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

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

```
import numpy as np
import pandas as pd
import os
print(os.listdir("/content/drive/MyDrive/animated faces/animeface-character-dataset/data/"))
```

```
['face_0_421_57.png', 'face_0_149_83.png', 'face_0_343_119.png', 'face_0_421_23.png', 'face_0_119_15.png', 'face_13_303_75.png', 'face_1
```

```
import numpy as np
import pandas as pd
import os
import time
import tensorflow as tf
import numpy as np
import glob
from glob import glob
import datetime
import random
from PIL import Image
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import keras
from keras.layers import Input, Dense, Reshape, Flatten, Dropout
from keras.layers import BatchNormalization, Activation, ZeroPadding2D
from keras.layers import LeakyReLU
from keras.layers import UpSampling2D, Conv2D , Conv2DTranspose
from keras.models import Sequential, Model
from keras.optimizers import Adam
import warnings
warnings.filterwarnings("ignore")
```

```
IMAGE_SIZE = 64
NOISE_SIZE = 100
LR_D = 0.00004
LR_G = 0.0004
BATCH_SIZE = 32
EPOCHS = 50
BETA1 = 0.5
WEIGHT_INIT_STDDEV = 0.02
EPSILON = 0.00005
SAMPLES_TO_SHOW = 8
```

```
def get_generator(z=(NOISE_SIZE,)):
    # 4 x 4 x 512
    input_layer = Input(z)
    hid = Dense(4*4*512, activation='relu', name ="Dense")(input_layer)
    hid = LeakyReLU(alpha=0.2)(hid)
    hid = Reshape((4, 4, 512))(hid)
    hid = Conv2DTranspose(512, kernel_size=[5,5],
                          strides=[2,2],
                          padding="same",
                          kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),name ="trans_conv1")(hid)
    hid = BatchNormalization(momentum=0.9, epsilon=EPSILON, name="batch_trans_conv1")(hid)
    hid = LeakyReLU(alpha=0.2,name ="trans_conv1_out")(hid)
    hid = Conv2DTranspose(256, kernel_size=[5,5],
                          strides=[2,2],
                          padding="same",
                          kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),name ="trans_conv2")(hid)
    hid = BatchNormalization(momentum=0.9, epsilon=EPSILON, name="batch_trans_conv2")(hid)
    hid = LeakyReLU(alpha=0.2,name ="trans_conv2_out")(hid)
    hid = Conv2DTranspose(128, kernel_size=[5,5],
                          strides=[2,2],
                          padding="same",
                          kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),name ="trans_conv3")(hid)
    hid = BatchNormalization(momentum=0.9, epsilon=EPSILON, name="batch_trans_conv3")(hid)
    hid = LeakyReLU(alpha=0.2,name ="trans_conv3_out")(hid)
    hid = Conv2DTranspose(64, kernel_size=[5,5],
```

```
                              strides=[2,2],
                              padding="same",
                              kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),name ="trans_conv4")(hid)
    hid = BatchNormalization(momentum=0.9, epsilon=EPSILON, name="batch_trans_conv4")(hid)
    hid = LeakyReLU(alpha=0.2,name ="trans_conv4_out")(hid)
    hid = Conv2DTranspose(3, kernel_size=[5,5],
                              strides=[1,1],
                              padding="same",
                              kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),name ="logits")(hid)
    out = Activation("tanh", name ="out")(hid)
    model = Model(inputs=input_layer, outputs=out)
    model.summary()
    return model
```

```
def get_discriminator(input_shape=(IMAGE_SIZE, IMAGE_SIZE,3)):
    input_layer = Input(input_shape)
    hid = Conv2D(filters=32,
                 kernel_size=[5,5],
                 strides=[2,2],
                 padding="same",
                 kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),
                 name = "conv1")(input_layer)
    hid = BatchNormalization(momentum=0.9, epsilon=EPSILON, name="batch_norm1")(hid)
    hid = LeakyReLU(alpha=0.2, name="conv1_out")(hid)
    hid = Conv2D(filters=64,
                     kernel_size=[5,5],
                     strides=[2,2],
                     padding="same",
                     kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),
                     name = "conv2")(hid)
    hid = BatchNormalization(momentum=0.9, epsilon=EPSILON, name="batch_norm2")(hid)
    hid = LeakyReLU(alpha=0.2, name="conv2_out")(hid)
    hid = Conv2D(filters=128,
                 kernel_size=[5,5],
                 strides=[2,2],
                 padding="same",
                 kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),
                 name = "conv3")(hid)
    hid = BatchNormalization(momentum=0.9, epsilon=EPSILON, name="batch_norm3")(hid)
    hid = LeakyReLU(alpha=0.2, name="conv3_out")(hid)
    hid = Conv2D(filters=256,
                 kernel_size=[5,5],
                 strides=[1,1],
                 padding="same",
                 kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),
                 name = "conv4")(hid)
    hid = BatchNormalization(momentum=0.9, epsilon=EPSILON, name="batch_norm4")(hid)
    hid = LeakyReLU(alpha=0.2, name="conv4_out")(hid)
    hid = Conv2D(filters=512,
                 kernel_size=[5,5],
                 strides=[2,2],
                 padding="same",
                 kernel_initializer= keras.initializers.TruncatedNormal(stddev=WEIGHT_INIT_STDDEV),
                 name = "conv5")(hid)
    hid = BatchNormalization(momentum=0.9, epsilon=EPSILON, name="batch_norm5")(hid)
    hid = LeakyReLU(alpha=0.2, name="conv5_out")(hid)
    hid = Flatten(name = "flatten")(hid)
    out = Dense(1, activation='sigmoid', name = "ligit")(hid)
    model = Model(inputs= input_layer, outputs=out)
    model.summary()
    return model
```

```
discriminator = get_discriminator((IMAGE_SIZE, IMAGE_SIZE,3))
discriminator.compile(loss='binary_crossentropy',optimizer=Adam(lr=LR_D, beta_1=BETA1),metrics=['accuracy'])
discriminator.trainable = False
generator = get_generator((NOISE_SIZE,))
gan_input = Input(shape=(NOISE_SIZE,))
x = generator(gan_input)
gan_out = discriminator(x)
gan = Model(gan_input, gan_out)
gan.summary()
gan.compile(loss='binary_crossentropy',optimizer=Adam(lr=LR_G, beta_1=BETA1))

```

```
)

trans_conv2 (Conv2DTranspo  (None, 16, 16, 256)       3277056
se)

batch_trans_conv2 (BatchNo  (None, 16, 16, 256)       1024
rmalization)

trans_conv2_out (LeakyReLU  (None, 16, 16, 256)       0
)

trans_conv3 (Conv2DTranspo  (None, 32, 32, 128)       819328
se)

batch_trans_conv3 (BatchNo  (None, 32, 32, 128)       512
rmalization)

trans_conv3_out (LeakyReLU  (None, 32, 32, 128)       0
)

trans_conv4 (Conv2DTranspo  (None, 64, 64, 64)        204864
se)

batch_trans_conv4 (BatchNo  (None, 64, 64, 64)        256
rmalization)

trans_conv4_out (LeakyReLU  (None, 64, 64, 64)        0
)

logits (Conv2DTranspose)    (None, 64, 64, 3)         4803

out (Activation)            (None, 64, 64, 3)         0

=================================================================
Total params: 11691395 (44.60 MB)
Trainable params: 11689475 (44.59 MB)
Non-trainable params: 1920 (7.50 KB)
_________________________________________________________________
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 100)]             0

 model_1 (Functional)        (None, 64, 64, 3)         11691395

 model (Functional)          (None, 1)                 4367553

=================================================================
Total params: 16058948 (61.26 MB)
Trainable params: 11689475 (44.59 MB)
Non-trainable params: 4369473 (16.67 MB)
_________________________________________________________________
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers
```

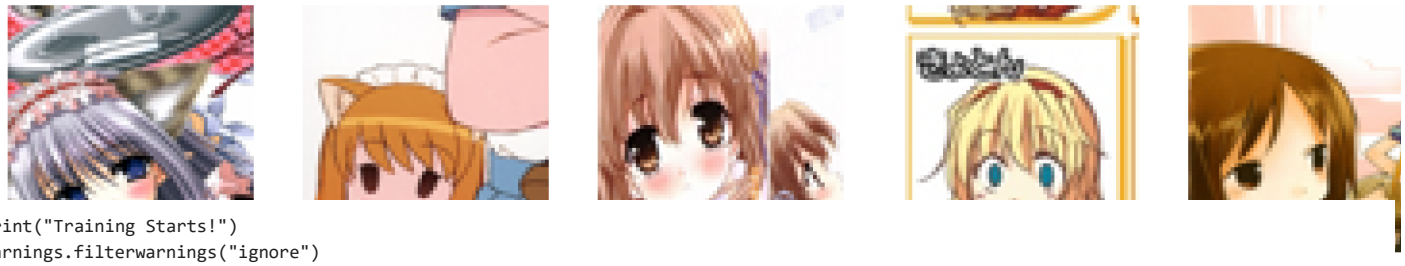
```
def show_samples(sample_images, name, epoch):
    figure, axes = plt.subplots(1, len(sample_images), figsize = (IMAGE_SIZE, IMAGE_SIZE))
    for index, axis in enumerate(axes):
        axis.axis('off')
        image_array = sample_images[index]
        axis.imshow(image_array)
        image = Image.fromarray(image_array)
    plt.show()
    plt.close()
def test(input_z, epoch):
    samples = generator.predict(input_z[:SAMPLES_TO_SHOW])
    sample_images = [((sample + 1.0) * 127.5).astype(np.uint8) for sample in samples]
    show_samples(sample_images, OUTPUT_DIR + "samples", epoch)
def summarize_epoch(d_losses, g_losses , data_shape, epoch, duration, input_z):
    minibatch_size = int(data_shape[0]//BATCH_SIZE)
    print("Epoch {}/{}".format(epoch, EPOCHS),
          "\nDuration: {:.5f}".format(duration),
          "\nD Loss: {:.5f}".format(np.mean(d_losses[-minibatch_size:])),
          "\nG Loss: {:.5f}".format(np.mean(g_losses[-minibatch_size:])))
    fig, ax = plt.subplots()
    plt.plot(d_losses, label='Discriminator', alpha=0.6)
    plt.plot(g_losses, label='Generator', alpha=0.6)
    plt.title("Losses")
    plt.legend()
    plt.savefig(OUTPUT_DIR + "losses_" + str(epoch) + ".png")
```

```
    plt.show()
    plt.close()
    test(input_z, epoch)
def get_batches(data):
    batches = []
    for i in range(int(data.shape[0]//BATCH_SIZE)):
        batch = data[i * BATCH_SIZE:(i + 1) * BATCH_SIZE]
        augmented_images = []
        for img in batch:
            image = Image.fromarray(img)
            if random.choice([True, False]):
                image = image.transpose(Image.FLIP_LEFT_RIGHT)
            augmented_images.append(np.asarray(image))
        batch = np.asarray(augmented_images)
        normalized_batch = (batch / 127.5) - 1.0
        batches.append(normalized_batch)
    return np.array(batches)
```

```
INPUT_DATA_DIR = "/content/drive/MyDrive/animated faces/animeface-character-dataset/data/"
OUTPUT_DIR =""
```

```
from PIL import Image
import re
exclude_img = []
exclude_img = [s + ".png" for s in exclude_img]
print("Image Samples")
input_images = np.asarray([np.asarray(Image.open(file).resize((IMAGE_SIZE, IMAGE_SIZE))) for file in glob(INPUT_DATA_DIR + '*') if file no
print ("Input: " + str(input_images.shape))
np.random.shuffle(input_images)
sample_images = random.sample(list(input_images), SAMPLES_TO_SHOW)
show_samples(sample_images, OUTPUT_DIR + "inputs", 0)
```

```
Image Samples
Input: (60, 64, 64, 3)
```



```
print("Training Starts!")
warnings.filterwarnings("ignore")
d_losses = []
g_losses = []
cum_d_loss = 0
cum_g_loss = 0
for epoch in range(EPOCHS):
    epoch += 1
    start_time = time.time()
    for batch_images in get_batches(input_images):
        noise_data = np.random.normal(0, 1, size=(BATCH_SIZE, NOISE_SIZE))
        generated_images = generator.predict(noise_data)
        noise_prop = 0.05
        real_labels = np.zeros((BATCH_SIZE, 1)) + np.random.uniform(low=0.0, high=0.1, size=(BATCH_SIZE, 1))
        flipped_idx = np.random.choice(np.arange(len(real_labels)), size=int(noise_prop*len(real_labels)))
        real_labels[flipped_idx] = 1 - real_labels[flipped_idx]
        d_loss_real = discriminator.train_on_batch(batch_images, real_labels)
        fake_labels = np.ones((BATCH_SIZE, 1)) - np.random.uniform(low=0.0, high=0.1, size=(BATCH_SIZE, 1))
        flipped_idx = np.random.choice(np.arange(len(fake_labels)), size=int(noise_prop*len(fake_labels)))
        fake_labels[flipped_idx] = 1 - fake_labels[flipped_idx]
        d_loss_fake = discriminator.train_on_batch(generated_images, fake_labels)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
        cum_d_loss += d_loss
        d_losses.append(d_loss[0])
        noise_data = np.random.normal(0, 1, size=(BATCH_SIZE, NOISE_SIZE))
        g_loss = gan.train_on_batch(noise_data, np.zeros((BATCH_SIZE, 1)))
        cum_g_loss += g_loss
        g_losses.append(g_loss)
    if epoch > 0 and epoch % 20 == 0 :
        print("saving model")
        discriminator.save_weights("desc-simposon-model.h5-" + str(epoch))
        gan.save_weights("gan-simposon-model.h5-" + str(epoch))
    summarize_epoch(d_losses, g_losses, input_images.shape, epoch, time.time()-start_time, noise_data)
```