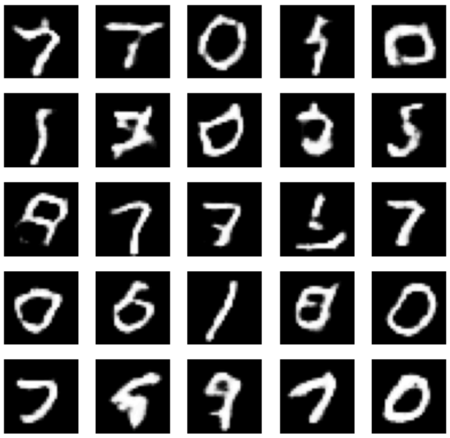
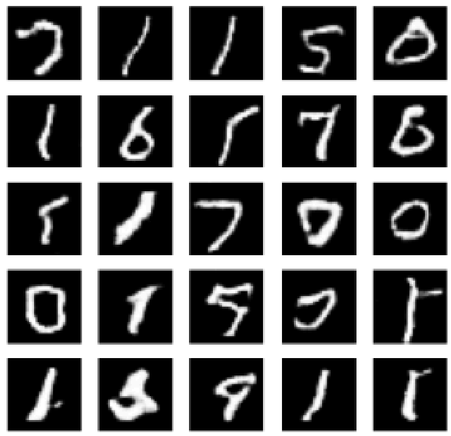
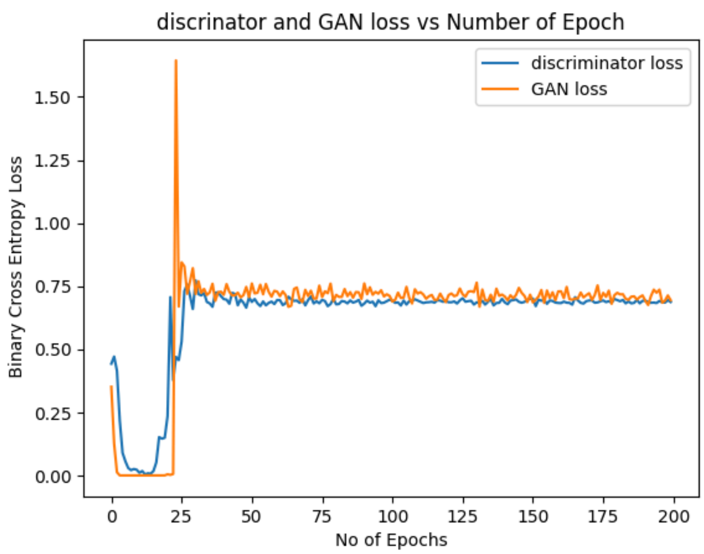
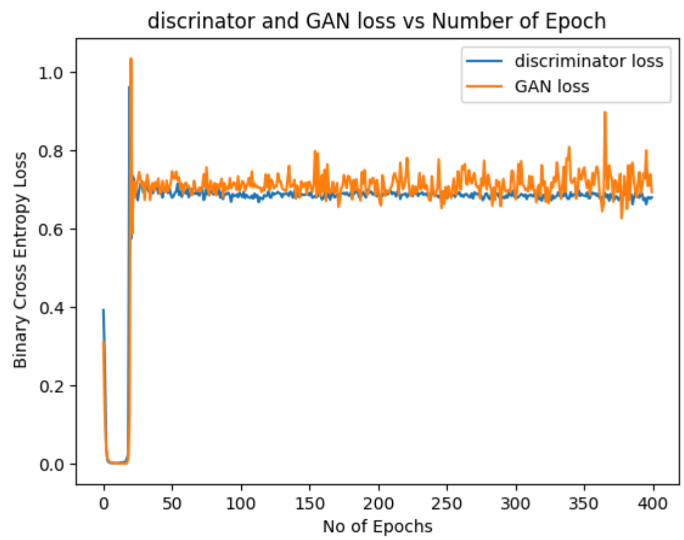
ME 793 Assignment 10
Roll No.: 19D100011

Answer 1:

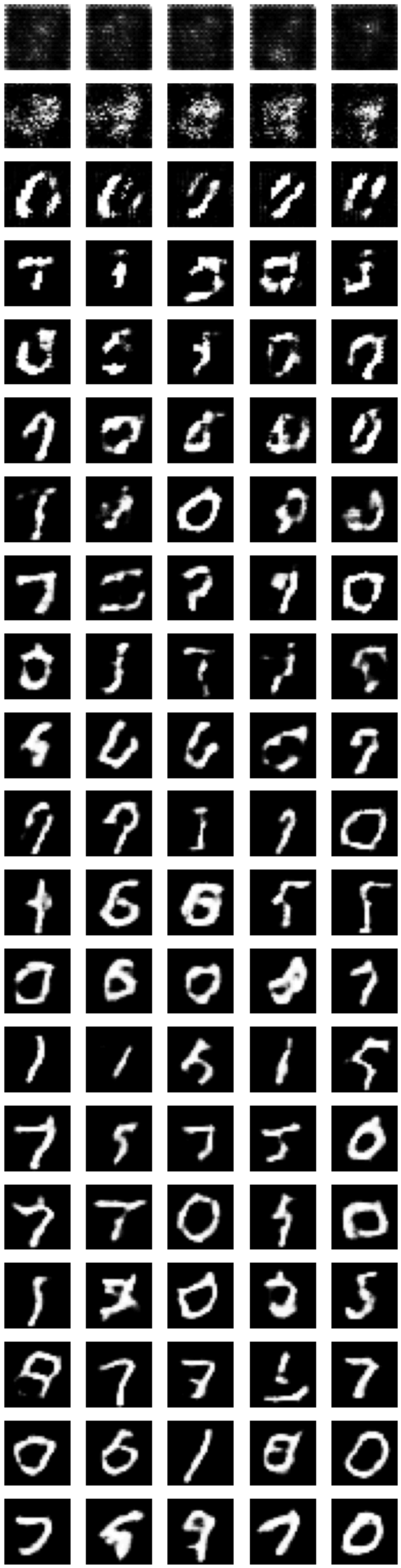


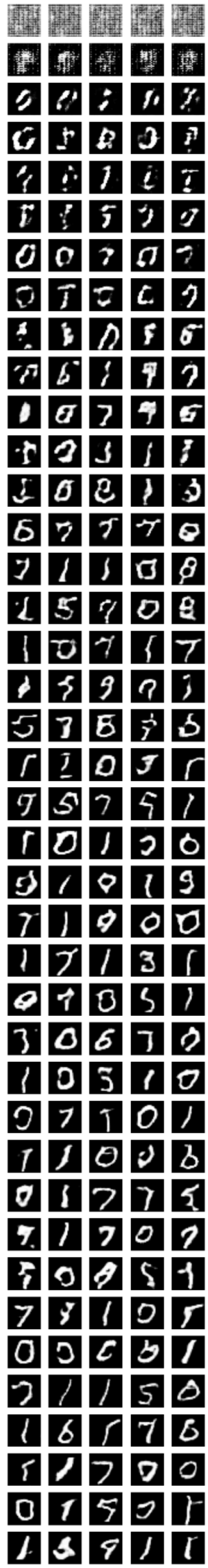






After doubling the epochs from 200 to 400 and running the notebook again, we observe that the generated images appear to have more refined details and clearer shapes compared to the images generated with the default number of epochs.





ME 793 - Assignment 10

Answer 2

19D100011

In [23]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
```

In [24]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras import layers
import tensorflow_datasets as tfds
from keras.models import Sequential , Model
from keras.layers import Dense,Flatten,Reshape,Dropout,LeakyReLU,Conv2DTranspose,Co
from keras.optimizers import Adam,SGD
from keras.layers import  LeakyReLU, BatchNormalization, Input
```

I used structural similarity index (SSIM) as a quantitative measure of similarity. I used the compare_ssim function from the scikit-image library to compute the SSIM score.

In [25]:
```
from skimage.metrics import structural_similarity as ssim
```

In [26]:
```
from keras.datasets import mnist
(train_images, y_train), (_, _) = mnist.load_data()
print(len(train_images))
train_images=train_images[(y_train==0) | (y_train==1) | (y_train==5) | (y_train==7)
train_images=train_images[:5000]
```

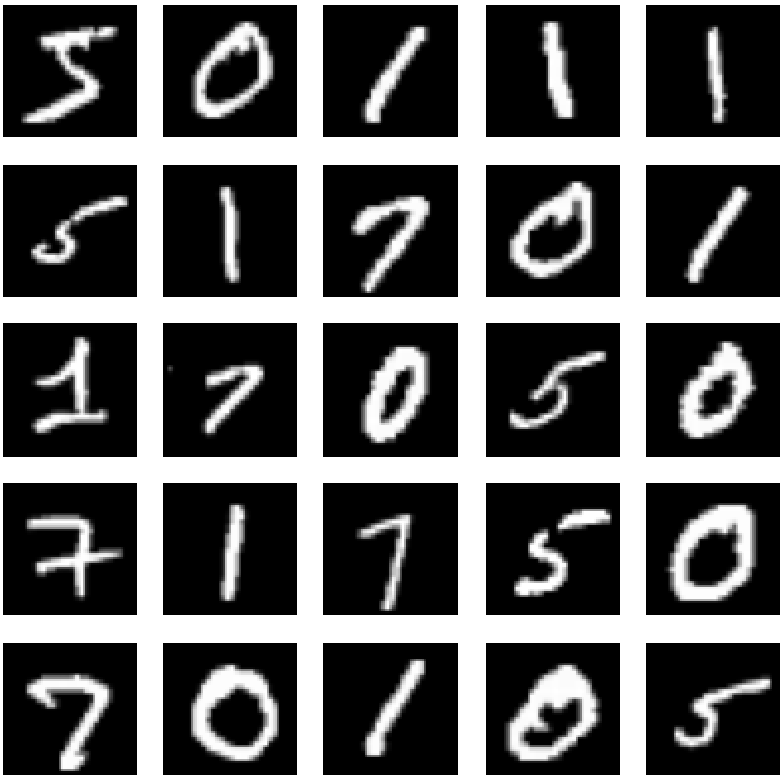
```
60000
```

In [27]:
```
len(train_images)
```

Out[27]:
```
5000
```

In [28]:
```
def plot_images(images):
    fig, axes = plt.subplots(5, 5, figsize=(5, 5))

    for i, ax in enumerate(axes.flat):
        ax.imshow(images[i], cmap='gray')
        ax.axis('off')

    plt.show()
```

In [29]:
```
plot_images(train_images)
```



In [30]:
```
train_images=train_images.astype("float32")/127.5-1                          # norma
train_images=train_images.reshape(train_images.shape[0],28,28,1)             # now t
```

In [31]:
```
def build_generator(latent_dim,input_shape):
    generator=Sequential([

        Dense(7*7*128 , input_dim=latent_dim),
        LeakyReLU(alpha=.2),
        Reshape((7, 7, 128)),
        BatchNormalization(),

        Conv2DTranspose(64 , (4,4) , padding="same" , strides=(2,2)),
        LeakyReLU(alpha=.2),
        BatchNormalization(),

        Conv2DTranspose(32 , (4,4) , padding="same" , strides=(2,2)),
        LeakyReLU(alpha=.2),
        BatchNormalization(),

        Conv2DTranspose(1 , (4,4) , padding="same" , activation="tanh"),
    ])

    return generator
```

In [32]:
```
def build_discriminator(input_shape):
    discriminator=Sequential([

        Conv2D(64 , (3,3) , strides=(2,2) , padding="same" , input_shape=input_shap
        LeakyReLU(alpha=.2),
        Dropout(rate=.4),

        Conv2D(64 , (3,3) , strides=(2,2) , padding="same" , input_shape=input_shap
        LeakyReLU(alpha=.2),
        Dropout(rate=.4),

        Flatten(),
```

```
        Dense(1 , activation="sigmoid" )

    ])
    return discriminator
```

In [33]:
```
def build_gan(generator, discriminator):
    discriminator.trainable = False  # Freeze discriminator during GAN training

    gan_input = Input(shape=(latent_dim,))
    x = generator(gan_input)
    gan_output = discriminator(x)

    gan = Model(gan_input, gan_output)
    return gan
```

