USN NUMBER:1RVU22CSE018
NAME:Akshay B

Ex No: 6.5 Date: 18-09-2024	Lab 6.4: Understanding General Adversarial Networks

Objective:

The goal of this lab is to develop a General Adversarial Network model to create new images. An GAN is a type of neural network used for generating images by training two new models that are trained simultaneously by a process that is adversarial. A generator learns to create new images that look real while a discriminator learns to tell real images apart from the fake images.

Descriptions:

The model architecture comprises a discriminator and a generator. During training the generator becomes progressively better at generating new content that looks real while the discriminator becomes better at telling apart the images that are not real from the images that are real.The process is started by images that are noise and progressively resemble hand written digits.

Steps to Build the Model:

Import Libraries:

Import necessary libraries such as TensorFlow, TensorFlow Datasets, NumPy, and Matplotlib.

Data Preparation:

Load the MNIST dataset using tensorflow_datasets.
Reshape the image to 28x28 and normalize it

Define Generator:

USN NUMBER:1RVU22CSE018
NAME:Akshay B

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 12544)	1,254,400
batch_normalization (BatchNormalization)	(None, 12544)	50,176
leaky_re_lu (LeakyReLU)	(None, 12544)	0
reshape (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose (Conv2DTranspose)	(None, 7, 7, 128)	819,200
batch_normalization_1 (BatchNormalization)	(None, 7, 7, 128)	512
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 14, 14, 64)	204,800
batch_normalization_2 (BatchNormalization)	(None, 14, 14, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 28, 28, 1)	1,600

The generator uses upsampling layers to produce an image from random noise. Start with a Dense layer that takes noise as input, then upsamples it several times until you reach the desired image size of 28x28x1

Define Discriminator:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	1,664
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	204,928
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)	6,273

The discriminator uses 3 convolution layers followed by activation function of LeakyReLU and a dropout layer followed by a Dense layer at the end.

USN NUMBER:1RVU22CSE018
NAME:Akshay B

Define Discriminator Loss:

```
def discriminator_loss(real_output, fake_output):
    # START YOUR CODE HERE
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    # END YOUR CODE HERE
    return total_loss
```

It compares the discriminator's predictions on real images to an array of 1s, and the discriminator's predictions on fake (generated) images to an array of 0s

Define Generator Loss:

```
def generator_loss(fake_output):
    # START YOUR CODE HERE
    return cross_entropy(tf.ones_like(fake_output), fake_output)
    # END YOUR CODE HERE
```

The generator's loss quantifies how well it was able to trick the discriminator by comparing discriminators decisions on the generated images to an array of 1s.

Define Optimizer:

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

Save Checkpoints

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint                                                    =
tf.train.Checkpoint(generator_optimizer=generator_optimizer,

discriminator_optimizer=discriminator_optimizer,
                                    generator=generator,
                                    discriminator=discriminator)
```

This is use to save checkpoints of the training and is helpful in case a long running training task is interrupted.

Training of model:

USN NUMBER:1RVU22CSE018
NAME:Akshay B

```python
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

The training loop begins with generator receiving a random seed as input. That seed is used to produce an image. The discriminator is then used to classify real images (drawn from the training set) and fakes images (produced by the generator). The loss is calculated for each of these models, and the gradients are used to update the generator and discriminator.

Generate Images:

```python
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)

  fig = plt.figure(figsize=(4, 4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

We generate images using the model trained and plot them

Traing the model:

```python
train(train_dataset, EPOCHS)
```

The train model function is used to train the generator and the discriminator. We train it for 50 epochs

GitHubLink:
https://github.com/Akb-25/Foundations-of-Deep-Learning/blob/main/dcgan.ipynb