WEEK - 12

Aim:-

To implement a Deep convolutional Generative Adversi-al Network (DCGAN) of generating complex RGB color image that resemble real-world images..

Objective:-

1.) To understand the working of Generative Adversial Networks (GAN's).

2.) To use convolutional layers for image generation and discrimination.

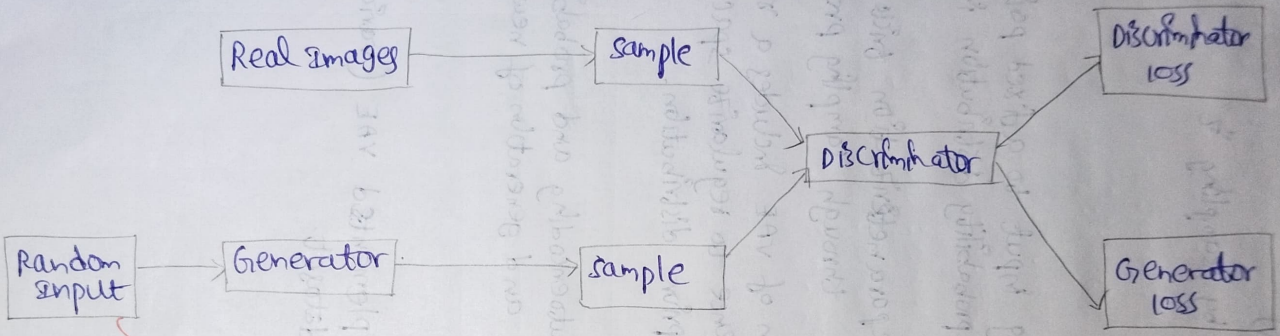
3.) To train the generator to produce realistic image and the discriminator to distinguish real from fake.

4.) To evaluate the quality of generated images visually after training.

Pseudo code:-

```
Begin
    load dataset of real color images
        → Resize and normalize images to (-1, 1]
    Define Generator Network (G):
            Input: random wise vector (z)
        layers: series of Conv Transpose 2D + Batch No + Relu
        output: RGB image
    Define Discriminator Network (D):-
            Input: RGB Image
        layers: series of conv 2D + Batch Norma + Leaky Relu
        output: single probability.

    Set loss function = Binary cross entropy
    Set optimizer = Adam (r=0.0002, B1=0.5)
    For each epoch:
            Train discriminator
            Train Generator
    Repeat until generator produces realistic color images
    END
```

Architecture of GAN



Algorithm :-

1.) Load a color image dataset
2.) Normalize pixel values to [-1,1].
3.) Define generator using conv 2D Transpose layers.
4.) Define Discriminator using conv 2D layers.
5.) Combine both to form DCGAN.
6.) Train using adversarial loss – generator tries to fool discriminator.
7.) periodically generate and display fake images.

observation :-

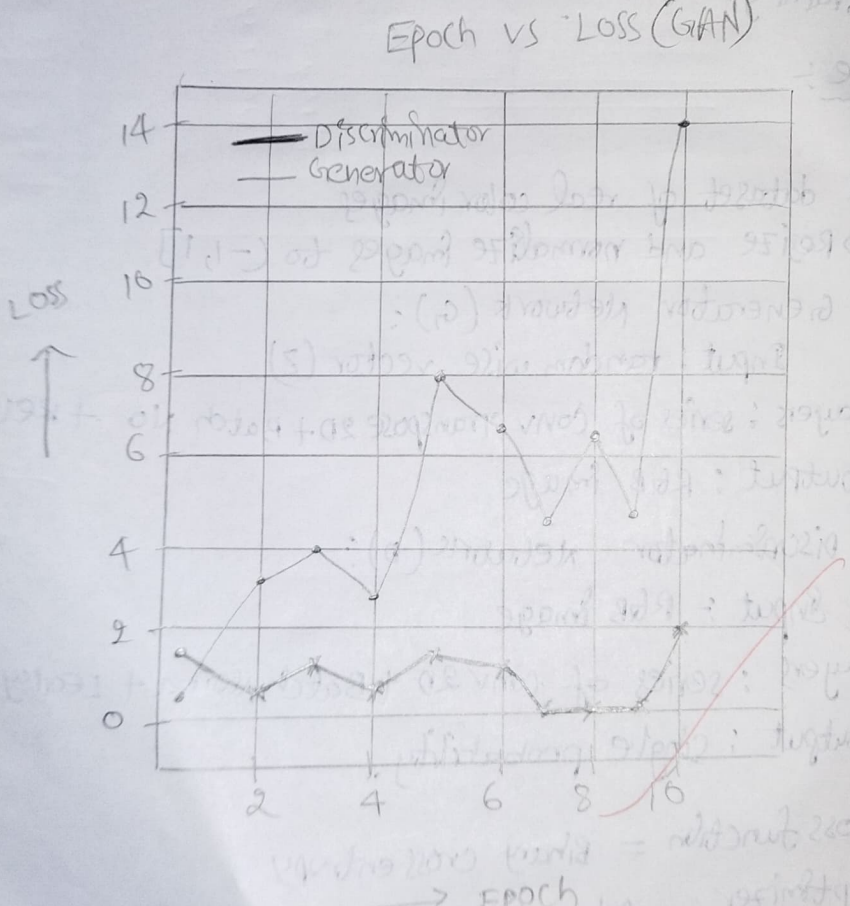
→ The generator gradually learned to produce realistic color images.
→ The discriminator improved in distinguishing real and fake images.
→ Generated images were visually similar to the training dataset after sufficient epochs.
→ Training demonstrated the adversarial learning process of GAN effectively.