In [34]:
```
latent_dim=100
image_shape=(28,28,1)
```

In [35]:
```
generator = build_generator(latent_dim, image_shape)
generator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))

discriminator = build_discriminator(image_shape)
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5), metr

gan = build_gan(generator, discriminator)
gan.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
```

In [36]:
```
epochs = 400
batch_size = 256
```

In [37]:
```
def generate_real_samples(dataset, n_samples):
    idx = np.random.randint(0, train_images.shape[0], n_samples)
    X = train_images[idx]
    y = np.ones((n_samples, 1))
    return X, y

# Generate fake samples using the generator
def generate_fake_samples(generator, latent_dim, n_samples):
    noise = np.random.normal(0, 1, size=(n_samples, latent_dim))
    X = generator.predict(noise)
    y = np.zeros((n_samples, 1))
    return X, y

# Generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    noise = np.random.normal(0, 1, size=(n_samples, latent_dim))
    return noise
```

In [38]:
```
discriminator_loss=[]
gan_loss=[]
generated_list=[]
```

In [39]:
```
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs, n_batch):
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)
    discriminator_loss=[]
    gan_loss=[]
    generated_list=[]
    # Manually enumerate epochs
    for i in range(n_epochs):
        # Enumerate batches over the training set
```

```
        for j in range(bat_per_epo):
            batch_loss_d=[]
            batch_loss_g=[]
            # Get randomly selected 'real' samples
            X_real, y_real = generate_real_samples(dataset, half_batch)

            # Generate 'fake' examples
            X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)

            # Create training set for the discriminator
            X, y = np.vstack((X_real, X_fake)), np.vstack((y_real, y_fake))

            # Update discriminator model weights
            d_loss, _ = d_model.train_on_batch(X, y)
            batch_loss_d.append(d_loss)
            # Prepare points in latent space as input for the generator
            X_gan = generate_latent_points(latent_dim, n_batch)

            # Create inverted labels for the fake samples
            y_gan = np.ones((n_batch, 1))

            # Update the generator via the discriminator's error
            g_loss = gan_model.train_on_batch(X_gan, y_gan)
            batch_loss_g.append(g_loss)
        gan_los=np.mean(batch_loss_g)
        gan_loss.append(gan_los)
        dis_loss=np.mean(batch_loss_d)
        discriminator_loss.append(dis_loss)

        # Summarize loss on this batch

        if (i + 1) % 10 == 0:
                print(f'Epoch {i + 1}/{n_epochs}, Batch {j + 1}/{bat_per_epo}, D Lo
        if (i + 1) % 10 == 0:
                num_samples=5
                noise = generate_latent_points(latent_dim, num_samples)
                generated_images = generator.predict(noise)
                generated_list.append(generated_images)

    return discriminator_loss, gan_loss, generated_list
```

In [40]:
```
# discriminator_loss, gan_loss, generated_list=train(generator, discriminator, gan,
```

In [41]:
```
# fig, axes = plt.subplots(len(generated_list), 5, figsize=(5, len(generated_list))
# for i in range(len(generated_list)):
#     image_list=generated_list[i]
#     for j in range(5):
#         axes[i,j].imshow(image_list[j],cmap="gray")
#         axes[i, j].axis('off')
# #plt.tight_layout()
# plt.show()
```

In [42]:
```
# plt.plot(discriminator_loss,label="discriminator loss")
# plt.plot(gan_loss,label="GAN loss")
# plt.legend()
# plt.title("discrinator and GAN loss vs Number of Epoch")
# plt.xlabel("No of Epochs")
# plt.ylabel(" Binary Cross Entropy Loss")
# plt.show()
```

In [43]:
```
# Select a random input image of digit '7'
input_image = train_images[np.random.randint(0, train_images.shape[0])]
```

```
input_image = cv2.resize(input_image, (28, 28), interpolation=cv2.INTER_AREA)
input_image = input_image.reshape(1, 28, 28, 1)

# Train the GAN for 200 and 400 epochs
discriminator_loss_200, gan_loss_200, generated_list_200 = train(generator, discrim
discriminator_loss_400, gan_loss_400, generated_list_400 = train(generator, discrim
```

```
Streaming output truncated to the last 5000 lines.
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
Epoch 140/400, Batch 19/19, D Loss: 0.688, G Loss: 0.730
1/1 [==============================] - 0s 19ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 7ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 4ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 6ms/step
4/4 [==============================] - 0s 4ms/step
```

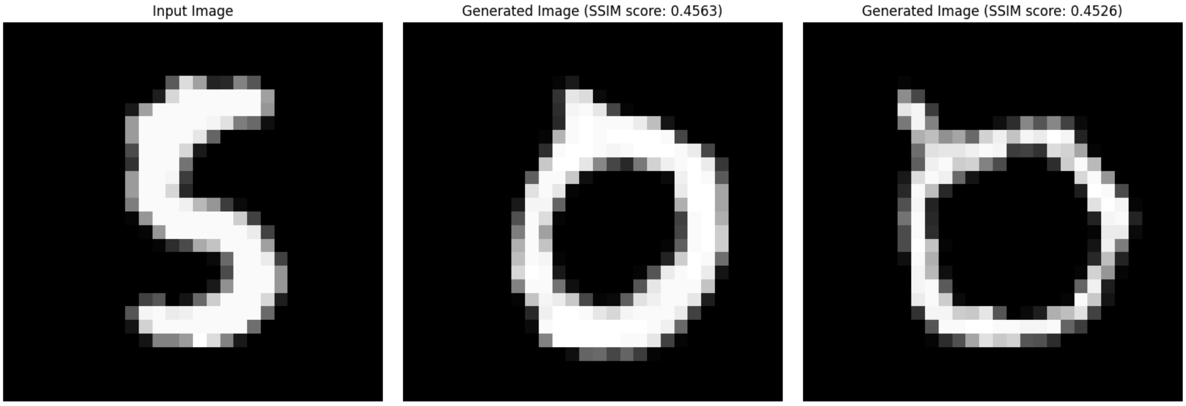
```
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 5ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 3ms/step
Epoch 400/400, Batch 19/19, D Loss: 0.692, G Loss: 0.715
1/1 [==============================] - 0s 19ms/step
```

In [44]:

```python
# Generate images after 200 and 400 epochs
generated_images_200 = generated_list_200[-1]
generated_images_400 = generated_list_400[-1]

# Select a random generated image of digit '7' after 200 and 400 epochs
generated_image_200 = generated_images_200[np.random.randint(0, generated_images_20
generated_image_400 = generated_images_400[np.random.randint(0, generated_images_40

# Reshape the images to 28x28 for comparison
input_image = input_image.squeeze()
generated_image_200 = generated_image_200.squeeze()
generated_image_400 = generated_image_400.squeeze()

# Calculate SSIM score
ssim_score_200 = ssim(input_image, generated_image_200, multichannel=True)
ssim_score_400 = ssim(input_image, generated_image_400, multichannel=True)

print(f"SSIM score after 200 epochs: {ssim_score_200:.4f}")
print(f"SSIM score after 400 epochs: {ssim_score_400:.4f}")

# Plot the images
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(input_image, cmap="gray")
axes[0].set_title("Input Image")
axes[0].axis('off')
axes[1].imshow(generated_image_200, cmap="gray")
axes[1].set_title(f"Generated Image (SSIM score: {ssim_score_200:.4f})")
axes[1].axis('off')
axes[2].imshow(generated_image_400, cmap="gray")
axes[2].set_title(f"Generated Image (SSIM score: {ssim_score_400:.4f})")
axes[2].axis('off')
plt.tight_layout()
plt.show()
```

```
SSIM score after 200 epochs: 0.4563
SSIM score after 400 epochs: 0.4526
```

```
<ipython-input-44-e577e0a3c159>:15: FutureWarning: `multichannel` is a deprecated argument name for `structural_similarity`. It will be removed in version 1.0. Please use `channel_axis` instead.
  ssim_score_200 = ssim(input_image, generated_image_200, multichannel=True)
<ipython-input-44-e577e0a3c159>:16: FutureWarning: `multichannel` is a deprecated argument name for `structural_similarity`. It will be removed in version 1.0. Please use `channel_axis` instead.
  ssim_score_400 = ssim(input_image, generated_image_400, multichannel=True)
```


Input Image
Generated Image (SSIM score: 0.4563)
Generated Image (SSIM score: 0.4526)

In [ ]: