```
# Mount drive to load celeb_a data
from google.colab import drive
drive.mount('/content/drive')
```

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

```
import numpy as np
import os
from keras.layers import Input, Conv2D, Flatten, Dense, Conv2DTranspose, Reshape, L
from keras.layers import Activation, BatchNormalization, LeakyReLU, Dropout, ZeroPa
from keras.layers.advanced_activations import LeakyReLU
from keras.models import Sequential, Model
from keras.optimizers import Adam, RMSprop
from keras.preprocessing.image import array_to_img, img_to_array, load_img
from keras import backend as K
from keras.callbacks import ModelCheckpoint
from keras.utils import plot_model
from keras.initializers import RandomNormal
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from numpy.random import randn
from numpy.random import randint
```

```
# load data from saved file
data = np.load('/content/drive/MyDrive/celebA/test1.npy')
data.shape
```

```
(10000, 128, 128, 3)
```

```
x_train = data
```

```
# Shape x_train - Totol # of images: 10000, dim = (128,128,3)
len(x_train), len(x_train[0]), x_train.shape
```

```
(10000, 128, (10000, 128, 128, 3))
```

```
# Preview Image from dataset
plt.imshow((x_train[0]))
```

```
<matplotlib.image.AxesImage at 0x7ff96219bda0>
```


```
#Define input image dimensions
#Large images take too much time and resources. taking dim = (128, 128, 3)
img_rows = 128
img_cols = 128
channels = 3
img_shape = (img_rows, img_cols, channels)


# Resets all state generated by Keras
K.clear_session()

#initializing weights
weight_init = RandomNormal(mean=0., stddev=0.02)

# Disciminator model
discriminator_input = Input(shape= img_shape, name='discriminator_input')
x = discriminator_input

# First convolutional layer
x = Conv2D(filters= 64, kernel_size= 5, strides= 2, padding= 'same', name= 'discrim
           kernel_initializer= weight_init)(x)
x = LeakyReLU(alpha = 0.2)(x)
x = Dropout(rate = 0.2)(x)

# Second convolutional layer
x = Conv2D(filters= 128, kernel_size= 5, strides= 2, padding= 'same', name= 'discri
           kernel_initializer= weight_init)(x)
x = LeakyReLU(alpha = 0.2)(x)
x = Dropout(rate = 0.2)(x)


# Third convolutional layer
x = Conv2D(filters= 256, kernel_size= 5, strides= 2, padding= 'same', name= 'discri
           kernel_initializer= weight_init)(x)
x = LeakyReLU(alpha = 0.2)(x)
x = Dropout(rate = 0.2)(x)


# Fourth convolutional layer
x = Conv2D(filters= 512, kernel_size= 5, strides= 2, padding= 'same', name= 'discri
           kernel_initializer= weight_init)(x)
x = LeakyReLU(alpha = 0.2)(x)
x = Dropout(rate = 0.2)(x)


x = Flatten()(x)

discriminator_output = Dense(1, activation= 'sigmoid', kernel_initializer= weight_i
```

```
discriminator_output = Dense(1, activation= 'sigmoid', kernel_initializer= weight_i

discriminator = Model(discriminator_input, discriminator_output)


# Discriminator model summary
discriminator.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
discriminator_input (InputLa [(None, 128, 128, 3)]     0
_________________________________________________________________
discriminator_conv_1 (Conv2D (None, 64, 64, 64)        4864
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 64, 64, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 64, 64, 64)        0
_________________________________________________________________
discriminator_conv_2 (Conv2D (None, 32, 32, 128)       204928
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 128)       0
_________________________________________________________________
discriminator_conv_3 (Conv2D (None, 16, 16, 256)       819456
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 256)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 256)       0
_________________________________________________________________
discriminator_conv_4 (Conv2D (None, 8, 8, 512)         3277312
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 512)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 1)                 32769
=================================================================
Total params: 4,339,329
Trainable params: 4,339,329
Non-trainable params: 0
_________________________________________________________________
```

```
# Generator model
generator_input = Input(shape=(100,), name='generator_input')

x = generator_input

x = Dense(np.prod(32768), kernel_initializer= weight_init)(x)
x = BatchNormalization(momentum = 0.9)(x)
x = LeakyReLU(alpha = 0.2)(x)
x = Reshape((8, 8, 512))(x)

x = UpSampling2D()(x)
x = Conv2D(filters= 512, kernel_size= 5, padding= 'same', strides= 1, name = 'gener
```

```
            kernel_initializer= weight_init)(x)
x = BatchNormalization(momentum = 0.9)(x)
x = LeakyReLU(alpha = 0.2)(x)

x = UpSampling2D()(x)
x = Conv2D(filters= 256, kernel_size= 5, padding= 'same', strides= 1, name = 'gener
            kernel_initializer= weight_init)(x)
x = BatchNormalization(momentum = 0.9)(x)
x = LeakyReLU(alpha = 0.2)(x)

x = UpSampling2D()(x)
x = Conv2D(filters= 64, kernel_size= 5, padding= 'same', strides= 1, name = 'genera
            kernel_initializer= weight_init)(x)
x = BatchNormalization(momentum = 0.9)(x)
x = LeakyReLU(alpha = 0.2)(x)


x = Conv2DTranspose(filters= 3, kernel_size= 5, padding= 'same', strides= 2, name =
                    kernel_initializer= weight_init)(x)
x = Activation('sigmoid')(x)

generator_output = x

generator = Model(generator_input, generator_output)


# Generator model summary
generator.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
generator_input (InputLayer) [(None, 100)]             0
_________________________________________________________________
dense_1 (Dense)              (None, 32768)             3309568
_________________________________________________________________
batch_normalization (BatchNo (None, 32768)             131072
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 32768)             0
_________________________________________________________________
reshape (Reshape)            (None, 8, 8, 512)         0
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 16, 16, 512)       0
_________________________________________________________________
generator_conv_1 (Conv2D)    (None, 16, 16, 512)       6554112
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 512)       2048
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 16, 16, 512)       0
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 32, 32, 512)       0
_________________________________________________________________
generator_conv_2 (Conv2D)    (None, 32, 32, 256)       3277056
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 256)       1024
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
```

```
up_sampling2d_2 (UpSampling2 (None, 64, 64, 256)       0
_________________________________________________________________
generator_conv_3 (Conv2D)    (None, 64, 64, 64)        409664
_________________________________________________________________
batch_normalization_3 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 64, 64, 64)        0
_________________________________________________________________
generator_conv_4 (Conv2DTran (None, 128, 128, 3)       4803
_________________________________________________________________
activation (Activation)      (None, 128, 128, 3)       0
=================================================================
Total params: 13,689,603
Trainable params: 13,622,403
Non-trainable params: 67,200
_________________________________________________________________
```

```
# freezing weights of generator
generator.trainable = False
for layer in generator.layers:
    layer.trainable = False

discriminator.compile(optimizer= RMSprop(lr= 0.0008, decay=6e-8),
                      loss= 'binary_crossentropy')

# freezing weights of discriminator
discriminator.trainable = False
for layer in discriminator.layers:
    layer.trainable = False

# unfreezing weights of generator
generator.trainable = True
for layer in generator.layers:
    layer.trainable = True

model_input = Input(shape=(100,), name='model_input')
model_output = discriminator(generator(model_input))

model = Model(model_input, model_output)

model.compile(optimizer= RMSprop(lr=0.0004, decay=3e-8),
              loss= 'binary_crossentropy')

# unfreezing weights of discriminator
discriminator.trainable = True
for layer in discriminator.layers:
    layer.trainable = True


# method to train discriminator
def train_discriminator(x_train, batch_size):

    valid = np.ones((batch_size,1))
    fake = np.zeros((batch_size,1))

    noise = np.random.normal(0, 1, (batch_size, 100))
    gen_imgs = generator.predict(noise)
```

```
        gen_imgs = generator.predict(noise)

        d_loss_real = discriminator.train_on_batch(x_train, valid)
        d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
        d_loss = 0.5 * (d_loss_real + d_loss_fake)

        return [d_loss, d_loss_real, d_loss_fake]

    # method to train generator(
    def train_generator(batch_size):
        valid = np.ones((batch_size,1))
        noise = np.random.normal(0, 1, (batch_size, 100))
        return model.train_on_batch(noise, valid)


    # Method to save images by generator in drive
    def save_imgs(epoch):
        r, c = 6, 6
        noise = np.random.normal(0, 1, (r * c, 100))

        #generate fake image
        gen_imgs = generator.predict(noise)

        fig, axs = plt.subplots(r, c)
        cnt1 = 0
        cnt2 = 0

        #save grey images
        for i in range(r):
            for j in range(c):
                axs[i,j].imshow(gen_imgs[cnt1, :,:,0],  cmap='gray')
                axs[i,j].axis('off')
                cnt1 += 1
        fig.savefig("/content/drive/MyDrive/celeb_a_predict_img/celeb_a_grey_%d.png" %

        #save color images
        for i in range(r):
            for j in range(c):
                axs[i,j].imshow(gen_imgs[cnt2, :,:,0])
                axs[i,j].axis('off')
                cnt2 += 1
        fig.savefig("/content/drive/MyDrive/celeb_a_predict_img/celeb_a_color_%d.png" %
        plt.close()


    import warnings
    warnings.filterwarnings("ignore");
    epoch = 0

    d_losses = []
    g_losses = []

    data_gen = ImageDataGenerator(preprocessing_function=lambda x: (x.astype('float32')
    for e in range(7000):
        print('Epoch', e)
        batches = 0
```

```
        for x_batch in data_gen.flow(x_train, batch_size=64):
            batches += 1
            if batches >= len(x_train) / 64:
                break
            d_loss = train_discriminator(x_batch, 64)
            g_loss = train_generator(64)

            d_losses.append(d_loss)
            g_losses.append(g_loss)

        if epoch%20 == 0:
            print ("%d [D loss: (%.3f)(R %.3f, F %.3f)]  [G loss: %.3f] " % (epoch, d_l
            save_imgs(e)
```

```
0 [D loss: (0.519)(R 1.037, F 0.002)]  [G loss: 16.811]
Epoch 342
0 [D loss: (0.187)(R 0.001, F 0.372)]  [G loss: 21.625]
Epoch 343
0 [D loss: (1.345)(R 2.140, F 0.550)]  [G loss: 30.689]
Epoch 344
0 [D loss: (0.172)(R 0.089, F 0.256)]  [G loss: 17.526]
Epoch 345
0 [D loss: (0.162)(R 0.083, F 0.240)]  [G loss: 15.959]
Epoch 346
0 [D loss: (0.052)(R 0.080, F 0.024)]  [G loss: 18.080]
Epoch 347
0 [D loss: (0.075)(R 0.062, F 0.088)]  [G loss: 24.649]
Epoch 348
0 [D loss: (0.542)(R 1.080, F 0.003)]  [G loss: 14.694]
Epoch 349
0 [D loss: (0.691)(R 1.029, F 0.354)]  [G loss: 14.568]
Epoch 350
0 [D loss: (0.353)(R 0.297, F 0.408)]  [G loss: 17.521]
Epoch 351
0 [D loss: (0.393)(R 0.452, F 0.334)]  [G loss: 19.794]
Epoch 352
0 [D loss: (0.333)(R 0.064, F 0.602)]  [G loss: 23.681]
Epoch 353
0 [D loss: (0.013)(R 0.005, F 0.020)]  [G loss: 19.736]
Epoch 354
0 [D loss: (0.891)(R 1.358, F 0.424)]  [G loss: 16.799]
Epoch 355
0 [D loss: (0.162)(R 0.147, F 0.176)]  [G loss: 17.691]
Epoch 356
0 [D loss: (0.001)(R 0.000, F 0.002)]  [G loss: 18.371]
Epoch 357
0 [D loss: (0.560)(R 0.201, F 0.919)]  [G loss: 24.274]
Epoch 358
0 [D loss: (0.441)(R 0.872, F 0.009)]  [G loss: 15.510]
Epoch 359
0 [D loss: (0.422)(R 0.257, F 0.586)]  [G loss: 25.565]
Epoch 360
0 [D loss: (0.745)(R 0.646, F 0.845)]  [G loss: 21.630]
Epoch 361
0 [D loss: (0.324)(R 0.401, F 0.247)]  [G loss: 19.506]
Epoch 362
0 [D loss: (0.065)(R 0.053, F 0.078)]  [G loss: 14.408]
Epoch 363
0 [D loss: (0.048)(R 0.005, F 0.091)]  [G loss: 18.793]
Epoch 364
```

```
0 [D loss: (0.247)(R 0.167, F 0.328)]  [G loss: 17.751]
Epoch 365
0 [D loss: (0.263)(R 0.198, F 0.327)]  [G loss: 15.868]
Epoch 366
0 [D loss: (0.253)(R 0.040, F 0.467)]  [G loss: 25.376]
Epoch 367
0 [D loss: (0.657)(R 0.236, F 1.078)]  [G loss: 24.840]
Epoch 368
0 [D loss: (0.024)(R 0.006, F 0.042)]  [G loss: 21.483]
Epoch 369
0 [D loss: (0.186)(R 0.007, F 0.365)]  [G loss: 17.171]
Epoch 370
0 [D loss: (0.656)(R 0.211, F 1.101)]  [G loss: 18.639]
```