deep-cnn-image-classifier

February 16, 2024

1 Deep CNN Image Classifier with ANY Images.

1.0.1 1. Install Dependencies and Setup

[1]:
```
import tensorflow as tf
import os

# Avoid OOM errors by setting GPU Memory Consumption Growth
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
  tf.config.experimental.set_memory_growth(gpu, True)
```

[2]:
```
gpus
```

[2]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

[3]:
```
gpus = tf.config.experimental.list_physical_devices('CPU')
gpus
```

[3]:
```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```

1.1 2. Remove Dodgy images

[4]:
```
import cv2
import imghdr
```

[5]:
```
data_dir = 'drive/MyDrive/Project 13 Classification/data'
```

[6]:
```
# cheking no of images in the dataset
total_images = 0

# Loop over each subdirectory
for folder_name in os.listdir(data_dir):
    folder_path = os.path.join(data_dir, folder_name)

    # Ensure the path is a directory before counting images
    if os.path.isdir(folder_path):
```

```
        num_images = len(os.listdir(folder_path))
        total_images += num_images

print(f'Total number of images: {total_images}')
```

```
Total number of images: 225
```

[7]:
```
image_exts = ['jpeg','jpg','bmp','png']
```

This code is used to read and process images from a directory. It checks the file type of each image and deletes any images that are not of the expected file type or cannot be read.

[8]:
```
for image_class in os.listdir(data_dir):
  for image in os.listdir(os.path.join(data_dir, image_class)):
    image_path = os.path.join(data_dir, image_class, image)
    try:
      img = cv2.imread(image_path)
      tip = imghdr.what(image_path)
      if tip not in image_exts:
        print('Image not ext list {}'.format(image_path))
        os.remove(image_path)
    except Exception as e:
      print('Issue with image {}'.formt(image_path))
      os.remove(image_path)
```

1.1.1 3. Load Data

[9]:
```
import numpy as np
from matplotlib import pyplot as plt
```

[10]:
```
data = 'drive/MyDrive/Project 13 Classification/data'
```

Using the Keras.Utilit function to preprocess the data into shape and size that I need in its default formart

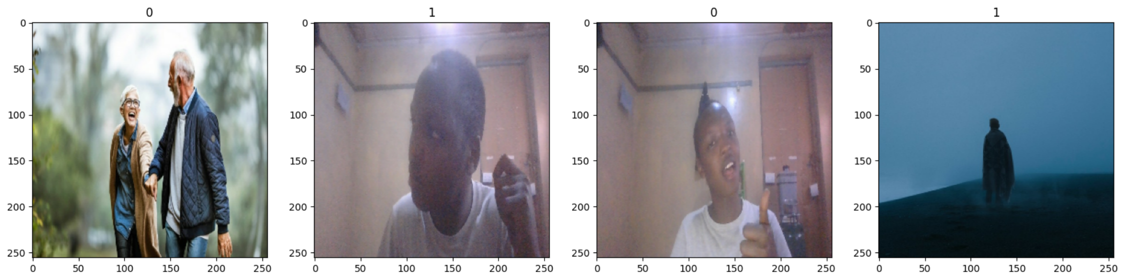
[11]:
```
data = tf.keras.utils.image_dataset_from_directory(data)
```

```
Found 225 files belonging to 2 classes.
```

[12]:
```
data_iterator = data.as_numpy_iterator()
```

[13]:
```
batch = data_iterator.next()
```

[14]:
```
fig, ax = plt.subplots(ncols = 4, figsize = (20,20))
for idx, img in enumerate(batch[0][:4]):
  ax[idx].imshow(img.astype(int))
  ax[idx].title.set_text(batch[1][idx])
```



1 represents Sad people and 0 represents Happy people

1.1.2 4. Scale The data

[15]:
```
data = data.map(lambda x, y: (x/255, y))
```

[16]:
```
data.as_numpy_iterator().next()
```

[16]:
```
(array([[[[0.08583793, 0.12127757, 0.23587623],
          [0.0723269 , 0.11285233, 0.25447303],
          [0.05271906, 0.10660999, 0.28235295],
          …,
          [0.1279565 , 0.1279565 , 0.2769761 ],
          [0.12941177, 0.12941177, 0.2714614 ],
          [0.12941177, 0.12941177, 0.27058825]],

         [[0.08583793, 0.11662266, 0.22897713],
          [0.07282658, 0.10387561, 0.24325214],
          [0.06012222, 0.09804967, 0.27029926],
          …,
          [0.12612872, 0.12571232, 0.27556473],
          [0.12716758, 0.12716758, 0.26921722],
          [0.12716758, 0.12716758, 0.26834404]],

         [[0.09159287, 0.115437  , 0.22269455],
          [0.08308484, 0.10066916, 0.23348859],
          [0.08546917, 0.09615503, 0.25874773],
          …,
          [0.12497147, 0.12220091, 0.27359277],
          [0.1254902 , 0.12323835, 0.26841506],
          [0.1254902 , 0.12323835, 0.26779258]],

         …,

         [[0.1102405 , 0.13376991, 0.2749464 ],
          [0.10590253, 0.12899536, 0.27104497],
```

```
  [0.10048043, 0.12008827, 0.2691079 ],
  …,
  [0.13012198, 0.15602367, 0.3416973 ],
  [0.12378217, 0.15092191, 0.33193654],
  [0.12000613, 0.14745711, 0.32784927]],

