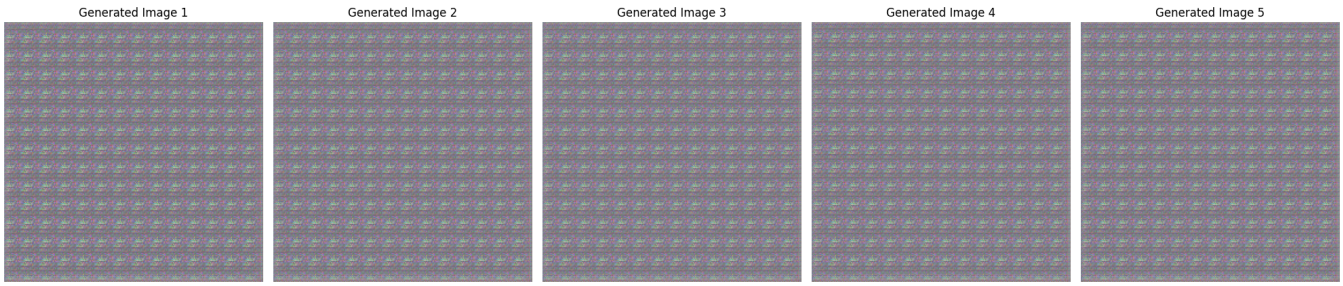
FID SCORE: 2.02

- Sucheta- Building the generator, discriminator
- Sucheta- Wasserstein Loss for the critic, clipping weights
- Ayesha- Optimizer, combining and compiling the model
- Ayesha- Training Wgan
- Ayesha- FID Score




```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Conv2D, LeakyReLU, Dropout, Flatten, Dense, Reshape, Conv2DTranspose
from tensorflow.keras.optimizers import Adam
import tensorflow.keras.backend as K
from sklearn.model_selection import train_test_split
import os
import matplotlib.pyplot as plt
from tensorflow.keras.applications.inception_v3 import InceptionV3
from scipy.linalg import sqrtm
import os
import pickle
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Loading images from file
images = np.load('/content/drive/My Drive/images.npy')

# Loading labels from file
labels = np.load('/content/drive/My Drive/labels.npy')

print(f"Images loaded: {images.shape}")
print(f"Labels loaded: {labels.shape}")
```

```
Images loaded: (8390, 224, 224, 3)
Labels loaded: (8390,)
```

```
labels_unique = np.unique(labels)
print(f"Unique labels: {labels_unique}")
```

```
Unique labels: ['drink' 'food' 'inside' 'menu' 'outside']
```

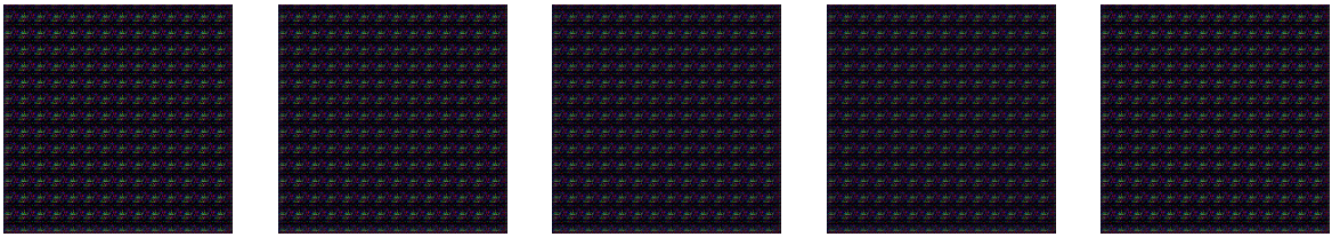
```
# Separate images by class
selected_class = 'menu'
class_indices = np.where(labels == selected_class)[0]
class_images = images[class_indices]

print(f"Total images for class '{selected_class}': {class_images.shape}")

# Plot the images
num_images = 5
fig, axs = plt.subplots(1, num_images, figsize=(15, 15))
for i in range(num_images):
    axs[i].imshow(class_images[i, :, :, :])  # Display RGB images
    axs[i].axis('off')
plt.show()
```

```
Total images for class 'menu': (1678, 224, 224, 3)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```

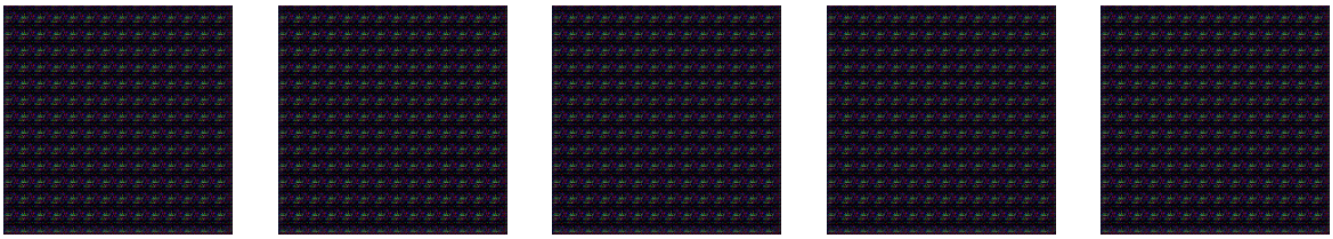

```
from tensorflow.keras.layers import BatchNormalization

def build_generator():
    model = Sequential()
    model.add(Dense(512 * 14 * 14, activation="relu", input_dim=100))
    model.add(Reshape((14, 14, 512)))
    model.add(Conv2DTranspose(256, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(BatchNormalization())  # BatchNormalization
    model.add(Conv2DTranspose(128, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(BatchNormalization())  # BatchNormalization
    model.add(Conv2DTranspose(64, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(BatchNormalization())  # BatchNormalization
    model.add(Conv2DTranspose(3, kernel_size=4, strides=2, activation="tanh", padding="same"))
    return model


def build_critic():
    model = Sequential()
    model.add(Conv2D(64, kernel_size=4, strides=2, input_shape=(224, 224, 3), padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.3))
    model.add(Conv2D(128, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.3))
    model.add(Conv2D(256, kernel_size=4, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.3))
    model.add(Flatten())
    model.add(Dense(1))  # No activation function for unbounded output
    return model


# Normalize images to [-1, 1]
class_images = class_images / 127.5 - 1  # Assuming images are in the range [0, 255]


# Wasserstein Loss for the critic
def wasserstein_loss(y_true, y_pred):
    return K.mean(y_true * y_pred)


# Optimizer
optimizer_critic = Adam(learning_rate=0.00005, beta_1=0.0, beta_2=0.9)
optimizer_generator = Adam(learning_rate=0.00005, beta_1=0.0, beta_2=0.9)


# Build the generator and critic models
generator = build_generator()
critic = build_critic()


# Compile the critic model with the Wasserstein loss
critic.compile(loss=wasserstein_loss, optimizer=optimizer_critic)



# Build the combined model (generator + critic)
z = Input(shape=(100,))
img = generator(z)
critic.trainable = False  # Freeze the critic during generator training
```

```
valid = critic(img)
combined = Model(z, valid)



# Compile the combined model (make sure it's compiled before training)
combined.compile(loss=wasserstein_loss, optimizer=optimizer_generator)


# Function for clipping critic weights
def clip_weights(critic, clip_value=0.01):
    for layer in critic.layers:
        if hasattr(layer, 'kernel'):
            K.set_value(layer.kernel, K.clip(layer.kernel, -clip_value, clip_value))


# Generator Output Check
def generate_and_plot_images(generator, num_images=5):
    noise = np.random.normal(0, 1, (num_images, 100))  # Generate random noise for the generator
    generated_images = generator.predict(noise)  # Generate images from noise

    # Debugging: Print out the shapes of the generated images
    print(f"Generated image shape: {generated_images.shape}")

    # Plot the generated images
    fig, axs = plt.subplots(1, num_images, figsize=(15, 15))
    for i in range(num_images):
        axs[i].imshow(generated_images[i])  # Display the generated images
        axs[i].axis('off')
    plt.show()


epochs = 300
batch_size = 64
save_interval = 50

# Train WGAN
for epoch in range(epochs):
    # Train Critic on real and fake images
    idx = np.random.randint(0, class_images.shape[0], batch_size)
    realhalf = class_images[idx]  # Make sure this has the shape (batch_size, 224, 224, 3)

