|  |  |
| --- | --- |
| **Ex No: 8**  **Date: 18-09-2024** | **GANs using MNIST Dataset** |

**Objective:**

To implement a Generative Adversarial Network (GAN) to generate images of handwritten digits using the MNIST dataset. The goal is to train two neural networks – a generator and a discriminator – in an adversarial manner to produce realistic images that resemble handwritten digits.

**Description:**

Generative Adversarial Networks (GANs) are a class of machine learning frameworks where two neural networks, the generator and the discriminator, are trained simultaneously through an adversarial process. The generator learns to produce realistic images, while the discriminator tries to distinguish between real and fake images.

In this lab, we demonstrate the GAN training process using the MNIST dataset. The generator network generates images from random noise, and the discriminator network classifies images as either real (from the MNIST dataset) or fake (produced by the generator). The GAN training process involves the generator trying to "fool" the discriminator, and the discriminator attempting to identify the fake images.

Over time, the generator improves its ability to create realistic images, while the discriminator becomes better at distinguishing real images from fakes. The training process reaches an equilibrium when the discriminator can no longer effectively distinguish between real and fake images.

**Model Architecture:**

1. Generator Model:

* The generator uses a series of Dense and Conv2DTranspose (upsampling) layers to create a 28x28 pixel image from a random noise input (a 100-dimensional vector). Key layers in the generator include:
* Dense Layer: Takes the random noise vector as input and projects it into a dense representation (7x7x256).
* Batch Normalization: Normalizes the inputs for stable and faster training.
* Leaky ReLU: Allows a small gradient when the unit is not active.
* Reshape Layer: Reshapes the dense representation into a 7x7x256 tensor.
* Conv2DTranspose Layers: Perform upsampling to convert the tensor to the final image size (28x28x1). The final layer uses a tanh activation function to normalize the output pixel values between -1 and 1.

def make\_generator\_model():

model = tf.keras.Sequential()

model.add(layers.Dense(7\*7\*256, use\_bias=False, input\_shape=(100,)))

model.add(layers.BatchNormalization())

model.add(layers.LeakyReLU())

model.add(layers.Reshape((7, 7, 256)))

model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use\_bias=False))

model.add(layers.BatchNormalization())

model.add(layers.LeakyReLU())

model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use\_bias=False))

model.add(layers.BatchNormalization())

model.add(layers.LeakyReLU())

model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use\_bias=False, activation='tanh'))

return model

**2. Discriminator Model:**

The discriminator is a Convolutional Neural Network (CNN) that classifies images as real or fake. Key components include:

* **Conv2D Layers:** Extract features using convolutional filters, with strides set to 2 to reduce image dimensions.
* **Leaky ReLU:** Allows gradients when the unit is inactive, helping stabilize GAN training.
* **Dropout:** Prevents overfitting by randomly dropping units during training.
* **Dense Layer:** Outputs a single value indicating whether the input image is real or fake

def make\_discriminator\_model():

model = tf.keras.Sequential()

model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input\_shape=[28, 28, 1]))

model.add(layers.LeakyReLU())

model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))

model.add(layers.LeakyReLU())

model.add(layers.Dropout(0.3))

model.add(layers.Flatten())

model.add(layers.Dense(1))

return model

**Loss Functions:**

* **Discriminator Loss:** Measures how well the discriminator distinguishes real images from fake ones. It compares the discriminator's predictions on real images to an array of 1s and predictions on fake images to an array of 0s.

def discriminator\_loss(real\_output, fake\_output):

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

return total\_loss

* **Generator Loss:** Quantifies how well the generator is able to fool the discriminator. The generator's objective is to make the discriminator classify its fake images as real.

def generator\_loss(fake\_output):

return cross\_entropy(tf.ones\_like(fake\_output), fake\_output)

**Training Loop:**

The training process involves:

1. Sampling random noise to generate an image using the generator.
2. Passing real images from the training set and generated images to the discriminator.
3. Computing the loss for both the generator and discriminator.
4. Updating the generator and discriminator using back propagation based on their respective losses.

**Generating Images:**

After every few epochs, the generator creates images from random noise to monitor its progress. This allows visualization of how the generated images improve over time, starting as random noise and gradually resembling handwritten digits.

**Observations:**

* Initially, the generated images are random noise.
* As training progresses, the images become more defined and begin to resemble real handwritten digits.
* GAN training is sensitive and requires careful tuning of hyperparameters and balancing the generator and discriminator training rates.

**Github Link:**

**https://github.com/Bhargava-Srinivasan-26/Deep\_learning\_elective/tree/main/Unit%202/Lab%207**