Result :-

The DCGAN successfully generated realistic-looking color images, proving the effectiveness of adversarial learning.

output:

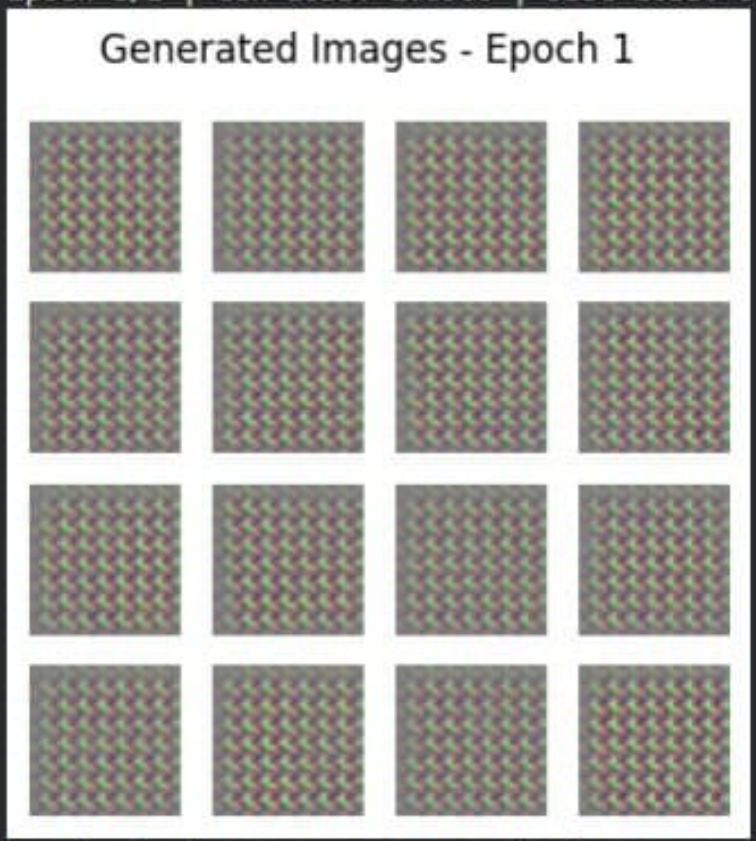
Epoch [1/10] D Loss : 1.0992, G Loss : 0.5678
Epoch [2/10] D Loss : 0.5060, G Loss : 2.7113
Epoch [3/10] D Loss : 1.2431, G Loss : 3.9827
Epoch [4/10] D Loss : 0.4229, G Loss : 2.2684
Epoch [5/10] D Loss : 1.486, G Loss : 8.1004
Epoch [6/10] D Loss : 1.0081, G Loss : 6.7625
Epoch [7/10] D Loss : 0.0252, G Loss : 4.5718
Epoch [8/10] D Loss : 0.0043, G Loss : 6.3371
Epoch [9/10] D Loss : 0.1880, G Loss : 4.73157
Epoch [10/10] D Loss : 2.3294, G Loss : 14.0363