    # Generate fake images
    noise = np.random.normal(0, 1, (batch_size, 100))
    fake_imgs = generator.predict(noise)

    # Train the critic on real images with label -1 (real)
    critic.trainable = True
    d_loss_real = critic.train_on_batch(realhalf, -np.ones((batch_size, 1)))  # Real images should be labeled -1

    # Train the critic on fake images with label 1 (fake)
    d_loss_fake = critic.train_on_batch(fake_imgs, np.ones((batch_size, 1)))   # Fake images should be labeled 1

    # Clip critic weights after every batch to enforce the 1-Lipschitz constraint
    clip_weights(critic)

    # Train Generator (via the combined model)
    noise = np.random.normal(0, 1, (batch_size, 100))
    g_loss = combined.train_on_batch(noise, -np.ones((batch_size, 1)))  # Generator aims to fool the critic (label -1)

    # Print progress
    if epoch % save_interval == 0:
        print(f"{epoch} [D loss real: {d_loss_real}] [D loss fake: {d_loss_fake}] [G loss: {g_loss}]")
        generate_and_plot_images(generator, num_images=5)
```

```
2/2 [==============================] - 1s 382ms/step
0 [D loss real: -19046.36328125] [D loss fake: -26.18267059326172] [G loss: -7448.857421875]
1/1 [==============================] - 0s 111ms/step
Generated image shape: (5, 224, 224, 3)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```

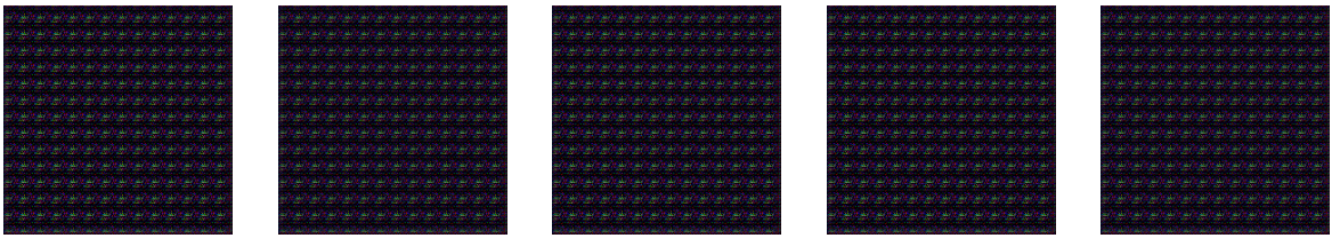


```
2/2 [==============================] - 1s 379ms/step
2/2 [==============================] - 1s 413ms/step
2/2 [==============================] - 1s 352ms/step
2/2 [==============================] - 1s 316ms/step
2/2 [==============================] - 1s 313ms/step
2/2 [==============================] - 1s 301ms/step
2/2 [==============================] - 1s 356ms/step
2/2 [==============================] - 1s 323ms/step
2/2 [==============================] - 1s 320ms/step
2/2 [==============================] - 1s 406ms/step
2/2 [==============================] - 1s 318ms/step
2/2 [==============================] - 1s 357ms/step
2/2 [==============================] - 1s 357ms/step
2/2 [==============================] - 1s 307ms/step
2/2 [==============================] - 1s 329ms/step
2/2 [==============================] - 1s 341ms/step
2/2 [==============================] - 1s 355ms/step
2/2 [==============================] - 1s 320ms/step
2/2 [==============================] - 1s 370ms/step
2/2 [==============================] - 1s 359ms/step
2/2 [==============================] - 1s 358ms/step
2/2 [==============================] - 1s 355ms/step
2/2 [==============================] - 1s 320ms/step
2/2 [==============================] - 1s 302ms/step
2/2 [==============================] - 1s 396ms/step
2/2 [==============================] - 1s 364ms/step
2/2 [==============================] - 1s 374ms/step
2/2 [==============================] - 1s 399ms/step
2/2 [==============================] - 1s 314ms/step
2/2 [==============================] - 1s 324ms/step
2/2 [==============================] - 1s 412ms/step
2/2 [==============================] - 1s 392ms/step
2/2 [==============================] - 1s 348ms/step
2/2 [==============================] - 1s 342ms/step
2/2 [==============================] - 1s 318ms/step
2/2 [==============================] - 1s 311ms/step
2/2 [==============================] - 1s 319ms/step
2/2 [==============================] - 1s 322ms/step
2/2 [==============================] - 1s 311ms/step
2/2 [==============================] - 1s 345ms/step
2/2 [==============================] - 1s 366ms/step
2/2 [==============================] - 1s 338ms/step
2/2 [==============================] - 1s 393ms/step
2/2 [==============================] - 1s 326ms/step
2/2 [==============================] - 1s 324ms/step
2/2 [==============================] - 1s 341ms/step
2/2 [==============================] - 1s 312ms/step
2/2 [==============================] - 1s 352ms/step
2/2 [==============================] - 1s 384ms/step
2/2 [==============================] - 1s 368ms/step
50 [D loss real: -44420.5078125] [D loss fake: -59.25751495361328] [G loss: -18964.833984375]
1/1 [==============================] - 0s 91ms/step
Generated image shape: (5, 224, 224, 3)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```



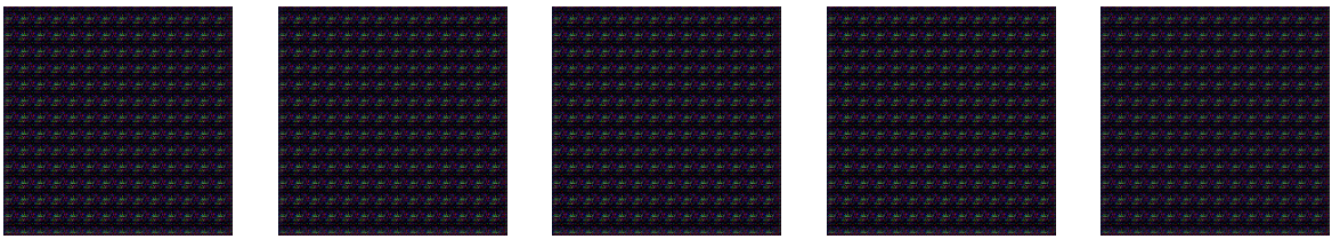
```
2/2 [==============================] - 1s 324ms/step
2/2 [==============================] - 1s 335ms/step
```

