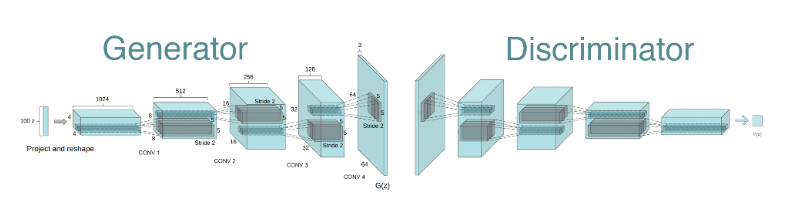
|  |  |
| --- | --- |
| **Ex No: 6**  **Date: 26Sep** | **Deep Convolutional Generative Adversarial Network** |

**Objective:**The main objective of this experiment is to showcase the generation of handwritten digit images through a Deep Convolutional Generative Adversarial Network (DCGAN). This project emphasizes building a DCGAN from scratch utilizing the Keras Sequential API. A custom training loop is implemented using tf.GradientTape to provide precise control over the generator and discriminator training steps, helping to better illustrate the adversarial learning dynamics. The ultimate aim is to create a model that can synthesize realistic images of handwritten digits, enhancing the understanding of generative model architectures and training processes.

**Description:**

DCGANs, or Deep Convolutional Generative Adversarial Networks, are a specialized form of Generative Adversarial Network (GAN) where both the generator and discriminator models employ deep convolutional neural networks, making them highly effective for generating realistic images from random noise inputs.

In a GAN, two networks are trained together:

* **Generator:** This network is responsible for creating realistic images from random noise, aiming to “fool” the discriminator by making the images appear as real as possible.
* **Discriminator:** This network’s task is to differentiate between real images and those generated by the generator, improving its ability to detect fake images over time.

The adversarial nature arises as these networks train in direct opposition: the generator constantly enhances its image synthesis abilities, while the discriminator becomes more proficient at identifying generated (fake) images.

**Key Characteristics of DCGANs:**

* **Convolutional Layers:** The generator network upscales data using transposed convolutions, while the discriminator uses standard convolutions to downsample and classify images as real or fake.
* **Batch Normalization:** This technique is applied to stabilize the training process, improving convergence.
* **Leaky ReLU Activations:** These activations, used in the discriminator, help avoid dead neurons and maintain gradient flow during training.

**Building The Parts Of The Algorithm:**

**1. Generator Network:**

* **Input:** A random noise vector, typically sampled from a normal distribution.
* Layers:
  + A dense layer that reshapes the noise vector into a small spatial configuration.
  + Multiple transposed convolutional layers (Conv2DTranspose) for upsampling the data, gradually shaping it into a full-sized image.
  + Activation functions (ReLU) and batch normalization layers to stabilize training.
* **Output:** A generated image matching the dimensions of real images from the dataset.

**2. Discriminator Network:**

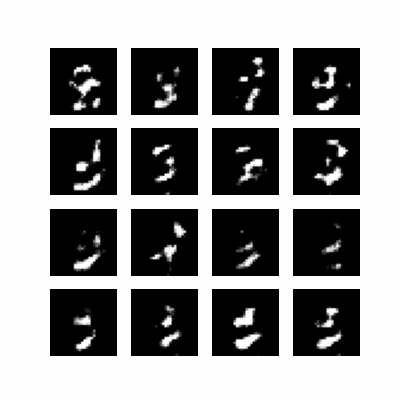
* **Input:** An image, which can be either real or generated.
* Layers:
  + Convolutional layers (Conv2D) for downsampling the image and extracting distinguishing features to aid in recognizing real from fake images.
  + Leaky ReLU activations and batch normalization to stabilize learning and prevent dead neurons.
* **Output:** A single probability score indicating the likelihood that the image is real.

**3. Loss Functions:**

* **Generator Loss:** Evaluates the generator’s ability to deceive the discriminator, calculated as the binary cross-entropy between the discriminator’s prediction on generated images and the label ‘real’ (since the generator aims to make its images appear real).
* **Discriminator Loss**: Assesses how well the discriminator can classify images, using binary cross-entropy between its predictions on real images (as real) and generated images (as fake).

**4. Training Procedure:**

* The generator trains to improve its image generation to the point of fooling the discriminator.
* The discriminator trains to become better at distinguishing real images from generated ones.
* A custom training loop using tf.GradientTape provides detailed control over the training process, allowing for targeted adjustments during each step.



An animated gif using the images saved during training.

**Conclusion:**

This lab provides a comprehensive, hands-on guide to building a DCGAN using TensorFlow. Key aspects of the DCGAN, such as the architecture of the generator and discriminator, loss functions, and custom training loops, are implemented and explained within the context of generating images of handwritten digits. Through this lab, participants will gain an understanding of how adversarial networks and convolutional architectures function in generating realistic images, making it a valuable resource for learning advanced deep learning concepts..

**GitHub Link: https://github.com/Kashishvarmaa/DL-CS3232/tree/main/Lab\_6**