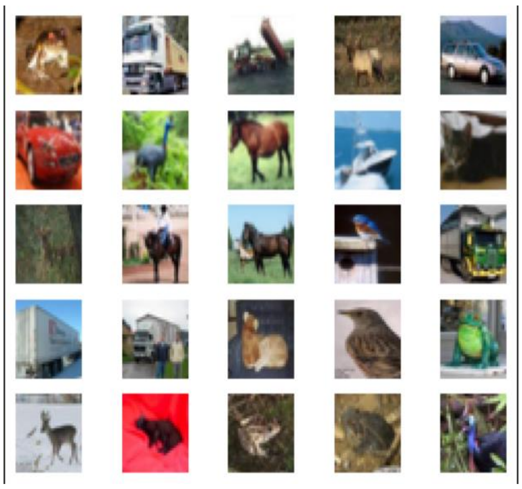
3. DATA PREPROCESSING

Data preprocessing techniques are an essential component of building GAN models, and there are several techniques used to preprocess data for GANs. Here are some common data preprocessing techniques used in GANs for this project:

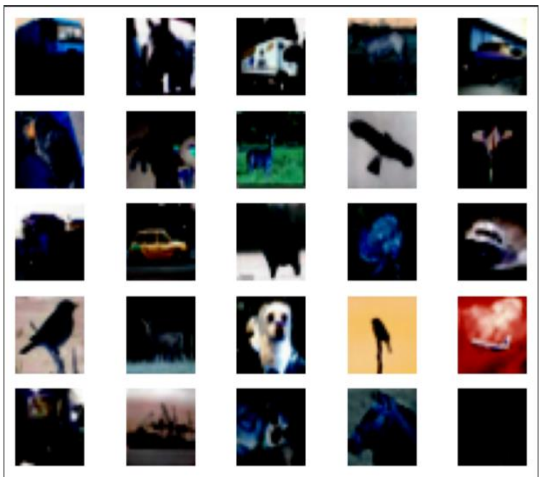
1. *Data normalization:* Normalization is a common preprocessing technique used to scale the data to a common range, usually between 0 and 1 or -1 and 1. Normalizing the data can help to improve the stability and convergence of the GAN model during training.

   The Formula used for Normalization(N) is :

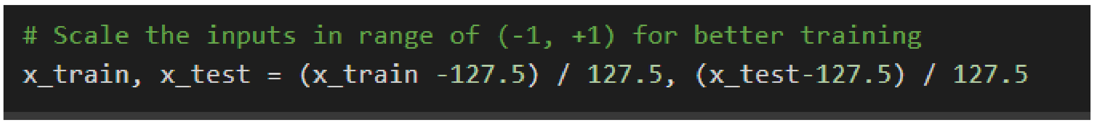
$$N(image) = \frac{[image\_arr\ ]-127.5}{127.5}$$



(a) Before Normalization

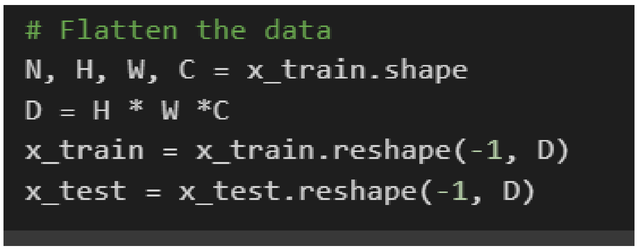


(b) After Normalization

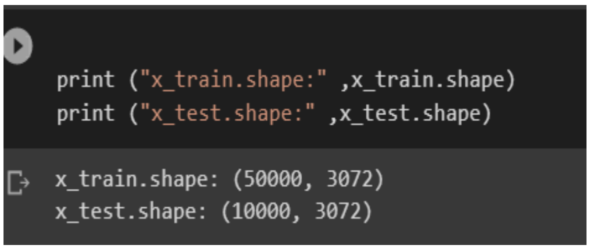


```
# Scale the inputs in range of (-1, +1) for better training
x_train, x_test = (x_train -127.5) / 127.5, (x_test-127.5) / 127.5
```

2. *Image resizing*: In some cases, images in the dataset may be of different sizes and aspect ratios. Cropping and resizing images to a fixed size and aspect ratio can help to improve the performance and stability of the GAN model.

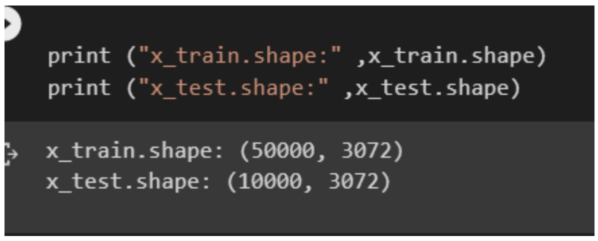


```
# Flatten the data
N, H, W, C = x_train.shape
D = H * W *C
x_train = x_train.reshape(-1, D)
x_test = x_test.reshape(-1, D)
```



```
print ("x_train.shape:" ,x_train.shape)
print ("x_test.shape:" ,x_test.shape)

x_train.shape: (50000, 3072)
x_test.shape: (10000, 3072)
```

a) Reshaping before



```
print ("x_train.shape:" ,x_train.shape)
print ("x_test.shape:" ,x_test.shape)

x_train.shape: (50000, 3072)
x_test.shape: (10000, 3072)
```