```
2/2 [==============================] - 1s 323ms/step
2/2 [==============================] - 1s 359ms/step
2/2 [==============================] - 1s 319ms/step
2/2 [==============================] - 1s 311ms/step
2/2 [==============================] - 1s 314ms/step
2/2 [==============================] - 1s 380ms/step
2/2 [==============================] - 1s 320ms/step
2/2 [==============================] - 1s 304ms/step
2/2 [==============================] - 1s 388ms/step
2/2 [==============================] - 1s 315ms/step
2/2 [==============================] - 1s 318ms/step
2/2 [==============================] - 1s 389ms/step
2/2 [==============================] - 1s 326ms/step
2/2 [==============================] - 1s 314ms/step
2/2 [==============================] - 1s 380ms/step
2/2 [==============================] - 1s 354ms/step
2/2 [==============================] - 1s 320ms/step
2/2 [==============================] - 1s 362ms/step
2/2 [==============================] - 1s 361ms/step
2/2 [==============================] - 1s 328ms/step
2/2 [==============================] - 1s 329ms/step
2/2 [==============================] - 1s 314ms/step
2/2 [==============================] - 1s 369ms/step
2/2 [==============================] - 1s 442ms/step
2/2 [==============================] - 1s 323ms/step
2/2 [==============================] - 1s 323ms/step
2/2 [==============================] - 1s 335ms/step
2/2 [==============================] - 1s 327ms/step
2/2 [==============================] - 1s 302ms/step
2/2 [==============================] - 1s 331ms/step
2/2 [==============================] - 1s 383ms/step
2/2 [==============================] - 1s 315ms/step
2/2 [==============================] - 1s 385ms/step
2/2 [==============================] - 1s 327ms/step
2/2 [==============================] - 1s 303ms/step
2/2 [==============================] - 1s 337ms/step
2/2 [==============================] - 1s 328ms/step
2/2 [==============================] - 1s 353ms/step
2/2 [==============================] - 1s 363ms/step
2/2 [==============================] - 1s 318ms/step
2/2 [==============================] - 1s 313ms/step
2/2 [==============================] - 1s 342ms/step
2/2 [==============================] - 1s 306ms/step
2/2 [==============================] - 1s 312ms/step
2/2 [==============================] - 1s 415ms/step
2/2 [==============================] - 1s 376ms/step
2/2 [==============================] - 1s 315ms/step
2/2 [==============================] - 1s 387ms/step
100 [D loss real: -76098.015625] [D loss fake: -99.89936828613281] [G loss: -34944.4453125]
1/1 [==============================] - 0s 93ms/step
Generated image shape: (5, 224, 224, 3)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```



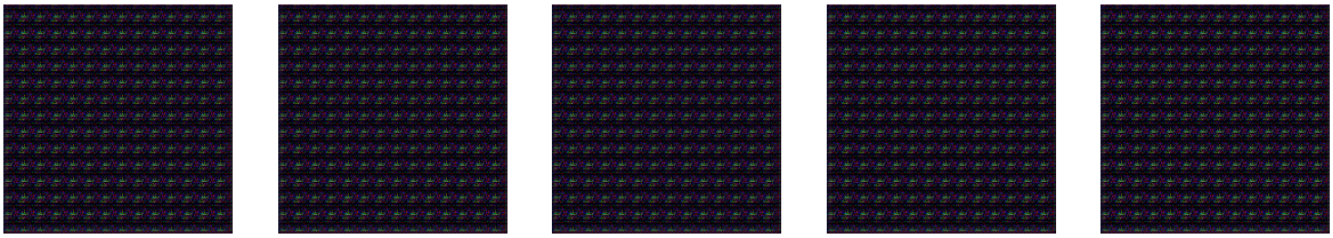
```
2/2 [==============================] - 1s 373ms/step
2/2 [==============================] - 1s 318ms/step
2/2 [==============================] - 1s 402ms/step
2/2 [==============================] - 1s 314ms/step
2/2 [==============================] - 1s 370ms/step
2/2 [==============================] - 1s 319ms/step
2/2 [==============================] - 1s 315ms/step
2/2 [==============================] - 1s 365ms/step
2/2 [==============================] - 1s 394ms/step
2/2 [==============================] - 1s 305ms/step
2/2 [==============================] - 1s 304ms/step
2/2 [==============================] - 1s 382ms/step
2/2 [==============================] - 1s 332ms/step
2/2 [==============================] - 1s 407ms/step
2/2 [==============================] - 1s 394ms/step
2/2 [==============================] - 1s 379ms/step
2/2 [==============================] - 1s 332ms/step
2/2 [==============================] - 1s 305ms/step
2/2 [==============================] - 1s 359ms/step
2/2 [==============================] - 1s 321ms/step
2/2 [==============================] - 1s 345ms/step
2/2 [==============================] - 1s 320ms/step
2/2 [==============================] - 1s 372ms/step
```

```
2/2 [==============================] - 1s 381ms/step
2/2 [==============================] - 1s 386ms/step
2/2 [==============================] - 1s 318ms/step
2/2 [==============================] - 1s 316ms/step
2/2 [==============================] - 1s 313ms/step
2/2 [==============================] - 1s 324ms/step
2/2 [==============================] - 1s 335ms/step
2/2 [==============================] - 1s 317ms/step
2/2 [==============================] - 1s 322ms/step
2/2 [==============================] - 1s 336ms/step
2/2 [==============================] - 1s 375ms/step
2/2 [==============================] - 1s 328ms/step
2/2 [==============================] - 1s 382ms/step
2/2 [==============================] - 1s 304ms/step
2/2 [==============================] - 1s 329ms/step
2/2 [==============================] - 1s 350ms/step
2/2 [==============================] - 1s 337ms/step
2/2 [==============================] - 1s 320ms/step
2/2 [==============================] - 1s 337ms/step
2/2 [==============================] - 1s 320ms/step
2/2 [==============================] - 1s 313ms/step
2/2 [==============================] - 1s 370ms/step
2/2 [==============================] - 1s 329ms/step
2/2 [==============================] - 1s 319ms/step
2/2 [==============================] - 1s 382ms/step
2/2 [==============================] - 1s 370ms/step
2/2 [==============================] - 1s 367ms/step
150 [D loss real: -98452.0390625] [D loss fake: -126.76513671875] [G loss: -47652.6875]
1/1 [==============================] - 0s 111ms/step
Generated image shape: (5, 224, 224, 3)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```

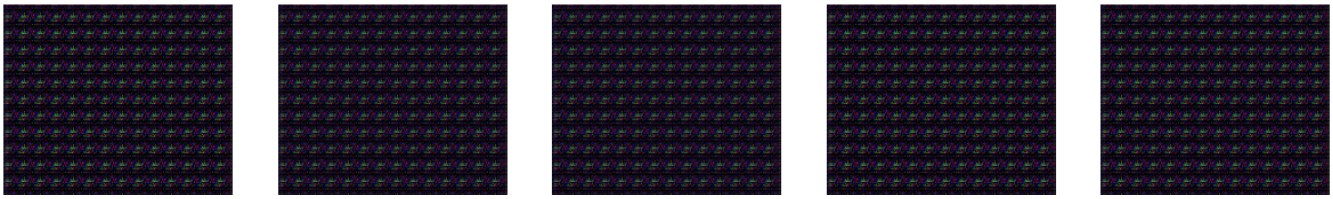


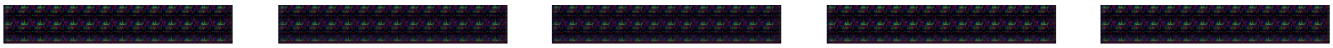
```
2/2 [==============================] - 1s 405ms/step
2/2 [==============================] - 1s 327ms/step
2/2 [==============================] - 1s 302ms/step
2/2 [==============================] - 1s 466ms/step
2/2 [==============================] - 1s 325ms/step
2/2 [==============================] - 1s 306ms/step
2/2 [==============================] - 1s 408ms/step
2/2 [==============================] - 1s 369ms/step
2/2 [==============================] - 1s 376ms/step
2/2 [==============================] - 1s 385ms/step
2/2 [==============================] - 1s 312ms/step
2/2 [==============================] - 1s 332ms/step
2/2 [==============================] - 1s 346ms/step
2/2 [==============================] - 1s 321ms/step
2/2 [==============================] - 1s 356ms/step
2/2 [==============================] - 1s 343ms/step
2/2 [==============================] - 1s 312ms/step
2/2 [==============================] - 1s 309ms/step
2/2 [==============================] - 1s 335ms/step
2/2 [==============================] - 1s 417ms/step
2/2 [==============================] - 1s 322ms/step
2/2 [==============================] - 1s 347ms/step
2/2 [==============================] - 1s 375ms/step
2/2 [==============================] - 1s 368ms/step
2/2 [==============================] - 1s 362ms/step
2/2 [==============================] - 1s 309ms/step
2/2 [==============================] - 1s 316ms/step
2/2 [==============================] - 1s 345ms/step
2/2 [==============================] - 1s 370ms/step
2/2 [==============================] - 1s 323ms/step
2/2 [==============================] - 1s 393ms/step
2/2 [==============================] - 1s 326ms/step
2/2 [==============================] - 1s 305ms/step
2/2 [==============================] - 1s 336ms/step
2/2 [==============================] - 1s 327ms/step
2/2 [==============================] - 1s 319ms/step
2/2 [==============================] - 1s 349ms/step
2/2 [==============================] - 1s 316ms/step
2/2 [==============================] - 1s 315ms/step
2/2 [==============================] - 1s 332ms/step
2/2 [==============================] - 1s 297ms/step
2/2 [==============================] - 1s 314ms/step
2/2 [==============================] - 1s 374ms/step
2/2 [==============================] - 1s 326ms/step
2/2 [==============================] - 1s 325ms/step
```

```
2/2 [==============================] - 1s 341ms/step
2/2 [==============================] - 1s 375ms/step
2/2 [==============================] - 1s 293ms/step
2/2 [==============================] - 1s 409ms/step
2/2 [==============================] - 1s 317ms/step
200 [D loss real: -99294.7265625] [D loss fake: -136.42401123046875] [G loss: -51170.421875]
1/1 [==============================] - 0s 95ms/step
Generated image shape: (5, 224, 224, 3)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```



```
2/2 [==============================] - 1s 370ms/step
2/2 [==============================] - 1s 401ms/step
2/2 [==============================] - 1s 314ms/step
2/2 [==============================] - 1s 355ms/step
2/2 [==============================] - 1s 364ms/step
2/2 [==============================] - 1s 309ms/step
2/2 [==============================] - 1s 401ms/step
2/2 [==============================] - 1s 382ms/step
2/2 [==============================] - 1s 311ms/step
2/2 [==============================] - 1s 310ms/step
2/2 [==============================] - 1s 392ms/step
2/2 [==============================] - 1s 337ms/step
2/2 [==============================] - 1s 322ms/step
2/2 [==============================] - 1s 347ms/step
2/2 [==============================] - 1s 360ms/step
2/2 [==============================] - 1s 333ms/step
2/2 [==============================] - 1s 349ms/step
2/2 [==============================] - 1s 318ms/step
2/2 [==============================] - 1s 316ms/step
2/2 [==============================] - 1s 336ms/step
2/2 [==============================] - 1s 359ms/step
2/2 [==============================] - 1s 375ms/step
2/2 [==============================] - 1s 355ms/step
2/2 [==============================] - 1s 311ms/step
2/2 [==============================] - 1s 380ms/step
2/2 [==============================] - 1s 347ms/step
2/2 [==============================] - 1s 315ms/step
2/2 [==============================] - 1s 336ms/step
2/2 [==============================] - 1s 345ms/step
2/2 [==============================] - 1s 316ms/step
2/2 [==============================] - 1s 327ms/step
2/2 [==============================] - 1s 362ms/step
2/2 [==============================] - 1s 383ms/step
2/2 [==============================] - 1s 324ms/step
2/2 [==============================] - 1s 390ms/step
2/2 [==============================] - 1s 367ms/step
2/2 [==============================] - 1s 320ms/step
2/2 [==============================] - 1s 355ms/step
2/2 [==============================] - 1s 380ms/step
2/2 [==============================] - 1s 370ms/step
2/2 [==============================] - 1s 330ms/step
2/2 [==============================] - 1s 361ms/step
2/2 [==============================] - 1s 368ms/step
2/2 [==============================] - 1s 380ms/step
2/2 [==============================] - 1s 387ms/step
2/2 [==============================] - 1s 327ms/step
2/2 [==============================] - 1s 334ms/step
2/2 [==============================] - 1s 365ms/step
2/2 [==============================] - 1s 351ms/step
2/2 [==============================] - 1s 344ms/step
250 [D loss real: -100131.453125] [D loss fake: -138.494384765625] [G loss: -54224.421875]
1/1 [==============================] - 0s 109ms/step
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
Generated image shape: (5, 224, 224, 3)
```





```
2/2 [==============================] - 1s 333ms/step
2/2 [==============================] - 1s 350ms/step
2/2 [==============================] - 1s 359ms/step
2/2 [==============================] - 1s 372ms/step
2/2 [==============================] - 1s 373ms/step
2/2 [==============================] - 1s 355ms/step
2/2 [==============================] - 1s 328ms/step
2/2 [==============================] - 1s 316ms/step
2/2 [==============================] - 1s 342ms/step
2/2 [==============================] - 1s 315ms/step
2/2 [==============================] - 1s 308ms/step
2/2 [==============================] - 1s 337ms/step
2/2 [==============================] - 1s 326ms/step
2/2 [==============================] - 1s 316ms/step
2/2 [==============================] - 1s 340ms/step
2/2 [==============================] - 1s 313ms/step
2/2 [==============================] - 1s 321ms/step
2/2 [==============================] - 1s 334ms/step
2/2 [==============================] - 1s 334ms/step
2/2 [==============================] - 1s 341ms/step
2/2 [==============================] - 1s 349ms/step
2/2 [==============================] - 1s 329ms/step
