Happy_or_Sad_Image_Classifier

December 23, 2023

1 Happy or Sad Face Image Classifier Model

In this upcoming project, our goal is to create a deep learning image classifier model to determine whether a given face image depicts a happy or sad expression.

To accomplish this, we will leverage `TensorFlow` along with additional libraries such as `cv2`, `os`, `matplotlib`, and more.

1.1 Import Libraries

[1]:
```
import tensorflow as tf
import os
import cv2
import imghdr
import numpy as np
from matplotlib import pyplot as plt
```

[2]:
```
def PRINT(text) -> None: print(f"{80*'-'}\n{text}\n{80*'-'}")
```

[3]:
```
!nvidia-smi
```

```
Sat Dec 23 07:48:02 2023
+-------------------------------------------------------------------------------
--------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version:
12.2     |
|-----------------------------------------+----------------------+--------------
--------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util
Compute M. |
|                                         |                      |
MIG M. |
|=========================================+======================+==============
========|
|   0  Tesla V100-SXM2-16GB           Off | 00000000:00:04.0 Off |
0 |
| N/A   35C    P0              25W / 300W |      0MiB / 16384MiB |      0%
```

```
Default |
|                                          |                      |
N/A |
+-----------------------------------------+----------------------+--------------
--------+

+-------------------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                            GPU
Memory |
|        ID   ID
Usage      |
|===============================================================================
========|
|  No running processes found
|
+-------------------------------------------------------------------------------
--------+
```

[4]:
```python
# 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)
```

[5]:
```python
tf.config.list_physical_devices('GPU')
```

[5]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

1.2 Clone GitHub Repository

To access all of our data, including images and their labels, we will clone our GitHub repository.

We can execute this command after finishing the preprocessing of our images in the *Jupyter Lab* environment. Once we have completed building our deep learning model, we will transition to the *Google Colab* environment to train our model on the `GPU` instead of the computer's `CPU`.

[7]:
```
!git clone https://github.com/Gavision97/Computer-Vision.git
```

```
Cloning into 'Computer-Vision'…
remote: Enumerating objects: 21573, done.
remote: Counting objects: 100% (1085/1085), done.
remote: Compressing objects: 100% (1053/1053), done.
remote: Total 21573 (delta 28), reused 1083 (delta 26), pack-reused 20488
Receiving objects: 100% (21573/21573), 883.04 MiB | 33.39 MiB/s, done.
Resolving deltas: 100% (48/48), done.
Updating files: 100% (21160/21160), done.
Downloading Object Detection Projects/Facial Verification
```

```
Project/_siamese_model_.h5 (468 MB)
Error downloading object: Object Detection Projects/Facial Verification
Project/_siamese_model_.h5 (a59b72d): Smudge error: Error downloading Object
Detection Projects/Facial Verification Project/_siamese_model_.h5
(a59b72df08135b6d5cdbfe1516e585e2f9b2e1b6de56acf11c31f5fb02d8cf67): batch
response: This repository is over its data quota. Account responsible for LFS
bandwidth should purchase more data packs to restore access.

Errors logged to /content/Computer-
Vision/.git/lfs/logs/20231223T075102.236865717.log
Use `git lfs logs last` to view the log.
error: external filter 'git-lfs filter-process' failed
fatal: Object Detection Projects/Facial Verification Project/_siamese_model_.h5:
smudge filter lfs failed
warning: Clone succeeded, but checkout failed.
You can inspect what was checked out with 'git status'
and retry with 'git restore --source=HEAD :/'
```

[8]:
```
cd "/content/Computer-Vision/Image Classification Projects/Happy or Sad Face␣
↪Image Classifier"
```

```
/content/Computer-Vision/Image Classification Projects/Happy or Sad Face Image
Classifier
```

1.3 Get Data

The first step is to get data. Our data is going to be taked from websites like *Google*. One simple way is just to search for happy/sad people pictures.

