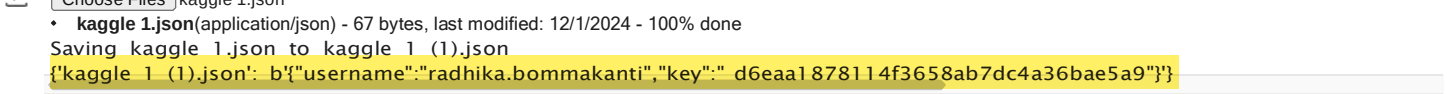
```
!pip install tensorflow numpy matplotlib
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorfl
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/loca
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.68.0)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.1)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.5.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorf
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensor
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.13.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tens
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tens
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->ten
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensor
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->ten
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboa
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->te
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->kera
```

```
from google.colab import files
files.upload()  # Select your kaggle.json file
```




```
Saving kaggle 1.json to kaggle 1 (1).json
{'kaggle 1 (1).json': b'{"username":"radhika.bommakanti","key":"d6eaa1878114f3658ab7dc4a36bae5a9"}'}
```

```
!pip install kaggle
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.8.30)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.6)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.10)
mkdir: cannot create directory ‘/root/.kaggle’: File exists
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

```
!kaggle datasets download -d jessicali9530/celeba-dataset
!unzip celeba-dataset.zip -d /content/dataset
```

```
  inflating: /content/dataset/img_align_celeba/img_align_celeba/201558.jpg
  inflating: /content/dataset/img_align_celeba/img_align_celeba/201559.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201560.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201561.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201562.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201563.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201564.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201565.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201566.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201567.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201568.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201569.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201570.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201571.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201572.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201573.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201574.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201575.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201576.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201577.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201578.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201579.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201580.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201581.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201582.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201583.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201584.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201585.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201586.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201587.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201588.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201589.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201590.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201591.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201592.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201593.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201594.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201595.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201596.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201597.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201598.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201599.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201600.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201601.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201602.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201603.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201604.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201605.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201606.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201607.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201608.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201609.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201610.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201611.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201612.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201613.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201614.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201615.jpg
 inflating: /content/dataset/img_align_celeba/img_align_celeba/201616.jpg
```

```
dataset_path = '/content/img_align_celeba'
```

```
import os

dataset_path = '/content/dataset'
if os.path.exists(dataset_path):
    print("Dataset folder exists. Files:", os.listdir(dataset_path)[:5])
else:
    print("Dataset folder not found.")
```

```
Dataset folder exists. Files: ['list_landmarks_align_celeba.csv', 'list_bbox_celeba.csv', 'img_align_celeba', 'list_attr_cel
```

```
import numpy as np
from PIL import Image
import os

def preprocess_images(image_dir, output_size=(256, 256), max_images=10000):
```

```
    images = []
    for i, file in enumerate(os.listdir(image_dir)):
        if i >= max_images:
            break
        img_path = os.path.join(image_dir, file)
        try:
            img = Image.open(img_path).convert("RGB")
            img = img.resize(output_size)
            img = np.array(img) / 127.5 - 1.0 # Normalize to [-1, 1]
            images.append(img)
        except:
            continue
    return np.array(images)

dataset_path = '/content/dataset/img_align_celeba'
images = preprocess_images(dataset_path, output_size=(256, 256))
print(f"Processed {len(images)} images with shape {images.shape}.")
```

```
Processed 0 images with shape (0,).
```

```
from tensorflow.keras import layers, models

def build_generator(input_dim=100, initial_size=(4, 4, 512)):
    model = models.Sequential()
    model.add(layers.Dense(np.prod(initial_size), input_dim=input_dim))
    model.add(layers.Reshape(initial_size))
    model.add(layers.Conv2DTranspose(256, (4, 4), strides=(2, 2), padding='same', activation='relu'))
    model.add(layers.Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same', activation='relu'))
    model.add(layers.Conv2D(3, (3, 3), padding='same', activation='tanh'))
    return model

generator = build_generator()
generator.summary()
```

```
Model: "sequential_2"
┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 8192)           │       827,392 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ reshape_1 (Reshape)             │ (None, 4, 4, 512)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_transpose_2 (Conv2DTranspose) │ (None, 8, 8, 256)      │     2,097,408 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_transpose_3 (Conv2DTranspose) │ (None, 16, 16, 128)    │       524,416 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_3 (Conv2D)               │ (None, 16, 16, 3)      │         3,459 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
Total params: 3,452,675 (13.17 MB)
Trainable params: 3,452,675 (13.17 MB)
Non-trainable params: 0 (0.00 B)
```

