Date of Performance :

Date of Submission :

EXPERIMENT NUMBER: 8

Aim: **Implement a GAN model for Image generation or video generation.**

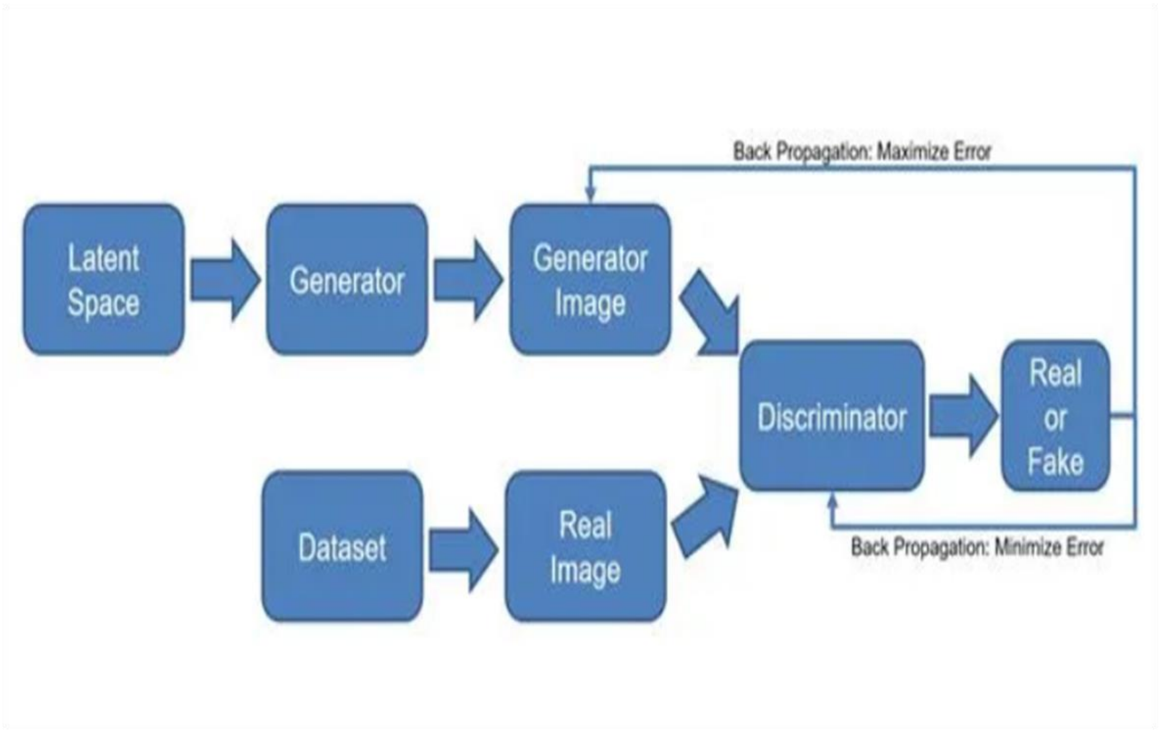
Objective:

Students will be gain familiarity with recent trends and applications of Deep Learning.

Software Used : Python

Theory:

- Generative Adversarial Network (GAN)



●The two neural networks that make up a GAN are referred to as the *generator* and the *discriminator*.

●The goal of the generator is to artificially manufacture outputs that could easily be mistaken for real data.

•The goal of the discriminator is to identify which of the outputs it receives have been artificially created.

●Generative models create their own training data. While the generator is trained to produce false data, the discriminator network is taught to distinguish between the generator's manufactured data and true examples.

• If the discriminator rapidly recognizes the fake data that the generator produces -- such as an image that isn't a human face -- the generator suffers a penalty.

• As the feedback_loop between the adversarial networks continues, the generator will begin to produce higher-quality and more believable output and the discriminator will become better at flagging data that has been artificially created.

Algorithm:

1) Load BigGAN generator module from TF Hub
2) sample truncated normal distribution based on seed and truncation parameter
3) using vectors of noise seeds and category labels, generate images
4) Create a TensorFlow session and initialize variables
5) Create video or images of interpolated BigGAN generator samples

