USN NUMBER: 1RVU22CSE128
NAME: Reethu RG Thota

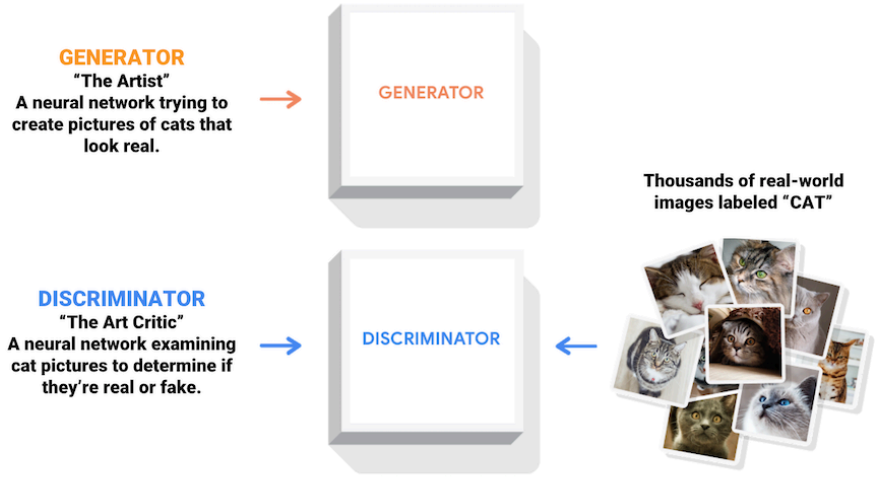
Ex No: 7 **Date:** 18/09/24	**Deep Convolutional Generative Adversarial Network (DCGAN)**

Objective

To train a Deep Convolutional Generative Adversarial Network (DCGAN) to generate images of handwritten digits. The model leverages adversarial training, where a generator learns to create realistic images, and a discriminator attempts to differentiate between real and generated images.

Descriptions

In this experiment, we aim to implement a DCGAN to tackle the problem of image generation. Generative Adversarial Networks (GANs) consist of two neural networks—***the generator and the discriminator***—competing against each other in a **zero-sum game**. The generator learns to create new images by generating data that mimics the distribution of the training dataset. The discriminator, on the other hand, classifies images as either real (from the training set) or fake (from the generator). ***Through adversarial training, the generator improves at creating realistic images, while the discriminator improves at identifying real versus fake images.***



DCGAN is an advanced type of GAN where convolutional layers are used for image synthesis. It helps generate more detailed and realistic images by learning from the spatial hierarchy of image features.

USN NUMBER: 1RVU22CSE128
NAME: Reethu RG Thota

Model

The model consists of two main parts:

- **Generator**: This model takes random noise as input and generates an image. It uses a series of transposed convolutions to upsample the noise into a realistic image.
- **Discriminator**: This model takes an image (either real or generated) as input and classifies it as real or fake. It uses convolutional layers to learn the spatial features of real vs. generated images.

The key components of the model include:

1. **Input Layer (Noise)**: The generator takes random noise as input to create images.
2. **Convolutional Layers**: Both models use convolutional layers for feature extraction (in the discriminator) and image synthesis (in the generator).
3. **Activation Functions**: Leaky ReLU is used in the discriminator to handle negative inputs, while ReLU and Tanh are used in the generator to produce images with pixel values between -1 and 1.

Building the parts of algorithm

The steps involved in building the CNN with transfer learning are:

1. **Install Dependencies:** Install the necessary libraries, including TensorFlow Hub.
```
!pip install tensorflow
```
2. **Load and Preprocess Dataset:** The training images are normalized to the range [-1, 1] to help the generator output images with similar pixel values.
```
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28,
1).astype('float32')
train_images = (train_images - 127.5) / 127.5
```
3. **Build the Generator**: The generator starts with a dense layer that maps random noise to a low-resolution image and upscales it using transposed convolutions.
```
generator = tf.keras.Sequential([
    tf.keras.layers.Dense(7*7*256, use_bias=False, input_shape=(100,)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.LeakyReLU(),
    tf.keras.layers.Reshape((7, 7, 256)),
```

```
    tf.keras.layers.Conv2DTranspose(128, (5, 5), strides=(1, 1),
padding='same', use_bias=False),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.LeakyReLU(),
    tf.keras.layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),
padding='same', use_bias=False),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.LeakyReLU(),
    tf.keras.layers.Conv2DTranspose(1, (5, 5), strides=(2, 2),
padding='same', use_bias=False, activation='tanh')
])
```

4. **Build the Discriminator:** The discriminator is a standard convolutional neural network for binary classification (real or fake).

```
discriminator = tf.keras.Sequential([
    tf.keras.layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
input_shape=[28, 28, 1]),
    tf.keras.layers.LeakyReLU(),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'),
    tf.keras.layers.LeakyReLU(),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1)
])
```

5. **Compile and Train the Model:** Both models are trained using a custom training loop with `tf.GradientTape`, where the generator tries to fool the discriminator, and the discriminator tries to correctly classify real and fake images.

```
for epoch in range(epochs):
    for image_batch in train_dataset:
        train_step(image_batch)
```

USN NUMBER: 1RVU22CSE128
NAME: Reethu RG Thota

Result

Training Performance:

- The generator progressively improved at producing images that resembled the MNIST dataset.
- The discriminator learned to differentiate between real and fake images, with its accuracy fluctuating during training, as is typical in adversarial learning.

Generated Images:

- After sufficient training, the generator produced realistic images of handwritten digits, although occasional artifacts or blurring were observed.

Loss:

- Both generator and discriminator losses fluctuated throughout training, indicating the adversarial dynamic, but both models improved over time.

GitHub Link

https://github.com/reethuthota/Deep_Learning/tree/main/Lab7