Next we can use *Download All Images* extension which can be downloaded easily from the extenstion. That extension makes the process of downloading images from google much easier.

1.4 Remove Redundant File

The next step is to remove redundant file, i.e., file which not one of the images format files (.jpg , .png …).

We saw that there are lots of *.svg* files. In order to remove those files we will run the command : `rm *.svg` from the command line which was opened from the wanted directory (*happy* or *sad* directory)

1.5 Remove Small Images

The next step is to remove all of the images that are of the size lower that 9kb. The way we are going to achive that is by running the commad:

`Get-ChildItem | Where-Object { $_.Length -lt 9216 } | Remove-Item`

from the command line which was opened again from the wanted directory.

```
[ ]: data_dir = 'data'

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

```
[ ]: 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 in ext list {}'.format(image_path))
                     os.remove(image_path)
             except Exception as e:
                 print(e)
                 print('Issue with image {}'.format(image_path))
                 # os.remove(image_path)
```

```
Image not in ext list data\happy\happy-home.jpg
Image not in ext list data\sad\182c9f72579b4bc6a5f2da710cba7918.webp
Image not in ext list data\sad\713b3140ec884011bac5813ea28d0f24.webp
Image not in ext list data\sad\close-sad-female-human-face-
footage-147131792_prevstill.jpeg
Image not in ext list data\sad\depositphotos_218926250-stock-photo-human-face-
can-different-strong.jpg
Image not in ext list data\sad\depositphotos_2444604-stock-photo-very-sad-
little-boy.jpg
Image not in ext list data\sad\fddbda04f5304f4ba58e2cde8311dd54.webp
```

1.6 Load Data

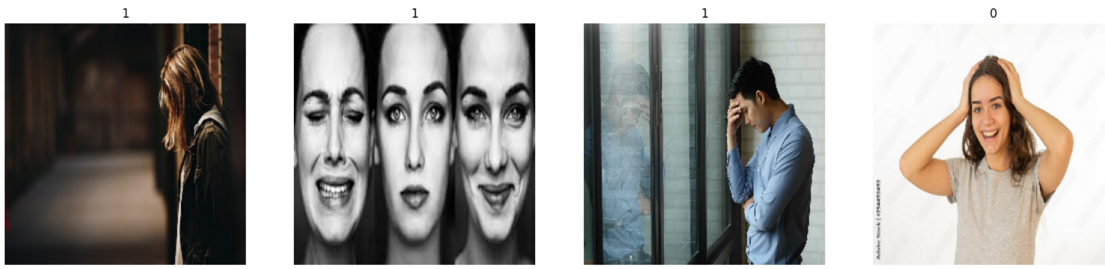
```
[9]: data = tf.keras.utils.image_dataset_from_directory('data')
```

```
Found 452 files belonging to 2 classes.
```

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

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

```
[17]: 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])
          ax[idx].axis(False)
```




```
[ ]: batch[0].shape
```

```
[ ]: (32, 256, 256, 3)
```

```
[ ]: for indx,val in enumerate(batch):
         if indx == 2 :
             break
         PRINT(f'Image number {indx+1} from the first bach in tensor representation:␣
      ↪{batch[0][indx]}\n\n And the corresponding class: {batch[1][indx]}')
```

```
---------------------------------------------------------------------------------
Image number 1 from the first bach in tensor representation: [[[227.       172.
46.       ]
  [228.14062  174.14062   48.140625]
  [230.9336   179.9336    51.933594]
  …
  [253.       253.       253.      ]
  [253.       253.       253.      ]
  [253.       253.       253.      ]]

 [[227.69922  170.21484   45.04297 ]
  [227.44922  172.6211    46.621094]
  [228.04297  174.21484   47.871094]
  …
  [253.       253.       253.      ]
  [253.       253.       253.      ]
  [253.       253.       253.      ]]

 [[228.01172  170.01172   45.01172 ]
  [227.75     172.75      46.75     ]
  [227.95312  172.95312   46.953125]
  …
  [254.       254.       254.      ]
  [253.04688  253.04688  253.04688 ]
  [254.       254.       254.      ]]

 …
```

```
 [[250.04688  171.04688   26.046875]
  [250.       172.         22.      ]
  [247.       172.         18.      ]
  …
  [246.       175.         59.5     ]
  [248.       173.         54.      ]
  [250.       173.         55.      ]]

 [[249.20703  170.20703   26.207031]
  [249.       171.         23.      ]
  [247.82812  169.82812   17.828125]
  …
  [247.       174.         61.      ]
  [248.       173.         54.      ]
  [250.       173.         55.      ]]

 [[248.75     169.75      25.75    ]
  [248.       170.         22.      ]
  [246.39062  168.39062   16.390625]
  …
  [248.       175.         62.      ]
  [249.       174.         55.      ]
  [250.       173.         55.      ]]]

 And the corresponding class: 0
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Image number 2 from the first bach in tensor representation: [[[ 94.43945
131.81445   156.68945  ]
  [ 95.84961   131.93164   155.89062  ]
  [ 93.625     130.41211   154.85938  ]
  …
  [ 88.17383   133.78711   159.98047  ]
  [ 88.40234   133.02734   163.14062  ]
  [ 89.11133   134.11133   163.11133  ]]

 [[ 96.91211   132.91211   157.16211  ]
  [ 93.10742   131.10742   154.10742  ]
  [ 93.58008   130.54883   154.06445  ]
  …
  [ 90.265625  134.93945   163.56445  ]
  [ 88.52344   135.27344   163.64844  ]
  [ 88.        133.03711   161.92578  ]]

 [[ 94.865234  132.86523   155.3457   ]
  [ 94.42578   131.41211   157.41211  ]
  [ 93.57617   131.57617   154.32617  ]
```

```
  …
  [ 89.484375  133.54492   162.54492  ]
  [ 87.234375  132.23438   162.79297  ]
  [ 89.28125   134.45703   163.36914  ]]

 …

 [[ 26.435547   72.859375   18.892578 ]
  [ 25.822266   72.708984   14.9296875]
  [ 15.388672   48.970703   11.175781 ]
  …
  [  6.6777344  38.597656   10.308594 ]
  [ 11.884766   29.640625    9.095703 ]
  [ 12.255859   33.785156    9.748047 ]]

 [[ 11.453125   39.0625      9.892578 ]
  [ 26.179688   61.242188   10.931641 ]
  [ 43.373047   87.65039    33.722656 ]
  …
  [  8.3046875  50.25586    12.244141 ]
  [ 25.623047   43.796875   16.314453 ]
  [ 23.648438   58.210938   16.847656 ]]

 [[ 39.583984   96.54883    26.03125  ]
  [ 22.199219   74.35742    15.814453 ]
  [ 43.009766   89.70703    41.060547 ]
  …
  [ 20.044922   52.853516   18.886719 ]
  [ 21.56836    44.695312   20.080078 ]
  [ 16.265625   39.67578    12.699219 ]]]

 And the corresponding class: 0
--------------------------------------------------------------------------------
```

1.7 Preprocess Data

In order to improve our model efficiency, we will take few data preprocessing steps such as:

- Scale out Date: Scale the images from [0,255] -> [0.1]

After we finish preprocess the data, we can visualize four images and their corresponding from single batch.

Our labels are:

- 0 -> Happy
- 1 -> Sad

[18]:
```
data = data.map(lambda x,y: (x/255,y)) # x for images and y for labels, i.e, 0␣
↪haapy 1 sad
```

[21]:
```
scaled_itr = data.as_numpy_iterator()

scaled_batch = scaled_itr.next()
```

[22]:
```
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(scaled_batch[0][:4]):
    ax[idx].imshow(img)
    ax[idx].title.set_text(scaled_batch[1][idx])
    ax[idx].axis(False)
```

[23]:
```
scaled_batch[0].max(), scaled_batch[0].min()
```

[23]:
```
(1.0, 0.0)
```

1.7.1 Split Data

Following that, we can partition our data into training, validation, and test sets.

[25]:
```
# The size of train dataset
train_size = int(len(data)*.7)

