In [1]:
```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/dock
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list a

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets
# You can also write temporary files to /kaggle/temp/, but they won't be saved ou
```

In [1]:
```python
import tensorflow as tf
import tensorflow.keras.layers as tfl
import os
from tqdm import tqdm
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import keras
import cv2
from keras.models import Sequential
import keras
import os
from tqdm import tqdm
import re
import matplotlib.pyplot as plt
from tensorflow.keras.utils import img_to_array
```

In [2]:
```python
path='C:/Users/user/Desktop/Age and Gender/super_resol_data/Humans'

def images_upload(path):
    images=[]
    for root,subfolders,files in os.walk(path):
        for file in tqdm(files):
            filename=root+os.sep+file
            if filename.endswith('jpg') or filename.endswith('png'):
                images.append(filename)
    return images
images=images_upload(path)
```

```
100%|██████████████████████████████████████████████████|
7219/7219 [00:00<00:00, 2385650.85it/s]
```

In [3]:
```
HSIZE =256
LSIZE =64
```

In [4]:
```
def convert_high_image_labels(images):
    high_labels=[]
    j=0
    for i in tqdm(images):
        if(j==6000):
            break
        i = cv2.imread(i)
        i=cv2.cvtColor(i, cv2.COLOR_BGR2RGB)
        res_i=cv2.resize(i,(HSIZE,HSIZE))
        res_i = res_i.astype('float32') / 255.0

        del i
        high_labels.append(img_to_array(res_i))
        j = j+1
    return high_labels
high_labels = convert_high_image_labels(images)
```

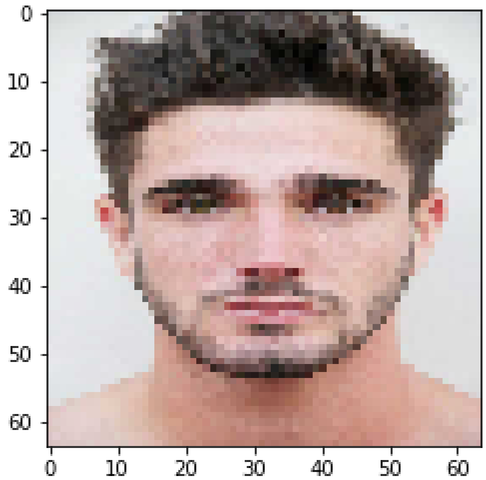
```
 84%|████████████████████████████████████████████▉
| 6000/7123 [02:12<00:24, 45.33it/s]
```

In [5]:
```
def convert_low_image_labels(images):
    low_labels=[]
    j=0
    for i in tqdm(images):
        if(j==6000):
            break
        i = cv2.imread(i)
        i=cv2.cvtColor(i, cv2.COLOR_BGR2RGB)
        res_i=cv2.resize(i,(LSIZE,LSIZE))
        res_i = res_i.astype('float32') / 255.0
        del i
        j= j+1
        low_labels.append(img_to_array(res_i))
    return low_labels
low_labels = convert_low_image_labels(images)
```

```
 84%|████████████████████████████████████████████▉
| 6000/7123 [01:46<00:20, 56.10it/s]
```

In [6]:
```
low_labels[0].shape
plt.imshow(low_labels[0])
```

Out[6]: <matplotlib.image.AxesImage at 0x15583064bb0>



In [7]:

```python
from keras import layers
def down(filters , kernel_low_size, apply_batch_normalization = True):
    downsample = tf.keras.models.Sequential()
    downsample.add(layers.Conv2D(filters,kernel_low_size,padding = 'same', stride
    if apply_batch_normalization:
        downsample.add(layers.BatchNormalization())
    downsample.add(keras.layers.LeakyReLU())
    return downsample


def up(filters, kernel_low_size, dropout = False):
    upsample = tf.keras.models.Sequential()
    upsample.add(layers.Conv2DTranspose(filters, kernel_low_size,padding = 'same'
    if dropout:
        upsample.dropout(0.2)
    upsample.add(keras.layers.LeakyReLU())
    return upsample

def model():
    inputs = layers.Input(shape= [LSIZE,LSIZE,3])
    print(inputs.shape)
    d1 = down(128,(3,3),False)(inputs)
    print(d1.shape)
    d2 = down(128,(3,3),False)(d1)
    print(d2.shape)
    d3 = down(256,(3,3),True)(d2)
    print(d3.shape)
    d4 = down(512,(3,3),True)(d3)
    print(d4.shape)

    d5 = down(512,(3,3),True)(d4)
    print(d5.shape)
    #upsampling
    u1 = up(512,(3,3),False)(d5)
    print(u1.shape)
    u1 = layers.concatenate([u1,d4])
    u2 = up(256,(3,3),False)(u1)
    print(u2.shape)
    u2 = layers.concatenate([u2,d3])
    u3 = up(128,(3,3),False)(u2)
    print(u3.shape)
    u3 = layers.concatenate([u3,d2])
    u4 = up(128,(3,3),False)(u3)
    u4 = layers.concatenate([u4,d1])
    u5 = up(3,(3,3),False)(u4)
    u6 = up(3,(3,3),False)(u5)
    u7 = up(3,(3,3),False)(u6)
    output = layers.Conv2D(3,(2,2),strides = 1, padding = 'same')(u7)
    return tf.keras.Model(inputs=inputs, outputs=output)

model = model()
model.summary()
```

```
(None, 64, 64, 3)
(None, 32, 32, 128)
(None, 16, 16, 128)
```

```
(None, 8, 8, 256)
(None, 4, 4, 512)
(None, 2, 2, 512)
(None, 4, 4, 512)
(None, 8, 8, 256)
(None, 16, 16, 128)
Model: "model"
_____________________________________________________________________________
_____________________
 Layer (type)                   Output Shape          Param #     Connected to
=============================================================================
=====================
 input_1 (InputLayer)           [(None, 64, 64, 3)]  0           []

 sequential (Sequential)        (None, 32, 32, 128)  3584        ['input_1[0]
[0]']

 sequential_1 (Sequential)      (None, 16, 16, 128)  147584      ['sequential
[0][0]']

 sequential_2 (Sequential)      (None, 8, 8, 256)    296192      ['sequential
_1[0][0]']

 sequential_3 (Sequential)      (None, 4, 4, 512)    1182208     ['sequential
_2[0][0]']

 sequential_4 (Sequential)      (None, 2, 2, 512)    2361856     ['sequential
_3[0][0]']

 sequential_5 (Sequential)      (None, 4, 4, 512)    2359808     ['sequential
_4[0][0]']

 concatenate (Concatenate)      (None, 4, 4, 1024)   0           ['sequential
_5[0][0]',
                                                                  'sequential
_3[0][0]']

 sequential_6 (Sequential)      (None, 8, 8, 256)    2359552     ['concatenat
e[0][0]']

 concatenate_1 (Concatenate)    (None, 8, 8, 512)    0           ['sequential
_6[0][0]',
                                                                  'sequential
_2[0][0]']

 sequential_7 (Sequential)      (None, 16, 16, 128)  589952      ['concatenat
e_1[0][0]']

 concatenate_2 (Concatenate)    (None, 16, 16, 256)  0           ['sequential
_7[0][0]',
                                                                  'sequential
_1[0][0]']

 sequential_8 (Sequential)      (None, 32, 32, 128)  295040      ['concatenat
e_2[0][0]']

 concatenate_3 (Concatenate)    (None, 32, 32, 256)  0           ['sequential
```

```
_8[0][0]',
                                                                               'sequential
[0][0]']

 sequential_9 (Sequential)      (None, 64, 64, 3)    6915        ['concatenat
e_3[0][0]']

 sequential_10 (Sequential)     (None, 128, 128, 3)  84          ['sequential
_9[0][0]']

 sequential_11 (Sequential)     (None, 256, 256, 3)  84          ['sequential
_10[0][0]']

 conv2d_5 (Conv2D)              (None, 256, 256, 3)  39          ['sequential
_11[0][0]']

=============================================================================
=====================
Total params: 9,602,898
Trainable params: 9,600,338
Non-trainable params: 2,560
_____________________________________________________________________________
_____________________
```

In [8]:
```python
len(high_labels)
```

Out[8]:
```
6000
```

In [9]:
```python

train_high_image = high_labels[:5000]
train_low_image = low_labels[:5000]
validation_high_image = high_labels[5000:]
validation_low_image = low_labels[5000:]

# train_high_image = np.reshape(train_high_image,(len(train_high_image),HSIZE,HSI
# train_low_image = np.reshape(train_low_image,(len(train_low_image),LSIZE,LSIZE,


# validation_high_image= np.reshape(validation_high_image,(len(validation_high_im
# validation_low_image = np.reshape(validation_low_image,(len(validation_low_imag


# test_high_image = high_labels[6500:]
# test_low_image = low_labels[6500:]
# test_high_image= np.reshape(test_high_image,(len(test_high_image),HSIZE,HSIZE,3
# test_low_image = np.reshape(test_low_image,(len(test_low_image),SIZE,SIZE,3))

# print("Shape of training images:",train_high_image.shape)
# # print("Shape of test images:",test_high_image.shape)
# print("Shape of validation images:",validation_high_image.shape)
```

In [10]:
```python
high_labels.clear()
low_labels.clear()
```

In [13]:

```
train_high_image = np.reshape(train_high_image[:5000],(5000,HSIZE,HSIZE,3))
train_low_image = np.reshape(train_low_image[:5000],(5000,LSIZE,LSIZE,3))


validation_high_image= np.reshape(validation_high_image[:1000],(1000,HSIZE,HSIZE,
validation_low_image = np.reshape(validation_low_image[:1000],(1000,LSIZE,LSIZE,3
```

In [14]:

```
print(train_high_image.shape)
print(train_low_image.shape)
print(validation_high_image.shape)
print(validation_low_image.shape)
```

```
(5000, 256, 256, 3)
(5000, 64, 64, 3)
(1000, 256, 256, 3)
(1000, 64, 64, 3)
```

In [15]:
```
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss =
              metrics = ['acc'])
H =model.fit(train_low_image, train_high_image, epochs = 20, batch_size = 1,
          validation_data = (validation_low_image,validation_high_image))
```

```
Epoch 1/20
5000/5000 [==============================] - 378s 75ms/step - loss: 0.0645 - ac
c: 0.7441 - val_loss: 0.0413 - val_acc: 0.8261
Epoch 2/20
5000/5000 [==============================] - 376s 75ms/step - loss: 0.0377 - ac
c: 0.7736 - val_loss: 0.0367 - val_acc: 0.8396
Epoch 3/20
5000/5000 [==============================] - 360s 72ms/step - loss: 0.0355 - ac
c: 0.7803 - val_loss: 0.0344 - val_acc: 0.8379
Epoch 4/20
5000/5000 [==============================] - 362s 72ms/step - loss: 0.0344 - ac
c: 0.7837 - val_loss: 0.0355 - val_acc: 0.8341
Epoch 5/20
5000/5000 [==============================] - 361s 72ms/step - loss: 0.0336 - ac
c: 0.7859 - val_loss: 0.0341 - val_acc: 0.8586
Epoch 6/20
5000/5000 [==============================] - 362s 72ms/step - loss: 0.0326 - ac
c: 0.7873 - val_loss: 0.0340 - val_acc: 0.8295
Epoch 7/20
5000/5000 [==============================] - 357s 71ms/step - loss: 0.0319 - ac
c: 0.7898 - val_loss: 0.0337 - val_acc: 0.8682
Epoch 8/20
5000/5000 [==============================] - 353s 71ms/step - loss: 0.0315 - ac
c: 0.7893 - val_loss: 0.0330 - val_acc: 0.8530
Epoch 9/20
5000/5000 [==============================] - 358s 72ms/step - loss: 0.0310 - ac
c: 0.7928 - val_loss: 0.0343 - val_acc: 0.8525
Epoch 10/20
5000/5000 [==============================] - 361s 72ms/step - loss: 0.0306 - ac
c: 0.7930 - val_loss: 0.0338 - val_acc: 0.8271
Epoch 11/20
5000/5000 [==============================] - 361s 72ms/step - loss: 0.0303 - ac
c: 0.7922 - val_loss: 0.0370 - val_acc: 0.8497
Epoch 12/20
5000/5000 [==============================] - 361s 72ms/step - loss: 0.0301 - ac
c: 0.7946 - val_loss: 0.0332 - val_acc: 0.8717
Epoch 13/20
5000/5000 [==============================] - 360s 72ms/step - loss: 0.0298 - ac
c: 0.7951 - val_loss: 0.0329 - val_acc: 0.8461
Epoch 14/20
5000/5000 [==============================] - 361s 72ms/step - loss: 0.0294 - ac
c: 0.7988 - val_loss: 0.0341 - val_acc: 0.8695
Epoch 15/20
5000/5000 [==============================] - 361s 72ms/step - loss: 0.0291 - ac
c: 0.7982 - val_loss: 0.0327 - val_acc: 0.8567
Epoch 16/20
5000/5000 [==============================] - 361s 72ms/step - loss: 0.0288 - ac
c: 0.8004 - val_loss: 0.0359 - val_acc: 0.8647
Epoch 17/20
5000/5000 [==============================] - 361s 72ms/step - loss: 0.0286 - ac
c: 0.7991 - val_loss: 0.0328 - val_acc: 0.8557
Epoch 18/20
```

```
5000/5000 [==============================] - 361s 72ms/step - loss: 0.0285 - ac
c: 0.7981 - val_loss: 0.0337 - val_acc: 0.8292
Epoch 19/20
5000/5000 [==============================] - 371s 74ms/step - loss: 0.0283 - ac
c: 0.7991 - val_loss: 0.0319 - val_acc: 0.8535
Epoch 20/20
5000/5000 [==============================] - 377s 75ms/step - loss: 0.0281 - ac
c: 0.7990 - val_loss: 0.0344 - val_acc: 0.8755
```

In [16]:

```
new_low_images = model.predict(train_low_image)
```

```
157/157 [==============================] - 34s 214ms/step
```

In [17]:

```
new_validation_low_image = model.predict(validation_low_image)
```

```
32/32 [==============================] - 7s 200ms/step
```

In [18]:

```python
def second_model():
    inputs = layers.Input(shape= [HSIZE,HSIZE,3])
    print(inputs.shape)
    d1 = down(128,(3,3),False)(inputs)
    print(d1.shape)
    d2 = down(128,(3,3),False)(d1)
    print(d2.shape)
    d3 = down(256,(3,3),True)(d2)
    print(d3.shape)
    d4 = down(512,(3,3),True)(d3)
    print(d4.shape)

    d5 = down(512,(3,3),True)(d4)
    print(d5.shape)
    #upsampling
    u1 = up(512,(3,3),False)(d5)
    print(u1.shape)
    u1 = layers.concatenate([u1,d4])
    u2 = up(256,(3,3),False)(u1)
    print(u2.shape)
    u2 = layers.concatenate([u2,d3])
    u3 = up(128,(3,3),False)(u2)
    print(u3.shape)
    u3 = layers.concatenate([u3,d2])
    u4 = up(128,(3,3),False)(u3)
    u4 = layers.concatenate([u4,d1])
    u5 = up(3,(3,3),False)(u4)

    output = layers.Conv2D(3,(2,2),strides = 1, padding = 'same')(u5)
    return tf.keras.Model(inputs=inputs, outputs=output)

smodel = second_model()
smodel.summary()
```

```
(None, 256, 256, 3)
(None, 128, 128, 128)
(None, 64, 64, 128)
(None, 32, 32, 256)
(None, 16, 16, 512)
(None, 8, 8, 512)
(None, 16, 16, 512)
(None, 32, 32, 256)
(None, 64, 64, 128)
Model: "model_1"
__________________________________________________________________________________
________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
=====================
 input_2 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 sequential_12 (Sequential)     (None, 128, 128, 12  3584        ['input_2[0]
[0]']
                                8)
```

```
 sequential_13 (Sequential)     (None, 64, 64, 128)  147584      ['sequential
_12[0][0]']

 sequential_14 (Sequential)     (None, 32, 32, 256)  296192      ['sequential
_13[0][0]']

 sequential_15 (Sequential)     (None, 16, 16, 512)  1182208     ['sequential
_14[0][0]']

 sequential_16 (Sequential)     (None, 8, 8, 512)    2361856     ['sequential
_15[0][0]']

 sequential_17 (Sequential)     (None, 16, 16, 512)  2359808     ['sequential
_16[0][0]']

 concatenate_4 (Concatenate)    (None, 16, 16, 1024  0           ['sequential
_17[0][0]',
                                )                                 'sequential
_15[0][0]']

 sequential_18 (Sequential)     (None, 32, 32, 256)  2359552     ['concatenat
e_4[0][0]']

 concatenate_5 (Concatenate)    (None, 32, 32, 512)  0           ['sequential
_18[0][0]',
                                                                  'sequential
_14[0][0]']

 sequential_19 (Sequential)     (None, 64, 64, 128)  589952      ['concatenat
e_5[0][0]']

 concatenate_6 (Concatenate)    (None, 64, 64, 256)  0           ['sequential
_19[0][0]',
                                                                  'sequential
_13[0][0]']

 sequential_20 (Sequential)     (None, 128, 128, 12  295040      ['concatenat
e_6[0][0]']
                                8)

 concatenate_7 (Concatenate)    (None, 128, 128, 25  0           ['sequential
_20[0][0]',
                                6)                                'sequential
_12[0][0]']

 sequential_21 (Sequential)     (None, 256, 256, 3)  6915        ['concatenat
e_7[0][0]']

 conv2d_11 (Conv2D)             (None, 256, 256, 3)  39          ['sequential
_21[0][0]']

==============================================================================
=====================
Total params: 9,602,730
Trainable params: 9,600,170
Non-trainable params: 2,560
```

```
In [19]: smodel.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss
                       metrics = ['acc'])
         h= smodel.fit(new_low_images, train_high_image, epochs = 7, batch_size = 1,
                   validation_data = (new_validation_low_image,validation_high_image))
```

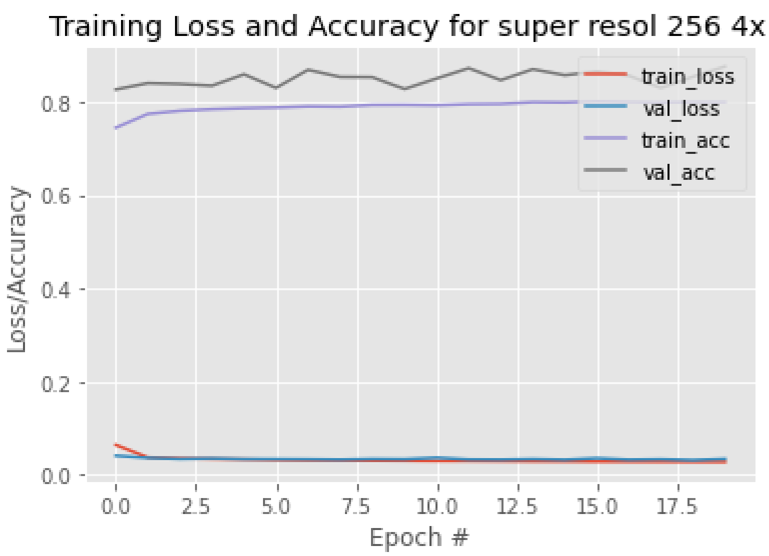
```
Epoch 1/7
5000/5000 [==============================] - 1115s 223ms/step - loss: 0.0361 -
acc: 0.7559 - val_loss: 0.0337 - val_acc: 0.8641
Epoch 2/7
5000/5000 [==============================] - 855s 171ms/step - loss: 0.0301 - a
cc: 0.7874 - val_loss: 0.0319 - val_acc: 0.8727
Epoch 3/7
5000/5000 [==============================] - 848s 170ms/step - loss: 0.0286 - a
cc: 0.7910 - val_loss: 0.0313 - val_acc: 0.8673
Epoch 4/7
5000/5000 [==============================] - 856s 171ms/step - loss: 0.0275 - a
cc: 0.7917 - val_loss: 0.0315 - val_acc: 0.8533
Epoch 5/7
5000/5000 [==============================] - 864s 173ms/step - loss: 0.0267 - a
cc: 0.7939 - val_loss: 0.0322 - val_acc: 0.8367
Epoch 6/7
5000/5000 [==============================] - 854s 171ms/step - loss: 0.0259 - a
cc: 0.7932 - val_loss: 0.0315 - val_acc: 0.8259
Epoch 7/7
5000/5000 [==============================] - 850s 170ms/step - loss: 0.0254 - a
cc: 0.7943 - val_loss: 0.0321 - val_acc: 0.8610
```

```
In [ ]: datagen = ImageDataGenerator(
                featurewise_center=False,  # set input mean to 0 over the dataset
                samplewise_center=False,  # set each sample mean to 0
                featurewise_std_normalization=False,  # divide inputs by std of the datas
                samplewise_std_normalization=False,  # divide each input by its std
                zca_whitening=False,  # dimesion reduction
                rotation_range=5,  # randomly rotate images in the range 5 degrees
                zoom_range = 0.1, # Randomly zoom image 10%
                width_shift_range=0.1,  # randomly shift images horizontally 10%
                height_shift_range=0.1,  # randomly shift images vertically 10%
                horizontal_flip=False,  # randomly flip images
                vertical_flip=False)  # randomly flip images

        datagen.fit(x)
```

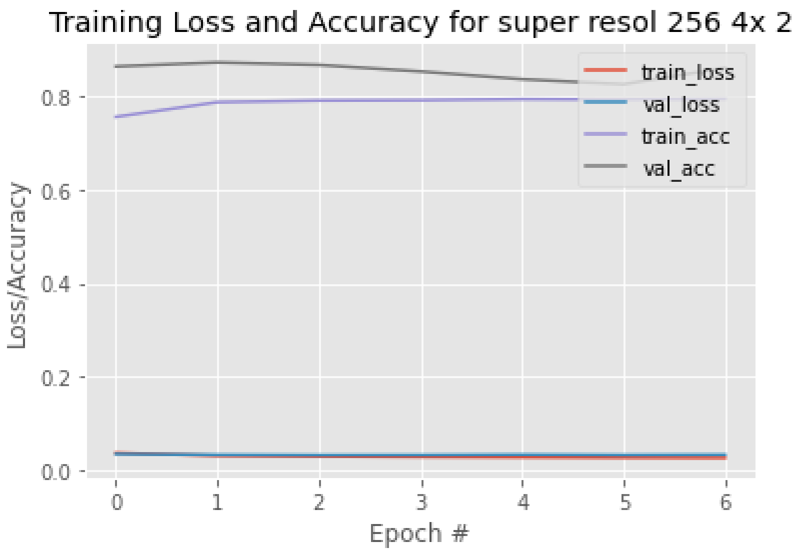
```
In [ ]:
```

In [25]:

```
plt.style.use("ggplot")
plt.figure()
N = 20
plt.plot(np.arange(0,N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0,N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0,N), H.history["acc"], label="train_acc")
plt.plot(np.arange(0,N), H.history["val_acc"], label="val_acc")

plt.title("Training Loss and Accuracy for super resol 256 4x 1")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="upper right")

# save plot to disk
plt.savefig('super resol 256 4x 1.png')
```



In [27]:
```
plt.style.use("ggplot")
plt.figure()
N = 7
plt.plot(np.arange(0,N), h.history["loss"], label="train_loss")
plt.plot(np.arange(0,N), h.history["val_loss"], label="val_loss")
plt.plot(np.arange(0,N), h.history["acc"], label="train_acc")
plt.plot(np.arange(0,N), h.history["val_acc"], label="val_acc")

plt.title("Training Loss and Accuracy for super resol 256 4x 2")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="upper right")

# save plot to disk
plt.savefig('super resol 256 4x 2.png')
```



In [ ]:
```

```

In [ ]:
```
model.save('super_resol.model')
```

In [ ]:
```
plt.imshow(low_labels[0])
```

In [ ]:
```
plt.imshow(high_labels[0])
```

In [ ]:
```
train_low_image[0].shape
```

In [ ]:
```
x =  np.reshape(train_l_image[0],(1,SIZE,SIZE,3))
xx = model.predict(x)[0]
```

In [ ]:
```

xx=cv2.cvtColor(xx, cv2.COLOR_BGR2RGB)
ply
```

In [ ]:
```
t = np.reshape(train_high_image[0],(196608))
```

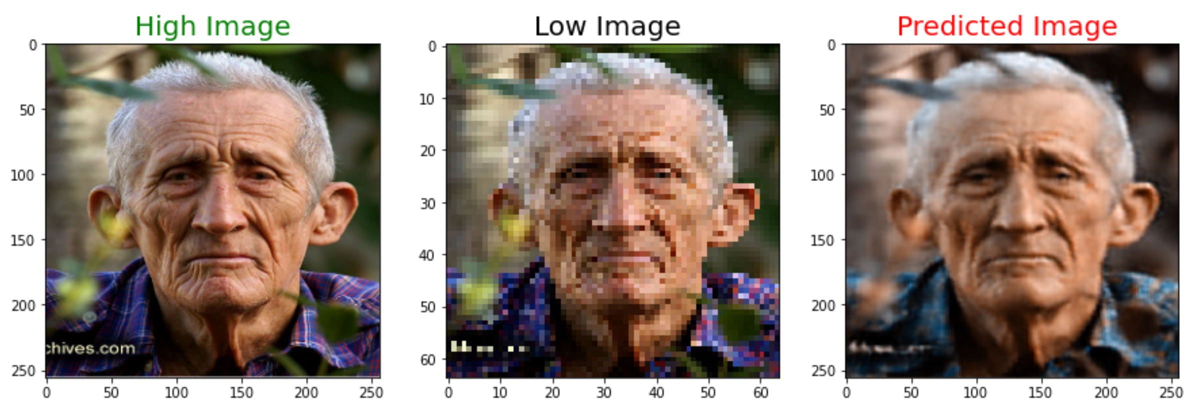
In [ ]:
```
import math
from collections import defaultdict
counts1 =defaultdict(lambda: 0)
for i in t:
    counts1[math.floor(i)] = counts1[math.floor(i)]+1
```

In [ ]:
```
plt.bar(range(len(counts1)), list(counts1.values()), align='center')
plt.xticks(range(len(counts1)), list(counts1.keys()))
plt.show()
```

In [ ]:
```
plt.bar(range(len(counts)), list(counts.values()), align='center')
plt.xticks(range(len(counts)), list(counts.keys()))
plt.show()
```

In [20]:
```
def plot_images(high,low,predicted):
    plt.figure(figsize=(15,15))
    plt.subplot(1,3,1)
    plt.title('High Image', color = 'green', fontsize = 20)
    plt.imshow(high)
    plt.subplot(1,3,2)
    plt.title('Low Image ', color = 'black', fontsize = 20)
    plt.imshow(low)
    plt.subplot(1,3,3)
    plt.title('Predicted Image ', color = 'Red', fontsize = 20)
    plt.imshow(predicted)

    plt.show()

for i in range(1,10):

    predicted = np.clip(model.predict(train_low_image[i].reshape(1,LSIZE, LSIZE,3
    plot_images(train_high_image[i],train_low_image[i],predicted)
```

```
1/1 [==============================] - 0s 28ms/step
```




```
1/1 [==============================] - 0s 29ms/step
```

In [ ]:
```
!tar -zcvf outputname.tar.gz /kaggle/working
```

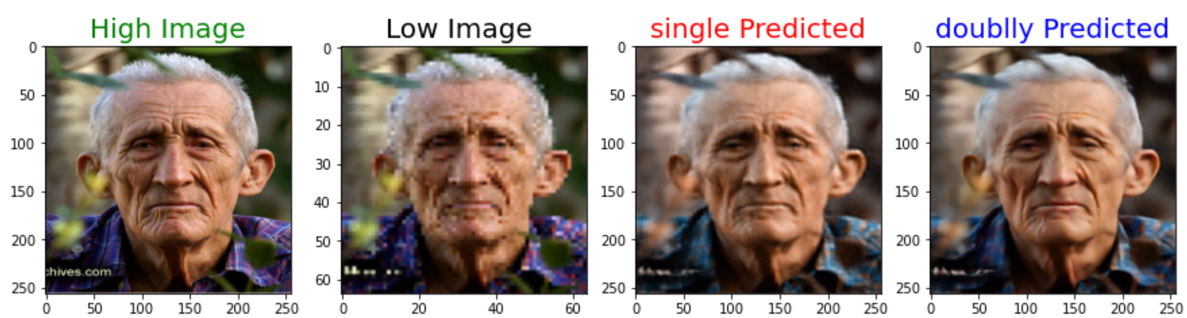
In [ ]:
```
import shutil
shutil.make_archive("OUTPUT_NAME", 'zip', DIRECTORY_TO_ZIP)
```

In [ ]:
```
from tensorflow.keras.models import load_model
m = load_model("super_resol.model")
```

In [ ]:
```
import tensorflow_hub as hub
import tensorflow as tf
esm = hub.load("https://tfhub.dev/captain-pool/esrgan-tf2/1")
# To add an extra dimension for batch, use tf.expand_dims()
low_resolution_image = train_low_image[1].reshape(1,SIZE, SIZE,3)# Low Resolutior
low_resolution_image = tf.cast(low_resolution_image, tf.float32)
predicted = np.clip(esm(train_low_image[1].reshape(1,SIZE, SIZE,3)),0.0,1.0).resh
```

In [21]:
```python
def plot_images(high,low,predicted1,predicted2):
    plt.figure(figsize=(15,15))
    plt.subplot(1,4,1)
    plt.title('High Image', color = 'green', fontsize = 20)
    plt.imshow(high)
    plt.subplot(1,4,2)
    plt.title('Low Image ', color = 'black', fontsize = 20)
    plt.imshow(low)
    plt.subplot(1,4,3)
    plt.title('single Predicted', color = 'Red', fontsize = 20)
    plt.imshow(predicted1)
    plt.subplot(1,4,4)
    plt.title('doublly Predicted', color = 'blue', fontsize = 20)
    plt.imshow(predicted2)

    plt.show()

for i in range(1,10):

    predicted1 = np.clip(model.predict(train_low_image[i].reshape(1,LSIZE, LSIZE,
    predicted2 = np.clip(smodel.predict(predicted1.reshape(1,HSIZE, HSIZE,3)),0.0

    plot_images(train_high_image[i],train_low_image[i],predicted1,predicted2)
```

```
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 177ms/step
```




```
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 56ms/step
```

In [ ]:

```
def plot_images(high,low,predicted):
    plt.figure(figsize=(15,15))
    plt.subplot(1,3,1)
    plt.title('High Image', color = 'green', fontsize = 20)
    plt.imshow(high)
    plt.subplot(1,3,2)
    plt.title('Low Image ', color = 'black', fontsize = 20)
    plt.imshow(low)
    plt.subplot(1,3,3)
    plt.title('Predicted Image ', color = 'Red', fontsize = 20)
    plt.imshow(predicted)

    plt.show()

for i in range(1,10):

    predicted = np.clip(smodel.predict(new_low_images[i].reshape(1,HSIZE, HSIZE,3
    plot_images(train_high_image[i],train_low_image[i],predicted)
```

In [22]:

```
model.save("high_resol1_256_4x.model")
smodel.save("high_resol2_256_4x.model")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _ji
t_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolut
ion_op, _jit_compiled_convolution_op while saving (showing 5 of 13). These func
tions will not be directly callable after loading.

INFO:tensorflow:Assets written to: high_resol1_256_4x.model\assets

INFO:tensorflow:Assets written to: high_resol1_256_4x.model\assets
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _ji
t_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolut
ion_op, _jit_compiled_convolution_op while saving (showing 5 of 11). These func
tions will not be directly callable after loading.

INFO:tensorflow:Assets written to: high_resol2_256_4x.model\assets

INFO:tensorflow:Assets written to: high_resol2_256_4x.model\assets
```

In [ ]: