GPU Access

```
Google colab offers 3 type of free GPU's with different compute capability.
1. Tesla K80 = 3.8
2. Telsa P100 = 6.0
3. Tesla T4 = 7.5

This notebook consist code that uses mixed precision :
required compute capability >= 7.0

⭐ Tesla T4 is the one that can works in colab for mixed precision.
```

mixed precision training `Mixed precision for training neural networks can reduce training time and memory requirements without affecting model performance`

> The **compute capability** of a GPU determines its general specifications and available features. see more

```
!nvidia-smi -L
```

```
GPU 0: Tesla K80 (UUID: GPU-19abf15a-50cb-2c3a-1745-05e965e0a14b)
```

Get Data

> clear cache

```
pip install --upgrade --no-cache-dir gdown
```

Get data

```
#@title Get data


!gdown --id "1-JVnG_wVJR3VgAwi6-Hhu2C-ZAyQ2-_9"
!gdown --id "1-7E0x-UGFjotUH8UJAWruM9Y0rwEzYzV"
!gdown --id "19li26wV60jhrf8UtUhGH6xuocDqiHqPG"
!gdown --id "179YgtbT7A0YFJsQyFULbQPgPZzdmnySA"
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1-JVnG_wVJR3VgAwi6-Hhu2C-ZAyQ2-_9
To: /content/1_half_face_labels.pickle
100% 179M/179M [00:03<00:00, 58.8MB/s]
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1-7E0x-UGFjotUH8UJAWruM9Y0rwEzYzV
To: /content/1_half_face_occluded.pickle
100% 179M/179M [00:02<00:00, 79.2MB/s]
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=19li26wV60jhrf8UtUhGH6xuocDqiHqPG
To: /content/unres_labels.pickle
100% 528M/528M [00:13<00:00, 38.9MB/s]
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=179YgtbT7A0YFJsQyFULbQPgPZzdmnySA
To: /content/unres_occluded.pickle
100% 528M/528M [00:09<00:00, 52.9MB/s]
```

Required Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import os
import pickle
import random
from tensorflow.keras import mixed_precision
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

Load data

Terminology used :

🐾 x : occluded face images (input data)

🐾 y : clear face images (labels)

```

y_path = '/content/1_half_face_labels.pickle'
x_path = '/content/1_half_face_occluded.pickle'
```

```python

pickle_in = open(x_path,"rb")
x = pickle.load(pickle_in)

pickle_in = open(y_path,"rb")
y = pickle.load(pickle_in)

```

format data

```python
print(f"shape of half face labels :{x.shape}")
print(f"shape of half face occluded : {y.shape}")
```

```
shape of half face labels :(4471, 200, 200, 1)
shape of half face occluded : (4471, 200, 200, 1)
```

we have total images : 4471

image size = (200,200)

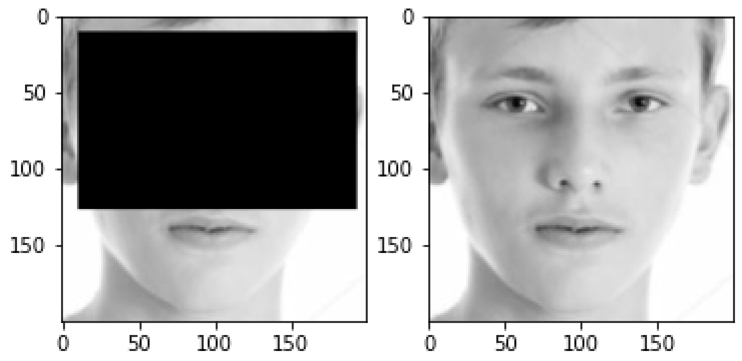
channel = 1

we are converting images into (64,64,1) images

to avoid resource exhausted problem 🤯

```python
fig=plt.figure(figsize=(6, 6))
fig.add_subplot(1, 2, 1)
plt.imshow(x[1,:,:,0],cmap="gray")
fig.add_subplot(1, 2, 2)
plt.imshow(y[1,:,:,0],cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f58c13eb490>
```




```python
def preprocess_img(image, img_shape=224):
  """
  Converts image datatype from 'uint8' -> 'float32' and reshapes image to
  [img_shape, img_shape]
  """

```

```
  image = tf.image.resize(image, [img_shape, img_shape])
  return tf.cast(image/255., tf.float32)
```

```
data=preprocess_img(x,64)
label=preprocess_img(y,64)
```

```
print("After perpocessing images:")
print(f"shape of half face labels :{data.shape}")
print(f"shape of half face occluded : {label.shape}")
```

```
After perpocessing images:
shape of half face labels :(4471, 64, 64, 1)
shape of half face occluded : (4471, 64, 64, 1)
```

Mixed precision

```
mixed_precision.set_global_policy('mixed_float16')
```

```
INFO:tensorflow:Mixed precision compatibility check (mixed_float16): OK
Your GPU will likely run quickly with dtype policy mixed_float16 as it has compute c
```

```
Why to use mixed precision ❓

using mixed precision we can get our task completed 3x times faster than  modern GPUs and 60% on
https://www.tensorflow.org/guide/mixed_precision
```

```
⭐ When using a normal GPU without mixed precision:

* training for 5 epochs takes ~ 6 min.

* with mixed precision it is done ~ 2 min.
```

[36]

```
train(data,label,5)
```

```
Epoch: 1 Gan Loss: 13.981639862060547      Disc Loss: 0.0599425733089447      Gen Loss: 4.6353302001953125
Epoch: 2 Gan Loss: 12.190213203430176      Disc Loss: 4.479439735412598      Gen Loss: 4.000351905822754
Epoch: 3 Gan Loss: 13.472700119018555      Disc Loss: 0.2273085117340088      Gen Loss: 3.441371440887451
Epoch: 4 Gan Loss: 5.26753044128418      Disc Loss: 1.6766669750213623      Gen Loss: 2.9352307319641113
Epoch: 5 Gan Loss: 8.885817527770996      Disc Loss: 0.2796383500099182      Gen Loss: 2.554171323776245
```

✓ 2m 18s completed at 4:53 PM

Create Model

```
Cosiderations after experiments:
1. using Batch normalization for stabilize training.
2. using Dropout to avoid overfitting.
3. Avoid max pooling for downsampling. Use convolution stride.
4. Adam optimizer usually works better than other optimizers.
5. Scale the image pixel value between -1 and 1. Use tanh as the output layer for the generator.
6. use LeakyRelu for to avoid sparse gradients.
```

Refernces : 1 2 3

```
img_shape=(64,64,1)
```

Generator model

Show code

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 64, 64, 1)]       0

 conv2d_14 (Conv2D)          (None, 32, 32, 64)        1664

 batch_normalization_16 (Bat  (None, 32, 32, 64)       256
 chNormalization)

 leaky_re_lu_11 (LeakyReLU)  (None, 32, 32, 64)        0

 dropout_14 (Dropout)        (None, 32, 32, 64)        0

 conv2d_15 (Conv2D)          (None, 16, 16, 128)       204928

 batch_normalization_17 (Bat  (None, 16, 16, 128)      512
 chNormalization)

 leaky_re_lu_12 (LeakyReLU)  (None, 16, 16, 128)       0

 dropout_15 (Dropout)        (None, 16, 16, 128)       0

 conv2d_16 (Conv2D)          (None, 8, 8, 256)         819456

 batch_normalization_18 (Bat  (None, 8, 8, 256)        1024
```

```
 chNormalization)

 leaky_re_lu_13 (LeakyReLU)  (None, 8, 8, 256)        0

 dropout_16 (Dropout)        (None, 8, 8, 256)        0

 conv2d_transpose_3 (Conv2DT  (None, 16, 16, 128)     819328
 ranspose)

 batch_normalization_19 (Bat  (None, 16, 16, 128)     512
 chNormalization)

 leaky_re_lu_14 (LeakyReLU)  (None, 16, 16, 128)      0

 conv2d_transpose_4 (Conv2DT  (None, 32, 32, 64)      204864
 ranspose)

 batch_normalization_20 (Bat  (None, 32, 32, 64)      256
 chNormalization)

 leaky_re_lu_15 (LeakyReLU)  (None, 32, 32, 64)       0

 conv2d_transpose_5 (Conv2DT  (None, 64, 64, 1)       1600
 ranspose)

=================================================================
Total params: 2,054,400
Trainable params: 2,053,120
Non-trainable params: 1,280
_________________________________________________________________
```

Discriminator model

Show code

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 64, 64, 1)]       0

 conv2d_17 (Conv2D)          (None, 32, 32, 64)        1664

 batch_normalization_21 (Bat  (None, 32, 32, 64)       256
 chNormalization)

 leaky_re_lu_16 (LeakyReLU)  (None, 32, 32, 64)        0

 dropout_17 (Dropout)        (None, 32, 32, 64)        0

 conv2d_18 (Conv2D)          (None, 16, 16, 128)       204928

 batch_normalization_22 (Bat  (None, 16, 16, 128)      512
 chNormalization)

 leaky_re_lu_17 (LeakyReLU)  (None, 16, 16, 128)       0
```

```
 dropout_18 (Dropout)        (None, 16, 16, 128)       0

 conv2d_19 (Conv2D)          (None, 8, 8, 256)         819456

 batch_normalization_23 (Bat  (None, 8, 8, 256)        1024
 chNormalization)

 leaky_re_lu_18 (LeakyReLU)  (None, 8, 8, 256)         0

 dropout_19 (Dropout)        (None, 8, 8, 256)         0

 conv2d_20 (Conv2D)          (None, 4, 4, 64)          409664

 batch_normalization_24 (Bat  (None, 4, 4, 64)         256
 chNormalization)

 leaky_re_lu_19 (LeakyReLU)  (None, 4, 4, 64)          0

 dropout_20 (Dropout)        (None, 4, 4, 64)          0

 flatten_3 (Flatten)         (None, 1024)              0

 dense_3 (Dense)             (None, 1)                 1025

=================================================================
Total params: 1,438,785
Trainable params: 1,437,761
Non-trainable params: 1,024
_________________________________________________________________
```

Compilation

Check layers and thier dtypes and dtype policy

- 🐾 layers dtype : layers stores information in this type.
- 🐾 layers dtype policy : layers perform calculation in this type.

```
for x in generator.layers:
  print(x.name,x.dtype,x.dtype_policy)
```

```
input_5 float32 <Policy "float32">
conv2d_14 float32 <Policy "mixed_float16">
batch_normalization_16 float32 <Policy "mixed_float16">
leaky_re_lu_11 float32 <Policy "mixed_float16">
dropout_14 float32 <Policy "mixed_float16">
conv2d_15 float32 <Policy "mixed_float16">
batch_normalization_17 float32 <Policy "mixed_float16">
leaky_re_lu_12 float32 <Policy "mixed_float16">
dropout_15 float32 <Policy "mixed_float16">
conv2d_16 float32 <Policy "mixed_float16">
batch_normalization_18 float32 <Policy "mixed_float16">
leaky_re_lu_13 float32 <Policy "mixed_float16">
```

```
dropout_16 float32 <Policy "mixed_float16">
conv2d_transpose_3 float32 <Policy "mixed_float16">
batch_normalization_19 float32 <Policy "mixed_float16">
leaky_re_lu_14 float32 <Policy "mixed_float16">
conv2d_transpose_4 float32 <Policy "mixed_float16">
batch_normalization_20 float32 <Policy "mixed_float16">
leaky_re_lu_15 float32 <Policy "mixed_float16">
conv2d_transpose_5 float32 <Policy "float32">
```

```
for x in discriminator.layers:
  print(x.name,x.dtype,x.dtype_policy)
```

```
input_6 float32 <Policy "float32">
conv2d_17 float32 <Policy "mixed_float16">
batch_normalization_21 float32 <Policy "mixed_float16">
leaky_re_lu_16 float32 <Policy "mixed_float16">
dropout_17 float32 <Policy "mixed_float16">
conv2d_18 float32 <Policy "mixed_float16">
batch_normalization_22 float32 <Policy "mixed_float16">
leaky_re_lu_17 float32 <Policy "mixed_float16">
dropout_18 float32 <Policy "mixed_float16">
conv2d_19 float32 <Policy "mixed_float16">
batch_normalization_23 float32 <Policy "mixed_float16">
leaky_re_lu_18 float32 <Policy "mixed_float16">
dropout_19 float32 <Policy "mixed_float16">
conv2d_20 float32 <Policy "mixed_float16">
batch_normalization_24 float32 <Policy "mixed_float16">
leaky_re_lu_19 float32 <Policy "mixed_float16">
dropout_20 float32 <Policy "mixed_float16">
flatten_3 float32 <Policy "mixed_float16">
dense_3 float32 <Policy "float32">
```