# The size of validation dataset
val_size = int(len(data)*.2)

# The size of test dataset
test_size = int(len(data)*.1)

train_size, val_size, test_size
```

[25]:
```
(10, 3, 1)
```

[34]:
```
# Define train dataset
train = data.take(train_size)

# Define validation dataset
val = data.skip(train_size).take(val_size)
```

```
# Define test dataset
test= data.skip(train_size + val_size).take(test_size)
```

1.8 Building our Deep Learning Model

1.8.1 Import Libraries

[35]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten,␣
 ↪Dropout, BatchNormalization
```

1.8.2 Building Model Class

The subsequent step involves constructing our image classifier model.

[71]:
```
class ImageClassifierModel:
    def __init__(self, input_shape=(256, 256, 3)):
        self.__model = self.build_model(input_shape)

    def build_model(self, input_shape):
        model = Sequential()

        model.add(Conv2D(32, (3, 3), 1, activation='relu', input_shape=(256,␣
 ↪256, 3)))
        model.add(MaxPooling2D())

        model.add(Conv2D(64, (3, 3), 1, activation='relu'))
        model.add(MaxPooling2D())

        model.add(Conv2D(128, (3, 3), 1, activation='relu'))
        model.add(MaxPooling2D())

        model.add(Flatten())

        model.add(Dense(256, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))

        return model

    def compile_model(self):
        self.__model.compile(optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy'])

    def get_model(self):
      return self.__model

    def summerize_model(self):
```

```
        self.__model.summary()
```

[72]:
```
classifier = ImageClassifierModel()
classifier.compile_model()
```

[73]:
```
classifier.summerize_model()
```

```
Model: "sequential_9"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 254, 254, 32)      896

 max_pooling2d_27 (MaxPooli  (None, 127, 127, 32)      0
 ng2D)

 conv2d_29 (Conv2D)          (None, 125, 125, 64)      18496

 max_pooling2d_28 (MaxPooli  (None, 62, 62, 64)        0
 ng2D)

 conv2d_30 (Conv2D)          (None, 60, 60, 128)       73856

 max_pooling2d_29 (MaxPooli  (None, 30, 30, 128)       0
 ng2D)

 flatten_9 (Flatten)         (None, 115200)            0

 dense_18 (Dense)            (None, 256)               29491456

 dense_19 (Dense)            (None, 1)                 257

=================================================================
Total params: 29584961 (112.86 MB)
Trainable params: 29584961 (112.86 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[75]:
```
model = classifier.get_model()

model
```

[75]: <keras.src.engine.sequential.Sequential at 0x7cec985d6890>

[76]:
```
logdir='logs'
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

1.8.3 Train our Model

Once we have created our model class and generated instance of it, as well as compiled the model, we are now ready to proceed with training the model.

[77]:
```
hist = model.fit(train, epochs=20, validation_data=val,
    callbacks=[tensorboard_callback])
```