2/2 [==============================] - 1s 312ms/step
2/2 [==============================] - 1s 336ms/step
2/2 [==============================] - 1s 304ms/step
2/2 [==============================] - 1s 335ms/step
2/2 [==============================] - 1s 362ms/step
2/2 [==============================] - 1s 343ms/step
2/2 [==============================] - 1s 314ms/step
2/2 [==============================] - 1s 342ms/step
2/2 [==============================] - 1s 311ms/step
2/2 [==============================] - 1s 318ms/step
2/2 [==============================] - 1s 325ms/step
2/2 [==============================] - 1s 321ms/step
2/2 [==============================] - 1s 316ms/step
2/2 [==============================] - 1s 335ms/step
2/2 [==============================] - 1s 313ms/step
2/2 [==============================] - 1s 305ms/step
2/2 [==============================] - 1s 369ms/step
2/2 [==============================] - 1s 322ms/step
2/2 [==============================] - 1s 322ms/step
2/2 [==============================] - 1s 350ms/step
2/2 [==============================] - 1s 344ms/step
2/2 [==============================] - 1s 324ms/step
2/2 [==============================] - 1s 331ms/step
2/2 [==============================] - 1s 358ms/step
2/2 [==============================] - 1s 310ms/step
2/2 [==============================] - 1s 343ms/step
2/2 [==============================] - 1s 328ms/step
```

```
# Function to calculate FID score
def calculate_fid(real_images, fake_images):
    # Resize images to 299x299 for InceptionV3
    real_images_resized = tf.image.resize(real_images, (299, 299))
    fake_images_resized = tf.image.resize(fake_images, (299, 299))

    # Preprocess images (for InceptionV3)
    real_images_resized = tf.keras.applications.inception_v3.preprocess_input(real_images_resized)
    fake_images_resized = tf.keras.applications.inception_v3.preprocess_input(fake_images_resized)

    # Load the InceptionV3 model pre-trained on ImageNet
    inception_model = InceptionV3(include_top=False, pooling='avg')

    # Get feature representations
    real_features = inception_model.predict(real_images_resized)
    fake_features = inception_model.predict(fake_images_resized)

    # Calculate the FID
    fid = calculate_frechet_inception_distance(real_features, fake_features)
    return fid

# Function to calculate FID
def calculate_frechet_inception_distance(real_features, fake_features):
    mean_real, cov_real = np.mean(real_features, axis=0), np.cov(real_features, rowvar=False)
    mean_fake, cov_fake = np.mean(fake_features, axis=0), np.cov(fake_features, rowvar=False)
    diff = mean_real - mean_fake
    covmean = sqrtm(cov_real.dot(cov_fake))
    fid = np.sum(diff**2) + np.trace(cov_real) + np.trace(cov_fake) - 2 * np.trace(covmean)
    return fid



# Save the generator and critic models
generator.save('/content/drive/MyDrive/wgan_generator.h5')
critic.save('/content/drive/MyDrive/wgan_critic.h5')
combined.save('/content/drive/MyDrive/wgan_combined.h5')

# Assuming class_images contains the real images for evaluation
real_images = class_images[:5]  # Take the first 5 real images for the comparison

# Generate random noise and create 5 images
noise = np.random.normal(0, 1, (5, 100))  # Random noise of shape (5, 100)
generated_images = generator.predict(noise)

# Rescale images to [0, 1] if they were scaled to [-1, 1]
generated_images = 0.5 * generated_images + 0.5

# Set up the plot to display images horizontally
fig, axes = plt.subplots(1, 5, figsize=(20, 5))  # 1 row, 5 columns for horizontal alignment

# Save the generated images
for i in range(5):
    ax = axes[i]  # Get the i-th axis
    ax.imshow(generated_images[i])  # Display the generated image
    ax.axis('off')  # Hide the axis
    ax.set_title(f"Generated Image {i+1}")  # Optional: add a title for each image

# Save the figure with all 5 images horizontally arranged
plt.tight_layout()
plt.savefig('/content/drive/MyDrive/generated_images_horizontal.png')  # Save as a single image
plt.show()

# Optionally, calculate FID and IS (FID calculation part needs real_images and generated_images)
fid = calculate_fid(real_images, generated_images)
print("FID Score:", fid)
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` wil
1/1 [==============================] - 0s 92ms/step
```


Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

```
1/1 [==============================] - 0s 138ms/step
```

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Double-click (or enter) to edit

Double-click (or enter) to edit