Mood Detection with OpenCV

John Carlos F. Camarao | 1920173
CPE312 - CPE32S7

In [1]:
```
#Libraries
import cv2
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from keras.applications.mobilenet import MobileNet, preprocess_input
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.models import Model, load_model
from keras.layers import Flatten, Dense

from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
```

Data preparation

In [2]:
```
train_gen = ImageDataGenerator(zoom_range=0.2,
                               shear_range=0.2,
                               horizontal_flip=True,
                               rescale=1./255)

train_data = train_gen.flow_from_directory(directory=r"C:\Users\johnc\OneDrive\Desktop\ComVis\train",
                                           target_size=(224, 224),
                                           batch_size=32,
                                           class_mode='categorical')
```

```
Found 350 images belonging to 7 classes.
```

In [3]:
```
val_datagen = ImageDataGenerator(rescale=1/255)
val_data = val_datagen.flow_from_directory(directory=r"C:\Users\johnc\OneDrive\Desktop\ComVis\train",
                                           target_size=(224, 224),
                                           batch_size=32,
                                           class_mode='categorical',
                                           shuffle=False)
```

```
Found 350 images belonging to 7 classes.
```

Here I implemented the necessary libraries that's needed for the project. Pandas, numpy and matplotlib are basic libraries needed to at least load, plot and do numbers with. Then keras modules for neural networks.

For the data preparation, here is where the module ImageDataGenerator was used. I created an ImageDataGenerator for data augmentation during training and all the parameters I used are commonly used. Data augmentation is used here to reduce overfitting and improve performance since the dataset I found is small.

Val_data is for validation images and was sent to a directory on my local machine.

Model

In [4]:
```
original_model = MobileNet(input_shape=(224, 224, 3), include_top=False)
for layer in original_model.layers:
    layer.trainable = False

x = Flatten()(original_model.output)
x = Dense(units=7, activation='softmax')(x)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf_no_top.h5
17225924/17225924 ———————————— 29s 2us/step
```