 [[0.09741212, 0.11926415, 0.26379538],
  [0.09500138, 0.11660357, 0.2616345 ],
  [0.09259064, 0.11219849, 0.26121807],
  …,
  [0.14298128, 0.17069827, 0.35683978],
  [0.13612132, 0.16499986, 0.34607843],
  [0.13234529, 0.16147365, 0.3418658 ]],

 [[0.08627451, 0.10588235, 0.25490198],
  [0.08627451, 0.10588235, 0.25490198],
  [0.08627451, 0.10588235, 0.25490198],
  …,
  [0.15237439, 0.18374693, 0.36806065],
  [0.14509805, 0.1764706 , 0.35729933],
  [0.141322  , 0.17269455, 0.3530867 ]]],


[[[0.1799326 , 0.7448836 , 0.7682292 ],
  [0.1799326 , 0.7448836 , 0.7682292 ],
  [0.1799326 , 0.7448836 , 0.7682292 ],
  …,
  [0.1799326 , 0.7448836 , 0.7682292 ],
  [0.1799326 , 0.7448836 , 0.7682292 ],
  [0.1799326 , 0.7448836 , 0.7682292 ]],

 [[0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  …,
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335]],

 [[0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  …,
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335]],
```

```
 …,

 [[0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  …,
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335]],

 [[0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  …,
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335]],

 [[0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  …,
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335],
  [0.05882353, 0.7058824 , 0.73333335]]],


[[[0.9421875 , 0.9186581 , 0.9244792 ],
  [0.94509804, 0.9254902 , 0.9106924 ],
  [0.93838847, 0.9187806 , 0.8952512 ],
  …,
  [0.9499344 , 0.93424815, 0.92248344],
  [0.9448065 , 0.92912024, 0.91735554],
  [0.94509804, 0.92941177, 0.91764706]],

 [[0.9421875 , 0.9186581 , 0.9244792 ],
  [0.94509804, 0.9254902 , 0.9106924 ],
  [0.93838847, 0.9187806 , 0.8952512 ],
  …,
  [0.94231004, 0.92662376, 0.91485906],
  [0.9499081 , 0.9342218 , 0.9224571 ],
  [0.94509804, 0.92941177, 0.91764706]],

 [[0.9421875 , 0.9186581 , 0.9244792 ],
  [0.94509804, 0.9254902 , 0.9106924 ],
  [0.93838847, 0.9187806 , 0.8952512 ],
  …,
```

```
   [0.94650733, 0.93082106, 0.91905636],
   [0.9476103 , 0.93192405, 0.92015934],
   [0.94509804, 0.92941177, 0.91764706]],

  …,

  [[0.11997549, 0.15919118, 0.16703431],
   [0.11352395, 0.15273964, 0.16058278],
   [0.11764706, 0.14901961, 0.16078432],
   …,
   [0.4392157 , 0.47058824, 0.6179534 ],
   [0.43836263, 0.4703867 , 0.6008885 ],
   [0.43308824, 0.4762255 , 0.6017157 ]],

  [[0.11368001, 0.1528957 , 0.16073884],
   [0.1106924 , 0.1499081 , 0.15775123],
   [0.11378676, 0.14515932, 0.15692402],
   …,
   [0.42371324, 0.46169052, 0.60631126],
   [0.42454043, 0.45983455, 0.58924633],
   [0.4272059 , 0.47034314, 0.59583336]],

  [[0.11014093, 0.14935662, 0.15719976],
   [0.12484585, 0.16406155, 0.17190468],
   [0.11335354, 0.14472608, 0.15649079],
   …,
   [0.4208946 , 0.45897672, 0.60355395],
   [0.41786152, 0.46099877, 0.58648896],
   [0.4143995 , 0.45753676, 0.58302695]]],


 …,


 [[[0.02331495, 0.01939338, 0.00370711],
   [0.04571078, 0.03002451, 0.0182598 ],
   [0.05892549, 0.04716079, 0.02755295],
   …,
   [0.04313726, 0.06666667, 0.06666667],
   [0.04283711, 0.05852338, 0.07028808],
   [0.03529412, 0.05251225, 0.06427696]],

  [[0.05690583, 0.04121955, 0.02945485],
   [0.05223652, 0.03655025, 0.02478554],
   [0.05900735, 0.04724265, 0.0276348 ],
   …,
   [0.04313726, 0.06666667, 0.06666667],
```

```
  [0.04683239, 0.06251867, 0.07428338],
  [0.03534917, 0.05256731, 0.06433202]],

 [[0.0836397 , 0.06403186, 0.05226716],
  [0.04712058, 0.0314343 , 0.0196696 ],
  [0.04808948, 0.04416791, 0.02456007],
  …,
  [0.04024634, 0.06377576, 0.06377576],
  [0.04705882, 0.0627451 , 0.07450981],
  [0.03953211, 0.05675025, 0.06851496]],

 …,

 [[0.04947917, 0.0573223 , 0.03771446],
  [0.04963235, 0.05747549, 0.04436275],
  [0.0579767 , 0.06581984, 0.05405513],
  …,
  [0.10283826, 0.10283826, 0.07118136],
  [0.06525687, 0.07702158, 0.0495706 ],
  [0.06935269, 0.0811174 , 0.05366642]],

 [[0.05775123, 0.0616728 , 0.04053309],
  [0.05165441, 0.05557598, 0.03596814],
  [0.05508579, 0.05508579, 0.04724265],
  …,
  [0.07337622, 0.06945466, 0.04984681],
  [0.05165441, 0.07126226, 0.04773284],
  [0.05729167, 0.07689951, 0.05183823]],

 [[0.06670544, 0.07062701, 0.04403387],
  [0.05165441, 0.05557598, 0.03204657],
  [0.05508579, 0.05900735, 0.03939951],
  …,
  [0.06481837, 0.06873995, 0.0491321 ],
  [0.05098039, 0.07058824, 0.04705882],
  [0.0562275 , 0.07583535, 0.05230593]]],


[[[0.0870481 , 0.13043046, 0.0755285 ],
  [0.0508004 , 0.09785922, 0.04295726],
  [0.07494638, 0.12984835, 0.07102481],
  …,
  [0.887343  , 0.89101946, 0.92116654],
  [0.8901961 , 0.89411765, 0.91158473],
  [0.89411765, 0.8901961 , 0.9135608 ]],

 [[0.10459942, 0.14828432, 0.08144148],
```

```
 [0.08870251, 0.1396829 , 0.07295496],
 [0.09021523, 0.1451172 , 0.08620941],
 …,
 [0.8877643 , 0.8940985 , 0.91099113],
 [0.89025736, 0.89411765, 0.9019608 ],
 [0.89411765, 0.89411765, 0.9019608 ]],

[[0.13482307, 0.18221508, 0.10087699],
 [0.12175628, 0.17273667, 0.09932981],
 [0.09508655, 0.15095742, 0.0839231 ],
 …,
 [0.8877451 , 0.89166665, 0.90832186],
 [0.89270836, 0.8936466 , 0.9059858 ],
 [0.89270836, 0.89411765, 0.90477943]],

…,

[[0.07266391, 0.12072993, 0.08167126],
 [0.09770986, 0.13974418, 0.10304075],
 [0.06666284, 0.10220205, 0.06813343],
 …,
 [0.2565602 , 0.30607   , 0.21317784],
 [0.25484067, 0.2893995 , 0.18572304],
 [0.22944623, 0.26973805, 0.15971585]],

[[0.05924479, 0.0868796 , 0.05544577],
 [0.06571309, 0.09334788, 0.06191406],
 [0.0639208 , 0.09511719, 0.06249617],
 …,
 [0.2859107 , 0.33419502, 0.2503715 ],
 [0.1971852 , 0.22860754, 0.13434054],
 [0.20478325, 0.24375384, 0.13787147]],

[[0.0313151 , 0.04851026, 0.02105928],
 [0.02414982, 0.04375766, 0.01630668],
 [0.05180377, 0.08293122, 0.05033318],
 …,
 [0.27328813, 0.32157245, 0.23951823],
 [0.21140854, 0.2427811 , 0.14939874],
 [0.17912453, 0.21785003, 0.12005591]]],


[[[0.22218138, 0.22218138, 0.19007353],
  [0.26813725, 0.23284313, 0.2125    ],
  [0.3348039 , 0.25674018, 0.29093137],
  …,
  [0.41789216, 0.40294117, 0.45747548],
```

```
         [0.39191177, 0.34485295, 0.3997549 ],
         [0.41066176, 0.36360294, 0.37144607]],

        [[0.24093138, 0.21960784, 0.21629901],
         [0.2617647 , 0.22254902, 0.22316177],
         [0.31875   , 0.25490198, 0.29693627],
         …,
         [0.4302696 , 0.41458333, 0.46948528],
         [0.41507354, 0.36409312, 0.43075982],
         [0.4509804 , 0.40563726, 0.43590686]],

        [[0.22058824, 0.19105393, 0.22683823],
         [0.25808823, 0.21495098, 0.24264705],
         [0.3262255 , 0.28125   , 0.34031862],
         …,
         [0.39791667, 0.37438726, 0.42928922],
         [0.41703433, 0.37389705, 0.44056374],
         [0.44154412, 0.39240196, 0.45355392]],

        …,

        [[0.5686275 , 0.49019608, 0.53333336],
         [0.53933823, 0.4569853 , 0.4922794 ],
         [0.56421566, 0.4897059 , 0.5132353 ],
         …,
         [0.41335785, 0.32316175, 0.35502452],
         [0.41458333, 0.32230392, 0.3677696 ],
         [0.4209559 , 0.34068626, 0.3822304 ]],

        [[0.5283088 , 0.4627451 , 0.50306374],
         [0.5598039 , 0.47855392, 0.5160539 ],
         [0.54791665, 0.47340685, 0.49693626],
         …,
         [0.38308823, 0.29571077, 0.3107843 ],
         [0.40551472, 0.3052696 , 0.32769608],
         [0.40563726, 0.31323528, 0.3345588 ]],

        [[0.50980395, 0.44705883, 0.49019608],
         [0.5254902 , 0.45882353, 0.49803922],
         [0.56531864, 0.49865195, 0.5221814 ],
         …,
         [0.3925245 , 0.30232844, 0.33370098],
         [0.41066176, 0.2930147 , 0.32438725],
         [0.40355393, 0.29767156, 0.32512254]]]], dtype=float32),
array([0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1], dtype=int32))
```

1.1.3 5. Split Data

[17]:
```
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)
test_size = int(len(data)*.1)
```

[18]:
```
train_size
```

[18]: 5

[19]:
```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size + val_size).take(test_size)
```

1.2 6. Build Deep Learning Model

[20]:
```
train
```

[20]:
```
<_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32,
name=None))>
```

[21]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten,␣
↪Dropout
```

[22]:
```
model = Sequential()
```

[23]:
```
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,3)))
model.add(MaxPooling2D())
model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

[24]:
```
# Compiling the model
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

[25]:
```
# Models summary
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 16)      448

 max_pooling2d (MaxPooling2  (None, 127, 127, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      4640

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 16)        4624

 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 16)        0
 g2D)

 flatten (Flatten)           (None, 14400)             0

 dense (Dense)               (None, 256)               3686656

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 3696625 (14.10 MB)
Trainable params: 3696625 (14.10 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

1.3 7. Train

[26]:
```
#save the view the logs
logdir = 'logs'
```

[27]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir = logdir)
```

[28]:
```
hist = model.fit(train, epochs = 60, validation_data = val, callbacks =␣
↪[tensorboard_callback])
```

```
Epoch 1/60
5/5 [==============================] - 12s 877ms/step - loss: 1.3285 - accuracy:
0.5375 - val_loss: 1.1755 - val_accuracy: 0.3750
Epoch 2/60
5/5 [==============================] - 6s 820ms/step - loss: 0.8052 - accuracy:
0.5375 - val_loss: 0.6359 - val_accuracy: 0.6562
Epoch 3/60
5/5 [==============================] - 8s 850ms/step - loss: 0.7123 - accuracy:
0.5625 - val_loss: 0.6619 - val_accuracy: 0.5938
Epoch 4/60
```

```
5/5 [==============================] - 8s 1s/step - loss: 0.6366 - accuracy:
0.6187 - val_loss: 0.5146 - val_accuracy: 0.7812
Epoch 5/60
5/5 [==============================] - 7s 840ms/step - loss: 0.6036 - accuracy:
0.5875 - val_loss: 0.5359 - val_accuracy: 0.6875
Epoch 6/60
5/5 [==============================] - 9s 1s/step - loss: 0.5356 - accuracy:
0.7125 - val_loss: 0.5782 - val_accuracy: 0.5938
Epoch 7/60
5/5 [==============================] - 6s 826ms/step - loss: 0.4605 - accuracy:
0.7688 - val_loss: 0.3986 - val_accuracy: 0.7500
Epoch 8/60
5/5 [==============================] - 10s 1s/step - loss: 0.3928 - accuracy:
0.8313 - val_loss: 0.3784 - val_accuracy: 0.8125
Epoch 9/60
5/5 [==============================] - 8s 1s/step - loss: 0.3527 - accuracy:
0.8125 - val_loss: 0.2903 - val_accuracy: 0.9062
Epoch 10/60
5/5 [==============================] - 8s 1s/step - loss: 0.3291 - accuracy:
0.8625 - val_loss: 0.2573 - val_accuracy: 0.8750
Epoch 11/60
5/5 [==============================] - 8s 940ms/step - loss: 0.2518 - accuracy:
0.9375 - val_loss: 0.1856 - val_accuracy: 0.9375
Epoch 12/60
5/5 [==============================] - 9s 1s/step - loss: 0.2275 - accuracy:
0.9187 - val_loss: 0.1235 - val_accuracy: 0.9375
Epoch 13/60
5/5 [==============================] - 8s 1s/step - loss: 0.1761 - accuracy:
0.9125 - val_loss: 0.1292 - val_accuracy: 0.9062
Epoch 14/60
5/5 [==============================] - 6s 828ms/step - loss: 0.1685 - accuracy:
0.9500 - val_loss: 0.1338 - val_accuracy: 0.9375
Epoch 15/60
5/5 [==============================] - 7s 1s/step - loss: 0.1241 - accuracy:
0.9375 - val_loss: 0.1961 - val_accuracy: 0.8750
Epoch 16/60
5/5 [==============================] - 8s 1s/step - loss: 0.1392 - accuracy:
0.9312 - val_loss: 0.0911 - val_accuracy: 0.9688
Epoch 17/60
5/5 [==============================] - 7s 849ms/step - loss: 0.1219 - accuracy:
0.9438 - val_loss: 0.0902 - val_accuracy: 0.9688
Epoch 18/60
5/5 [==============================] - 8s 1s/step - loss: 0.0904 - accuracy:
0.9750 - val_loss: 0.0655 - val_accuracy: 1.0000
Epoch 19/60
5/5 [==============================] - 7s 842ms/step - loss: 0.0767 - accuracy:
0.9688 - val_loss: 0.0686 - val_accuracy: 0.9688
Epoch 20/60
```

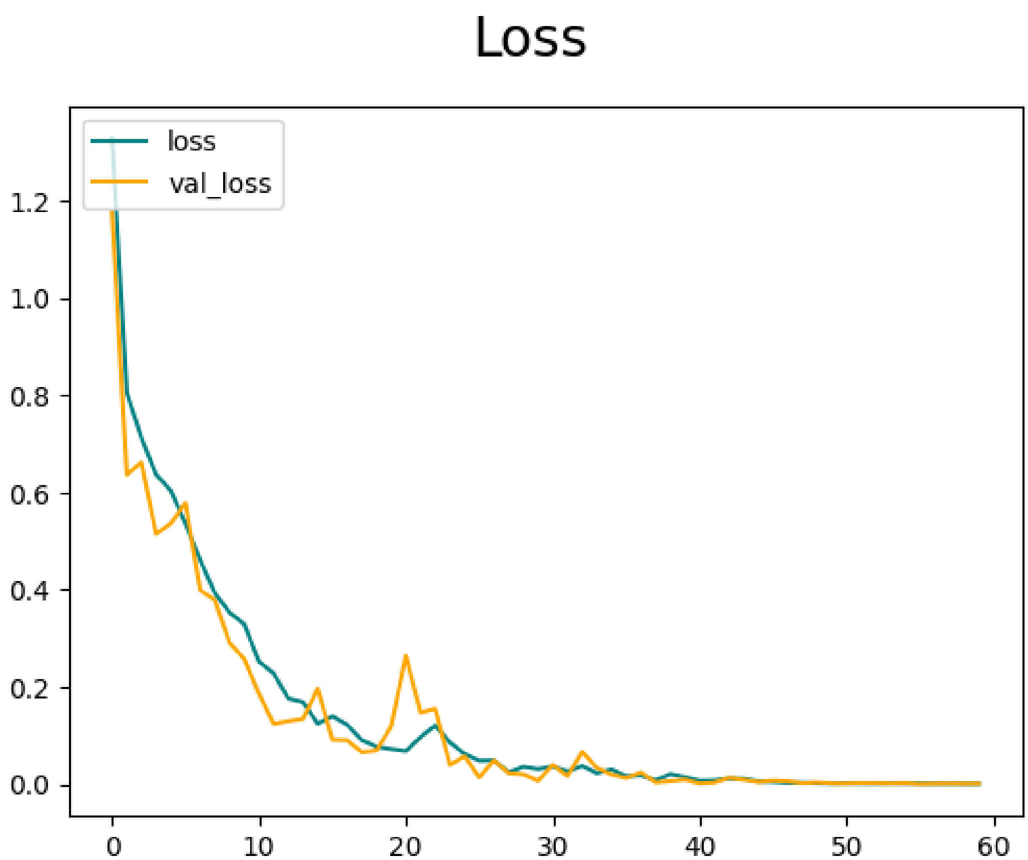
```
5/5 [==============================] - 7s 1s/step - loss: 0.0718 - accuracy:
0.9750 - val_loss: 0.1184 - val_accuracy: 0.9062
Epoch 21/60
5/5 [==============================] - 6s 857ms/step - loss: 0.0676 - accuracy:
0.9688 - val_loss: 0.2638 - val_accuracy: 0.8125
Epoch 22/60
5/5 [==============================] - 8s 953ms/step - loss: 0.0964 - accuracy:
0.9750 - val_loss: 0.1467 - val_accuracy: 1.0000
Epoch 23/60
5/5 [==============================] - 9s 1s/step - loss: 0.1204 - accuracy:
0.9750 - val_loss: 0.1550 - val_accuracy: 0.9062
Epoch 24/60
5/5 [==============================] - 6s 826ms/step - loss: 0.0857 - accuracy:
0.9812 - val_loss: 0.0392 - val_accuracy: 1.0000
Epoch 25/60
5/5 [==============================] - 7s 1s/step - loss: 0.0617 - accuracy:
0.9750 - val_loss: 0.0569 - val_accuracy: 1.0000
Epoch 26/60
5/5 [==============================] - 6s 837ms/step - loss: 0.0482 - accuracy:
0.9937 - val_loss: 0.0135 - val_accuracy: 1.0000
Epoch 27/60
5/5 [==============================] - 7s 998ms/step - loss: 0.0488 - accuracy:
0.9937 - val_loss: 0.0490 - val_accuracy: 1.0000
Epoch 28/60
5/5 [==============================] - 8s 1s/step - loss: 0.0237 - accuracy:
1.0000 - val_loss: 0.0221 - val_accuracy: 1.0000
Epoch 29/60
5/5 [==============================] - 6s 821ms/step - loss: 0.0355 - accuracy:
0.9875 - val_loss: 0.0199 - val_accuracy: 1.0000
Epoch 30/60
5/5 [==============================] - 7s 1s/step - loss: 0.0308 - accuracy:
0.9937 - val_loss: 0.0065 - val_accuracy: 1.0000
Epoch 31/60
5/5 [==============================] - 6s 841ms/step - loss: 0.0362 - accuracy:
0.9937 - val_loss: 0.0394 - val_accuracy: 1.0000
Epoch 32/60
5/5 [==============================] - 7s 999ms/step - loss: 0.0258 - accuracy:
0.9937 - val_loss: 0.0161 - val_accuracy: 1.0000
Epoch 33/60
5/5 [==============================] - 6s 822ms/step - loss: 0.0373 - accuracy:
0.9875 - val_loss: 0.0664 - val_accuracy: 0.9688
Epoch 34/60
5/5 [==============================] - 8s 901ms/step - loss: 0.0224 - accuracy:
1.0000 - val_loss: 0.0334 - val_accuracy: 1.0000
Epoch 35/60
5/5 [==============================] - 6s 826ms/step - loss: 0.0299 - accuracy:
0.9875 - val_loss: 0.0194 - val_accuracy: 1.0000
Epoch 36/60
```

```
5/5 [==============================] - 7s 822ms/step - loss: 0.0161 - accuracy:
0.9937 - val_loss: 0.0125 - val_accuracy: 1.0000
Epoch 37/60
5/5 [==============================] - 7s 864ms/step - loss: 0.0177 - accuracy:
0.9937 - val_loss: 0.0236 - val_accuracy: 1.0000
Epoch 38/60
5/5 [==============================] - 7s 847ms/step - loss: 0.0078 - accuracy:
1.0000 - val_loss: 0.0035 - val_accuracy: 1.0000
Epoch 39/60
5/5 [==============================] - 9s 1s/step - loss: 0.0202 - accuracy:
0.9937 - val_loss: 0.0060 - val_accuracy: 1.0000
Epoch 40/60
5/5 [==============================] - 6s 844ms/step - loss: 0.0143 - accuracy:
1.0000 - val_loss: 0.0091 - val_accuracy: 1.0000
Epoch 41/60
5/5 [==============================] - 9s 1s/step - loss: 0.0067 - accuracy:
1.0000 - val_loss: 0.0013 - val_accuracy: 1.0000
Epoch 42/60
5/5 [==============================] - 6s 826ms/step - loss: 0.0074 - accuracy:
1.0000 - val_loss: 0.0032 - val_accuracy: 1.0000
Epoch 43/60
5/5 [==============================] - 9s 1s/step - loss: 0.0114 - accuracy:
1.0000 - val_loss: 0.0129 - val_accuracy: 1.0000
Epoch 44/60
5/5 [==============================] - 7s 1s/step - loss: 0.0108 - accuracy:
1.0000 - val_loss: 0.0088 - val_accuracy: 1.0000
Epoch 45/60
5/5 [==============================] - 6s 830ms/step - loss: 0.0054 - accuracy:
1.0000 - val_loss: 0.0044 - val_accuracy: 1.0000
Epoch 46/60
5/5 [==============================] - 7s 1s/step - loss: 0.0047 - accuracy:
1.0000 - val_loss: 0.0070 - val_accuracy: 1.0000
Epoch 47/60
5/5 [==============================] - 6s 817ms/step - loss: 0.0026 - accuracy:
1.0000 - val_loss: 0.0057 - val_accuracy: 1.0000
Epoch 48/60
5/5 [==============================] - 9s 1s/step - loss: 0.0028 - accuracy:
1.0000 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 49/60
5/5 [==============================] - 8s 1s/step - loss: 0.0018 - accuracy:
1.0000 - val_loss: 0.0033 - val_accuracy: 1.0000
Epoch 50/60
5/5 [==============================] - 7s 834ms/step - loss: 0.0013 - accuracy:
1.0000 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 51/60
5/5 [==============================] - 8s 1s/step - loss: 0.0017 - accuracy:
1.0000 - val_loss: 0.0013 - val_accuracy: 1.0000
Epoch 52/60
```

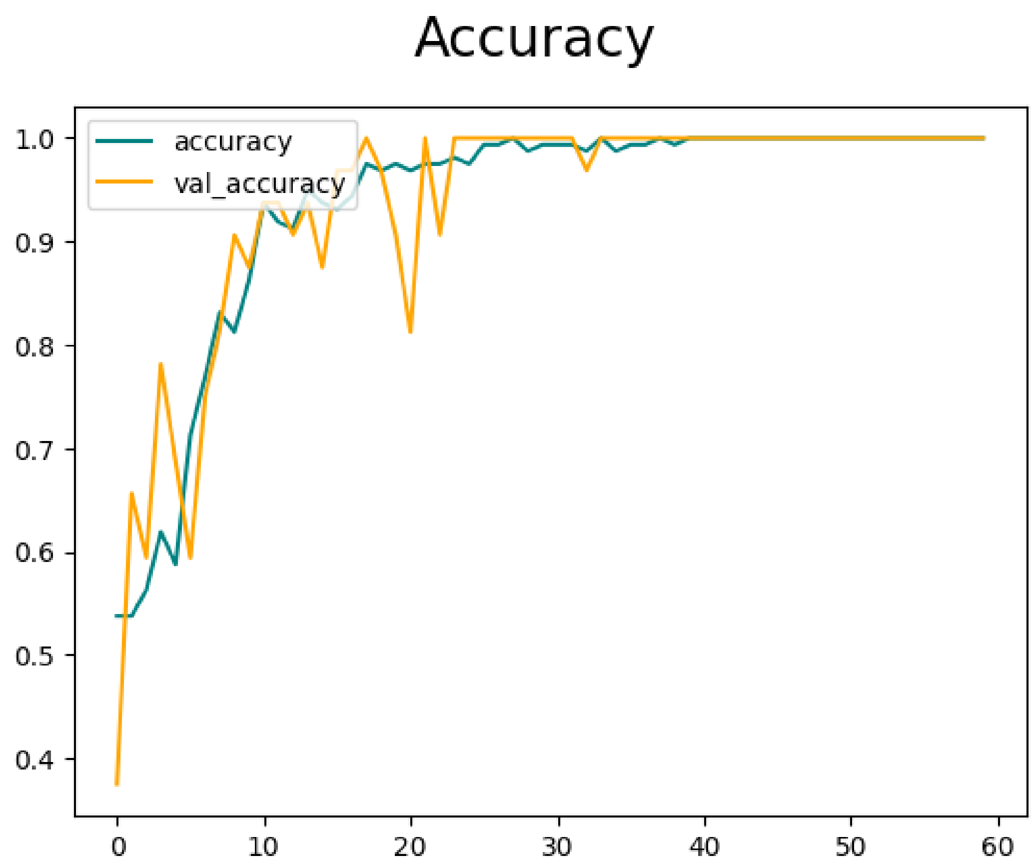
```
5/5 [==============================] - 6s 834ms/step - loss: 0.0013 - accuracy:
1.0000 - val_loss: 0.0020 - val_accuracy: 1.0000
Epoch 53/60
5/5 [==============================] - 7s 830ms/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 54/60
5/5 [==============================] - 7s 1s/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 55/60
5/5 [==============================] - 6s 830ms/step - loss: 0.0013 - accuracy:
1.0000 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 56/60
5/5 [==============================] - 8s 1s/step - loss: 0.0012 - accuracy:
1.0000 - val_loss: 2.4022e-04 - val_accuracy: 1.0000
Epoch 57/60
5/5 [==============================] - 7s 835ms/step - loss: 9.6568e-04 -
accuracy: 1.0000 - val_loss: 9.5465e-04 - val_accuracy: 1.0000
Epoch 58/60
5/5 [==============================] - 7s 841ms/step - loss: 9.1853e-04 -
accuracy: 1.0000 - val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 59/60
5/5 [==============================] - 8s 1s/step - loss: 7.4237e-04 - accuracy:
1.0000 - val_loss: 9.1980e-04 - val_accuracy: 1.0000
Epoch 60/60
5/5 [==============================] - 7s 837ms/step - loss: 5.8313e-04 -
accuracy: 1.0000 - val_loss: 9.6101e-04 - val_accuracy: 1.0000
```

1.4 Plot Performance

[29]:
```
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



[30]:
```
fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



1.5 Evaluate

[31]:
```
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
```

[32]:
```
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
```

[33]:
```
for batch in test.as_numpy_iterator():
    X, y = batch
    yhat = model.predict(X)
    pre.update_state(y, yhat)
    re.update_state(y, yhat)
    acc.update_state(y, yhat)
```

[34]:
```
print(pre.result(), re.result(), acc.result())
```

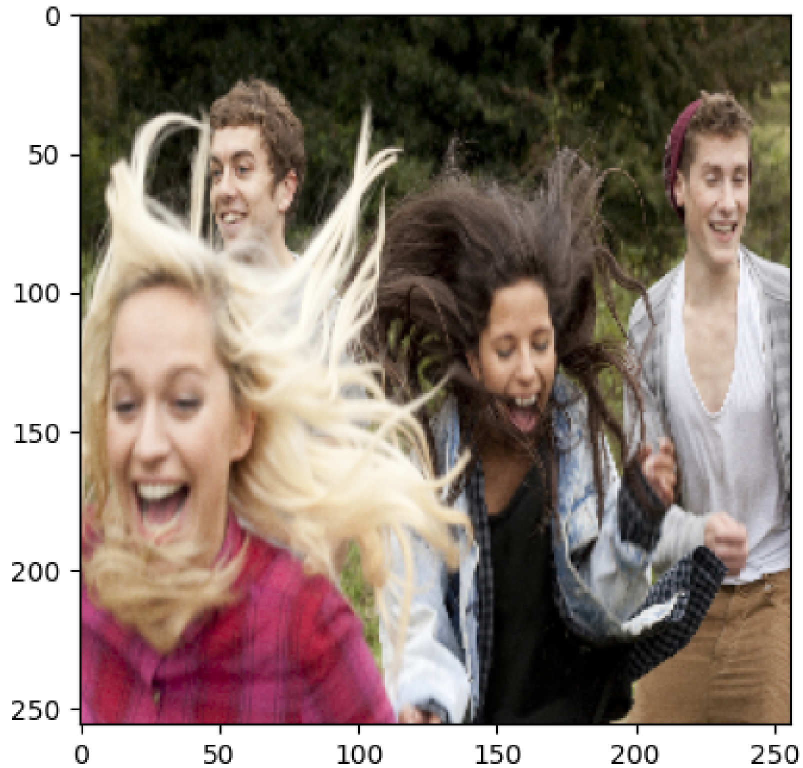
```
tf.Tensor(0.0, shape=(), dtype=float32) tf.Tensor(0.0, shape=(), dtype=float32)
tf.Tensor(0.0, shape=(), dtype=float32)
```

1.6 10. Test

[35]:
```
import cv2
```

[36]:
```
img = cv2.imread('drive/MyDrive/Project 13 Classification/154006829.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
if img is None:
    print("Image not loaded")
else:
    plt.imshow(img)
    plt.show()
```

[37]:
```
resize =tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```



[38]:
```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 [==============================] - 0s 323ms/step
```

[39]:
```
yhat
```

[39]:
```
array([[7.7254776e-07]], dtype=float32)
```

[40]:
```
if yhat > 0.5:
  print(f'Predicted class is Sad')
else:
  print (f'Predicted Class is happy')
```

