```python
# `pip3 install assemblyai` (macOS)
# `pip install assemblyai` (Windows)

import assemblyai as aai

aai.settings.api_key = "dfc5fb97b35e4fde9f97900564047539"
transcriber = aai.Transcriber()

transcript =
transcriber.transcribe("https://storage.googleapis.com/aai-web-
samples/news.mp4")
# transcript = transcriber.transcribe("./my-local-audio-file.wav")

print(transcript.text)
```

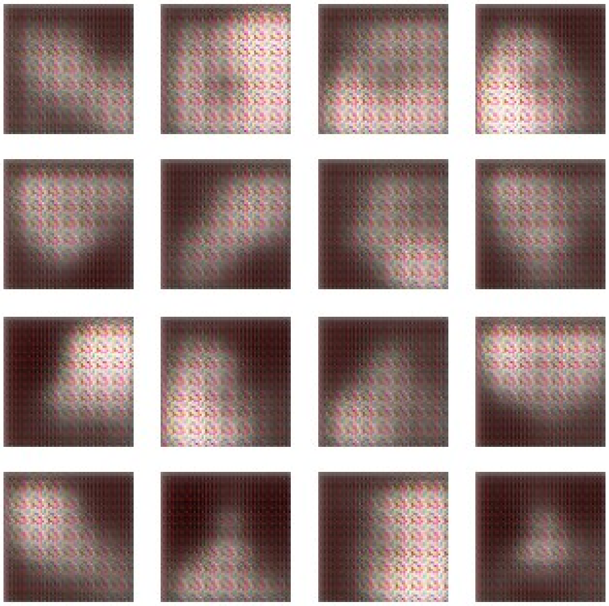
```
I'm David Curley at the Smithsonian Air and Space Museum, where we are
marking 50 years since man landed and walked on the moon in a lander
just like this one. We are going to show you some of the actual ABC
News coverage from 50 years ago. During that eight day mission of this
remarkable achievement, Apollo Eleven's lander, the Eagle, would be
the first man craft to land on the moon. For training, NASA came up
with an unusual contraption. Neil Armstrong actually had to eject from
it once, and then he had a couple of successful flights. ABC News
anchor at the time, 50 years ago, Frank Reynolds with a look at that
unusual trainer. Apollo Eleven Commander Neil Armstrong is at the
controls of a lunar landing training vehicle, testing the reaction
control jets. These thrusters stabilize the LEM during landing and
takeoff. The LLTV is designed to simulate the behavior of the LEM as
it lands in the moon's gravity. Lunar gravity is one 6th that of the
Earth's. Neil ArmstroNG flew one of these vehicles on May 6, 1968, and
that flight was nearly his last. Later reports by a NASA investigating
team said the crash, which we'll see soon, was caused by a loss of
fuel pressure compounded by a warning light that failed to work.
Armstrong's coolness under pressure saved himself and possibly months
of delay for the Apollo program. Later that same year, 1968, another
test pilot, Joseph Allegranti, also escaped from an LLTV just before
it crashed. The training vehicle then underwent several design
modifications and improvements. Engineers had to increase the
vehicle's rocket power to help stabilize the craft. That made the LLTV
less of a moon gravity simulator, but it improved pilot safety. On
June 16, 1969, Neil Armstrong flew the vehicle, sometimes called the
flying bedstead, to several perfect landings. The question was, how
does the machine fly? And the answer is that we're very pleased with
the way it flies. It's a significant improvement over the LLRV, which
we were flying here a year ago, and I think it does an excellent job
of actually capturing the handling characteristics of the lunar module
in landing maneuver. It's really a great deal different than any other
kind of aircraft that I've ever flown. The simulation of lunar gravity
has some aspects that make this type of flight sufficiently different
from anything else we've ever done to make this vehicle very
```

```
worthwhile. And I'm very pleased that I had the opportunity to get
some flights in it here just before the Apollo eleven flight.
```