Program:

```
# basics
import io
import os
import numpy as np

# deep learning
from scipy.stats import truncnorm
import tensorflow as tf
import tensorflow_hub as hub

# visualization
from IPython.core.display import HTML
#!pip install imageio
import imageio
import base64

# check that tensorflow GPU is enabled
tf.test.gpu_device_name() # returns empty string if using CPU
!pip install tensorflow==2.5.0
# comment out the TF Hub module path you would like to use
# module_path = 'https://tfhub.dev/deepmind/biggan-128/1'  # 128x128 BigGAN
# module_path = 'https://tfhub.dev/deepmind/biggan-256/1'  # 256x256 BigGAN
module_path = 'https://tfhub.dev/deepmind/biggan-512/1'  # 512x512 BigGAN
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()
tf.compat.v1.reset_default_graph()
##tf.reset_default_graph()
print('Loading BigGAN module from:', module_path)
module = hub.Module(module_path)
inputs = {k: tf.placeholder(v.dtype, v.get_shape().as_list(), k)
          for k, v in module.get_input_info_dict().items()}
output = module(inputs)
input_z = inputs['z']
input_y = inputs['y']
input_trunc = inputs['truncation']

dim_z = input_z.shape.as_list()[1]
vocab_size = input_y.shape.as_list()[1]
```

```
# sample truncated normal distribution based on seed and truncation parameter
def truncated_z_sample(truncation=1., seed=None):
    state = None if seed is None else np.random.RandomState(seed)
    values = truncnorm.rvs(-2, 2, size=(1, dim_z), random_state=state)
    return truncation * values

# convert `index` value to a vector of all zeros except for a 1 at `index`
def one_hot(index, vocab_size=vocab_size):
    index = np.asarray(index)
    if len(index.shape) == 0: # when it's a scale convert to a vector of size 1
        index = np.asarray([index])
    assert len(index.shape) == 1
    num = index.shape[0]
    output = np.zeros((num, vocab_size), dtype=np.float32)
    output[np.arange(num), index] = 1
    return output

def one_hot_if_needed(label, vocab_size=vocab_size):
    label = np.asarray(label)
    if len(label.shape) <= 1:
        label = one_hot(label, vocab_size)
    assert len(label.shape) == 2
    return label

# using vectors of noise seeds and category labels, generate images
def sample(sess, noise, label, truncation=1., batch_size=8,
vocab_size=vocab_size):
    noise = np.asarray(noise)
    label = np.asarray(label)
    num = noise.shape[0]
    if len(label.shape) == 0:
        label = np.asarray([label] * num)
    if label.shape[0] != num:
        raise ValueError('Got # noise samples ({}) != # label samples ({})'
                         .format(noise.shape[0], label.shape[0]))
    label = one_hot_if_needed(label, vocab_size)
    ims = []
    for batch_start in range(0, num, batch_size):
        s = slice(batch_start, min(num, batch_start + batch_size))
        feed_dict = {input_z: noise[s], input_y: label[s], input_trunc:
truncation}
```

```
        ims.append(sess.run(output, feed_dict=feed_dict))
    ims = np.concatenate(ims, axis=0)

    assert ims.shape[0] == num
    ims = np.clip(((ims + 1) / 2.0) * 256, 0, 255)
    ims = np.uint8(ims)
    return ims

def interpolate(a, b, num_interps):
    alphas = np.linspace(0, 1, num_interps)
    assert a.shape == b.shape, 'A and B must have the same shape to
interpolate.'
    return np.array([(1-x)*a + x*b for x in alphas])

def interpolate_and_shape(a, b, steps):
    interps = interpolate(a, b, steps)
    return (interps.transpose(1, 0, *range(2,
len(interps.shape))).reshape(steps, -1))
initializer = tf.global_variables_initializer()
sess = tf.Session()
sess.run(initializer)
# category options: https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a
category = 947 # mushroom

# important parameter that controls how much variation there is
truncation = 0.2 # reasonable range: [0.02, 1]

seed_count = 10
clip_secs = 36

seed_step = int(100 / seed_count)
interp_frames = int(clip_secs * 30 / seed_count)  # interpolation frames

cat1 = category
cat2 = category
all_imgs = []

for i in range(seed_count):
    seed1 = i * seed_step # good range for seed is [0, 100]
    seed2 = ((i+1) % seed_count) * seed_step
    z1, z2 = [truncated_z_sample(truncation, seed) for seed in [seed1, seed2]]
    y1, y2 = [one_hot([category]) for category in [cat1, cat2]]

    z_interp = interpolate_and_shape(z1, z2, interp_frames)
    y_interp = interpolate_and_shape(y1, y2, interp_frames)


    imgs = sample(sess, z_interp, y_interp, truncation=truncation)

    all_imgs.extend(imgs[:-1])

# save the video for displaying in the next cell, this is way more space
efficient than the gif animation
imageio.mimsave('gan.mp4', all_imgs, fps=30)
%%HTML
<video autoplay loop>
  <source src="gan.mp4" type="video/mp4">
</video>
```

Output:

Conclusion/Outcome:

Thus, we have implemented GAN model for Video generation

Marks & Signature:

R1 (4 Marks)	R2 (4 Marks)	R3 (4 Marks)	R4 (3 Mark)	Total (15 Marks)	Signature