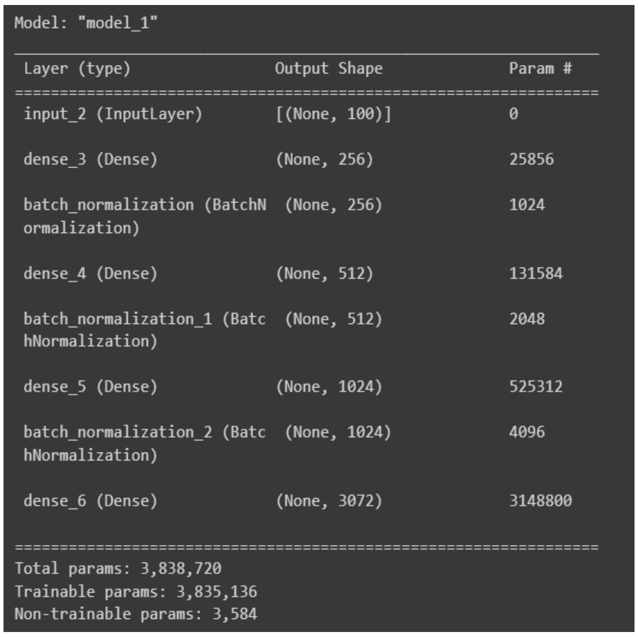
b) Reshaping After

4. MODEL DESIGN

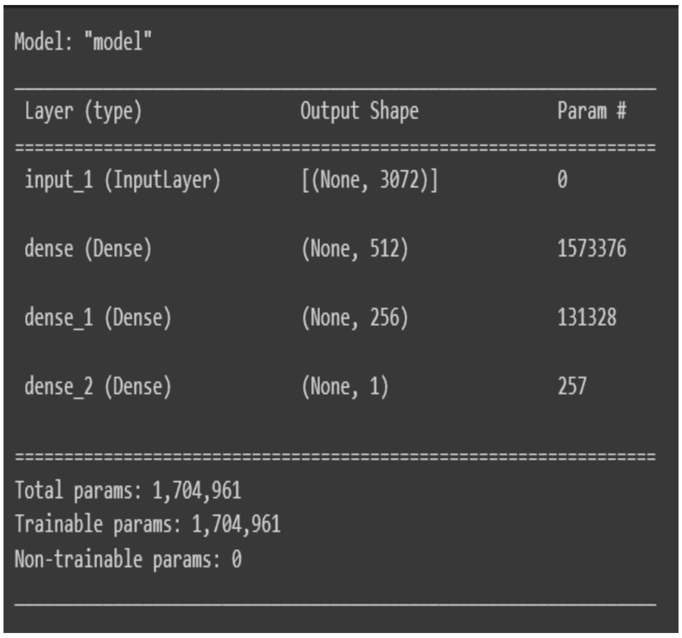
For this Project I have used two (2) different models for both Generator and Discriminator.

a) Model 1

Model 1 consists of a relatively simple generator model architecture that consists of several fully connected layers with LeakyReLU activation and batch normalization. Architectures of Model 1.



```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 100)]             0

 dense_3 (Dense)             (None, 256)               25856

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dense_4 (Dense)             (None, 512)               131584

 batch_normalization_1 (Batc  (None, 512)              2048
 hNormalization)

 dense_5 (Dense)             (None, 1024)              525312

 batch_normalization_2 (Batc  (None, 1024)             4096
 hNormalization)

 dense_6 (Dense)             (None, 3072)              3148800

=================================================================
Total params: 3,838,720
Trainable params: 3,835,136
Non-trainable params: 3,584
```