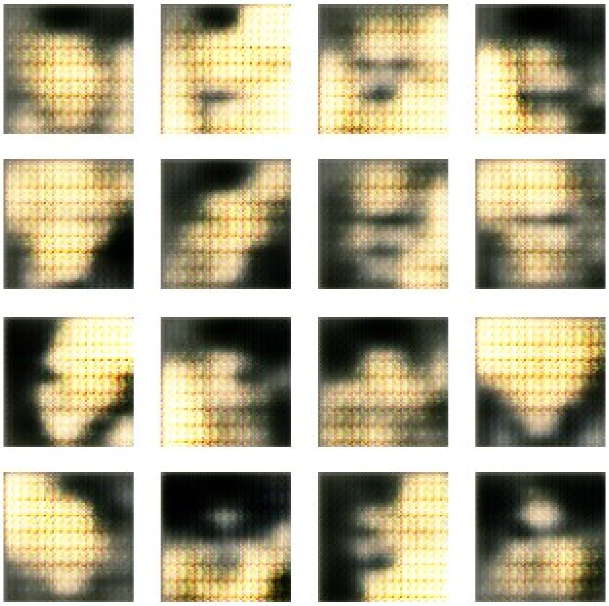
```
import tensorflow as tf
import os
from PIL import Image
import numpy as np

def preprocess_image(image):
    image = tf.image.resize(image, (64, 64))
    image = (image - 127.5) / 127.5  # Normalize to [-1, 1]
    return image

def load_data(data_dir):
    images = []
    for img_file in os.listdir(data_dir):
        img_path = os.path.join(data_dir, img_file)
        img = Image.open(img_path).convert('RGB')
        img = np.array(img)
        img = preprocess_image(img)
        images.append(img)
    dataset = tf.data.Dataset.from_tensor_slices(images).batch(128)
    return dataset

data_dir = 'CelebA-HQ-img'
dataset = load_data(data_dir)

def make_generator_model():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(8*8*256, use_bias=False,
input_shape=(100,)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.LeakyReLU())
    model.add(tf.keras.layers.Reshape((8, 8, 256)))
    model.add(tf.keras.layers.Conv2DTranspose(128, (5, 5), strides=(2,
2), padding='same', use_bias=False))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.LeakyReLU())
    model.add(tf.keras.layers.Conv2DTranspose(64, (5, 5), strides=(2,
2), padding='same', use_bias=False))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.LeakyReLU())
    model.add(tf.keras.layers.Conv2DTranspose(3, (5, 5), strides=(2,
2), padding='same', use_bias=False, activation='tanh'))
    return model

def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Conv2D(64, (5, 5), strides=(2, 2),
padding='same', input_shape=[64, 64, 3]))
```

```
    model.add(tf.keras.layers.LeakyReLU())
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Conv2D(128, (5, 5), strides=(2, 2),
padding='same'))
    model.add(tf.keras.layers.LeakyReLU())
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(1))
    return model

generator = make_generator_model()
discriminator = make_discriminator_model()

cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    return real_loss + fake_loss

generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)

import os
import matplotlib.pyplot as plt

EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

seed = tf.random.normal([num_examples_to_generate, noise_dim])

@tf.function
def train_step(images):
    noise = tf.random.normal([128, noise_dim])

    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 = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss,
generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, 
discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, 
generator.trainable_variables))

discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator
, discriminator.trainable_variables))

def train(dataset, epochs):
    for epoch in range(epochs):
        for image_batch in dataset:
            train_step(image_batch)
        print(f'Epoch {epoch+1} completed')
        generate_and_save_images(generator, epoch + 1, seed)

def generate_and_save_images(model, epoch, test_input):
    predictions = model(test_input, 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] * 127.5 + 
127.5).numpy().astype("uint8"))
        plt.axis('off')
    plt.savefig(f'image_at_epoch_{epoch:04d}.png')
    plt.show()

train(dataset, EPOCHS)
```

```
C:\Users\Medhavi\anaconda3\Lib\site-packages\keras\src\layers\core\
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` 
argument to a layer. When using Sequential models, prefer using an 
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, 
**kwargs)
C:\Users\Medhavi\anaconda3\Lib\site-packages\keras\src\layers\
convolutional\base_conv.py:107: UserWarning: Do not pass an 
`input_shape`/`input_dim` argument to a layer. When using Sequential 
models, prefer using an `Input(shape)` object as the first layer in 
the model instead.
  super().__init__(activity_regularizer=activity_regularizer, 
**kwargs)

Epoch 1 completed
```

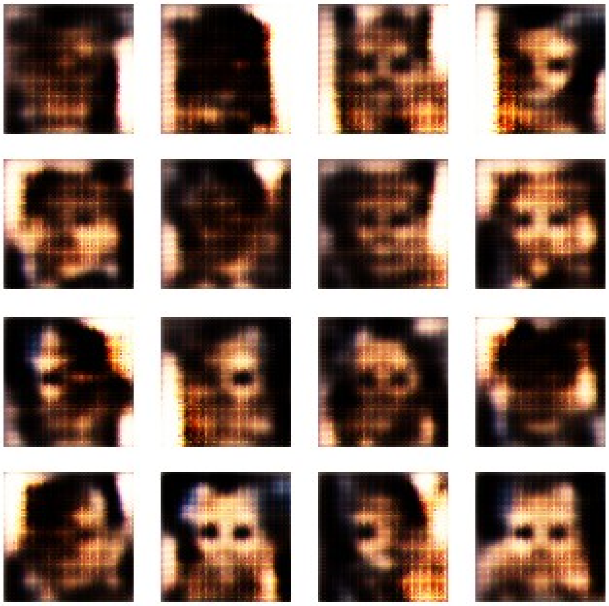


```
Epoch 2 completed
```



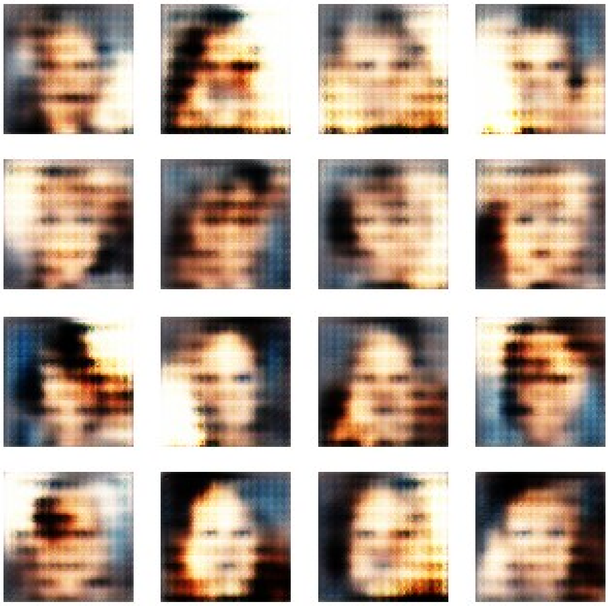
```
Epoch 3 completed
```

```
Epoch 4 completed
```

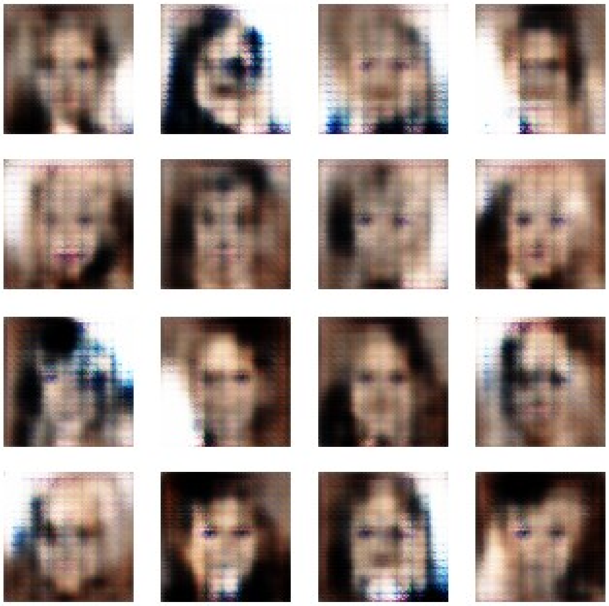


```
Epoch 5 completed
```

```
Epoch 6 completed
```



```
Epoch 7 completed
```



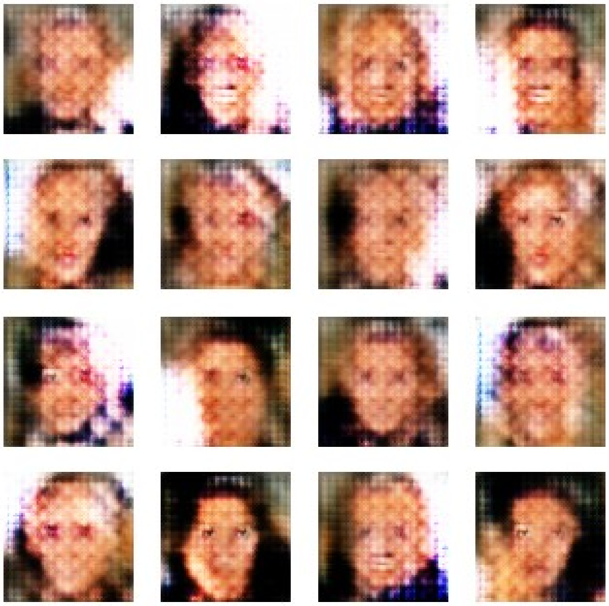
```
Epoch 8 completed
```

```
Epoch 9 completed
```

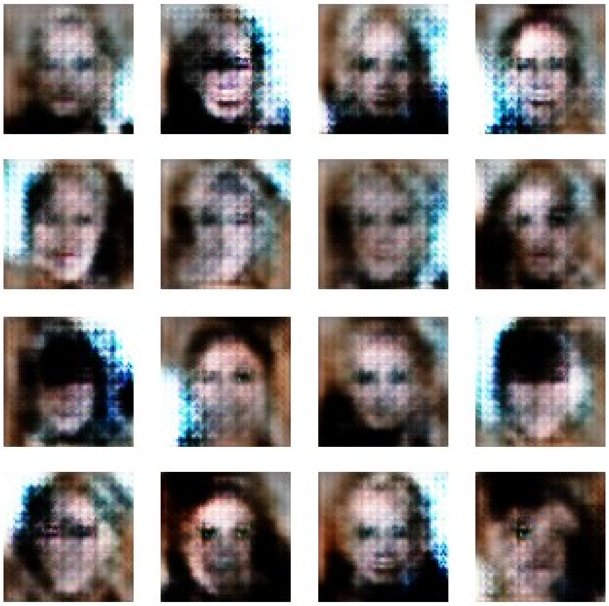
```
Epoch 10 completed
```

```
Epoch 11 completed
```

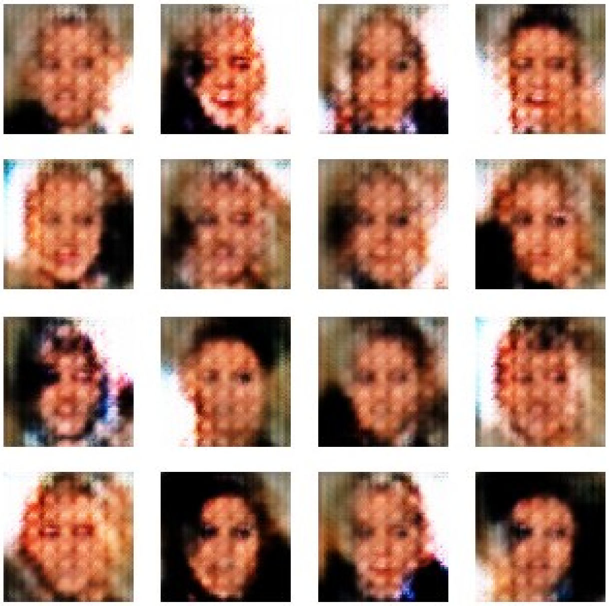
```
Epoch 12 completed
```



```
Epoch 13 completed
```



```
Epoch 14 completed
```



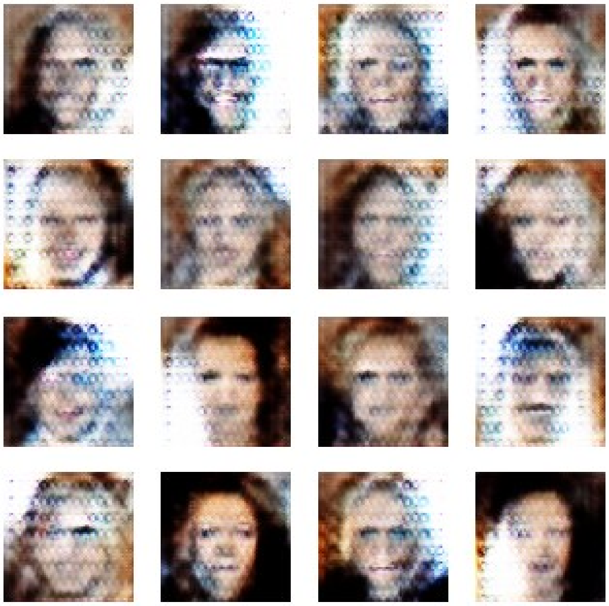
```
Epoch 15 completed
```

```
Epoch 16 completed
```

```
Epoch 17 completed
```

```
Epoch 18 completed
```

```
Epoch 19 completed
```

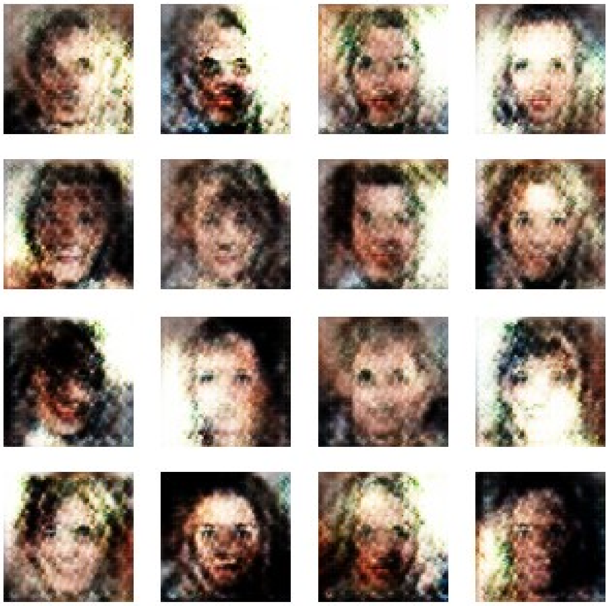


```
Epoch 20 completed
```

```
Epoch 21 completed
```

```
Epoch 22 completed
```

```
Epoch 23 completed
```



```
Epoch 24 completed
```

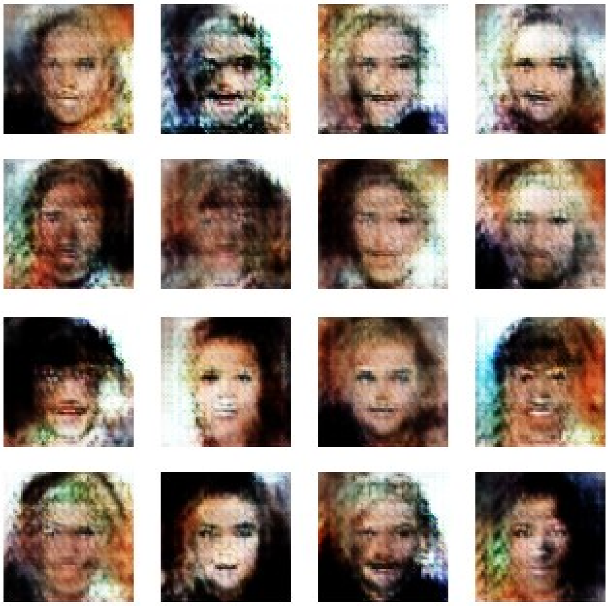
```
Epoch 25 completed
```

```
Epoch 26 completed
```

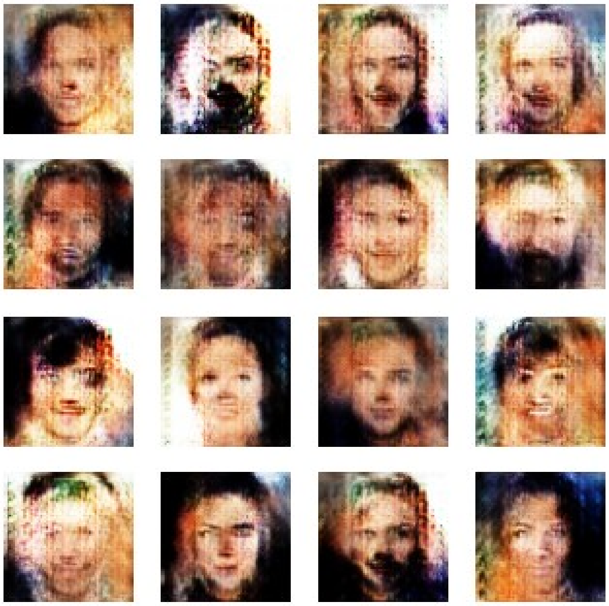
```
Epoch 27 completed
```

```
Epoch 28 completed
```

```
Epoch 29 completed
```



```
Epoch 30 completed
```



```
Epoch 31 completed
```

```
Epoch 32 completed
```

```
Epoch 33 completed
```

```
Epoch 34 completed
```

```
Epoch 35 completed
```

```
Epoch 36 completed
```

```
Epoch 37 completed
```

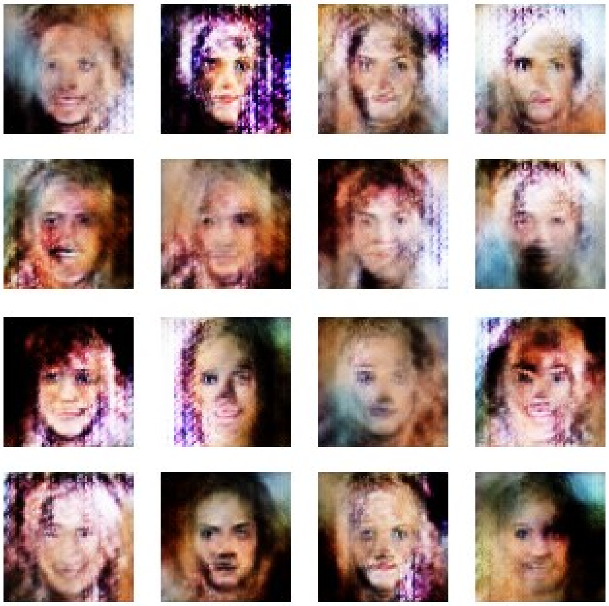
```
Epoch 38 completed
```

```
Epoch 39 completed
```

```
Epoch 40 completed
```

```
Epoch 41 completed
```

```
Epoch 42 completed
```

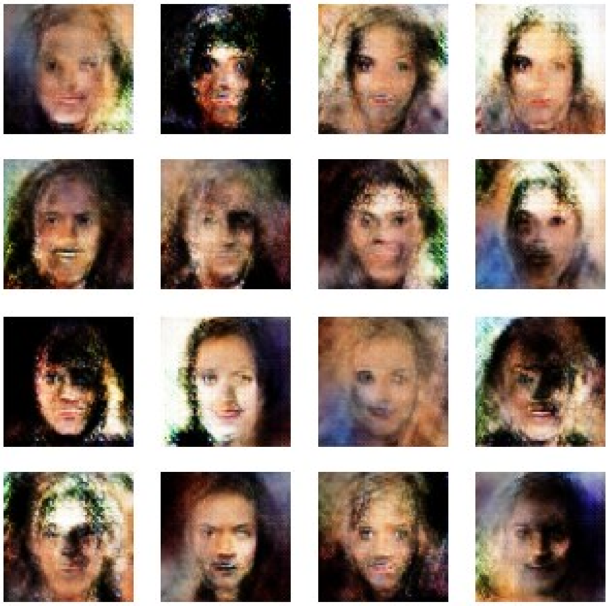


```
Epoch 43 completed
```

```
Epoch 44 completed
```

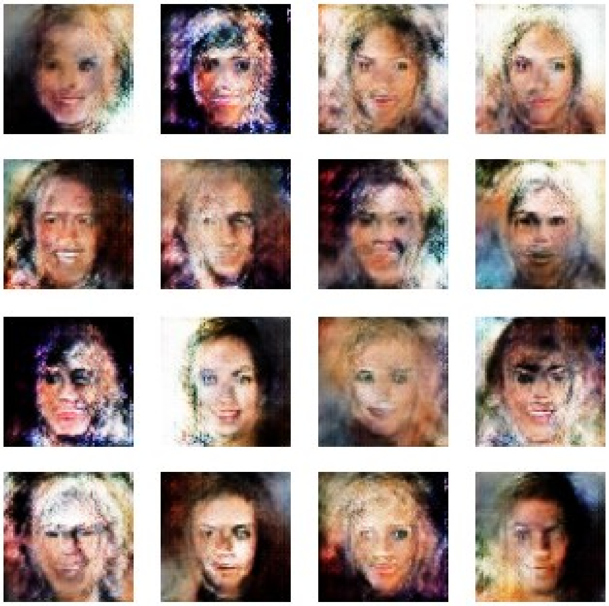
```
Epoch 45 completed
```

```
Epoch 46 completed
```

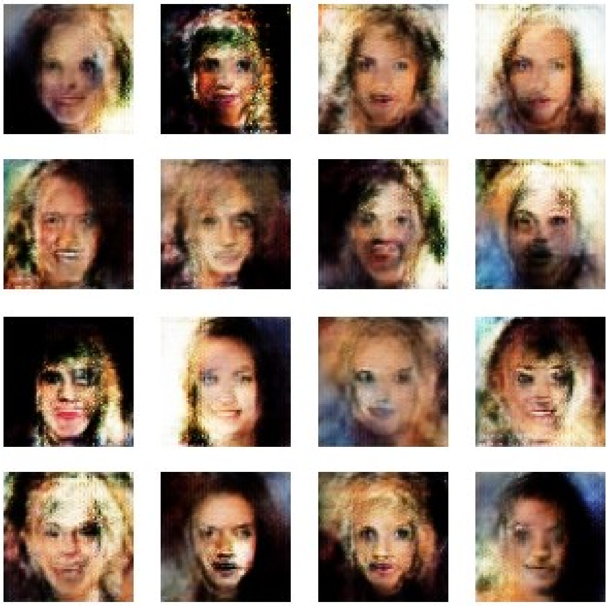


```
Epoch 47 completed
```

```
Epoch 48 completed
```



```
Epoch 49 completed
```



```
Epoch 50 completed
```

```python
import tensorflow as tf
from tensorflow.keras import layers
import tensorflow_datasets as tfds

# Load and preprocess the CelebA dataset
def preprocess_image(image, label):
    image = tf.image.resize(image, [64, 64])
    image = (image - 127.5) / 127.5  # Normalize to [-1, 1]
    return image
def load_data(data_dir):
    images = []
    for img_file in os.listdir(data_dir):
        img_path = os.path.join(data_dir, img_file)
        img = Image.open(img_path).convert('RGB')
        img = np.array(img)
        img = preprocess_image(img)
        images.append(img)
    dataset = tf.data.Dataset.from_tensor_slices(images).batch(128)
    return dataset

data_dir = 'CelebA-HQ-img'
dataset = load_data(data_dir)


# Define the generator model
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(8*8*256, use_bias=False,
input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((8, 8, 256)))
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(2, 2),
padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),
padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(3, (5, 5), strides=(2, 2),
padding='same', use_bias=False, activation='tanh'))

    return model

# Define the discriminator model
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2),
padding='same', input_shape=[64, 64, 3]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2),
padding='same'))
```

```
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model

# Define loss and optimizers
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

generator = make_generator_model()
discriminator = make_discriminator_model()

generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)

# Training loop
@tf.function
def train_step(images):
    noise = tf.random.normal([256, 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 = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss,
generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss,
discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables))

discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator
, discriminator.trainable_variables))
```

```
def train(dataset, epochs):
    for epoch in range(epochs):
        for image_batch in dataset:
            train_step(image_batch)

        # Produce images for the GIF as we go
        display.clear_output(wait=True)
        generate_and_save_images(generator, epoch + 1, seed)

    # Generate after the final epoch
    display.clear_output(wait=True)
    generate_and_save_images(generator, epochs, seed)

def generate_and_save_images(model, epoch, test_input):
    predictions = model(test_input, 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] * 127.5 +
127.5).numpy().astype("uint8"))
        plt.axis('off')

    plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
    plt.show()

# Load the dataset
dataset = load_data()

# Train the model
train(dataset, epochs=50)

----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Cell In[6], line 22
     19     return dataset
     21 data_dir = 'CelebA-HQ-img'
---> 22 dataset = load_data(data_dir)
     25 # Define the generator model
     26 def make_generator_model():

Cell In[6], line 16, in load_data(data_dir)
     14     img = Image.open(img_path).convert('RGB')
     15     img = np.array(img)
---> 16     img = preprocess_image(img)
     17     images.append(img)
     18 dataset = 
```

`tf.data.Dataset.from_tensor_slices(images).batch(128)`

`TypeError: preprocess_image() missing 1 required positional argument: 'label'`

```
import torch
import torchvision.transforms as transforms
from torchvision.models import inception_v3
import numpy as np
from scipy.linalg import sqrtm
from torch.utils.data import DataLoader, Dataset
from PIL import Image
import os
from tqdm import tqdm

class ImageDataset(Dataset):
    def __init__(self, image_paths, transform=None):
        self.image_paths = image_paths
        self.transform = transform

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image = Image.open(self.image_paths[idx]).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image

def load_images(image_folder):
    image_paths = [os.path.join(image_folder, img) for img in
os.listdir(image_folder)]
    transform = transforms.Compose([
        transforms.Resize((299, 299)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
    ])
    dataset = ImageDataset(image_paths, transform=transform)
    return DataLoader(dataset, batch_size=32, shuffle=False)

def get_inception_activations(dataloader, model, device):
    model.eval()
    activations = []
    with torch.no_grad():
        for batch in tqdm(dataloader):
            batch = batch.to(device)
            pred = model(batch)[0]
            activations.append(pred.cpu().numpy())
    activations = np.concatenate(activations, axis=0)
    return activations

def calculate_inception_score(activations, splits=10):
    scores = []
    for i in range(splits):
        part = activations[i * (activations.shape[0] // splits): (i +
1) * (activations.shape[0] // splits), :]
        py = np.mean(part, axis=0)
        scores.append(np.exp(np.mean(np.sum(part * (np.log(part + 1e-
10) - np.log(py + 1e-10)), axis=1))))
    return np.mean(scores), np.std(scores)

def calculate_fid(real_activations, fake_activations):
    mu1, sigma1 = np.mean(real_activations, axis=0),
np.cov(real_activations, rowvar=False)
    mu2, sigma2 = np.mean(fake_activations, axis=0),
np.cov(fake_activations, rowvar=False)
    ssdiff = np.sum((mu1 - mu2)**2.0)
    covmean = sqrtm(sigma1.dot(sigma2))
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    return ssdiff + np.trace(sigma1 + sigma2 - 2.0 * covmean)

def main(real_images_folder, generated_images_folder):
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    model = inception_v3(pretrained=True,
transform_input=False).to(device)

    real_dataloader = load_images(real_images_folder)
    generated_dataloader = load_images(generated_images_folder)

    real_activations = get_inception_activations(real_dataloader,
model, device)
    fake_activations = get_inception_activations(generated_dataloader,
model, device)

    inception_score_mean, inception_score_std =
calculate_inception_score(fake_activations)
    fid_score = calculate_fid(real_activations, fake_activations)

    print(f"Inception Score: {inception_score_mean} ±
{inception_score_std}")
    print(f"FID Score: {fid_score}")

if __name__ == "__main__":
    real_images_folder = "CelebA-HQ-img"
    generated_images_folder = "generated_images"
    main(real_images_folder, generated_images_folder)
```

```
C:\Users\Medhavi\anaconda3\Lib\site-packages\torchvision\models\
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
C:\Users\Medhavi\anaconda3\Lib\site-packages\torchvision\models\
_utils.py:223: UserWarning: Arguments other than a weight enum or
`None` for 'weights' are deprecated since 0.13 and may be removed in
the future. The current behavior is equivalent to passing
`weights=Inception_V3_Weights.IMAGENET1K_V1`. You can also use
`weights=Inception_V3_Weights.DEFAULT` to get the most up-to-date
weights.
  warnings.warn(msg)
100%|
██████████████████████████████████████████████████████████████████████
██████████| 938/938 [52:22<00:00,  3.35s/it]
100%|
██████████████████████████████████████████████████████████████████████
██████████████| 2/2 [00:03<00:00,  1.60s/it]

----------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
last)
Cell In[1], line 82
     80 real_images_folder = "CelebA-HQ-img"
     81 generated_images_folder = "generated_images"
---> 82 main(real_images_folder, generated_images_folder)

Cell In[1], line 73, in main(real_images_folder,
generated_images_folder)
     70 real_activations = get_inception_activations(real_dataloader,
model, device)
     71 fake_activations =
get_inception_activations(generated_dataloader, model, device)
---> 73 inception_score_mean, inception_score_std =
calculate_inception_score(fake_activations)
     74 fid_score = calculate_fid(real_activations, fake_activations)
     76 print(f"Inception Score: {inception_score_mean} ±
{inception_score_std}")

Cell In[1], line 49, in calculate_inception_score(activations, splits)
     47 scores = []
     48 for i in range(splits):
---> 49     part = activations[i * (activations.shape[0] // splits):
(i + 1) * (activations.shape[0] // splits), :]
     50     py = np.mean(part, axis=0)
     51     scores.append(np.exp(np.mean(np.sum(part * (np.log(part +
1e-10) - np.log(py + 1e-10)), axis=1))))
```

```
IndexError: too many indices for array: array is 1-dimensional, but 2
were indexed
```

```
import os
import cv2
import numpy as np
from skimage.metrics import structural_similarity as ssim

def calculate_psnr(img1, img2):
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * np.log10(PIXEL_MAX / np.sqrt(mse))

def calculate_ssim(img1, img2):
    return ssim(img1, img2, multichannel=True)

def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path)
        if img is not None:
            images.append((filename, img))
        else:
            print(f"Warning: {filename} could not be loaded.")
    return images

def main():
    folder = 'generated_images'
    reference_image_path = 'generated_images/50.png'

    reference_image = cv2.imread(reference_image_path)
    if reference_image is None:
        print(f"Error: Reference image at path {reference_image_path}
could not be loaded.")
        return

    images = load_images_from_folder(folder)

    results = []

    for (filename, image) in images:
        if image.shape != reference_image.shape:
            print(f"Warning: {filename} has a different shape than the
reference image. Skipping.")
            continue
        psnr_value = calculate_psnr(reference_image, image)
        ssim_value = calculate_ssim(reference_image, image)
```

```python
        results.append((filename, psnr_value, ssim_value))

    for result in results:
        print(f"Filename: {result[0]}, PSNR: {result[1]:.2f}, SSIM:
{result[2]:.4f}")

if __name__ == "__main__":
    main()
```

```
Error: Reference image at path generated_images/50.png could not be
loaded.
```