Deliverable Data Scientist

Submitted by : SHUBHAM SHARMA
2019ME20933 (IIT Delhi)

READ ME

Project 1 : Smile Classification CODE(Click here)

I have chosen Smile Classification for the deliverable assignment. Firstly I downloaded the dataset and split it into training and test reading CSV files given with the dataset.
Code i have used for this is :

```
from __future__ import print_function
import pandas as pd
import shutil
import os
import sys
train_labels = pd.read_csv("/content/drive/MyDrive/I'mbesideyou/data.zip (Unzipped Files)/I'mbesideyou/train.csv")
test_labels = pd.read_csv("/content/drive/MyDrive/I'mbesideyou/data.zip (Unzipped Files)/I'mbesideyou/test.csv")
# Create happy_images directory
happy_images ="/content/drive/MyDrive/I'mbesideyou/data.zip (Unzipped Files)/I'mbesideyou/happy_images"
train = "/content/drive/MyDrive/I'mbesideyou/data.zip (Unzipped Files)/I'mbesideyou/train/"
test = "/content/drive/MyDrive/I'mbesideyou/data.zip (Unzipped Files)/I'mbesideyou/test/"

if not os.path.exists (train):
  os.mkdir (train)

for filename, class_name in train_labels.values:
  # Create subdirectory with class name
  if not os.path.exists(train + str(class_name)):
    os.mkdir(train + str(class_name))
  src_path = happy_images +'/'+ filename +'.jpg'
  dst_path = train + str(class_name) + '/' + filename +'.jpg'
  try:
    shutil.copy(src_path, dst_path)
    print("sucessful")
  except IOError as e:
    print('Unable to copy file {1 to fl'
          .format(src_path, dst_path))
  except:
    print('when try copy file {1 to {, unexpected error: {}'
        .format (src_path, dst_path, sys.exc_info()))
```

```
if not os.path.exists (test):
  os.mkdir (test)

for filename, class_name in test_labels.values:
  # Create subdirectory with class name
  if not os.path.exists(test + str(class_name)):
    os.mkdir(test + str(class_name))
  src_path = happy_images +'/'+ filename +'.jpg'
  dst_path = test + str(class_name) + '/' + filename +'.jpg'
  try:
    shutil.copy(src_path, dst_path)
    print("sucessful")
  except IOError as e:
    print('Unable to copy file {1 to fl'
          .format (src_path, dst_path))
  except:
    print('when try copy file {1 to {, unexpected error: {}'
        .format (src_path, dst_path, sys.exc_info()))
```

Then I again splitted the train datasets into training and validation ratio taken = (0.8, 0.2). I used splitfolders library for this:

```
[5] import splitfolders # or import split folders
    input_folder = "/content/drive/MyDrive/I'mbesideyou/data.zip (Unzipped Files)/I'mbesideyou/train"

    splitfolders.ratio(input_folder, output="/content/drive/MyDrive/I'mbesideyou/data.zip (Unzipped Files)/I'mbesideyou/Val_Train", seed=42, ratio=(.8,.2),group_prefix=None)
```

Now the data collection and division is done.

Imported all important libraries required for the modeling.

```
from tensorflow.keras.layers import Conv2D,Flatten, Dense,MaxPool2D,BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow. keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
```

Data Preprocessing

As the input images (dataset) have different images, first it is converted into fixed image size of (224, 224), as transfer learning technique takes specific sizes as input images. And also declared the batch size here itself that would be used as train batch size.

Then I defined the paths for the training, validation and test data that i splitted earlier.

Now I am utilizing a datagenerator where I used multiple augmentations like shear range = 0.2, zoom range = 0.2, horizontal flip and vertical flip taken to be true and preprocessing function taken is preprocess input (this is imported from ResNet50 library).

