|  |  |
| --- | --- |
| **Ex No: 8**  **Date: 26th September 2024** | **Deep Convolutional Generative Adversarial Network** |

**Objective:** The primary goal of this experiment is to demonstrate how to generate images of handwritten digits using a Deep Convolutional Generative Adversarial Network (DCGAN). The focus is on building a DCGAN using the Keras Sequential API and training it with a custom training loop using tf.GradientTape.

**Description:**

DCGANs are a specific type of **Generative Adversarial Network (GAN)** where both the generator and discriminator models utilize **deep convolutional neural networks**. They are particularly effective for generating realistic images from random noise.

In a GAN, two networks are trained simultaneously:

1. **Generator**: This network tries to create realistic images from random noise. It is tasked with "fooling" the discriminator.
2. **Discriminator**: This network attempts to distinguish between real images and images generated by the generator.

The adversarial aspect comes from the fact that the generator and discriminator are trained in opposition to each other. Over time, the generator improves in producing more realistic images, while the discriminator becomes better at spotting fake ones.

Key characteristics of DCGANs:

* **Convolutional Layers**: The generator upsamples data using transposed convolutions, while the discriminator uses normal convolutions to downsample and classify the images as real or fake.
* **Batch Normalization**: Helps stabilize the training process.
* **Leaky ReLU Activations**: Used in the discriminator to prevent dead neurons.

**Building the parts of the algorithm:**

The DCGAN is composed of the following major components:

**1. Generator Network:**

* Input: Random noise vector (typically sampled from a normal distribution).
* Layers:
  + Dense layer to reshape the input noise vector into a small spatial configuration.
  + Several transposed convolutional layers (Conv2DTranspose) to upsample the data, eventually forming a full-size image.
  + Activation functions (ReLU) and batch normalization layers to improve the stability of training.
* Output: A generated image with the same dimensions as the real images in the dataset.

**2. Discriminator Network:**

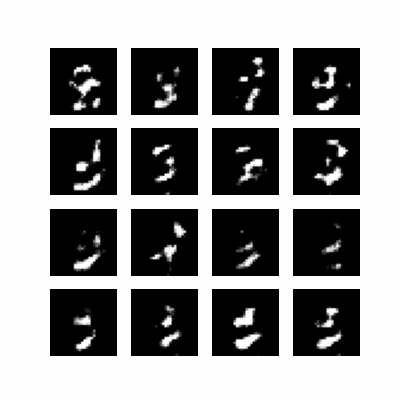
* Input: An image (either real or generated).
* Layers:
  + Convolutional layers (Conv2D) to downsample the image, extracting features that help in distinguishing real from fake.
  + LeakyReLU activations and batch normalization to ensure that learning remains stable and no neurons die out.
* Output: A single value representing the probability that the image is real.

**3. Loss Function:**

* Generator Loss: Measures how well the generator can fool the discriminator. This is calculated as the binary cross-entropy between the discriminator’s output for generated images and the label 'real' (i.e., the generator tries to make the discriminator classify generated images as real).
* Discriminator Loss: Measures how well the discriminator can differentiate between real and generated images. This is calculated as the binary cross-entropy between its classification of real images as real, and generated images as fake.

**4. Training Procedure:**

* The generator is trained to improve at creating images that can fool the discriminator.
* The discriminator is trained to become better at distinguishing real images from fake ones.
* A custom training loop is implemented using tf.GradientTape for fine-grained control over training.



An animated gif using the images saved during training.

**Conclusion:**

The notebook provides a hands-on approach to building a DCGAN using TensorFlow. The key aspects of DCGAN, such as the architecture of the generator and discriminator, loss functions, and training loops, are implemented and explained in the context of generating handwritten digit images. The tutorial is beneficial for understanding the workings of adversarial networks and convolutional architectures in generating realistic images.

**GitHub Link:** [**https://github.com/tulasigr/DeepLearning**](https://github.com/tulasigr/DeepLearning)