# FACIAL RECOGNITION WITH KERAS

***PROJECT REPORT***

INTRODUCTION TO DATA ANALYTICS

***By***

PARAS WAHI **E18ECE026**

*Submitted to*

**Dr. Manoj Sharma**

# INDEX

## WHAT IS DATA SCIENCE ?

Data Science is a process, not an event. It is the process of using data to understand different things, to understand the world. For me is when you have a model or hypothesis of a problem, and you try to validate that hypothesis or model with your data. Data science is the art of uncovering the insights and trends that are hiding behind data. It's when you translate data into a story.

# INTRODUCTION

The data consists of 48x48 pixel grayscale images of faces. The objective is to classify each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

We will use OpenCV to automatically detect faces in images and draw bounding boxes around them. Once we have trained, saved, and exported the CNN, we will directly serve the trained model to a web interface and perform real-time facial expression recognition on video and image data.

# IMPORTING LIBRARIES

**Task 1: Import Libraries**

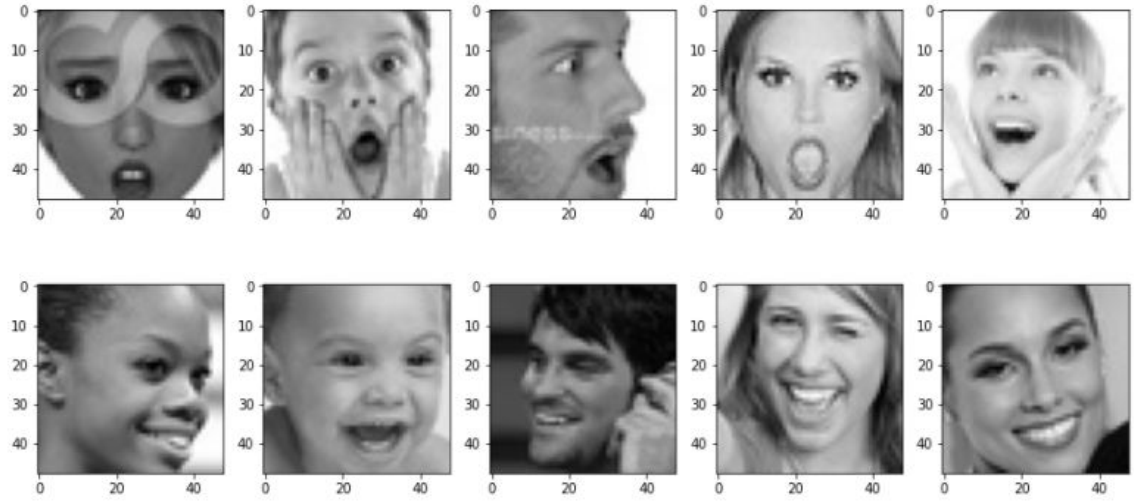
In [1]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import utils
import os
%matplotlib inline

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Input, Dropout,Flatten, Conv2D
from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooling2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.utils import plot_model

from IPython.display import SVG, Image
from livelossplot import PlotLossesKerasTF
import tensorflow as tf
print("Tensorflow version:", tf.__version__)
```

```
Tensorflow version: 2.2.0
```

# PLOT SAMPLE IMAGES

**Task 2: Plot Sample Images**

In [2]:
```
utils.datasets.fer.plot_example_images(plt).show()
```



In [3]:
```
for expression in os.listdir("train/"):
    print(str(len(os.listdir("train/" + expression))) + " " + expression + " images")
```

```
3171 surprise images
7214 happy images
4965 neutral images
3995 angry images
4830 sad images
436 disgust images
4097 fear images
```

# GENERATE TRAINING AND VALIDATION BATCHES

### Task 3: Generate Training and Validation Batches

In [4]:
```
img_size = 48
batch_size = 64

datagen_train = ImageDataGenerator(horizontal_flip=True)

train_generator = datagen_train.flow_from_directory("train/",
                                                    target_size=(img_size,img_size),
                                                    color_mode="grayscale",
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=True)

datagen_validation = ImageDataGenerator(horizontal_flip=True)
validation_generator = datagen_validation.flow_from_directory("test/",
                                                    target_size=(img_size,img_size),
                                                    color_mode="grayscale",
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=False)
```

```
Found 28708 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

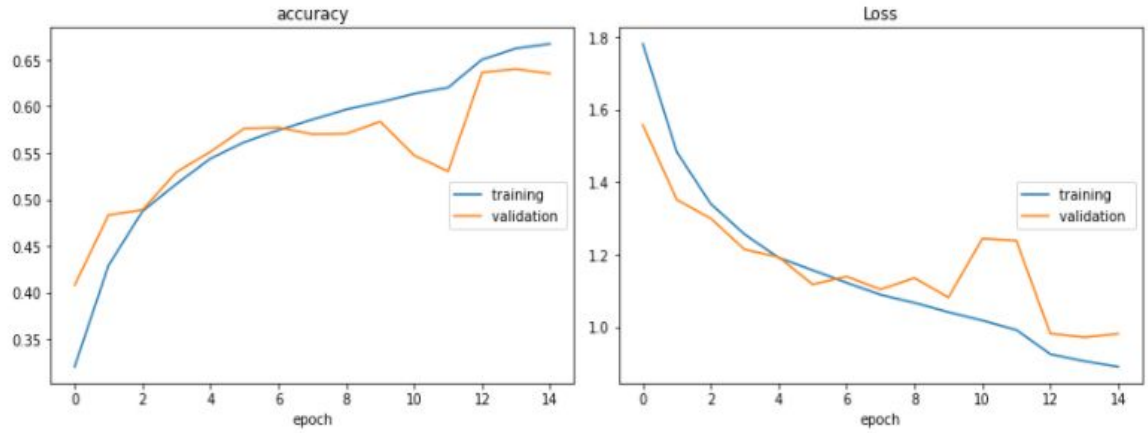
# CREATE A CNN MODEL

In [6]:
```
# Initialising the CNN
model = Sequential()

# 1 - Convolution
model.add(Conv2D(64,(3,3), padding='same', input_shape=(48, 48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 2nd Convolution layer
model.add(Conv2D(128,(5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 3rd Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
# 4th Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Flattening
model.add(Flatten())

# Fully connected layer 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(7, activation='softmax'))

opt = Adam(lr=0.0005)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

# TRAIN AND EVALUATE THE MODEL

```
%%time

epochs = 15
steps_per_epoch = train_generator.n//train_generator.batch_size
validation_steps = validation_generator.n//validation_generator.batch_size

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                              patience=2, min_lr=0.00001, mode='auto')
checkpoint = ModelCheckpoint("model_weights.h5", monitor='val_accuracy',
                             save_weights_only=True, mode='max', verbose=1)
callbacks = [PlotLossesKerasTF(), checkpoint, reduce_lr]

history = model.fit(
    x=train_generator,
    steps_per_epoch=steps_per_epoch,
    epochs=epochs,
    validation_data = validation_generator,
    validation_steps = validation_steps,
    callbacks=callbacks
)
```




```
accuracy
        training                (min:    0.320, max:    0.667, cur:    0.667)
        validation              (min:    0.408, max:    0.640, cur:    0.635)
Loss
        training                (min:    0.890, max:    1.781, cur:    0.890)
        validation              (min:    0.972, max:    1.557, cur:    0.981)

Epoch 00015: saving model to model_weights.h5
448/448 [==============================] - 26s 58ms/step - loss: 0.8904 - accuracy: 0.6672 - val_loss: 0.9813 - val_accuracy:
0.6353 - lr: 5.0000e-05
CPU times: user 6min 34s, sys: 56.7 s, total: 7min 30s
Wall time: 6min 51s
```

## REPRESENT AS JSON STRING

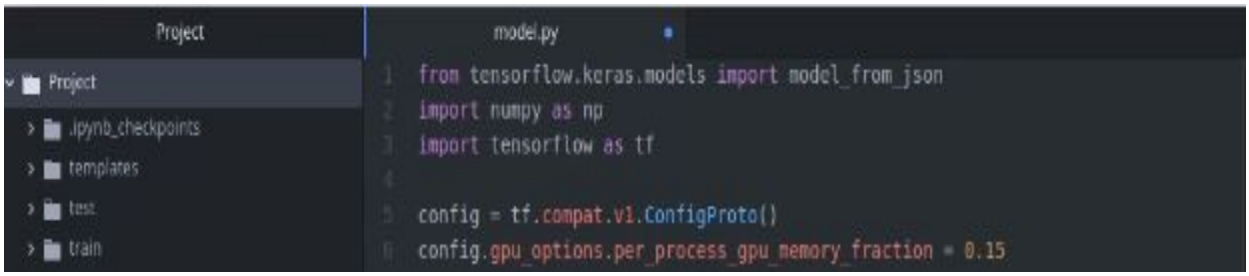
```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

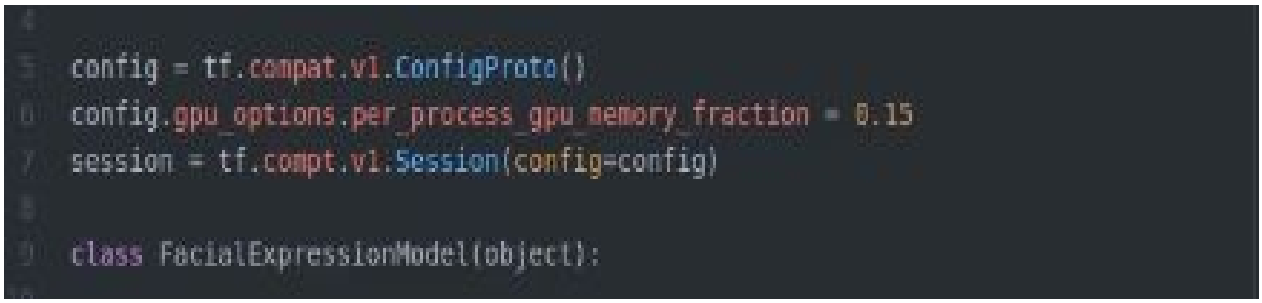
# CREATE A FLASK APP TO SERVE PREDICTIONS

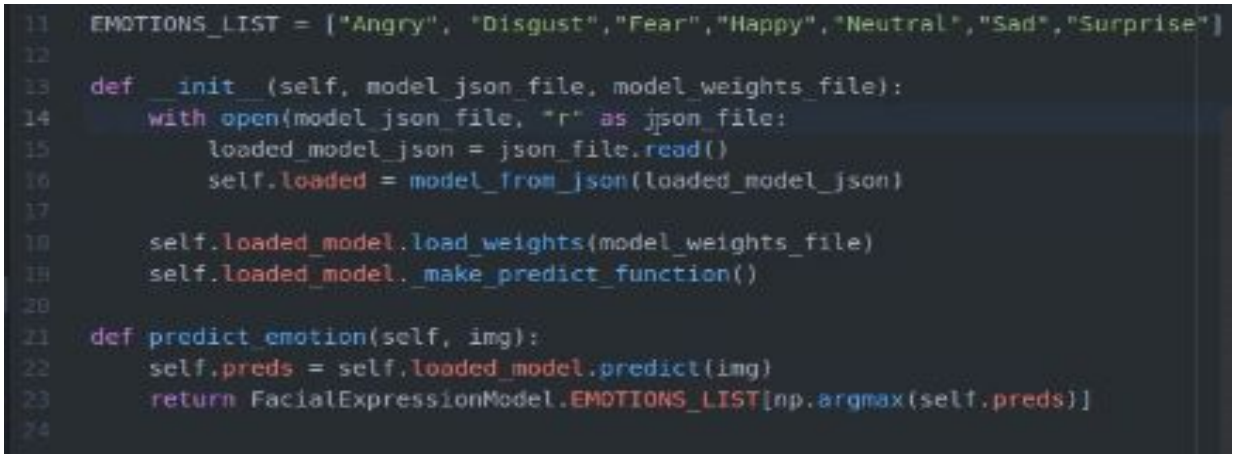
```python
import cv2
from model import FacialExpressionModel
import numpy as np

facec = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
model = FacialExpressionModel("model.json", "model_weights.h5")
font = cv2.FONT_HERSHEY_SIMPLEX

class VideoCamera(object):
    def __init__(self):
        self.video = cv2.VideoCapture("/home/rhyme/Desktop/Project/videos/facial_exp.mkv")

    def __del__(self):
        self.video.release()

    # returns camera frames along with bounding boxes and predictions
    def get_frame(self):
        _, fr = self.video.read()
        gray_fr = cv2.cvtColor(fr, cv2.COLOR_BGR2GRAY)
        faces = facec.detectMultiScale(gray_fr, 1.3, 5)

        for (x, y, w, h) in faces:
            fc = gray_fr[y:y+h, x:x+w]

            roi = cv2.resize(fc, (48, 48))
            pred = model.predict_emotion(roi[np.newaxis, :, :, np.newaxis])

            cv2.putText(fr, pred, (x, y), font, 1, (255, 255, 0), 2)
            cv2.rectangle(fr,(x,y),(x+w,y+h),(255,0,0),2)

        _, jpeg = cv2.imencode('.jpg', fr)
        return jpeg.tobytes()
```

```python
from flask import Flask, render_template, Response
from camera import VideoCamera


app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

def gen(camera):
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(gen(VideoCamera()),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

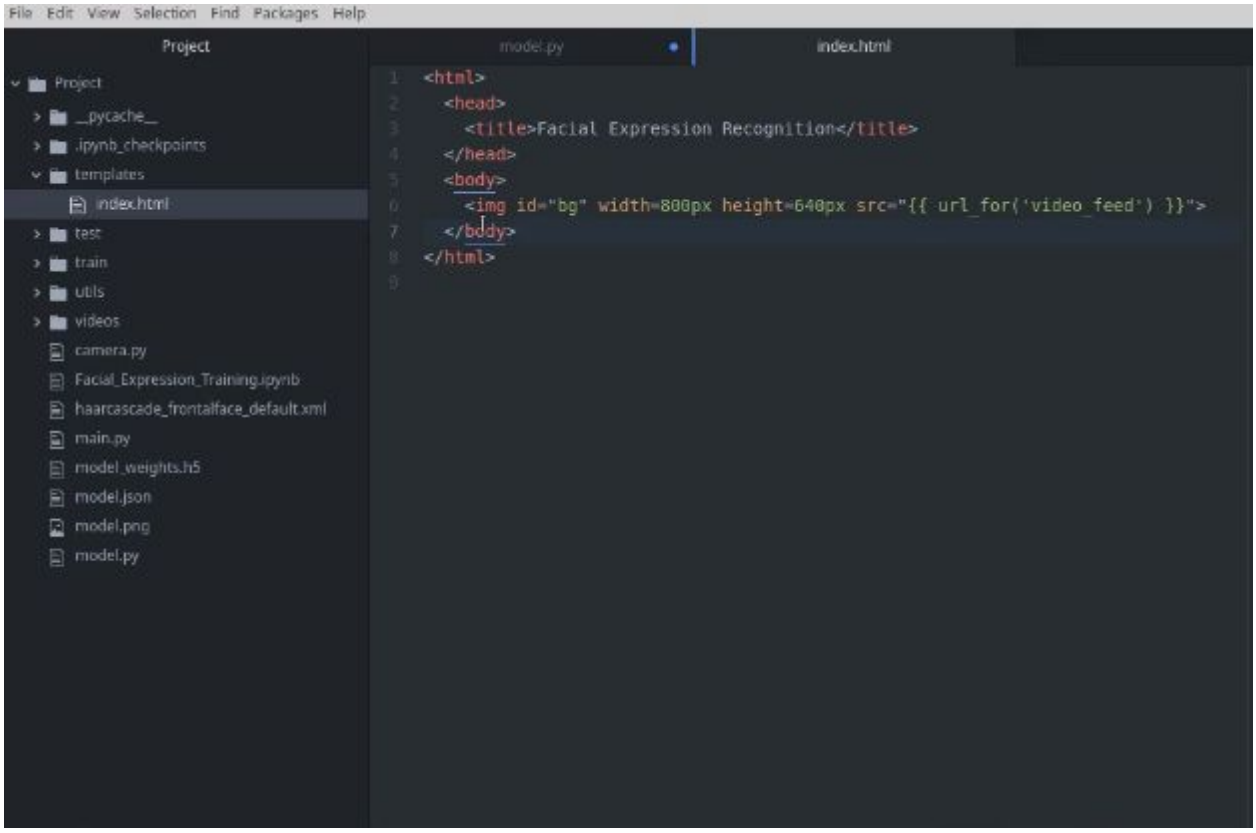
# CREATE A CLASS OUTPUT TO MODEL PREDICTIONS



## DESIGN A HTML TEMPLATE FOR THE FLASK APP




## Use Model to Recognize Facial Expressions in Videos

- Run the main.py script to create the Flask app and serve the model's predictions to a web interface.
- Apply the model to saved videos on disk.