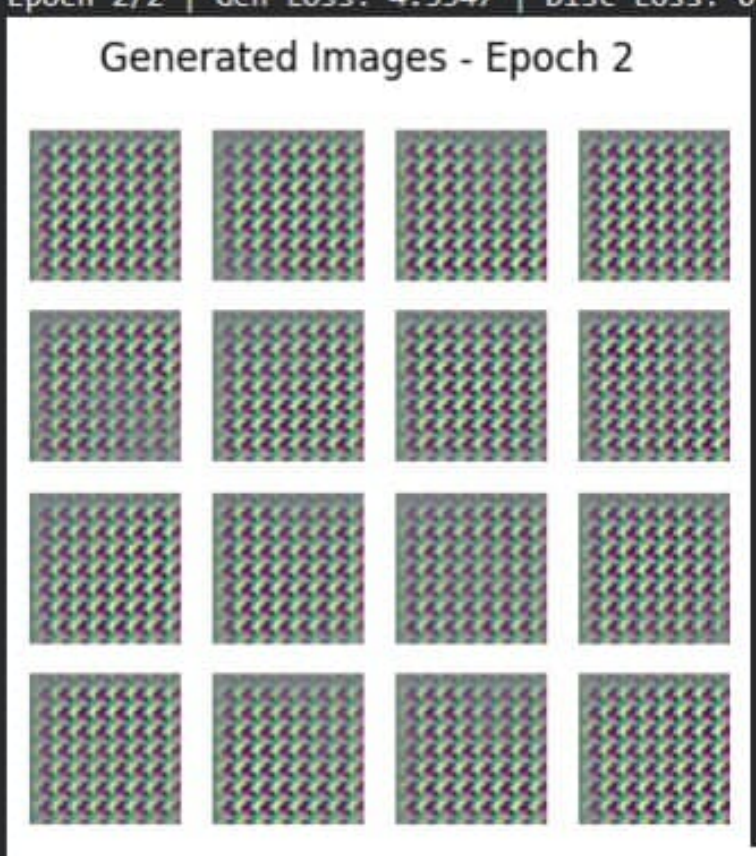
```python
# ✅ Lab 12 (Fast Version): DCGAN to Generate Color Images (CIFAR-10)
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import numpy as np
import time
(x_train, _), (_, _) = tf.keras.datasets.cifar10.load_data()
x_train = (x_train.astype("float32") - 127.5) / 127.5  # normalize to [-1,1]
x_train = x_train[:5000]  # only 5k images for faster training

BUFFER_SIZE = 5000
BATCH_SIZE = 32
train_dataset = tf.data.Dataset.from_tensor_slices(x_train).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

def build_generator():
    model = tf.keras.Sequential([
        layers.Input(shape=(100,)),
        layers.Dense(8*8*128, use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),
        layers.Reshape((8, 8, 128)),
        layers.Conv2DTranspose(64, (5,5), strides=(2,2), padding="same", use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),
        layers.Conv2DTranspose(3, (5,5), strides=(2,2), padding="same", use_bias=False, activation="tanh"),
    ])
    return model

def build_discriminator():
    model = tf.keras.Sequential([
        layers.Input(shape=(32, 32, 3)),
        layers.Conv2D(64, (5,5), strides=(2,2), padding="same"),
        layers.LeakyReLU(),
        layers.Dropout(0.3),
        layers.Conv2D(128, (5,5), strides=(2,2), padding="same"),
        layers.LeakyReLU(),
        layers.Dropout(0.3),
        layers.Flatten(),
        layers.Dense(1)
    ])
    return model

generator = build_generator()
discriminator = build_discriminator()

cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
gen_optimizer = tf.keras.optimizers.Adam(1e-4)
disc_optimizer = tf.keras.optimizers.Adam(1e-4)

@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, 100])
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)
        gen_loss = cross_entropy(tf.ones_like(fake_output), fake_output)
        disc_loss = (
            cross_entropy(tf.ones_like(real_output), real_output)
            + cross_entropy(tf.zeros_like(fake_output), fake_output)
```

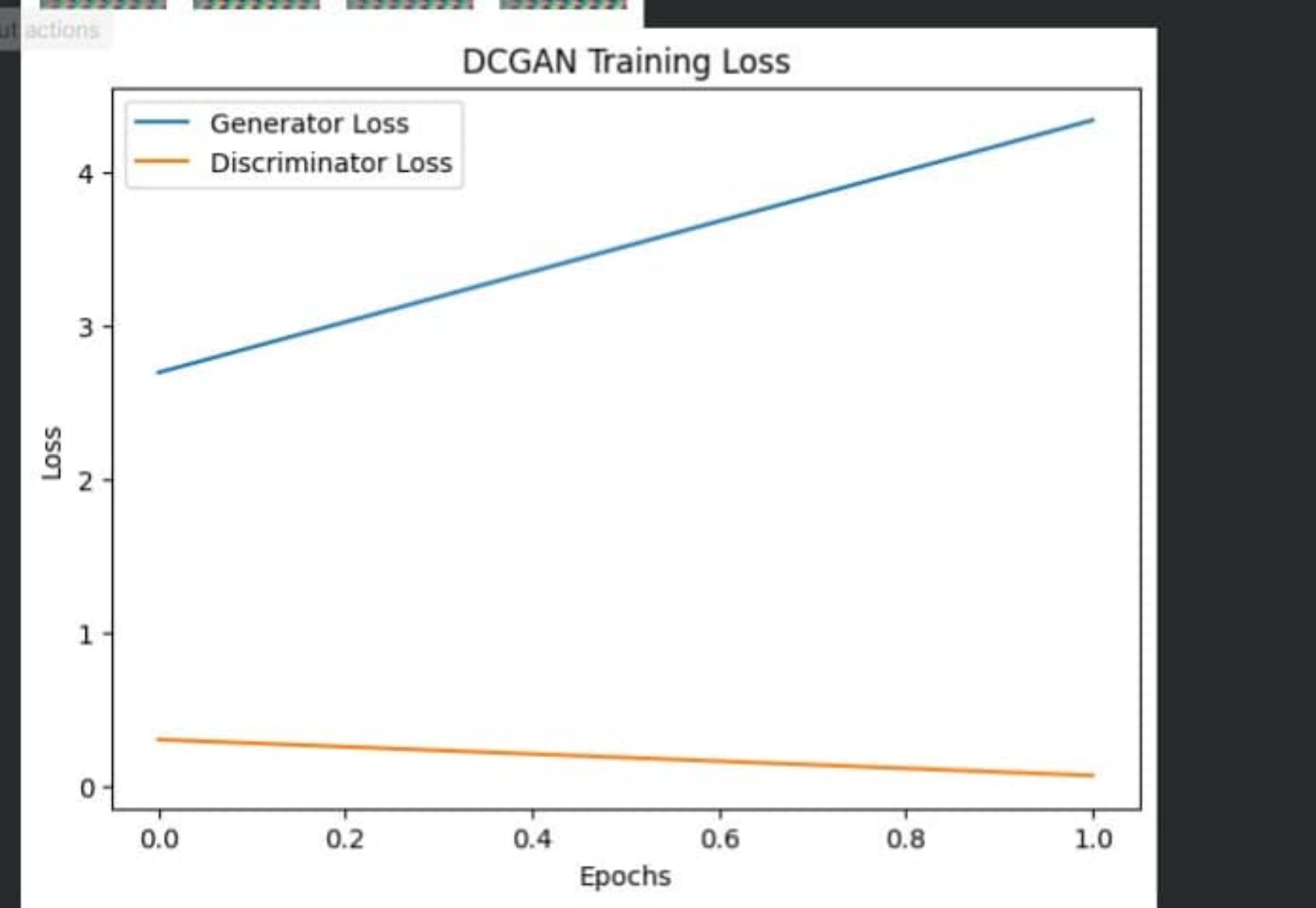
```
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)
        gen_loss = cross_entropy(tf.ones_like(fake_output), fake_output)
        disc_loss = (
            cross_entropy(tf.ones_like(real_output), real_output)
            + cross_entropy(tf.zeros_like(fake_output), fake_output)
        )
    grads_gen = gen_tape.gradient(gen_loss, generator.trainable_variables)
    grads_disc = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
    gen_optimizer.apply_gradients(zip(grads_gen, generator.trainable_variables))
    disc_optimizer.apply_gradients(zip(grads_disc, discriminator.trainable_variables))
    return gen_loss, disc_loss

EPOCHS = 2
seed = tf.random.normal([16, 100])
gen_losses, disc_losses = [], []

for epoch in range(EPOCHS):
    start = time.time()
    for image_batch in train_dataset.take(100):  # only 100 batches for speed
        g_loss, d_loss = train_step(image_batch)
    gen_losses.append(g_loss.numpy())
    disc_losses.append(d_loss.numpy())

    print(f"Epoch {epoch+1}/{EPOCHS} | Gen Loss: {g_loss:.4f} | Disc Loss: {d_loss:.4f} | Time: {time.time()-start:.2f}s")

    predictions = generator(seed, training=False)
    fig = plt.figure(figsize=(4, 4))
    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i + 1)
        plt.imshow((predictions[i] * 0.5 + 0.5))
        plt.axis("off")
    plt.suptitle(f"Generated Images - Epoch {epoch+1}")
    plt.show()

plt.figure(figsize=(7,5))
plt.plot(gen_losses, label="Generator Loss")
plt.plot(disc_losses, label="Discriminator Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.title("DCGAN Training Loss")
plt.show()
```

Epoch 1/2 | Gen Loss: 2.6906 | Disc Loss: 0.2989 | Time: 36.21s



Epoch 2/2 | Gen Loss: 4.3347 | Disc Loss: 0.0647 | Time: 29.22s




DCGAN Training Loss
Generator Loss
Discriminator Loss
Loss
Epochs
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0