```
def build_discriminator(input_shape=(4, 4, 3)):
    model = models.Sequential()
    model.add(layers.Conv2D(64, (3, 3), padding='same', input_shape=input_shape, activation='relu'))
    model.add(layers.Conv2D(128, (4, 4), strides=(2, 2), padding='same', activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(1, activation='sigmoid'))
    return model

discriminator = build_discriminator()
discriminator.summary()
```

```
Model: "sequential_3"
┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_4 (Conv2D)               │ (None, 4, 4, 64)       │         1,792 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_5 (Conv2D)               │ (None, 2, 2, 128)      │       131,200 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_1 (Flatten)             │ (None, 512)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 1)              │           513 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
Total params: 133,505 (521.50 KB)
Trainable params: 133,505 (521.50 KB)
Non-trainable params: 0 (0.00 B)
```

```
import tensorflow as tf

def wasserstein_loss(y_true, y_pred):
    return -tf.reduce_mean(y_true * y_pred)


optimizer = tf.keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5)


def progressive_training(generator, discriminator, dataset, epochs_per_stage=5):
    resolutions = [(4, 4), (8, 8), (16, 16), (32, 32), (64, 64), (128, 128), (256, 256)]
    for res in resolutions:
        print(f"Training at resolution {res}")
        # Resize images for current resolution
        resized_images = np.array([tf.image.resize(img, res).numpy() for img in dataset])
        train_progan(generator, discriminator, resized_images, epochs=epochs_per_stage)


def train_progan(generator, discriminator, dataset, epochs=10, batch_size=32):
    for epoch in range(epochs):
        for i in range(0, len(dataset), batch_size):
            real_images = dataset[i:i+batch_size]
            noise = tf.random.normal([batch_size, 100])
            fake_images = generator(noise)

            # Train Discriminator
            with tf.GradientTape() as tape:
                real_loss = wasserstein_loss(tf.ones((batch_size, 1)), discriminator(real_images))
                fake_loss = wasserstein_loss(tf.zeros((batch_size, 1)), discriminator(fake_images))
                d_loss = real_loss + fake_loss
            grads = tape.gradient(d_loss, discriminator.trainable_weights)
            optimizer.apply_gradients(zip(grads, discriminator.trainable_weights))

            # Train Generator
            with tf.GradientTape() as tape:
                gen_images = generator(noise)
                g_loss = wasserstein_loss(tf.ones((batch_size, 1)), discriminator(gen_images))
            grads = tape.gradient(g_loss, generator.trainable_weights)
            optimizer.apply_gradients(zip(grads, generator.trainable_weights))

        print(f"Epoch {epoch+1}/{epochs}, D Loss: {d_loss.numpy()}, G Loss: {g_loss.numpy()}")


def train_progan(generator, discriminator, dataset, epochs=10, batch_size=32):
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5)

    for epoch in range(epochs):
        d_loss, g_loss = None, None # Initialize losses
        for i in range(0, len(dataset), batch_size):
            real_images = dataset[i:i+batch_size]
            noise = tf.random.normal([batch_size, 100])
            fake_images = generator(noise)

            # Train Discriminator
            with tf.GradientTape() as tape:
                try:
                    real_loss = wasserstein_loss(tf.ones((batch_size, 1)), discriminator(real_images))
                    fake_loss = wasserstein_loss(tf.zeros((batch_size, 1)), discriminator(fake_images))
                    d_loss = real_loss + fake_loss
                except Exception as e:
                    print(f"Error during discriminator training: {e}")
                    continue
            grads = tape.gradient(d_loss, discriminator.trainable_weights)
            optimizer.apply_gradients(zip(grads, discriminator.trainable_weights))

            # Train Generator
            with tf.GradientTape() as tape:
                try:
                    gen_images = generator(noise)
                    g_loss = wasserstein_loss(tf.ones((batch_size, 1)), discriminator(gen_images))
                except Exception as e:
                    print(f"Error during generator training: {e}")
                    continue
            grads = tape.gradient(g_loss, generator.trainable_weights)
            optimizer.apply_gradients(zip(grads, generator.trainable_weights))

        # Print the losses after each epoch
        d_loss_value = d_loss.numpy() if d_loss is not None else "N/A"
```

```
        g_loss_value = g_loss.numpy() if g_loss is not None else "N/A"
        print(f"Epoch {epoch+1}/{epochs}, D Loss: {d_loss_value}, G Loss: {g_loss_value}")


def progressive_training(generator, discriminator, dataset, epochs_per_stage=5):
    resolutions = [(4, 4), (8, 8), (16, 16), (32, 32), (64, 64), (128, 128), (256, 256)]
    for res in resolutions:
        print(f"Training at resolution {res}")
        # Resize images for current resolution
        resized_images = np.array([tf.image.resize(img, res).numpy() for img in dataset])
        print(f"Dataset resized to {res}. Shape: {resized_images.shape}")
        train_progan(generator, discriminator, resized_images, epochs=epochs_per_stage)


def train_progan(generator, discriminator, dataset, epochs=10, batch_size=32):
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5)


    for epoch in range(epochs):
        d_loss, g_loss = None, None # Initialize losses
        for i in range(0, len(dataset), batch_size):
            # Ensure variables are defined
            real_images = np.zeros((batch_size, *dataset.shape[1:]))
            fake_images = np.zeros((batch_size, *dataset.shape[1:]))

            try:
                # Prepare real images and generate fake images
                real_images = dataset[i:i+batch_size]
                noise = tf.random.normal([batch_size, 100])
                fake_images = generator(noise)

                print(f"Real images shape: {real_images.shape}")
                print(f"Fake images shape: {fake_images.shape}")
            except Exception as e:
                print(f"Error in preparing images: {e}")
                continue

            # Train Discriminator
            with tf.GradientTape() as tape:
                try:
                    real_loss = wasserstein_loss(tf.ones((batch_size, 1)), discriminator(real_images))
                    fake_loss = wasserstein_loss(tf.zeros((batch_size, 1)), discriminator(fake_images))
                    d_loss = real_loss + fake_loss
                except Exception as e:
                    print(f"Error during discriminator training: {e}")
                    continue
            grads = tape.gradient(d_loss, discriminator.trainable_weights)
            optimizer.apply_gradients(zip(grads, discriminator.trainable_weights))

            # Train Generator
            with tf.GradientTape() as tape:
                try:
                    gen_images = generator(noise)
                    g_loss = wasserstein_loss(tf.ones((batch_size, 1)), discriminator(gen_images))
                except Exception as e:
                    print(f"Error during generator training: {e}")
                    continue
            grads = tape.gradient(g_loss, generator.trainable_weights)
            optimizer.apply_gradients(zip(grads, generator.trainable_weights))

        # Print losses after each epoch
        d_loss_value = d_loss.numpy() if d_loss is not None else "N/A"
        g_loss_value = g_loss.numpy() if g_loss is not None else "N/A"
        print(f"Epoch {epoch+1}/{epochs}, D Loss: {d_loss_value}, G Loss: {g_loss_value}")


def train_progan(generator, discriminator, dataset, epochs=10, batch_size=32):
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5)

    for epoch in range(epochs):
        d_loss, g_loss = None, None # Initialize losses
        for i in range(0, len(dataset), batch_size):
            # Prepare real images and generate fake images
            try:
                real_images = dataset[i:i+batch_size]
                if real_images.shape[0] == 0:
                    print("Empty batch encountered. Skipping.")
                    continue
```

```
                noise = tf.random.normal([real_images.shape[0], 100])
                fake_images = generator(noise)

                print(f"Real images shape: {real_images.shape}")
                print(f"Fake images shape: {fake_images.shape}")
            except Exception as e:
                print(f"Error in preparing images: {e}")
                continue

            # Train Discriminator
            with tf.GradientTape() as tape:
                try:
                    real_loss = wasserstein_loss(tf.ones((real_images.shape[0], 1)), discriminator(real_images))
                    fake_loss = wasserstein_loss(tf.zeros((real_images.shape[0], 1)), discriminator(fake_images))
                    d_loss = real_loss + fake_loss
                except Exception as e:
                    print(f"Error during discriminator training: {e}")
                    continue
            grads = tape.gradient(d_loss, discriminator.trainable_weights)
            optimizer.apply_gradients(zip(grads, discriminator.trainable_weights))

            # Train Generator
            with tf.GradientTape() as tape:
                try:
                    gen_images = generator(noise)
                    g_loss = wasserstein_loss(tf.ones((real_images.shape[0], 1)), discriminator(gen_images))
                except Exception as e:
                    print(f"Error during generator training: {e}")
                    continue
            grads = tape.gradient(g_loss, generator.trainable_weights)
            optimizer.apply_gradients(zip(grads, generator.trainable_weights))

        # Print losses after each epoch
        d_loss_value = d_loss.numpy() if d_loss is not None else "N/A"
        g_loss_value = g_loss.numpy() if g_loss is not None else "N/A"
        print(f"Epoch {epoch+1}/{epochs}, D Loss: {d_loss_value}, G Loss: {g_loss_value}")


progressive_training(generator, discriminator, images, epochs_per_stage=5)
```

```
Training at resolution (4, 4)
Dataset resized to (4, 4). Shape: (0,)
Epoch 1/5, D Loss: N/A, G Loss: N/A
Epoch 2/5, D Loss: N/A, G Loss: N/A
Epoch 3/5, D Loss: N/A, G Loss: N/A
Epoch 4/5, D Loss: N/A, G Loss: N/A
Epoch 5/5, D Loss: N/A, G Loss: N/A
Training at resolution (8, 8)
Dataset resized to (8, 8). Shape: (0,)
Epoch 1/5, D Loss: N/A, G Loss: N/A
Epoch 2/5, D Loss: N/A, G Loss: N/A
Epoch 3/5, D Loss: N/A, G Loss: N/A
Epoch 4/5, D Loss: N/A, G Loss: N/A
Epoch 5/5, D Loss: N/A, G Loss: N/A
Training at resolution (16, 16)
Dataset resized to (16, 16). Shape: (0,)
Epoch 1/5, D Loss: N/A, G Loss: N/A
Epoch 2/5, D Loss: N/A, G Loss: N/A
Epoch 3/5, D Loss: N/A, G Loss: N/A
Epoch 4/5, D Loss: N/A, G Loss: N/A
Epoch 5/5, D Loss: N/A, G Loss: N/A
Training at resolution (32, 32)
Dataset resized to (32, 32). Shape: (0,)
Epoch 1/5, D Loss: N/A, G Loss: N/A
Epoch 2/5, D Loss: N/A, G Loss: N/A
Epoch 3/5, D Loss: N/A, G Loss: N/A
Epoch 4/5, D Loss: N/A, G Loss: N/A
Epoch 5/5, D Loss: N/A, G Loss: N/A
Training at resolution (64, 64)
Dataset resized to (64, 64). Shape: (0,)
Epoch 1/5, D Loss: N/A, G Loss: N/A
Epoch 2/5, D Loss: N/A, G Loss: N/A
Epoch 3/5, D Loss: N/A, G Loss: N/A
Epoch 4/5, D Loss: N/A, G Loss: N/A
Epoch 5/5, D Loss: N/A, G Loss: N/A
Training at resolution (128, 128)
Dataset resized to (128, 128). Shape: (0,)
Epoch 1/5, D Loss: N/A, G Loss: N/A
Epoch 2/5, D Loss: N/A, G Loss: N/A
Epoch 3/5, D Loss: N/A, G Loss: N/A
```

```
Epoch 4/5, D Loss: N/A, G Loss: N/A
Epoch 5/5, D Loss: N/A, G Loss: N/A
Training at resolution (256, 256)
Dataset resized to (256, 256). Shape: (0,)
Epoch 1/5, D Loss: N/A, G Loss: N/A
Epoch 2/5, D Loss: N/A, G Loss: N/A
Epoch 3/5, D Loss: N/A, G Loss: N/A
Epoch 4/5, D Loss: N/A, G Loss: N/A
Epoch 5/5, D Loss: N/A, G Loss: N/A
```

```
import matplotlib.pyplot as plt

def save_and_display_images(generator, epoch, output_dir="/content/generated_images"):
    os.makedirs(output_dir, exist_ok=True)
    noise = tf.random.normal([16, 100])
    gen_images = generator(noise)
    gen_images = (gen_images + 1) / 2  # Rescale to [0, 1]

    for i, img in enumerate(gen_images):
        plt.imshow(img.numpy())
        plt.axis('off')
        plt.savefig(f"{output_dir}/image_epoch_{epoch}_{i}.png")
        plt.show()

save_and_display_images(generator,  epoch=1)
```