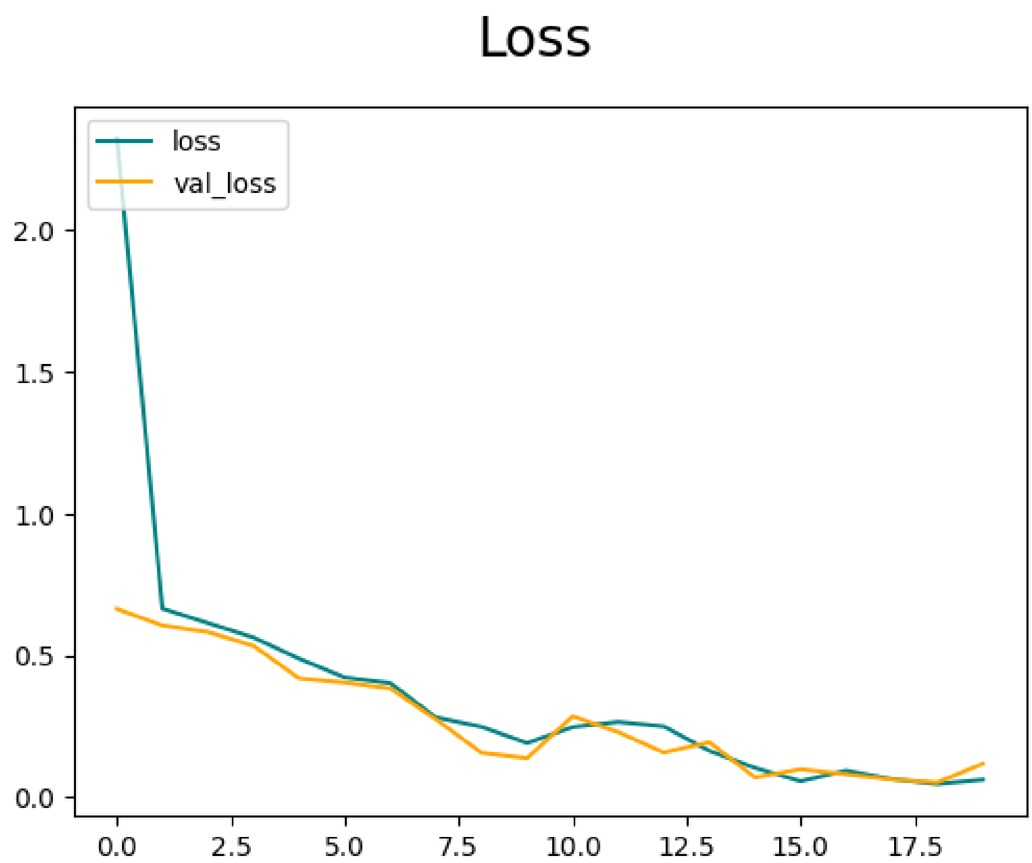
```
Epoch 1/20
10/10 [==============================] - 11s 762ms/step - loss: 2.3213 -
accuracy: 0.5250 - val_loss: 0.6633 - val_accuracy: 0.5938
Epoch 2/20
10/10 [==============================] - 8s 688ms/step - loss: 0.6642 -
accuracy: 0.5875 - val_loss: 0.6053 - val_accuracy: 0.6354
Epoch 3/20
10/10 [==============================] - 9s 670ms/step - loss: 0.6137 -
accuracy: 0.6844 - val_loss: 0.5822 - val_accuracy: 0.6979
Epoch 4/20
10/10 [==============================] - 9s 822ms/step - loss: 0.5627 -
accuracy: 0.7219 - val_loss: 0.5331 - val_accuracy: 0.6979
Epoch 5/20
10/10 [==============================] - 8s 666ms/step - loss: 0.4880 -
accuracy: 0.7781 - val_loss: 0.4188 - val_accuracy: 0.8438
Epoch 6/20
10/10 [==============================] - 11s 812ms/step - loss: 0.4222 -
accuracy: 0.8156 - val_loss: 0.4044 - val_accuracy: 0.7604
Epoch 7/20
10/10 [==============================] - 9s 800ms/step - loss: 0.4021 -
accuracy: 0.8469 - val_loss: 0.3837 - val_accuracy: 0.8333
Epoch 8/20
10/10 [==============================] - 11s 1s/step - loss: 0.2813 - accuracy:
0.8844 - val_loss: 0.2744 - val_accuracy: 0.9062
Epoch 9/20
10/10 [==============================] - 8s 661ms/step - loss: 0.2483 -
accuracy: 0.8969 - val_loss: 0.1568 - val_accuracy: 0.9479
Epoch 10/20
10/10 [==============================] - 9s 779ms/step - loss: 0.1912 -
accuracy: 0.9406 - val_loss: 0.1378 - val_accuracy: 0.9688
Epoch 11/20
10/10 [==============================] - 9s 768ms/step - loss: 0.2473 -
accuracy: 0.9094 - val_loss: 0.2854 - val_accuracy: 0.9271
Epoch 12/20
10/10 [==============================] - 8s 657ms/step - loss: 0.2654 -
accuracy: 0.8844 - val_loss: 0.2295 - val_accuracy: 0.9271
Epoch 13/20
10/10 [==============================] - 9s 800ms/step - loss: 0.2496 -
accuracy: 0.8813 - val_loss: 0.1578 - val_accuracy: 0.9271
Epoch 14/20
10/10 [==============================] - 9s 794ms/step - loss: 0.1635 -
```

```
accuracy: 0.9406 - val_loss: 0.1939 - val_accuracy: 0.9688
Epoch 15/20
10/10 [==============================] - 8s 666ms/step - loss: 0.1038 -
accuracy: 0.9563 - val_loss: 0.0700 - val_accuracy: 0.9688
Epoch 16/20
10/10 [==============================] - 9s 797ms/step - loss: 0.0569 -
accuracy: 0.9906 - val_loss: 0.0990 - val_accuracy: 0.9583
Epoch 17/20
10/10 [==============================] - 9s 794ms/step - loss: 0.0938 -
accuracy: 0.9656 - val_loss: 0.0808 - val_accuracy: 0.9688
Epoch 18/20
10/10 [==============================] - 8s 646ms/step - loss: 0.0644 -
accuracy: 0.9750 - val_loss: 0.0632 - val_accuracy: 0.9896
Epoch 19/20
10/10 [==============================] - 9s 778ms/step - loss: 0.0473 -
accuracy: 0.9844 - val_loss: 0.0525 - val_accuracy: 0.9792
Epoch 20/20
10/10 [==============================] - 9s 793ms/step - loss: 0.0620 -
accuracy: 0.9750 - val_loss: 0.1185 - val_accuracy: 0.9792
```

1.9 Visualize Training Preformance

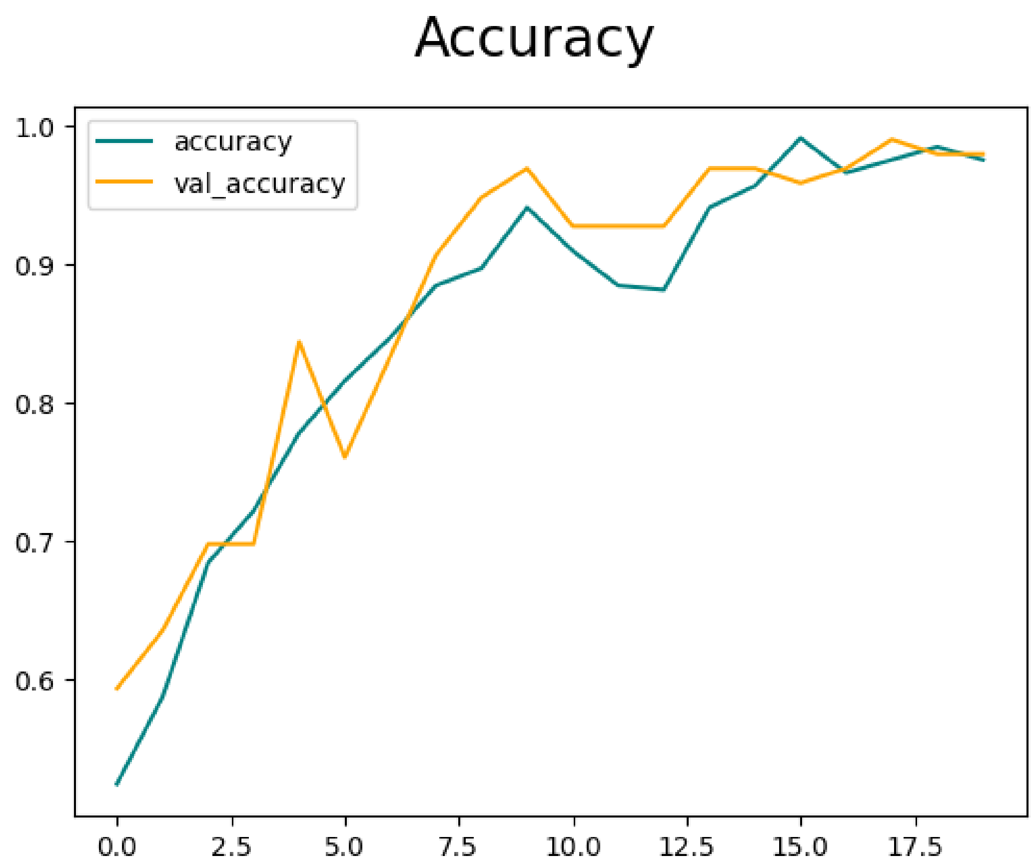
1.9.1 Visualize Training Loss

[78]:
```
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()
```



1.9.2 Visualize Training Accuracy

[79]:
```
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()
```



1.10 Evaluate Model

[81]:
```
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
```

[82]:
```
precision = Precision()
recall = Recall()
accuracy = BinaryAccuracy()
```

[83]:
```
metrics_list = [precision, recall, accuracy]
```

[85]:
```
for batch in test.as_numpy_iterator():
  X, y = batch
  yhat = model.predict(X)

  for metric in metrics_list:
    metric.update_state(y, yhat)
```

```
1/1 [==============================] - 0s 41ms/step
```

1.10.1 Visualize Metrics Results

[87]:
```
PRINT(f'Precision result -> {precision.result()}\nRecall result -> {recall.
    ↪result()}\nAccuracy result -> {accuracy.result()}')
```

```
--------------------------------------------------------------------------------
Precision result -> 1.0
Recall result -> 0.9333333373069763
Accuracy result -> 0.96875
--------------------------------------------------------------------------------
```

1.11 Test our Model

Once our model has been trained, we aim to evaluate its performance on new, unseen data.

To achieve this, we will undertake several steps, including downloading random facial expression images (both happy and sad), preprocessing these images, predicting their labels, and plotting the images alongside their predicted labels.

This process allows us to visualize and assess the performance of our trained model.

[88]:
```
import cv2
```

1.11.1 Upload Random Happy and Sad Face Images

The next step is to upload images depicting happy and sad facial expressions of humans.

Afterward, we will preprocess these images to facilitate real-time predictions.

[93]:
```
from google.colab import files

uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving sad_woman_1.jpg to sad_woman_1.jpg
```

1.11.2 Preprocess Images

After selecting four images (two happy and two sad), we will proceed to preprocess them.

As part of the preprocessing step, we will:

- Resize the images to 256x256 pixels.
- Convert the color channels from RGB (red, green, and blue) to BGR (blue, green, and red) so that we can visualize the images and their corresponding predicted labels using the `matplotlib` library later on.

[127]:
```
happy_woman_1_img = cv2.imread('happy_woman_1.jpg')
happy_woman_2_img = cv2.imread('happy_woman_2jpg.jpg')
sad_woman_1_img = cv2.imread('sad_woman_1.jpg')
sad_man_1_img = cv2.imread('sad_man_1.jpg')
```

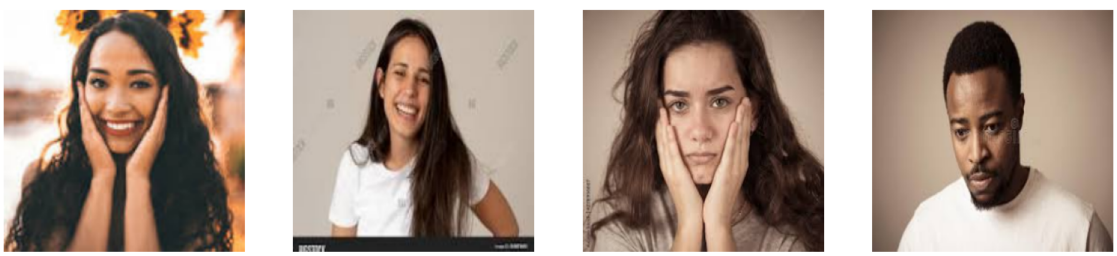
[128]:
```python
test_images_list = [happy_woman_1_img, happy_woman_2_img, sad_woman_1_img, sad_man_1_img]
```

Preprocess Step

[133]:
```python
resized_test_images_list = []

for img_path in test_images_list:

    # Resize the images to 256x256
    resized = tf.image.resize(img_path, (256, 256))

    # Switch the color channels, so we can later visualize the images.
    adjusted_img = resized[..., ::-1]

    # Append the adjusted image to 'resized_test_images_list
    resized_test_images_list.append(adjusted_img)
```

Visualization Step

[134]:
```python
fig, axes = plt.subplots(1, len(resized_test_images_list), figsize=(20, 20))

for idx, img in enumerate(resized_test_images_list):
    axes[idx].imshow(img.numpy().astype(int))  # Convert to int for proper display
    axes[idx].axis('off')

plt.show()
```



As observed in the image visualization, we currently lack labels for each image. This is because we have not yet predicted their labels using our trained model.

However, we can still discern them by ourselves simply by examining the facial expressions in the images above.

1.11.3 Generate Predictions

After completing the preprocessing step, we can utilize our model to predict the labels of images.

Subsequently, we will visualize the images along with their predicted labels to assess how well our model performed on new, unseen, randomly selected facial expressions of happy and sad individuals.

[136]:
```
images_predictions_list = []
```

[137]:
```
for img in resized_test_images_list:
  yhat = model.predict(np.expand_dims(img/255, 0))

  images_predictions_list.append(yhat)
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
```

[138]:
```
def print_prediction_value_and_visualize(img, pred) -> None:
  if pred == None:
    PRINT("Error! Got None value!")
    return

  if pred > 0.5:
    PRINT(f'Predicted class is  -> Sad')
  else:
    PRINT(f'Predicted class is -> Happy')

  plt.imshow(img.numpy().astype(int))
  plt.axis(False)

  plt.show()
```

[150]:
```
print_prediction_value_and_visualize(resized_test_images_list[0],images_predictions_list[0])
```

```
--------------------------------------------------------------------------------
Predicted class is -> Happy
--------------------------------------------------------------------------------
```

[149]:
```
print_prediction_value_and_visualize(resized_test_images_list[1],images_predictions_list[1])
```

```
--------------------------------------------------------------------------------
Predicted class is -> Happy
--------------------------------------------------------------------------------
```

[148]:
```
print_prediction_value_and_visualize(resized_test_images_list[2],images_predictions_list[2])
```

```
--------------------------------------------------------------------------------
Predicted class is  -> Sad
--------------------------------------------------------------------------------
```

[147]:
```
print_prediction_value_and_visualize(resized_test_images_list[3],images_predictions_list[3])
```

```
--------------------------------------------------------------------------------
Predicted class is  -> Sad
--------------------------------------------------------------------------------
```

As observed in our real-time test, we randomly selected four facial expression images that our model had not encountered before, and we accurately predicted each expression. The model correctly identified sad faces as sad and happy faces as happy.

1.12 Save Trained Model

[153]:
```
model.save('happy_sad_image_classifier.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(
```

[ ]:
```

```