Now, I defined the train generator, validation generator and test generator. In all them categorical class mode is used and the batch size is 32 for training and validation and 1 for testing. Target image size is (224, 224).

```
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   vertical_flip=True)

train_generator = train_datagen.flow_from_directory(
                    train_data_dir,
                    target_size=(img_height, img_width),
                    batch_size=batch_size,
                    class_mode='categorical')

valid_generator = train_datagen.flow_from_directory(
                    valid_data_dir,
                    target_size=(img_height, img_width),
                    batch_size=batch_size,
                    class_mode= 'categorical')
```

```
Found 3862 images belonging to 3 classes.
Found 968 images belonging to 3 classes.
```

```
[22] test_generator = train_datagen.flow_from_directory(
                     test_data_dir,
                     target_size=(img_height, img_width),
                     batch_size=1,
                     class_mode='categorical')
```

```
Found 1610 images belonging to 3 classes.
```

Model Choosing

Now define the model by giving the base model ResNet50, i have followed Residual Neural Network Technique for modeling. Here I put include_top = False because imagenet is trained for 1000 classes datasets but in our case we have to predict only 3 classes which are (Positive Smile, Negative Smile and Not Smile) so put our own dense layer for the output. Here weights are taken by imagenet because I wanted to have a transferlearn model from the imagenet datasets.
X here takes the output of base_model, now adding some extra layers to the base model taken GlobalAveragePooling2D()(x), i have also tried with Maxpool2D imported from tensorflow.keras.layers. Again adding three more dense layers for (256, 512, 1024) respectively activation function is relu in all.
Again used a dense for the prediction with softmax activation function as the classes are more than 2. After applying this the model is turned into a transferlearn model where I am utilizing our own classes.
What the model is doing is taking the input from ResNet50 and giving the output from the last layer which is the prediction layer.
Here I did not enable the transfer to learn trainable because it would increase the training time because it would be a higher number of parameters. Tough to increase accuracy we can enable it.
Now compiling the model, where i took optimizer = adam, loss function = categorical_crossentropy, and metrics = accuracy.

```
base_model = ResNet50(include_top=False, weights='imagenet')
x = base_model.output
x = GlobalAveragePooling2D()(x)

x = Dense(256, activation='relu')(x)

x = Dense(512, activation='relu')(x)

x = Dense(1024, activation='relu')(x)

predictions = Dense(train_generator.num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics = ['accuracy'])
```

Summary of model
##Only adding 1st and last page screenshot of summary here as the summary is very long it might take 5-6 pages.

```
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, None, None,  0           []
                                 3)]

 conv1_pad (ZeroPadding2D)      (None, None, None,   0           ['input_3[0][0]']
                                3)

 conv1_conv (Conv2D)            (None, None, None,   9472        ['conv1_pad[0][0]']
                                64)

 conv1_bn (BatchNormalization)  (None, None, None,   256         ['conv1_conv[0][0]']
                                64)

 conv1_relu (Activation)        (None, None, None,   0           ['conv1_bn[0][0]']
                                64)

 pool1_pad (ZeroPadding2D)      (None, None, None,   0           ['conv1_relu[0][0]']
                                64)

 pool1_pool (MaxPooling2D)      (None, None, None,   0           ['pool1_pad[0][0]']
                                64)

 conv2_block1_1_conv (Conv2D)   (None, None, None,   4160        ['pool1_pool[0][0]']
                                64)

 conv2_block1_1_bn (BatchNormal  (None, None, None,   256        ['conv2_block1_1_conv[0][0]']
 ization)                       64)

 conv2_block1_1_relu (Activatio  (None, None, None,   0          ['conv2_block1_1_bn[0][0]']
 n)                             64)

 conv2_block1_2_conv (Conv2D)   (None, None, None,   36928       ['conv2_block1_1_relu[0][0]']
                                64)

 conv2_block1_2_bn (BatchNormal  (None, None, None,   256        ['conv2_block1_2_conv[0][0]']
 ization)                       64)
```

```
 conv5_block3_1_relu (Activatio  (None, None, None,   0          ['conv5_block3_1_bn[0][0]']
 n)                             512)

 conv5_block3_2_conv (Conv2D)   (None, None, None,   2359808     ['conv5_block3_1_relu[0][0]']
                                512)

 conv5_block3_2_bn (BatchNormal  (None, None, None,   2048       ['conv5_block3_2_conv[0][0]']
 ization)                       512)

 conv5_block3_2_relu (Activatio  (None, None, None,   0          ['conv5_block3_2_bn[0][0]']
 n)                             512)

 conv5_block3_3_conv (Conv2D)   (None, None, None,   1050624     ['conv5_block3_2_relu[0][0]']
                                2048)

 conv5_block3_3_bn (BatchNormal  (None, None, None,   8192       ['conv5_block3_3_conv[0][0]']
 ization)                       2048)

 conv5_block3_add (Add)         (None, None, None,   0           ['conv5_block2_out[0][0]',
                                2048)                            'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activation)  (None, None, None,   0           ['conv5_block3_add[0][0]']
                                2048)

 global_average_pooling2d_2 (Gl  (None, 2048)        0           ['conv5_block3_out[0][0]']
 obalAveragePooling2D)

 dense_4 (Dense)                (None, 256)          524544      ['global_average_pooling2d_2[0][0
                                                                 ]']

 dense_5 (Dense)                (None, 512)          131584      ['dense_4[0][0]']

 dense_6 (Dense)                (None, 1024)         525312      ['dense_5[0][0]']

 dense_7 (Dense)                (None, 3)            3075        ['dense_6[0][0]']

==================================================================================================
Total params: 24,772,227
Trainable params: 1,184,515
Non-trainable params: 23,587,712
__________________________________________________________________________________________________
```

Here, it can be seen that total parameters used = 24+ Millions in which trainable parameters are about 1 million and rest are non-trainable.

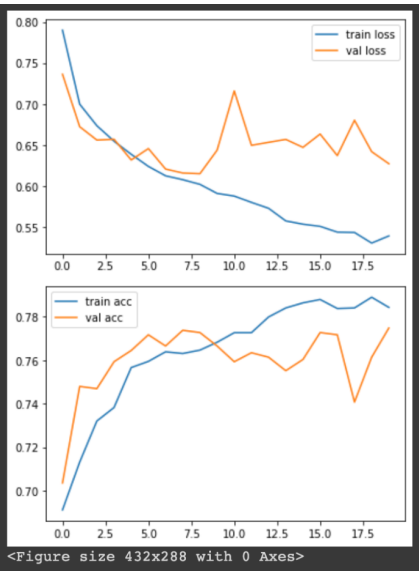
Training of Model

Now run the compiled model for multiple epochs (20). I have tried different epochs, but considering the run time and validation accuracy 20 is taken for the final model, tough accuracy could be improved by applying higher epochs.

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`
  after removing the cwd from sys.path.
Epoch 1/20
121/121 [==============================] - 84s 663ms/step - loss: 0.7900 - accuracy: 0.6911 - val_loss: 0.7363 - val_accuracy: 0.7035
Epoch 2/20
121/121 [==============================] - 77s 640ms/step - loss: 0.7000 - accuracy: 0.7131 - val_loss: 0.6725 - val_accuracy: 0.7479
Epoch 3/20
121/121 [==============================] - 77s 639ms/step - loss: 0.6738 - accuracy: 0.7320 - val_loss: 0.6563 - val_accuracy: 0.7469
Epoch 4/20
121/121 [==============================] - 78s 642ms/step - loss: 0.6549 - accuracy: 0.7382 - val_loss: 0.6573 - val_accuracy: 0.7593
Epoch 5/20
121/121 [==============================] - 77s 638ms/step - loss: 0.6390 - accuracy: 0.7566 - val_loss: 0.6321 - val_accuracy: 0.7645
Epoch 6/20
121/121 [==============================] - 77s 639ms/step - loss: 0.6243 - accuracy: 0.7595 - val_loss: 0.6460 - val_accuracy: 0.7717
Epoch 7/20
121/121 [==============================] - 77s 639ms/step - loss: 0.6128 - accuracy: 0.7639 - val_loss: 0.6211 - val_accuracy: 0.7665
Epoch 8/20
121/121 [==============================] - 77s 639ms/step - loss: 0.6081 - accuracy: 0.7631 - val_loss: 0.6162 - val_accuracy: 0.7738
Epoch 9/20
121/121 [==============================] - 77s 633ms/step - loss: 0.6024 - accuracy: 0.7646 - val_loss: 0.6153 - val_accuracy: 0.7727
Epoch 10/20
121/121 [==============================] - 76s 632ms/step - loss: 0.5914 - accuracy: 0.7683 - val_loss: 0.6440 - val_accuracy: 0.7665
Epoch 11/20
121/121 [==============================] - 77s 633ms/step - loss: 0.5882 - accuracy: 0.7727 - val_loss: 0.7161 - val_accuracy: 0.7593
Epoch 12/20
121/121 [==============================] - 77s 636ms/step - loss: 0.5805 - accuracy: 0.7727 - val_loss: 0.6499 - val_accuracy: 0.7634
Epoch 13/20
121/121 [==============================] - 77s 639ms/step - loss: 0.5733 - accuracy: 0.7799 - val_loss: 0.6536 - val_accuracy: 0.7614
Epoch 14/20
121/121 [==============================] - 77s 640ms/step - loss: 0.5579 - accuracy: 0.7840 - val_loss: 0.6571 - val_accuracy: 0.7552
Epoch 15/20
121/121 [==============================] - 77s 638ms/step - loss: 0.5540 - accuracy: 0.7864 - val_loss: 0.6474 - val_accuracy: 0.7603
Epoch 16/20
121/121 [==============================] - 77s 635ms/step - loss: 0.5513 - accuracy: 0.7879 - val_loss: 0.6636 - val_accuracy: 0.7727
Epoch 17/20
121/121 [==============================] - 76s 631ms/step - loss: 0.5443 - accuracy: 0.7838 - val_loss: 0.6374 - val_accuracy: 0.7717
Epoch 18/20
121/121 [==============================] - 77s 636ms/step - loss: 0.5439 - accuracy: 0.7840 - val_loss: 0.6804 - val_accuracy: 0.7407
Epoch 19/20
121/121 [==============================] - 77s 636ms/step - loss: 0.5310 - accuracy: 0.7890 - val_loss: 0.6422 - val_accuracy: 0.7614
Epoch 20/20
121/121 [==============================] - 77s 637ms/step - loss: 0.5396 - accuracy: 0.7843 - val_loss: 0.6275 - val_accuracy: 0.7748
```

Evaluating the model

Now, the graph is plotted for train loss with val loss and for train accuracy with val accuracy.




```
<Figure size 432x288 with 0 Axes>
```

Now, saved the model as a .h5 file.

```
[16] model.save("/content/drive/MyDrive/I'mbesideyou/data.zip (Unzipped Files)/I'mbesideyou/ResNet_model.h5")
```

Testing of Model

Now, I tested the model for the test dataset.

```
[29] test_loss, test_acc = model.evaluate(test_generator, verbose=2)
     print ('InTest accuracy:', test_acc)

     1610/1610 - 41s - loss: 0.6033 - accuracy: 0.7820 - 41s/epoch - 26ms/step
     InTest accuracy: 0.7819875478744507
```

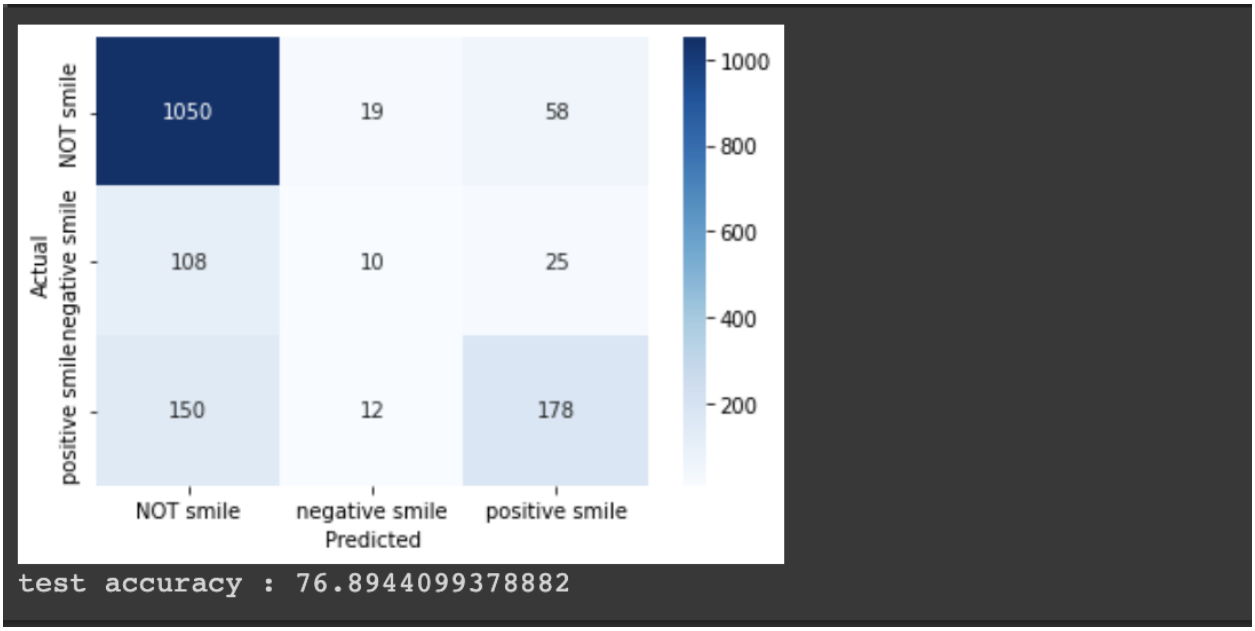
Test accuracy is more than 78%.

Now, confusion matrix is made here below is the code i used

```
import pandas as pd
import seaborn as sn
import tensorflow as tf
model = tf.keras.models.load_model("/content/drive/MyDrive/I'mbesideyou/data.zip (Unzipped Files)/I'mbesideyou/ResNet_model.h5")
filenames = test_generator.filenames
nb_samples = len(test_generator)
y_prob=[]
y_act=[]
test_generator.reset()
for _ in range(nb_samples):
  x_test, y_test = test_generator.next()
  y_prob.append(model.predict(x_test))
  y_act.append(y_test)
predicted_class = [list(train_generator.class_indices.keys())[i.argmax()] for i in y_prob]
actual_class = [list(train_generator.class_indices.keys())[i.argmax()] for i in y_act]
out_df = pd.DataFrame(np.vstack([predicted_class,actual_class]).T,columns=['predicted_class','actual_class'])
confusion_matrix = pd.crosstab(out_df['actual_class'],out_df['predicted_class'], rownames=['Actual'], colnames=['Predicted'])

sn.heatmap(confusion_matrix, cmap='Blues', annot=True, fmt='d')
plt.show()
print('test accuracy : {}'.format ((np.diagonal(confusion_matrix).sum()/confusion_matrix.sum().sum()*100)))
```

Confusion matrix :



Here also could be see that test accuracy is 76.89%

#Please Click below link to redirect to the source codes

Google Colab link for model code (ResNet50_Model)

#I have also tried CNN models also using InceptionV3 but didn't get better accuracy.
Google Colab link for model code (InceptionV3_Model)

Thank You!