```
Predicted Class is happy
```

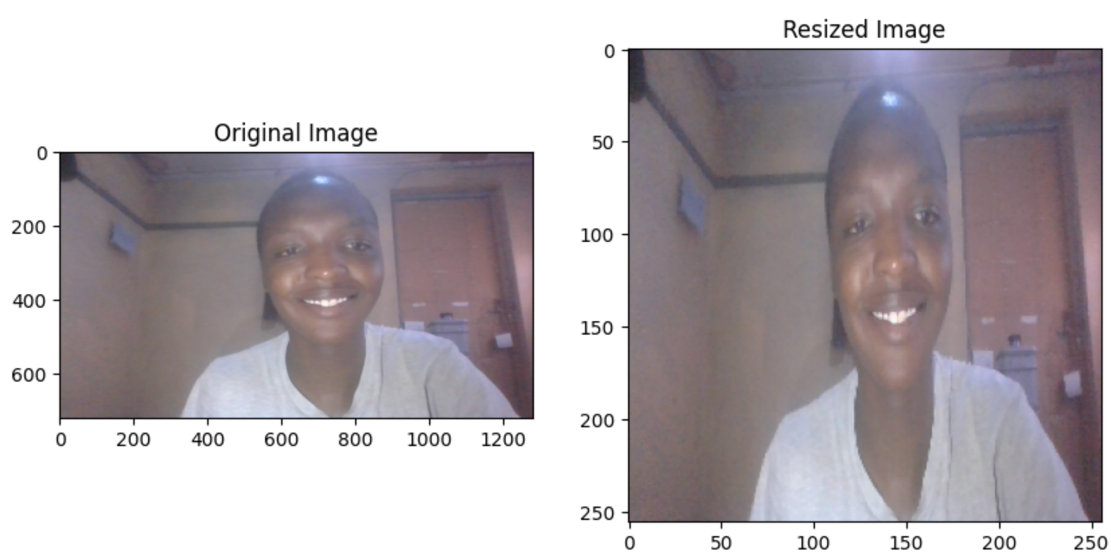
Function for predicting images if happy or sad

[41]:
```
def predict_emotion(image_path, model):
    # Load and convert the image
    img = cv2.imread(image_path)
    if img is None:
        print("Image not loaded")
        return
```

```
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Resize the image
    resize = tf.image.resize(img, (256,256))

    # Predict the class
    yhat = model.predict(np.expand_dims(resize/255, 0))
    print(yhat)
    if yhat > 0.5:
        print('Predicted class is Sad')
    else:
        print('Predicted class is Happy')

    # Show the original and resized images
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.title('Original Image')
    plt.subplot(1, 2, 2)
    plt.imshow(resize.numpy().astype(int))
    plt.title('Resized Image')
    plt.show()
```

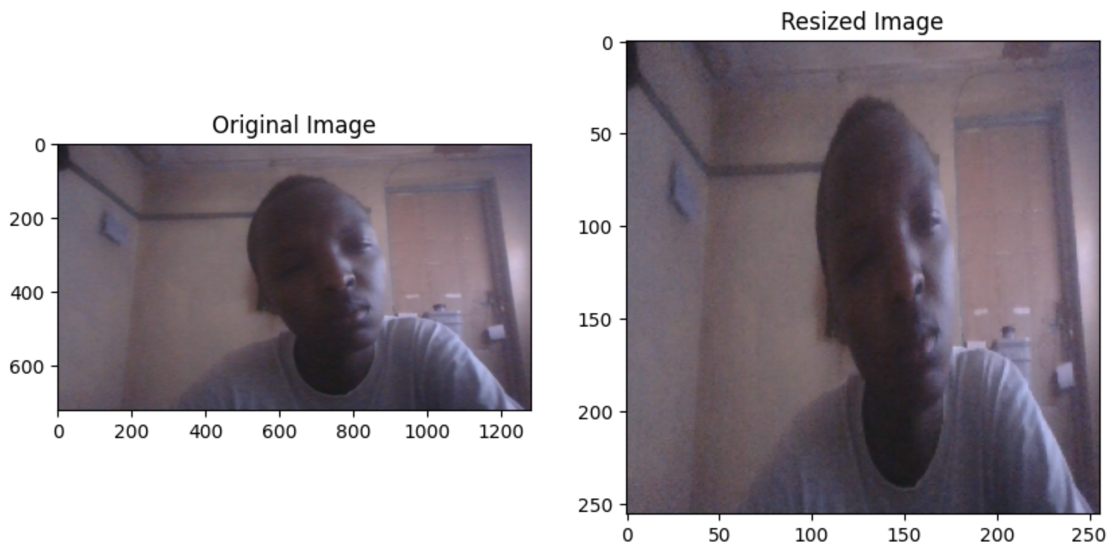
[42]:
```
image_path = 'drive/MyDrive/Project 13 Classification/WIN_20240216_13_46_13_Pro.
 ↪jpg'
predict_emotion(image_path, model)
```

```
1/1 [==============================] - 0s 36ms/step
[[0.00483828]]
Predicted class is Happy
```



```
[43]: image_path = 'drive/MyDrive/Project 13 Classification/WIN_20240216_13_14_07_Pro.
        ↪jpg'
      predict_emotion(image_path, model)
```

```
1/1 [==============================] - 0s 18ms/step
[[0.71629643]]
Predicted class is Sad
```



1.7 Save The Model

```
[44]: from tensorflow.keras.models import load_model
```

```
[45]: model.save(os.path.join('models', 'drive/MyDrive/Project 13 Classification/
        ↪model/imageclassifier.h5'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```