In [5]:
```
model = Model(original_model.input, x)
```

In [6]:
```
model.summary()
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
conv1 (Conv2D)	(None, 112, 112, 32)	864
conv1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv1_relu (ReLU)	(None, 112, 112, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_dw_1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	0
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2,048

In [7]:
```
model.compile(optimizer='adam',
              loss="categorical_crossentropy",
              metrics=['accuracy'])
```

For the model I choose MobileNet since it's said that it's computational work is efficient and lightweight. Given the time I have in making this assignment I opted for it for faster training and evaluation. For the weights, the input (224, 224, 3) are quite common for pre-trained models.then the freezing of the layers is common practice as demonstrated in previous HOA'. and is done to preserve the pretrained features that I will connect to the mood detection tasks

---

In [7]:
```
model.compile(optimizer='adam',
              loss="categorical_crossentropy",
              metrics=['accuracy'])
```

This is where I compile the model for use, using parameters like adams for better performance compared to SGD, and others that are commonly used.

Early stopping

In [9]:
```
early_stop = EarlyStopping(monitor='val_accuracy',
                           min_delta=0.01,
                           patience=5,
                           verbose=1,
                           mode='auto')
```

In [10]:
```
model_checkpoint = ModelCheckpoint(filepath=r"C:\Users\johnc\OneDrive\Desktop\ComVis\train\model1.keras",
                                   monitor = 'val_accuracy',
                                   verbose = 1,
                                   save_best_only =
                                   True, mode = 'auto')
```

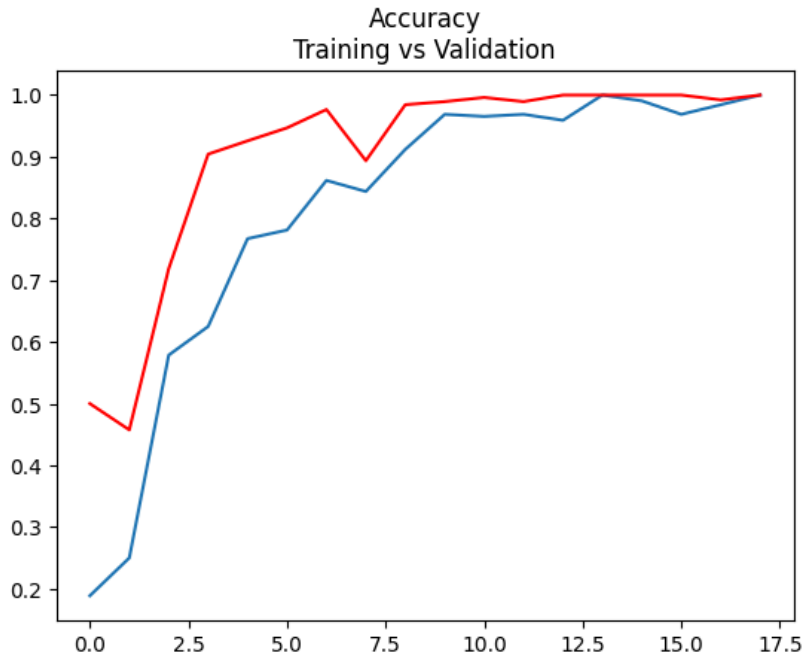
```
model_checkpoint = ModelCheckpoint(filepath=r"C:\Users\John Carlos Camarao\Desktop\ComVis\model1.h5", monitor = 'val_accuracy', verbose = 1,
save_best_only = True, mode = 'auto')
```

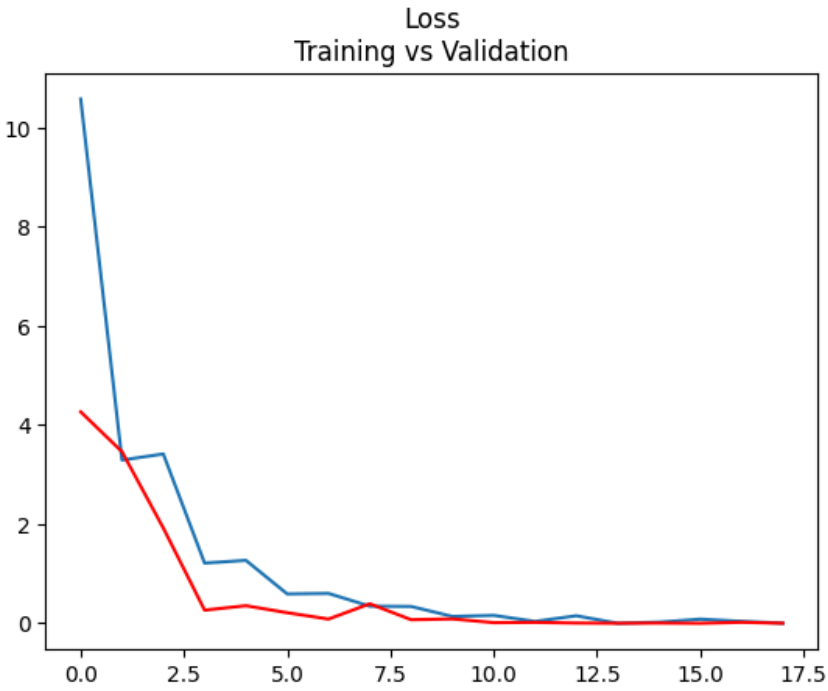
In [11]:
```
hist = model.fit(train_data,
                 steps_per_epoch=10,
                 epochs=30,
                 validation_data=val_data,
                 validation_steps=8,
                 callbacks=[early_stop, model_checkpoint])
```

```
Epoch 1/30
C:\Users\johnc\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.
py:122: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can inclu
de `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 518ms/step - accuracy: 0.1118 - loss: 10.9211
Epoch 1: val_accuracy improved from -inf to 0.50000, saving model to C:\Users\johnc\OneDrive\Desktop\ComVis\train\model1.kera
s
10/10 ━━━━━━━━━━━━━━━━━━━━ 14s 1s/step - accuracy: 0.1188 - loss: 10.8891 - val_accuracy: 0.5000 - val_loss: 4.2636
```

In here I also added “Early Stopping” as I mentioned before my dataset is small so I needed to compensate for it to avoid overfitting which was applied during the training on model.fit and Early stopping is one of the techniques demonstrated in our previous lessons and at the same time it’s a common practice to use in Deep Learning CNNs. It also improves train time.

ModelCheckpoint was used to save my model, and applied to the model training and recording.





So far, looking at these plots the training accuracy shows a consistent improvement where you can see as the number of epochs increases which means the model learns from the training data. Although it plateaus after a certain amount of epochs.

For the training loss it appears as expected where it decreases as the model learns more from the data.

Sample Output: