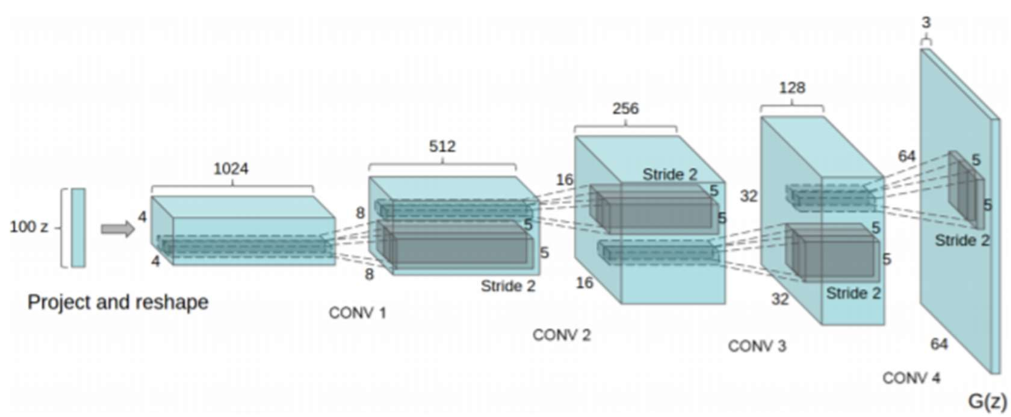
Experiment No. 3
Train a Deep Convolution Generative Multi-Layer (DCGAN) Network Model for MNIST dataset
Date of Performance:
Date of Submission:

Aim: Train a Deep Convolution Generative Multi-Layer (DCGAN) Network Model for MNIST dataset

Objective: Ability to implement Deep Convolution Generative Multi-Layer (DCGAN) Network Model.

Theory:

DCGAN uses convolutional and convolutional-transpose layers in the generator and discriminator, respectively. It was proposed by Radford et. al. in the paper Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks. Here the discriminator consists of strided convolution layers, batch normalization layers, and LeakyRelu as activation function. It takes a 3x64x64 input image. The generator consists of convolutional-transpose layers, batch normalization layers, and ReLU activations. The output will be a 3x64x64 RGB image.



Implementation:

Code:

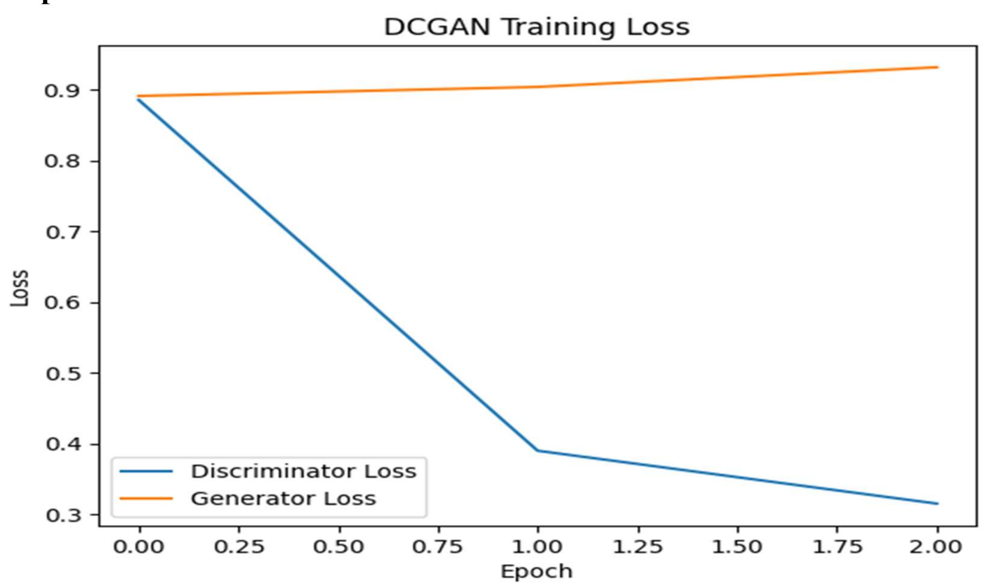
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization, Reshape, Flatten, Input
from tensorflow.keras.optimizers import Adam
```

```
# Load MNIST dataset
(X_train, ), (, _) = mnist.load_data()

# Normalize data
X_train = (X_train.astype(np.float32) - 127.5) / 127.5
X_train = np.expand_dims(X_train, axis=-1)

# Set up discriminator
def build_discriminator(input_shape=(28, 28, 1)):
    model = Sequential([
        Input(shape=input_shape),
        Flatten(),
        Dense(512),
        LeakyReLU(alpha=0.2),
        Dense(256),
        LeakyReLU(alpha=0.2),
        Dense(1, activation='sigmoid')
    ])
    return model

discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5),
metrics=['accuracy'])

# Set up generator
def build_generator(latent_dim):
    model = Sequential([
        Input(shape=(latent_dim,)),
        Dense(256),
        LeakyReLU(alpha=0.2),
        BatchNormalization(momentum=0.8),
        Dense(512),
        LeakyReLU(alpha=0.2),
        BatchNormalization(momentum=0.8),
        Dense(28*28*1, activation='tanh'),
        Reshape((28, 28, 1))
    ])
    return model

latent_dim = 100
generator = build_generator(latent_dim)

# Combined model
z = Input(shape=(latent_dim,))
img = generator(z)
discriminator.trainable = False
```

```
validity = discriminator(img)
combined = Model(z, validity)
combined.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))

# Training
epochs = 3
batch_size = 64
sample_interval = 1000

# Arrays to keep track of losses
d_loss_history = []
g_loss_history = []

for epoch in range(epochs):
   # Train Discriminator
   idx = np.random.randint(0, X_train.shape[0], batch_size)
   imgs = X_train[idx]

   noise = np.random.normal(0, 1, (batch_size, latent_dim))
   gen_imgs = generator.predict(noise)

   d_loss_real = discriminator.train_on_batch(imgs, np.ones((batch_size, 1)))
   d_loss_fake = discriminator.train_on_batch(gen_imgs, np.zeros((batch_size, 1)))
   d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

   # Train Generator
   noise = np.random.normal(0, 1, (batch_size, latent_dim))
   valid_y = np.array([1] * batch_size)

   g_loss = combined.train_on_batch(noise, valid_y)

   # Record losses
   d_loss_history.append(d_loss[0])
   g_loss_history.append(g_loss)

   # Print progress
   if epoch % 100 == 0:
      print(f"{epoch} [D loss: {d_loss[0]}, acc.: {100 * d_loss[1]}] [G loss: {g_loss}]")

   # Save generated images at sample interval
   if epoch % sample_interval == 0:
      r, c = 5, 5
      noise = np.random.normal(0, 1, (r * c, latent_dim))
      gen_imgs = generator.predict(noise) * 0.5 + 0.5  # Rescale images 0 - 1

      fig, axs = plt.subplots(r, c)
```

```
    cnt = 0
    for i in range(r):
        for j in range(c):
            axs[i,j].imshow(gen_imgs[cnt, :, :, 0], cmap='gray')
            axs[i,j].axis('off')
            cnt += 1
    fig.savefig(f"mnist_{epoch}.png")
    plt.close()

# Plot loss history
plt.plot(d_loss_history, label='Discriminator Loss')
plt.plot(g_loss_history, label='Generator Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('DCGAN Training Loss')
plt.legend()
plt.show()
```

Output:



Conclusion:

The accuracy of the Deep Convolutional Generative Adversarial Network (DCGAN) trained on the MNIST dataset varies based on factors like model architecture, training parameters, and dataset size. With appropriate tuning, DCGANs achieve high-quality image generation. The network comprises a generator and discriminator, leveraging convolutional layers to learn hierarchical representations, enabling realistic image synthesis.