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 3072)]            0

 dense (Dense)               (None, 512)               1573376

 dense_1 (Dense)             (None, 256)               131328

 dense_2 (Dense)             (None, 1)                 257

=================================================================
Total params: 1,704,961
Trainable params: 1,704,961
Non-trainable params: 0
_________________________________________________________________
```

a) Generator Architecture b) Discriminator Architecture

Explanation of Generator:

- latent_dim: The dimensionality of the random noise vector input to the generator.
- Input(shape=(latent_dim,)): Creates an input tensor for the generator with the shape (None, latent_dim) where None represents the batch size.
- Dense(256, activation=LeakyReLU(alpha=0.2)): Adds a dense layer with 256 units to the generator model and applies LeakyReLU activation with a small negative slope (0.2) to the output.
- BatchNormalization(momentum=0.7): Applies batch normalization to the output of the previous layer to improve the stability and speed of training.
- x = Dense(512, activation=LeakyReLU(alpha=0.2)): Adds another dense layer with 512 units and applies LeakyReLU activation to the output.
- BatchNormalization(momentum=0.7)(x): Applies batch normalization again.
- Dense(1024, activation=LeakyReLU(alpha=0.2)): Adds another dense layer with 1024 units and applies LeakyReLU activation to the output.
- BatchNormalization(momentum=0.7): Applies batch normalization again.

- Dense(D, activation='tanh'): Adds a dense layer with D units (where D is the dimensionality of the output image) and applies hyperbolic tangent activation to the output. The tanh activation function scales the output to the range (-1, 1), which is appropriate for image data.
- model = Model(i, x): Creates a Keras model object with the input tensor i and output tensor x.

The generator takes random noise as input and outputs synthetic images that are intended to resemble real images from a particular dataset.

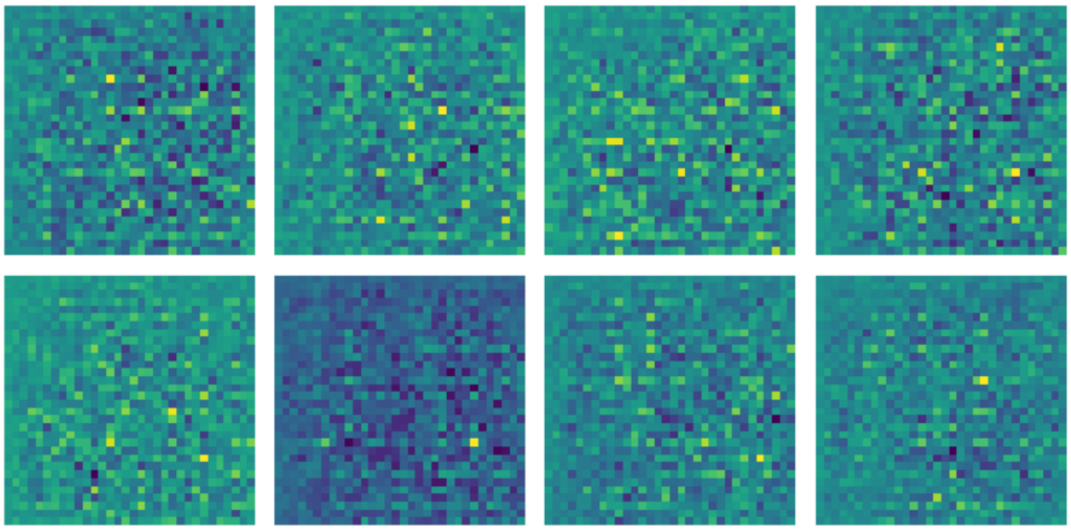


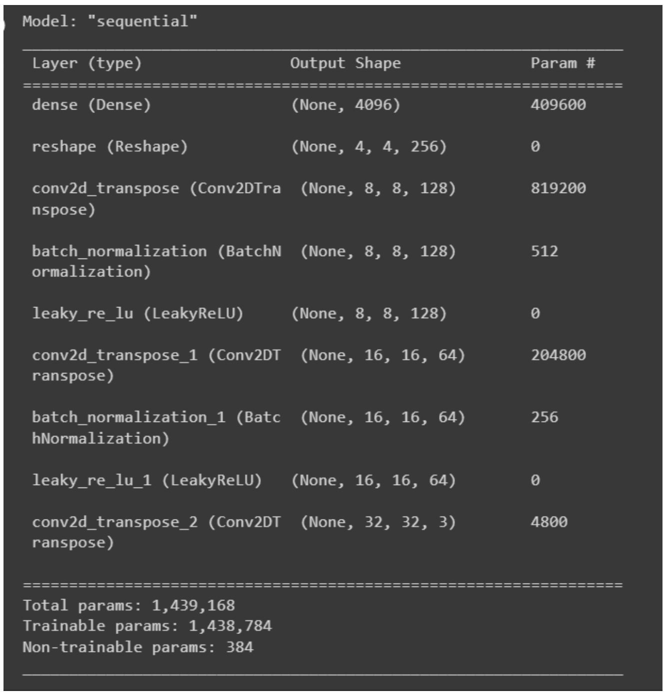
Fig. 3. Generator Takes Random Noises of images to output synthetic images of dimension size (8,100)

Explanation of Discriminator:

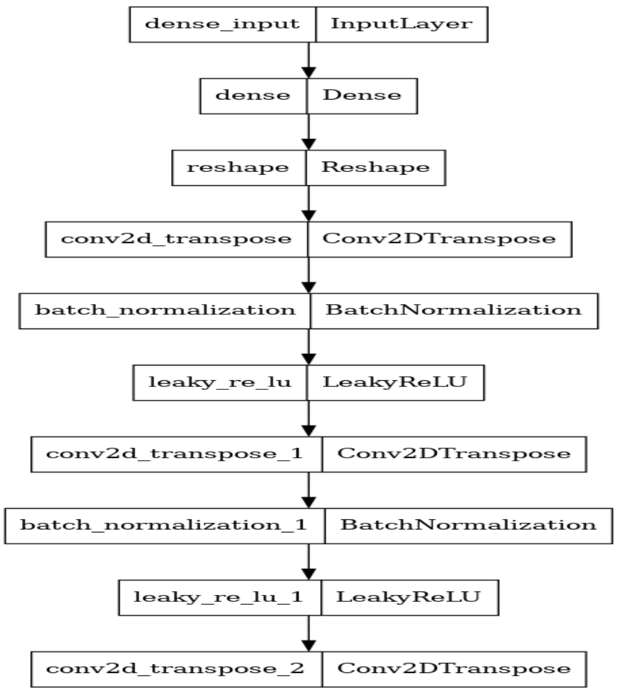
- **img_size**: The size of the input images to the discriminator.
- **Input(shape=(img_size,))**: Creates an input tensor for the discriminator with the shape **(None, img_size)** where **None** represents the batch size.
- **Dense(512, activation=LeakyReLU(alpha=0.2))**: Adds a dense layer with 512 units to the discriminator model and applies LeakyReLU activation with a small negative slope (0.2) to the output.
- **Dense(256, activation=LeakyReLU(alpha=0.2))**: Adds another dense layer with 256 units and applies LeakyReLU activation to the output.
- **Dense(1, activation='sigmoid')**: Adds a final dense layer with 1 unit and applies sigmoid activation to the output. The sigmoid activation function scales the output to the range (0, 1), which can be interpreted as the probability that the input image is real.

The discriminator takes an image as input and outputs a binary value indicating whether the image is real or fake (generated by the generator).

b) Model 2



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 4096)              409600

 reshape (Reshape)           (None, 4, 4, 256)         0

 conv2d_transpose (Conv2DTra  (None, 8, 8, 128)        819200
 nspose)

 batch_normalization (BatchN  (None, 8, 8, 128)        512
 ormalization)

 leaky_re_lu (LeakyReLU)     (None, 8, 8, 128)         0

 conv2d_transpose_1 (Conv2DT  (None, 16, 16, 64)       204800
 ranspose)

 batch_normalization_1 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 16, 16, 64)        0

 conv2d_transpose_2 (Conv2DT  (None, 32, 32, 3)        4800
 ranspose)

=================================================================
Total params: 1,439,168
Trainable params: 1,438,784
Non-trainable params: 384
_________________________________________________________________
```