Set hyper parametes

```
Q 1. Why discriminator learning rate is more than generator's learning rate ❓

If Generator and Discriminator both have same learning rate than it will be like ⁉
"A blind person have to guide a blind person"
So Either a Discriminator train earlier so it can have some discriminating nature
or Discriminator have high learning rate than Generator so it can learn faster
than Generator and discriminate between fake and real images so training of
generator will go in right direction.
Generator will get train properly..

Q 2. Why **BCE** and **MSE** ❓
look into these articels
```

1 2

```
discriminator.trainable = True
```

```
discriminator.compile(loss = "binary_crossentropy",
                      optimizer = tf.keras.optimizers.Adam(lr=0.00004))

generator.compile(loss = "mean_squared_error",
                      optimizer = tf.keras.optimizers.Adam(lr=0.00001))
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning:
  super(Adam, self).__init__(name, **kwargs)
```

GAN model

```
discriminator.trainable=False

gan=Sequential([
  generator,
  discriminator,
])


gan.compile(loss = "binary_crossentropy",
            optimizer = tf.keras.optimizers.Adam(lr=0.00001))
gan.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 model_4 (Functional)        (None, 64, 64, 1)         2054400

 model_5 (Functional)        (None, 1)                 1438785

=================================================================
Total params: 3,493,185
Trainable params: 2,053,120
Non-trainable params: 1,440,065
_________________________________________________________________
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning:
  super(Adam, self).__init__(name, **kwargs)
```

Training Function

Standard GAN loss function (min-max GAN loss)

$$\mathbf{min}\ G\ \mathbf{max}\ D\ \ \mathbf{V(D,G)} = E_{x \sim P_{data}(x)}\ log\, D_{\theta_d}(x) + E_{z \sim P(z)}\ log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

> reference : 1 2

Show code

```
import numpy as np
np.random.normal(loc=0,size=64)
```

```
array([ 0.35988645,  1.36301132,  0.23156339,  0.83325584, -0.36306104,
       -0.1816666 , -1.07263975, -0.77830716, -0.55314108, -0.72732645,
        0.28666999,  0.04438089, -1.11722925, -0.46831777,  0.30803816,
        1.52411784, -0.24072359,  0.30436013, -1.2091677 ,  0.23826835,
        0.98884626,  1.27566872,  1.56788431, -1.11146306,  0.83270448,
        0.8831319 , -0.05931619,  0.85453023, -0.28434114,  0.04260363,
        0.92493073, -1.66884768,  1.27625869,  0.31884669, -0.11643232,
        1.74582352,  2.32305117, -1.33638581, -0.05899278,  1.22414971,
        0.25997429, -0.99901731, -0.76464692, -0.234426  ,  1.45130397,
       -0.56998051, -0.37832206, -0.03790518, -2.26488101, -0.43959257,
        0.07101479,  0.66563436,  0.56439883, -0.54120988,  1.36552456,
       -0.20243779,  0.37067642,  0.27923343,  0.45833033,  1.29901581,
        1.95277731,  0.70391181,  2.43692158,  0.90609885])
```

training

```
print(len(data),len(label))
```

```
4471 4471
```

- train for 250 epochs

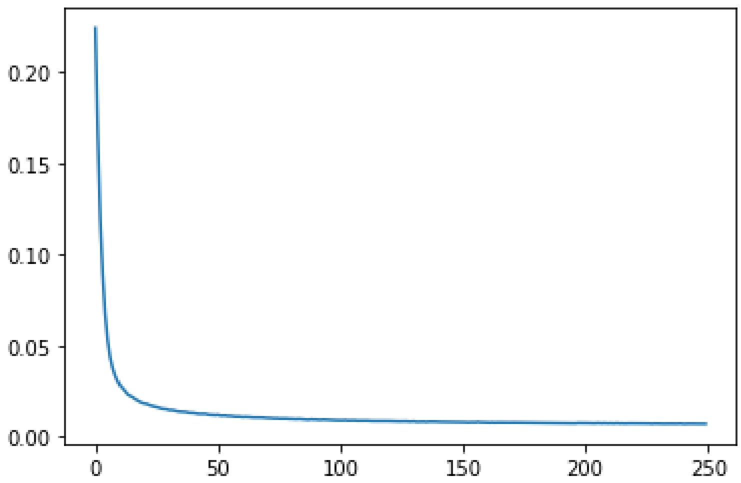
```
train(data,label,250)
```

```
Epoch: 1 Gan Loss: 15.424948692321777       Disc Loss: 0.22947844862937927
Epoch: 2 Gan Loss: 4.4394001960754395       Disc Loss: 0.22615490853786469
Epoch: 3 Gan Loss: 11.07776165008545       Disc Loss: -0.19493962824344635
Epoch: 4 Gan Loss: 5.0527753829956055       Disc Loss: 0.13403281569480896
Epoch: 5 Gan Loss: 14.552885055541992       Disc Loss: 0.030760429799556732
Epoch: 6 Gan Loss: 3.085127115249634       Disc Loss: 0.1411665678024292         G
Epoch: 7 Gan Loss: 15.211397171020508       Disc Loss: 0.4296594262123108
Epoch: 8 Gan Loss: 5.88374662399292       Disc Loss: -0.17771480977535248
Epoch: 9 Gan Loss: 14.273153305053711       Disc Loss: -0.26820245385169983
Epoch: 10 Gan Loss: 5.668078422546387       Disc Loss: 0.3387182652950287
Epoch: 11 Gan Loss: 14.151998519897461       Disc Loss: 0.6591202020645142
Epoch: 12 Gan Loss: 5.159114837646484       Disc Loss: 0.2815797030925751
Epoch: 13 Gan Loss: 14.765886306762695       Disc Loss: -0.09720595180988312
Epoch: 14 Gan Loss: 11.64935302734375       Disc Loss: 1.0278476476669312
Epoch: 15 Gan Loss: 13.888923645019531       Disc Loss: 0.6606805920600891
Epoch: 16 Gan Loss: 4.329955101013184       Disc Loss: 0.27323880791664124
Epoch: 17 Gan Loss: 12.123867988586426       Disc Loss: -0.0931006371974945
Epoch: 18 Gan Loss: 8.910452842712402       Disc Loss: 1.07106614112854         Ge
Epoch: 19 Gan Loss: 13.666898727416992       Disc Loss: -0.08394332230091095
Epoch: 20 Gan Loss: 10.948257446289062       Disc Loss: 0.34923821687698364
Epoch: 21 Gan Loss: 14.733949661254883       Disc Loss: 0.2966359257698059
Epoch: 22 Gan Loss: 9.036434173583984       Disc Loss: 0.7871143817901611
Epoch: 23 Gan Loss: 12.82349967956543       Disc Loss: 0.51315492391586 3         G
Epoch: 24 Gan Loss: 9.739767074584961       Disc Loss: 0.5666536092758179
Epoch: 25 Gan Loss: 13.317072868347168       Disc Loss: 0.40155404806137085
```

```
Epoch: 26 Gan Loss: 8.742391586303711      Disc Loss: 0.9781019687652588
Epoch: 27 Gan Loss: 13.636163711547852      Disc Loss: -0.09534776955842972
Epoch: 28 Gan Loss: 9.993696212768555      Disc Loss: 0.8349488973617554
Epoch: 29 Gan Loss: 13.373937606811523      Disc Loss: -0.1367550492286682
Epoch: 30 Gan Loss: 12.189014434814453      Disc Loss: 1.0596426725387573
Epoch: 31 Gan Loss: 14.101619720458984      Disc Loss: 0.08743734657764435
Epoch: 32 Gan Loss: 10.481380462646484      Disc Loss: 0.5308753848075867
Epoch: 33 Gan Loss: 14.562674522399902      Disc Loss: 0.14339226484298706
Epoch: 34 Gan Loss: 11.136268615722656      Disc Loss: 0.035227105021476746
Epoch: 35 Gan Loss: 14.281625747680664      Disc Loss: 0.022682808339595795
Epoch: 36 Gan Loss: 7.7298479080200195      Disc Loss: 0.8507931232452393
Epoch: 37 Gan Loss: 11.352651596069336      Disc Loss: 0.8795280456542969
Epoch: 38 Gan Loss: 10.911267280578613      Disc Loss: 0.32038116455078125
Epoch: 39 Gan Loss: 13.625067710876465      Disc Loss: -0.015256434679031372
Epoch: 40 Gan Loss: 9.955638885498047      Disc Loss: 1.0417312383651733
Epoch: 41 Gan Loss: 13.355147361755371      Disc Loss: 0.2122444361448288
Epoch: 42 Gan Loss: 11.674309730529785      Disc Loss: 0.9664750695228577
Epoch: 43 Gan Loss: 13.424057960510254      Disc Loss: 0.46878236532211304
Epoch: 44 Gan Loss: 10.968786239624023      Disc Loss: 0.7706108093261719
Epoch: 45 Gan Loss: 14.750448226928711      Disc Loss: 0.04425686597824097
Epoch: 46 Gan Loss: 13.425739288330078      Disc Loss: 0.524245023727417
Epoch: 47 Gan Loss: 12.643754959106445      Disc Loss: -0.059951379895210266
Epoch: 48 Gan Loss: 14.287469863891602      Disc Loss: 1.4496198892593384
Epoch: 49 Gan Loss: 14.721458435058594      Disc Loss: -0.08321405947208405
Epoch: 50 Gan Loss: 10.268197059631348      Disc Loss: 0.9218164682388306
Epoch: 51 Gan Loss: 13.581535339355469      Disc Loss: 0.7389094829559326
Epoch: 52 Gan Loss: 12.104586601257324      Disc Loss: 0.9932212233543396
Epoch: 53 Gan Loss: 14.777189254760742      Disc Loss: 0.3052419424057007
Epoch: 54 Gan Loss: 11.070615768432617      Disc Loss: 0.9217857122421265
Epoch: 55 Gan Loss: 14.127086639404297      Disc Loss: -0.1265900731086731
Epoch: 56 Gan Loss: 11.286212921142578      Disc Loss: 0.005463186651468277
```

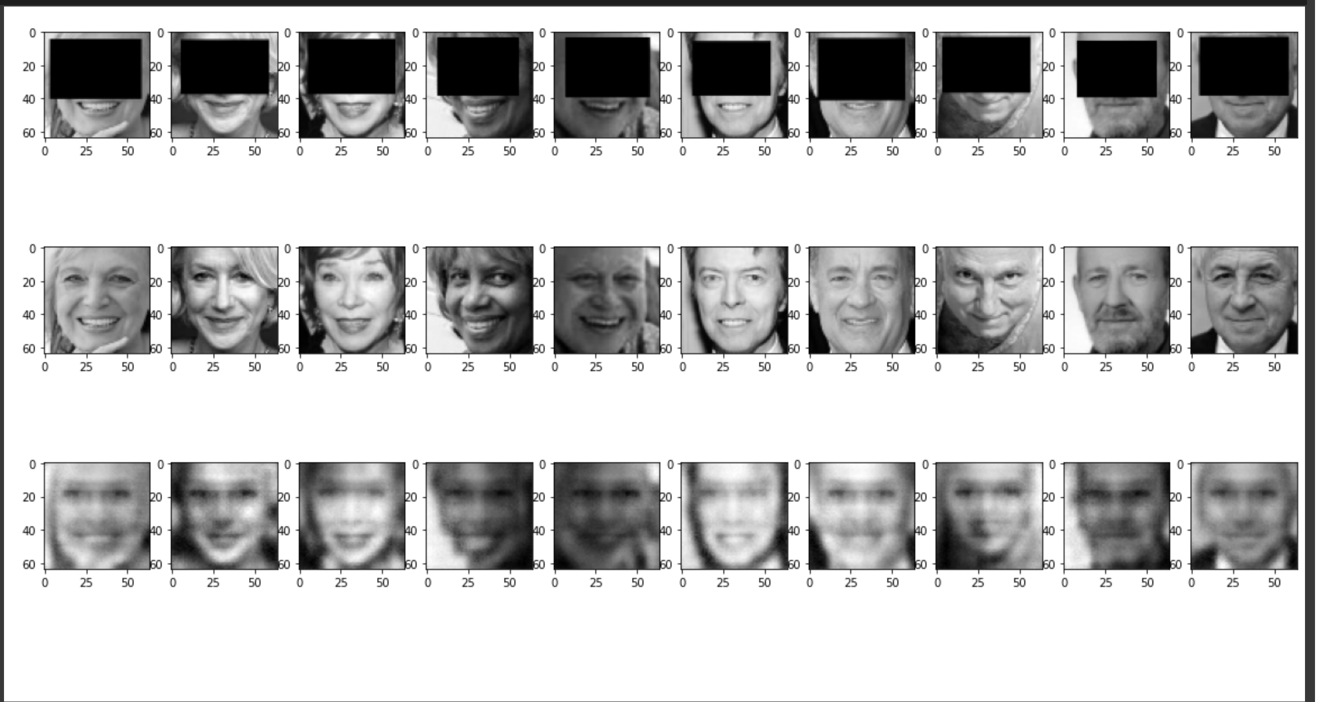
```
1 plt.plot(gen_loss_training)
```

[<matplotlib.lines.Line2D at 0x7fa06a708310>]




```
1
2 a = 4160
3 b = 4170
4 pred=generator.predict(data[a:b])
5
6 fig = plt.figure(figsize = (20,10))
```

```
for ctr in range(10):
  fig.add_subplot(3,10,ctr+1)
  plt.imshow(np.reshape(data[a + ctr],(64,64)),  cmap = "gray")



for ctr in range(10):
  fig.add_subplot(3,10,(10 + ctr + 1))
  plt.imshow(np.reshape(label[a + ctr]/255,(64,64)),  cmap = "gray")


for ctr in range(10):
  fig.add_subplot(3,10,(20 + ctr + 1))
  plt.imshow(np.reshape(pred[ctr],(64,64)),  cmap = "gray")


```



```

a = 4160
b = 4170
pred=generator.predict(data[a:b])

fig = plt.figure(figsize = (20,10))
for ctr in range(10):
  fig.add_subplot(3,10,ctr+1)
  plt.imshow(np.reshape(data[a + ctr],(64,64)),  cmap = "gray")



for ctr in range(10):
  fig.add_subplot(3,10,(10 + ctr + 1))
  plt.imshow(np.reshape(label[a + ctr]/255,(64,64)),  cmap = "gray")


for ctr in range(10):
  fig.add_subplot(3,10,(20 + ctr + 1))
```

```
  plt.imshow(np.reshape(pred[ctr],(64,64)),  cmap = "gray")

plt.title("250 epochs")
```

```
Text(0.5, 1.0, '250 epochs')
```

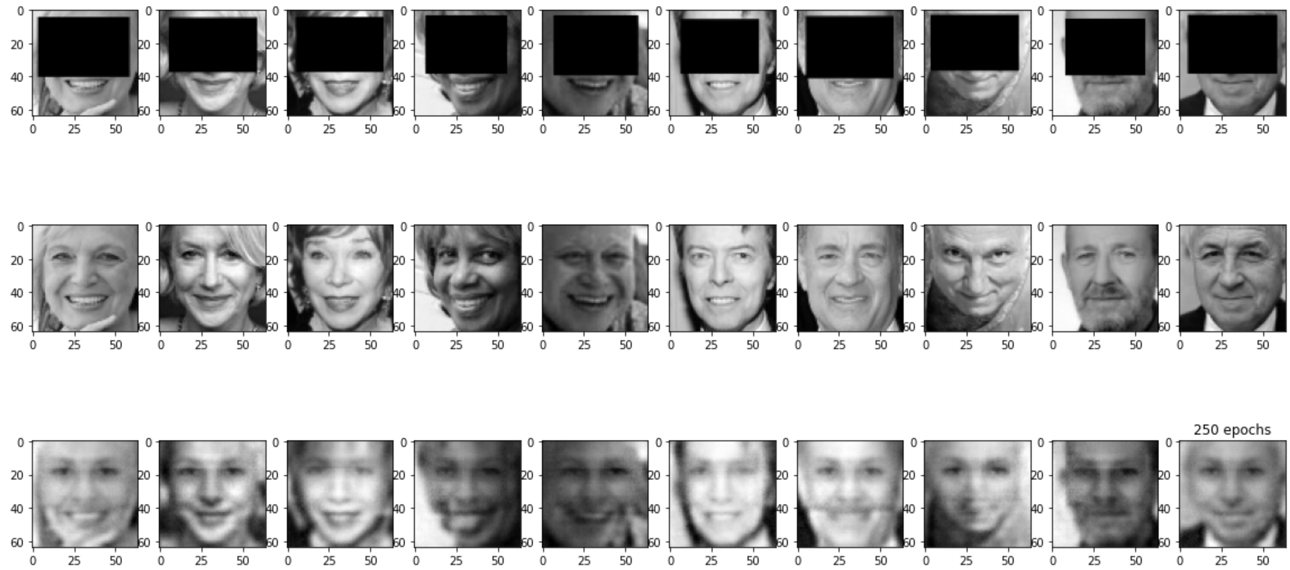
save model

```
tf.keras.models.save_model(generator,'/content/drive/MyDrive/Colab Notebooks/PROJECT/mo
```

```
gen_load_model=tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/PROJE
```

```

a = 4160
b = 4170
pred=gen_load_model.predict(data[a:b])

fig = plt.figure(figsize = (20,10))
for ctr in range(10):
  fig.add_subplot(3,10,ctr+1)
  plt.imshow(np.reshape(data[a + ctr],(64,64)),  cmap = "gray")



for ctr in range(10):
  fig.add_subplot(3,10,(10 + ctr + 1))
  plt.imshow(np.reshape(label[a + ctr]/255,(64,64)),  cmap = "gray")


for ctr in range(10):
  fig.add_subplot(3,10,(20 + ctr + 1))
  plt.imshow(np.reshape(pred[ctr],(64,64)),  cmap = "gray")

plt.title("250 epochs")
```

```
Text(0.5, 1.0, '250 epochs')
```

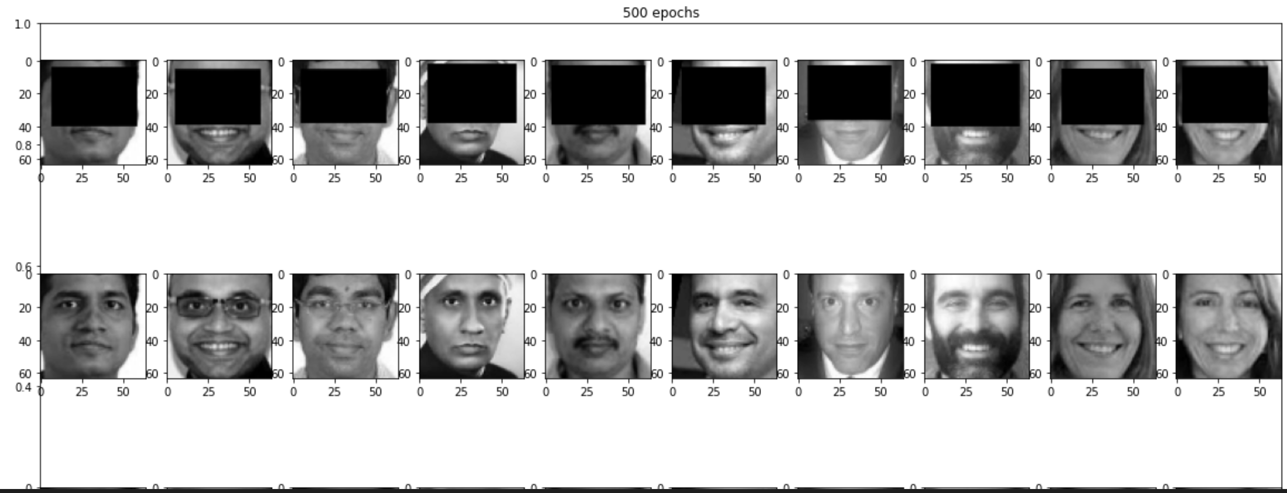


prediction

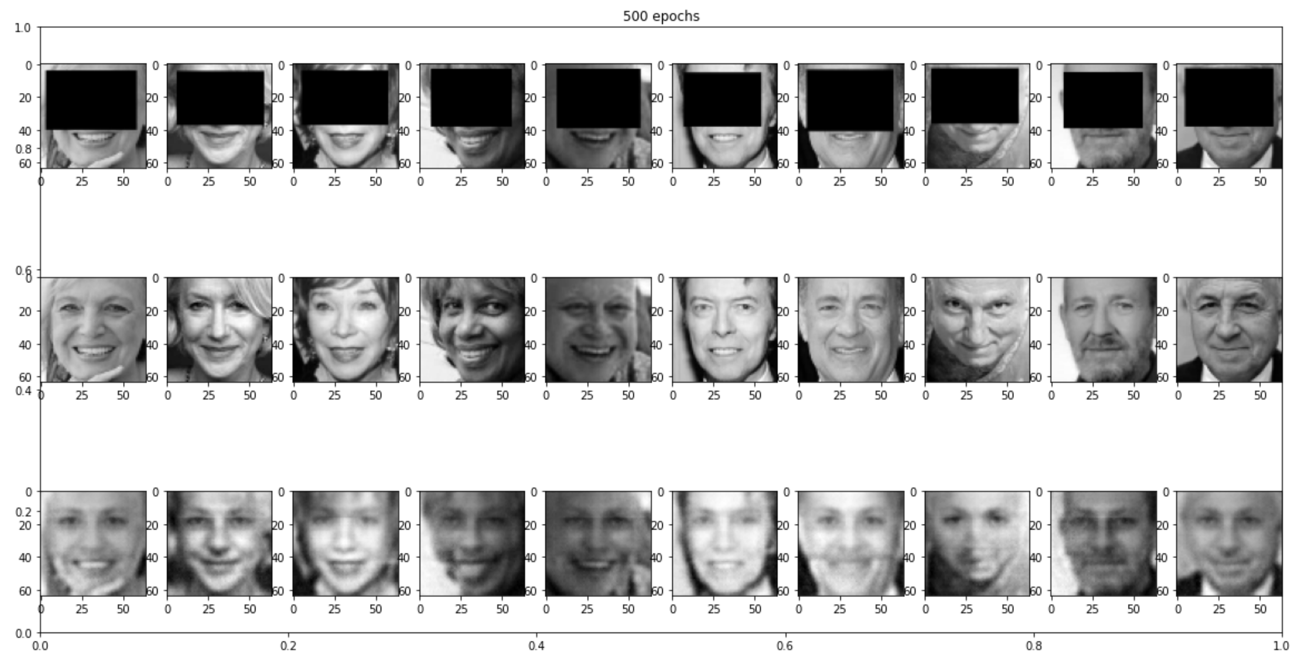
```
gen_500=tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/PROJECT/mode
```

```
gen_250=tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/PROJECT/mode
```

```
a = 3160
b = 3170
pred=gen_500.predict(data[a:b])

fig = plt.figure(figsize = (20,10))
plt.title("500 epochs")
for ctr in range(10):
  fig.add_subplot(3,10,ctr+1)
  plt.imshow(np.reshape(data[a + ctr],(64,64)),  cmap = "gray")

for ctr in range(10):
  fig.add_subplot(3,10,(10 + ctr + 1))
  plt.imshow(np.reshape(label[a + ctr]/255,(64,64)),  cmap = "gray")

for ctr in range(10):
  fig.add_subplot(3,10,(20 + ctr + 1))
  plt.imshow(np.reshape(pred[ctr],(64,64)),  cmap = "gray")
```




```
a = 4160
b = 4170
pred=gen_500.predict(data[a:b])

fig = plt.figure(figsize = (20,10))
plt.title("500 epochs")
for ctr in range(10):
  fig.add_subplot(3,10,ctr+1)
  plt.imshow(np.reshape(data[a + ctr],(64,64)),  cmap = "gray")

for ctr in range(10):
  fig.add_subplot(3,10,(10 + ctr + 1))
  plt.imshow(np.reshape(label[a + ctr]/255,(64,64)),  cmap = "gray")

for ctr in range(10):
  fig.add_subplot(3,10,(20 + ctr + 1))
  plt.imshow(np.reshape(pred[ctr],(64,64)),  cmap = "gray")
```




```
a = 4160
```

```
b = 4170
pred_500=gen_500.predict(data[a:b])
pred_250=gen_250.predict(data[a:b])

fig = plt.figure(figsize = (20,10))
plt.title("predictions")
for ctr in range(10):
  fig.add_subplot(4,10,ctr+1)
  plt.imshow(np.reshape(data[a + ctr],(64,64)),  cmap = "gray")

for ctr in range(10):
  fig.add_subplot(4,10,(10 + ctr + 1))
  plt.imshow(np.reshape(label[a + ctr]/255,(64,64)),  cmap = "gray",label="Original")

for ctr in range(10):
  fig.add_subplot(4,10,(20 + ctr + 1))
  plt.imshow(np.reshape(pred_250[ctr],(64,64)),  cmap = "gray")

plt.title("250 epochs training")

for ctr in range(10):
  fig.add_subplot(4,10,(30 + ctr + 1))
  plt.imshow(np.reshape(pred_500[ctr],(64,64)),  cmap = "gray")
plt.title("500 epochs training")
```

```
Text(0.5, 1.0, '500 epochs training')
```


```
gen_100_128_img=tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/PROJ
```

```
gen_250_128_img=tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/PROJ
```

```
y_path = '/content/1_half_face_labels.pickle'
x_path = '/content/1_half_face_occluded.pickle'
```

```
pickle_in = open(x_path,"rb")
x = pickle.load(pickle_in)
pickle_in = open(y_path,"rb")
y = pickle.load(pickle_in)
```

```
data=preprocess_img(x,128)
label=preprocess_img(y,128)
```

```
a = 4160
b = 4170
pred_100=gen_100_128_img.predict(data[a:b])
pred_250=gen_250_128_img.predict(data[a:b])
fig = plt.figure(figsize = (20,10))
plt.title("predictions")
for ctr in range(10):
  fig.add_subplot(4,10,ctr+1)
  plt.imshow(np.reshape(data[a + ctr],(128,128)),  cmap = "gray")
for ctr in range(10):
  fig.add_subplot(4,10,(10 + ctr + 1))
  plt.imshow(np.reshape(label[a + ctr]/255,(128,128)),  cmap = "gray",label="Original")
for ctr in range(10):
  fig.add_subplot(4,10,(20 + ctr + 1))
  plt.imshow(np.reshape(pred_100[ctr],(128,128)),  cmap = "gray")
plt.title("100 epochs training")
for ctr in range(10):
  fig.add_subplot(4,10,(30 + ctr + 1))
  plt.imshow(np.reshape(pred_250[ctr],(128,128)),  cmap = "gray")
plt.title("250 epochs training")
```

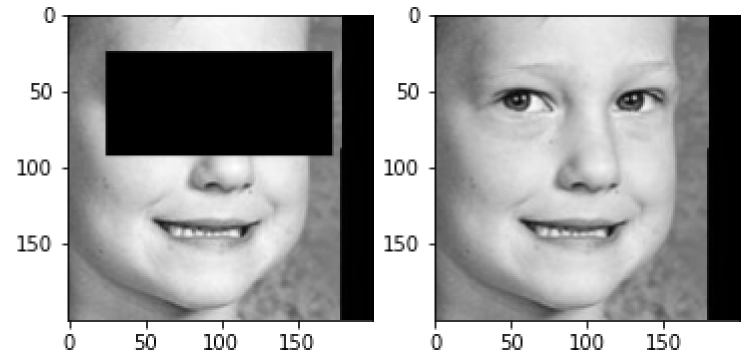
```
Text(0.5, 1.0, '250 epochs training')
```

New set of images

```
y_path = '/content/unres_labels.pickle'
x_path = '/content/unres_occluded.pickle'

pickle_in = open(x_path,"rb")
x = pickle.load(pickle_in)

pickle_in = open(y_path,"rb")
y = pickle.load(pickle_in)
```

```
fig=plt.figure(figsize=(6, 6))
fig.add_subplot(1, 2, 1)
plt.imshow(x[1,:,:,0],cmap="gray")
fig.add_subplot(1, 2, 2)
plt.imshow(y[1,:,:,0],cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f580ddfaed0>
```

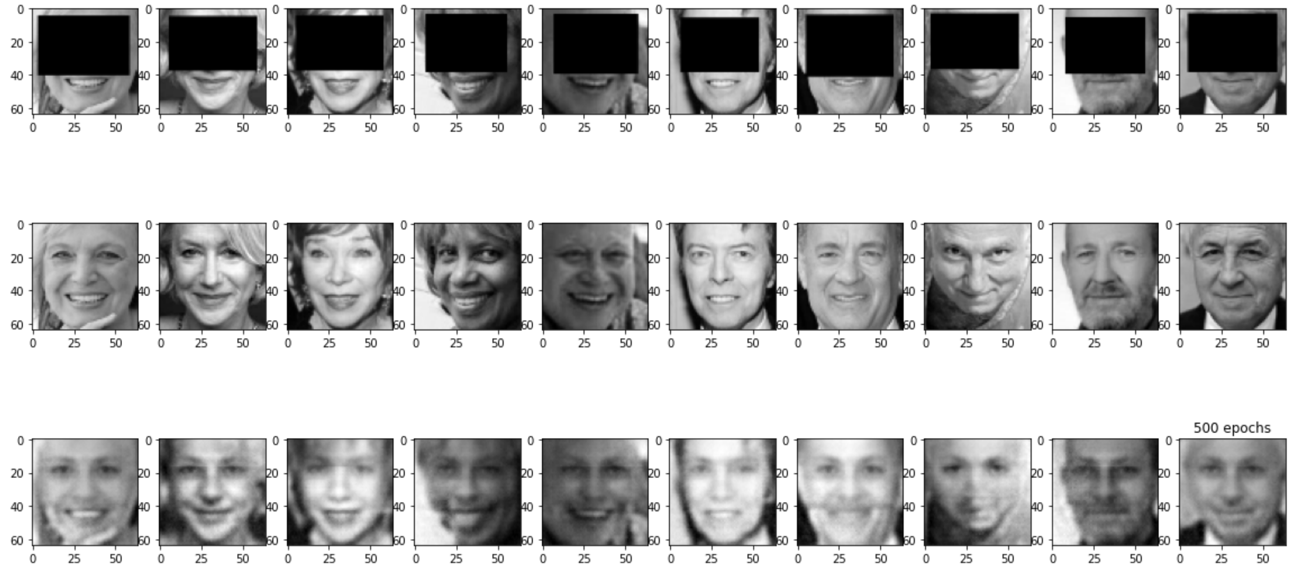


Evaluation

```
# laod save model
gen_500=tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/PROJECT/mode
```

```
a = 4160
b = 4170
pred=gen_500.predict(data[a:b])

fig = plt.figure(figsize = (20,10))
for ctr in range(10):
  fig.add_subplot(3,10,ctr+1)
  plt.imshow(np.reshape(data[a + ctr],(64,64)),  cmap = "gray")

for ctr in range(10):
  fig.add_subplot(3,10,(10 + ctr + 1))
  plt.imshow(np.reshape(label[a + ctr]/255,(64,64)),  cmap = "gray")

for ctr in range(10):
  fig.add_subplot(3,10,(20 + ctr + 1))
  plt.imshow(np.reshape(pred[ctr],(64,64)),  cmap = "gray")

plt.title("500 epochs")
```

```
Text(0.5, 1.0, '500 epochs')
```




```
import numpy
from numpy import cov
from numpy import trace
from numpy import iscomplexobj
from numpy.random import random
from scipy.linalg import sqrtm
```

```
# calculate frechet inception distance
def calculate_fid(act1, act2):

    # calculate mean and covariance statistics
    mean1, sigma1 = act1.mean(axis=0), cov(act1, rowvar=False)
    mean2, sigma2 = act2.mean(axis=0), cov(act2, rowvar=False)

    # calculate sum squared difference between means
    ssdiff = numpy.sum((mean1 - mean2)**2.0)

    # calculate sqrt of product between cov
    covmean = sqrtm(sigma1.dot(sigma2))

    # check and correct imaginary numbers from sqrt
    if iscomplexobj(covmean):
        covmean = covmean.real
    # calculate score
    fid = ssdiff + trace(sigma1 + sigma2 - 2.0 * covmean)
    return fid
```

```
fid_scores=[]
for i in range(10):
  fid_scores.append(calculate_fid(label[a+i,:,:,0].numpy(),np.reshape(pred[i],(64,64)))
```

```
fid=(1-tf.reduce_mean(fid_scores))*100
```

```
print(f"Generated images are {int(numpy.ceil(fid))}% same as Original image")
```

```
Generated images are 82% same as Original image
```