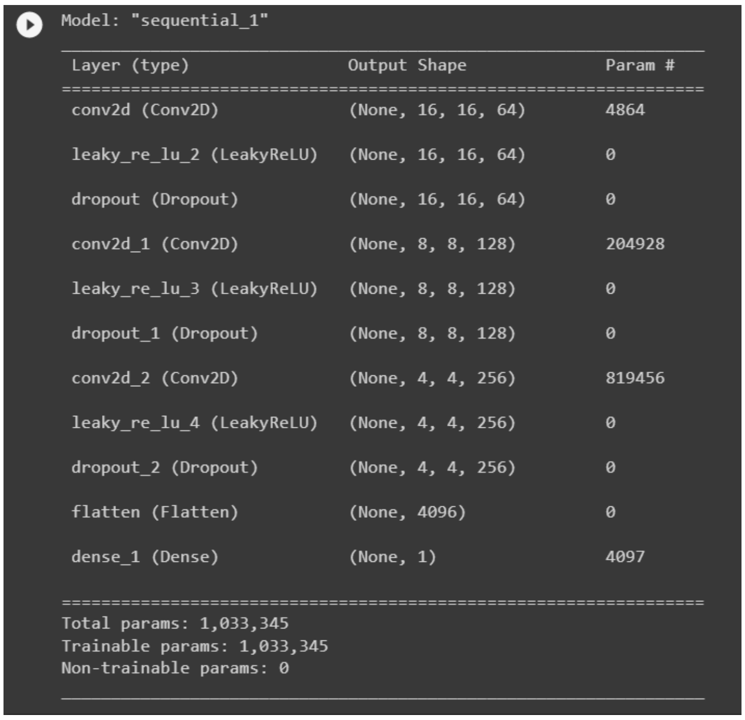

dense_input | InputLayer
dense | Dense
reshape | Reshape
conv2d_transpose | Conv2DTranspose
batch_normalization | BatchNormalization
leaky_re_lu | LeakyReLU
conv2d_transpose_1 | Conv2DTranspose
batch_normalization_1 | BatchNormalization
leaky_re_lu_1 | LeakyReLU
conv2d_transpose_2 | Conv2DTranspose

a) Generator Architecture of Model 2

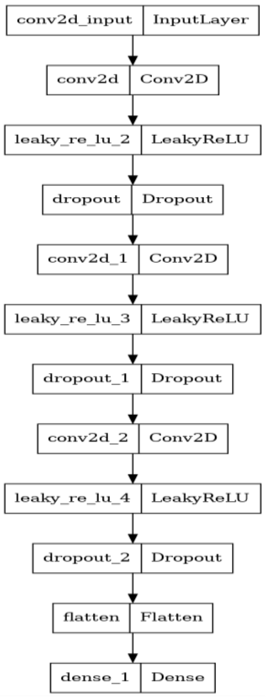
Explanation of Generator

- **model.add(Dense(4*4*256, use_bias=False, input_shape=(100,)))**: Adds a fully connected layer to the generator model with 4x4x256=4096 units, no bias term, and an input shape of (100,) for the random noise vector. The purpose of this layer is to transform the random noise vector into a higher-dimensional representation that can be reshaped into a small image.
- **model.add(Reshape((4, 4, 256)))**: Reshapes the output of the previous layer into a 4x4x256 tensor, which can be interpreted as a small feature map.
- **model.add(Conv2DTranspose(128, (5, 5), strides=(2, 2), padding='same', use_bias=False))**: Adds a transpose convolutional layer to the generator model with 128 filters, a kernel size of 5x5, a stride of (2,2) to upsample the feature map, 'same' padding to preserve the spatial dimensions, and no bias term.
- **model.add(BatchNormalization())**: Adds a batch normalization layer to the generator model, which normalizes the activations of the previous layer along the channel axis to improve the stability and convergence of the training process.
- **model.add(LeakyReLU(alpha=0.2))**: Adds a leaky rectified linear unit activation function to the generator model, which introduces a small negative slope to the negative part of the activation to avoid dead neurons and improve the expressive power of the model.
- **model.add(Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))**: Adds another transpose convolutional layer to the generator model with 64 filters, a kernel size of 5x5, a stride of (2,2) to upsample the feature map further, 'same' padding, and no bias term.

- **model.add(BatchNormalization())**: Adds another batch normalization layer to the generator model.
- **model.add(LeakyReLU(alpha=0.2))**: Adds another leaky ReLU activation function to the generator model.
- **model.add(Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))**: Adds a final transpose convolutional layer to the generator model with 3 filters (corresponding to the RGB channels of the image), a kernel size of 5x5, a stride of (2,2), 'same' padding, no bias term, and a hyperbolic tangent activation function to scale the output to the range (-1, 1), which corresponds to the pixel values of the image.



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 16, 16, 64)        4864

 leaky_re_lu_2 (LeakyReLU)   (None, 16, 16, 64)        0

 dropout (Dropout)           (None, 16, 16, 64)        0

 conv2d_1 (Conv2D)           (None, 8, 8, 128)         204928

 leaky_re_lu_3 (LeakyReLU)   (None, 8, 8, 128)         0

 dropout_1 (Dropout)         (None, 8, 8, 128)         0

 conv2d_2 (Conv2D)           (None, 4, 4, 256)         819456

 leaky_re_lu_4 (LeakyReLU)   (None, 4, 4, 256)         0

 dropout_2 (Dropout)         (None, 4, 4, 256)         0

 flatten (Flatten)           (None, 4096)              0

 dense_1 (Dense)             (None, 1)                 4097

=================================================================
Total params: 1,033,345
Trainable params: 1,033,345
Non-trainable params: 0
_________________________________________________________________
```



b) Discriminator Architecture of Model 2

Explanation of Discriminator Architecture

The discriminator model consists of convolutional layers with increasing numbers of filters to extract features from the input image. The **Conv2D** layers use 5x5 filters with a stride of 2 to downsample the input image. The **LeakyReLU** activation function is used to introduce non-linearity in the output of the convolutional layers. **Dropout** is used to prevent overfitting by randomly dropping some of the neurons during training.

The **Flatten()** layer is used to flatten the output from the convolutional layers to a 1D vector. The **Dense** layer with a single neuron and sigmoid activation function is used to output a probability score indicating whether the input image is real or fake.

5. MODEL TRAINING

a) Hyper parameters considered for Model 1.

- Batch size: 64
- Number of epochs: 20000
- Latent dimension: 100
- Optimizer : Adam
- Learning rate: 0.0002
- Alpha : 0.5
- Weight initialization: Random Weights [batch size, Latent_dimension]
- Training time: 1hr :15 min

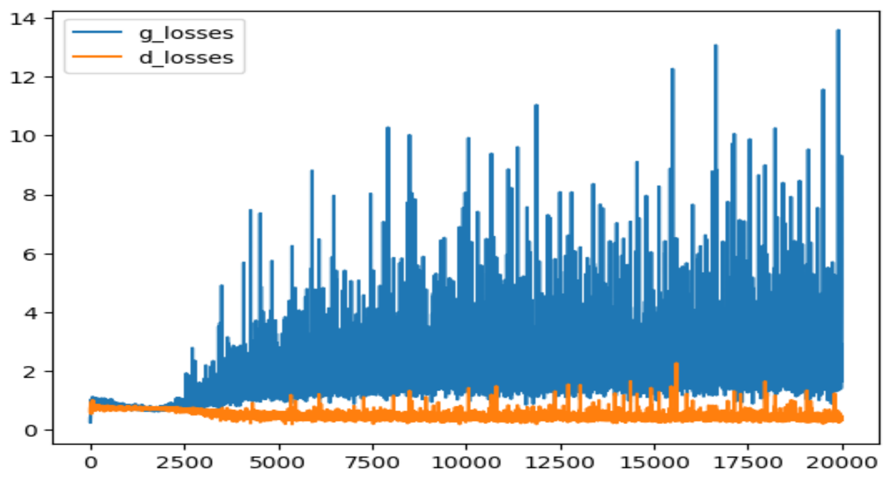


Fig. 4. Losses of Both Generator and Discriminator Losses

Adam: An adaptive learning rate optimizer that uses a combination of gradient information and momentum to update the weights. The best optimizer for a particular task depends on several factors, such as the size of the dataset, the complexity of the model, and the nature of the problem being solved. In this project we have a large dataset of 50000, Layers exceeds 7 which is little complex. In practice, Adam is a popular choice for many deep learning tasks due to its ease of use and effectiveness in a wide range of scenarios.

b) Hyper parameters considered for Model 2

- Batch size: 256
- Number of epochs: 1,000
- Latent dimension: 100
- Optimizer : Adam
- Learning rate: 0.0001
- Weight initialization: Random Weights [batch size, Latent_dimension]

Training time: 5 hrs :35 min

6. MODEL EVALUATION

Qualitative evaluation by visually inspecting a sample of generated images from both the **Model 1 and Model 2.**

a) Results of Model 1

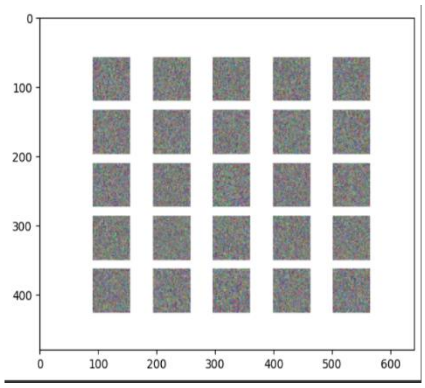


Fig. 5. Noise Images input to Image Generator and trained for 20000 epochs for a very simple architecture viz Model 1.

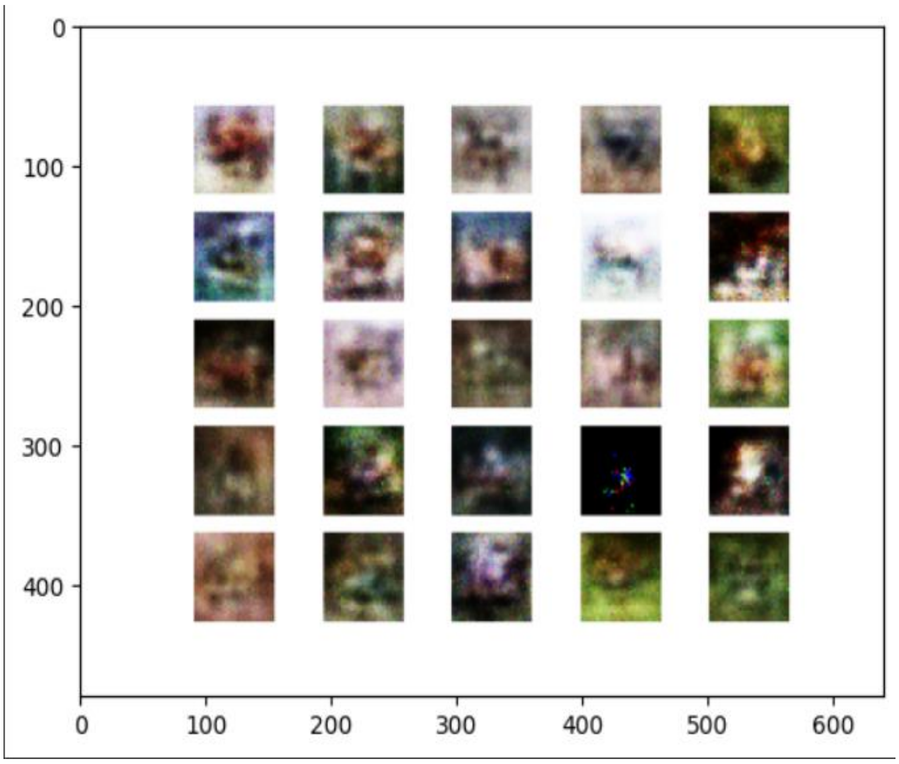


Fig. 6. Generator takes random noise as input and outputs synthetic images that are intended to resemble real images from a particular dataset.

b) Results of Model 2

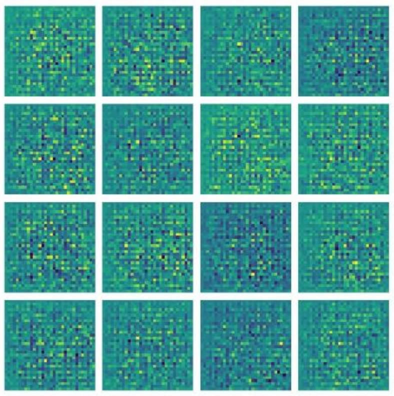


Fig. 7. Noise Images input to Image Generator and trained for 1000 epochs for a very simple architecture viz Model 2.

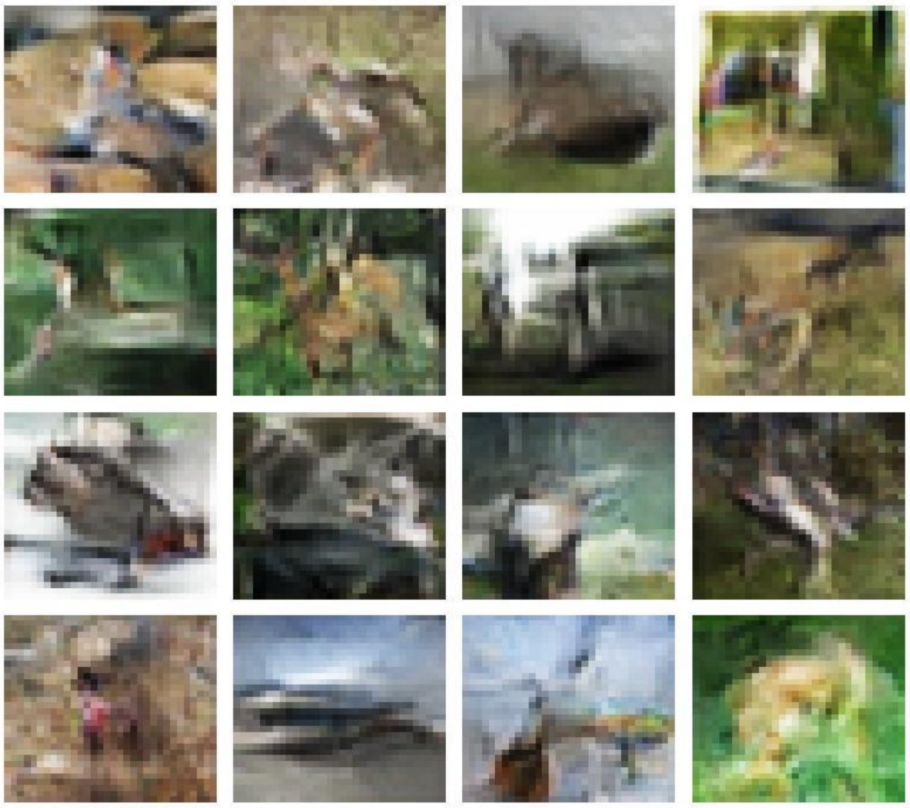


Fig. 8. Generator takes random noise as input and outputs synthetic images that are intended to resemble real images from a particular dataset.

7. MODEL DEPLOYMENT

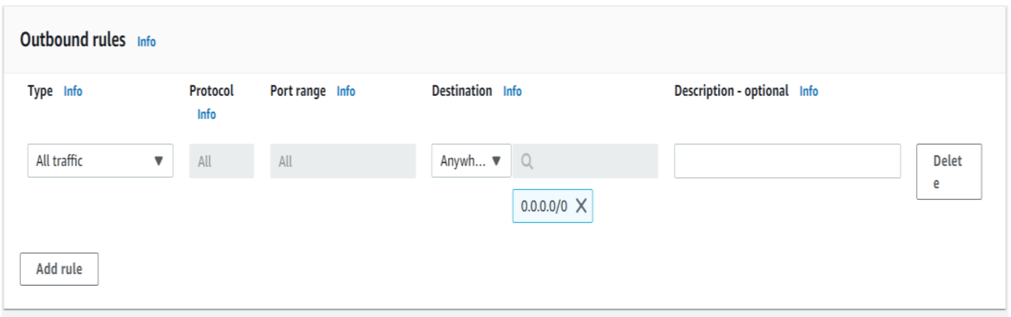
Tools used for Deployment :

- AWS EC2
- WinSCP
- PuTTYgen
- PuTTY

Step1: Launch Instance

Requirements:

1. Instance t2.micro
2. Hard Disk : 30GiB
3. RAM : 2 GB
4. OS : ubuntu
5. Key : .pem (create key for instance)
6. Security Group :
    a. Set Type "All Traffic"
    b. Set Destination " Access Anywhere (0.0.0.0/0)"

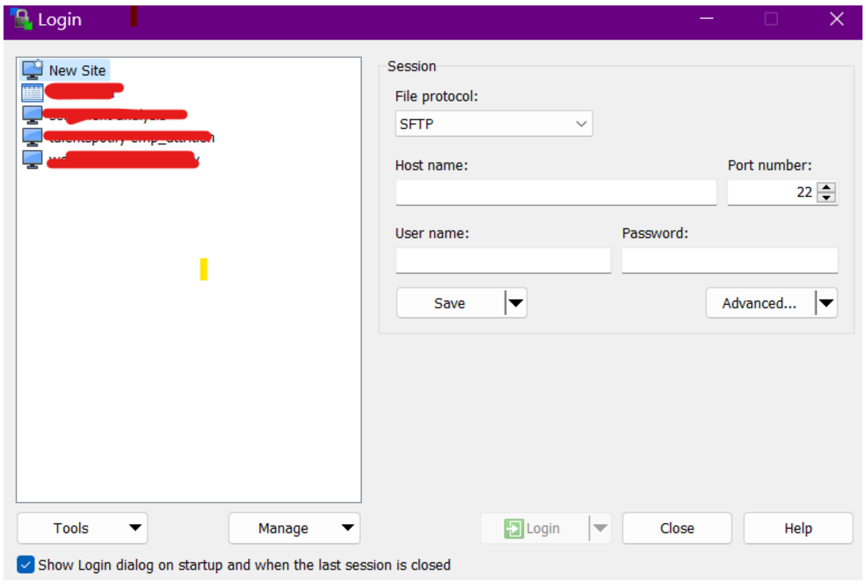

Outbound rules Info

Type	Protocol	Port range	Destination	Description - optional
All traffic	All	All	Anywh... 0.0.0.0/0	

Fig. 9 Security Group modifications in AWS

Step2: Start WinSCP

1. Enter Host Name, User Name
2. Copy the SSH (Secure Shell) Address and paste in the Host Name
3. Open PuTTYgen , load the .pem key file and generate .ppk private key, save it.
4. Insert .ppk in the password



File protocol: SFTP

Host name:

Port number: 22

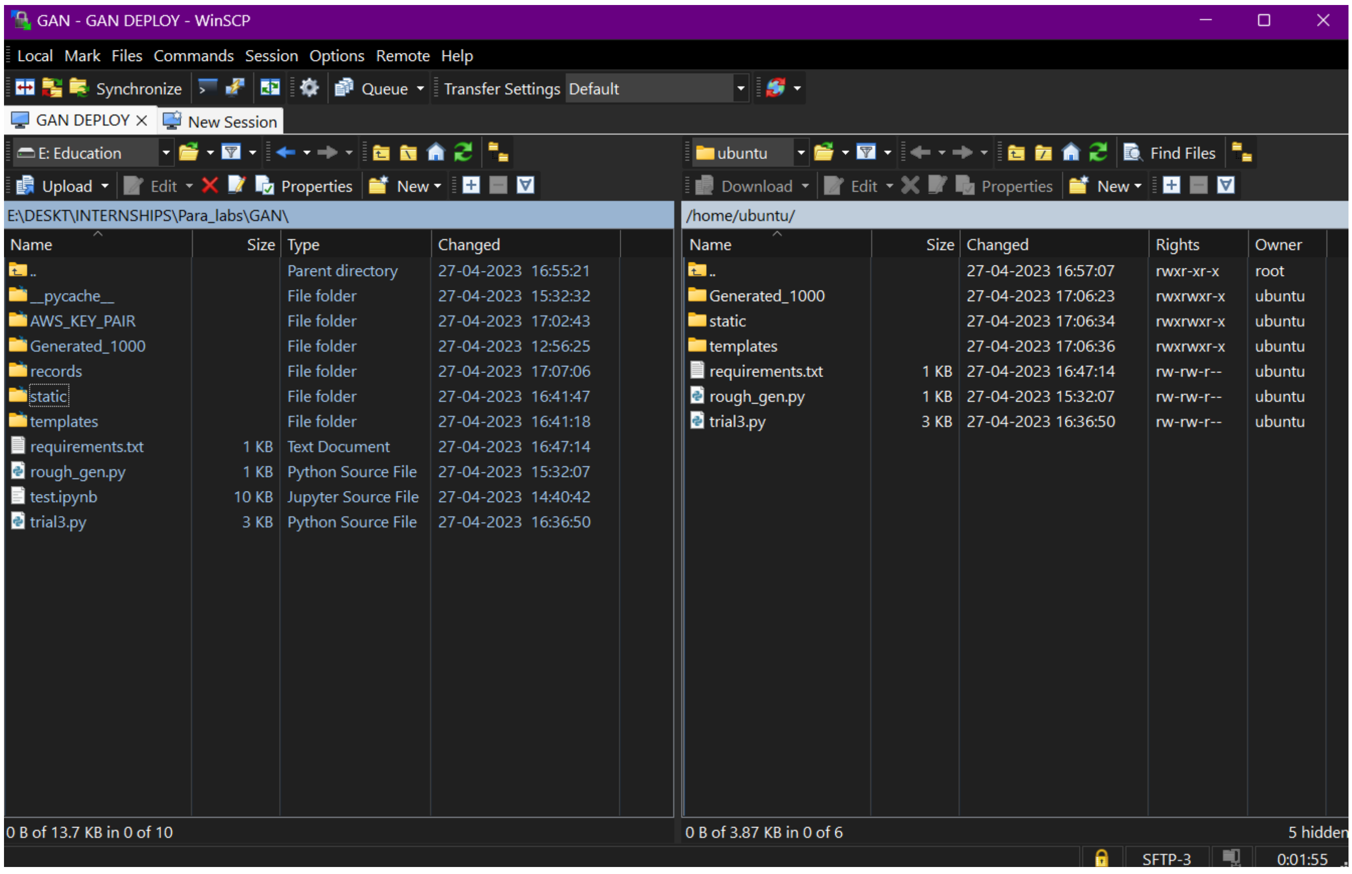
User name:

Password:

Show Login dialog on startup and when the last session is closed

Fig. 10. Interface of WinSCP

Step3: Insert all the files in WinSCP


GAN - GAN DEPLOY - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
GAN DEPLOY
New Session
E: Education
ubuntu
Find Files
Upload
Edit
Properties
New
Download

E:\DESKT\INTERNSHIPS\Para_labs\GAN\

Name	Size	Type	Changed
..		Parent directory	27-04-2023 16:55:21
__pycache__		File folder	27-04-2023 15:32:32
AWS_KEY_PAIR		File folder	27-04-2023 17:02:43
Generated_1000		File folder	27-04-2023 12:56:25
records		File folder	27-04-2023 17:07:06
static		File folder	27-04-2023 16:41:47
templates		File folder	27-04-2023 16:41:18
requirements.txt	1 KB	Text Document	27-04-2023 16:47:14
rough_gen.py	1 KB	Python Source File	27-04-2023 15:32:07
test.ipynb	10 KB	Jupyter Source File	27-04-2023 14:40:42
trial3.py	3 KB	Python Source File	27-04-2023 16:36:50

/home/ubuntu/

Name	Size	Changed	Rights	Owner
..		27-04-2023 16:57:07	rwxr-xr-x	root
Generated_1000		27-04-2023 17:06:23	rwxrwxr-x	ubuntu
static		27-04-2023 17:06:34	rwxrwxr-x	ubuntu
templates		27-04-2023 17:06:36	rwxrwxr-x	ubuntu
requirements.txt	1 KB	27-04-2023 16:47:14	rw-rw-r--	ubuntu
rough_gen.py	1 KB	27-04-2023 15:32:07	rw-rw-r--	ubuntu
trial3.py	3 KB	27-04-2023 16:36:50	rw-rw-r--	ubuntu

0 B of 13.7 KB in 0 of 10

0 B of 3.87 KB in 0 of 6 5 hidden

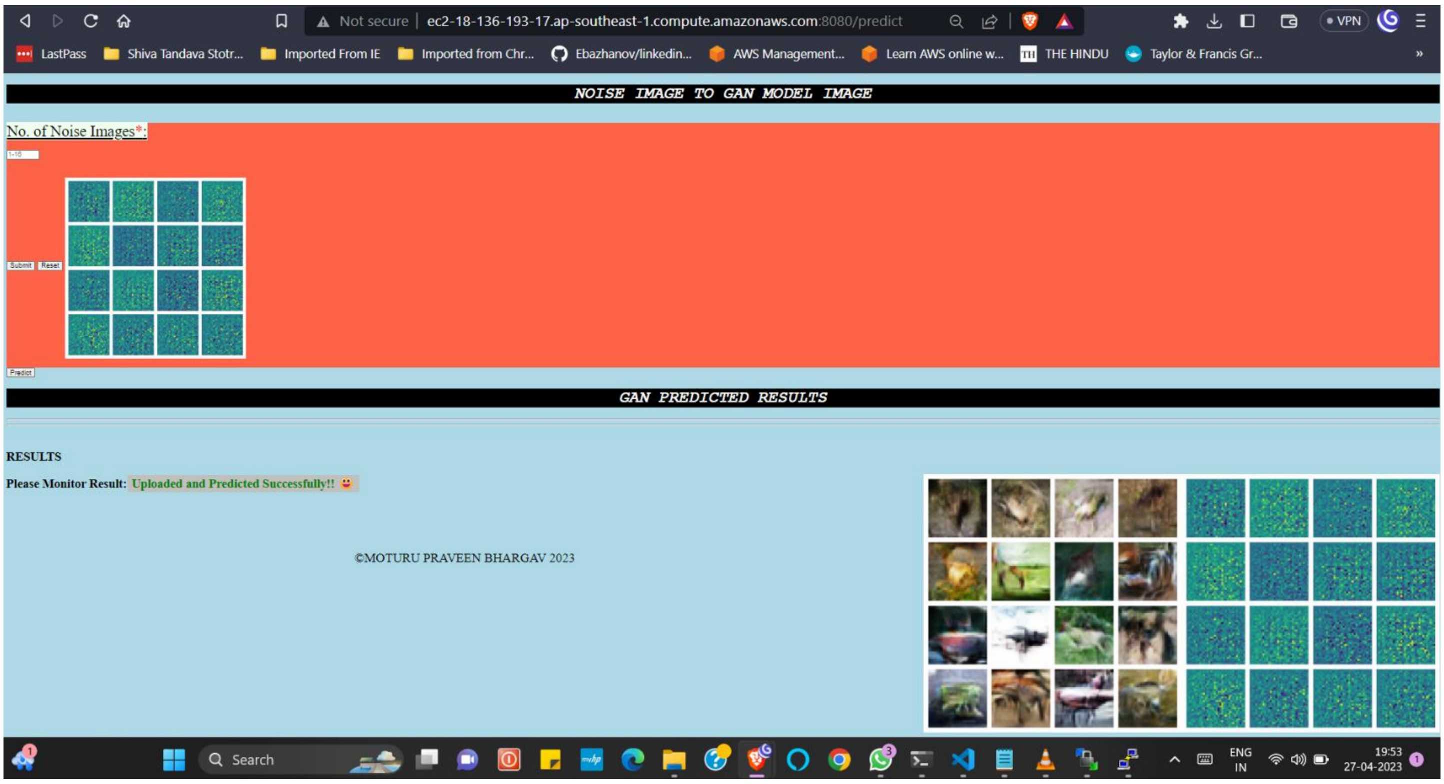
SFTP-3 0:01:55

Fig. 11. Open WinSCP and shift all the files which are mandatory for model deployment

Step4: Open PuTTY

1. Enter the SSH, .ppk key file to login into the EC2 instance
2. Install Python and Pip with "*sudo apt-get update && sudo apt-get install python3-pip*"
3. Install requirements.txt with "pip3 install -r requirements.txt"
4. Run Python Application, "*python app.py*".
5. Note the "Host Address:Port_Number/" and share to the public & colleagues.
   Eg: http://ec2-18-136-193-17.ap-southeast-1.compute.amazonaws.com:8080/

APPLICATION:


Not secure | ec2-18-136-193-17.ap-southeast-1.compute.amazonaws.com:8080/predict
NOISE IMAGE TO GAN MODEL IMAGE
No. of Noise Images*:
Submit
Reset
Predict
GAN PREDICTED RESULTS

RESULTS

Please Monitor Result: Uploaded and Predicted Successfully!!

©MOTURU PRAVEEN BHARGAV 2023

Fig. 12. The above deployed Application, we can see the URL to access the web application thought the world. We need to pass the No. of noises to synthesize by the generator from 1-16. Click 'Submit'. Then click "Predict" for the result images generated by the model.

8. OVERVIEW OF RESOURCES

Technologies Used:

- Python
- TensorFlow
- Visual Studio

- Google Colaboratory
- Kaggle Notebook
- Flask
- RESTAPI
- AWS

9. CODE

Please find the below GitHub Link to view the code and for also accessibility.

GitHub Link: https://github.com/MOTURUPRAVEENBHARGAV/GAN-s-using-Tensorflow.git

*************** THE END **************