Facial Expression Recognition with Keras

by Coursera Project Network

Completed by Suhaimi William Chan

Taught by: Snehan Kekre, Machine Learning Instructor

About this Course
Build and train a convolutional neural network (CNN) in Keras from scratch to recognize facial expressions. 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).

You will use OpenCV to automatically detect faces in images and draw bounding boxes around them. Once you have trained, saved, and exported the CNN, you will directly serve the trained model to a web interface and perform real-time facial expression recognition on video and image data. For this project, you'll get instant access to a cloud desktop with Python, Jupyter, and Keras pre-installed.

Course Objectives

In this course, we are going to focus on four learning objectives:

1. Develop a facial expression recognition model in Keras
2. Build and train a convolutional neural network (CNN)
3. Deploy the trained model to a web interface with Flask
4. Apply the model to real-time video streams and image data

By the end of this course, you will be able to build and train a convolutional neural network (CNN) in Keras from scratch to recognize facial expressions. Once you have trained, saved, and exported the CNN, you will directly serve the trained model to a web interface and perform real-time facial expression recognition on video and image data.

Course Structure

This course is divided into 3 parts:

1. Course Overview: This introductory reading material.
2. Facial Expression Recognition with Keras**:** This is the hands on project that we will work on in Rhyme.
3. Graded Quiz: This is the final assignment that you need to pass in order to finish the course successfully.

Project Structure

The hands on project on Facial Expression Recognition is divided into following tasks:

Task 1: Introduction and Overview

- Introduction to the data and and overview of the project.
- See a demo of the final product you will build by the end of this project.
- Introduction to the Rhyme interface.
- Import essential modules and helper functions from NumPy, Matplotlib, and Keras.

Task 2: Explore the Dataset

- Display some images from every expression type in the Emotion FER dataset.
- Check for class imbalance problems in the training data.

Task 3: Generate Training and Validation Batches

- Generate batches of tensor image data with real-time data augmentation.
- Specify paths to training and validation image directories and generates batches of augmented data.

Task 4: Create a Convolutional Neural Network (CNN) Model

- Design a convolutional neural network with 4 convolution layers and 2 fully connected layers to predict 7 types of facial expressions.
- Use Adam as the optimizer, categorical crossentropy as the loss function, and accuracy as the evaluation metric.

Task 5: Train and Evaluate Model

- Train the CNN by invoking the **model.fit()** method.
- Use **ModelCheckpoint()** to save the weights associated with the higher validation accuracy.
- Observe live training loss and accuracy plots in Jupyter Notebook for Keras.

Task 6: Save and Serialize Model as JSON String

- Sometimes, you are only interested in the architecture of the model, and you don't need to save the weight values or the optimizer.
- Use **to_json()**, which uses a JSON string, to store the model architecture.

Task 7: Create a Flask App to Serve Predictions

- Use open-source code from "Video Streaming with Flask Example" to create a flask app to serve the model's prediction images directly to a web interface.

Task 8: Create a Class to Output Model Predictions

- Create a FacialExpressionModel class to load the model from the JSON file, load the trained weights into the model, and predict facial expressions.

Task 9: Design an HTML Template for the Flask App

- Design a basic template in HTML to create the layout for the Flask app.

Task 10: 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.

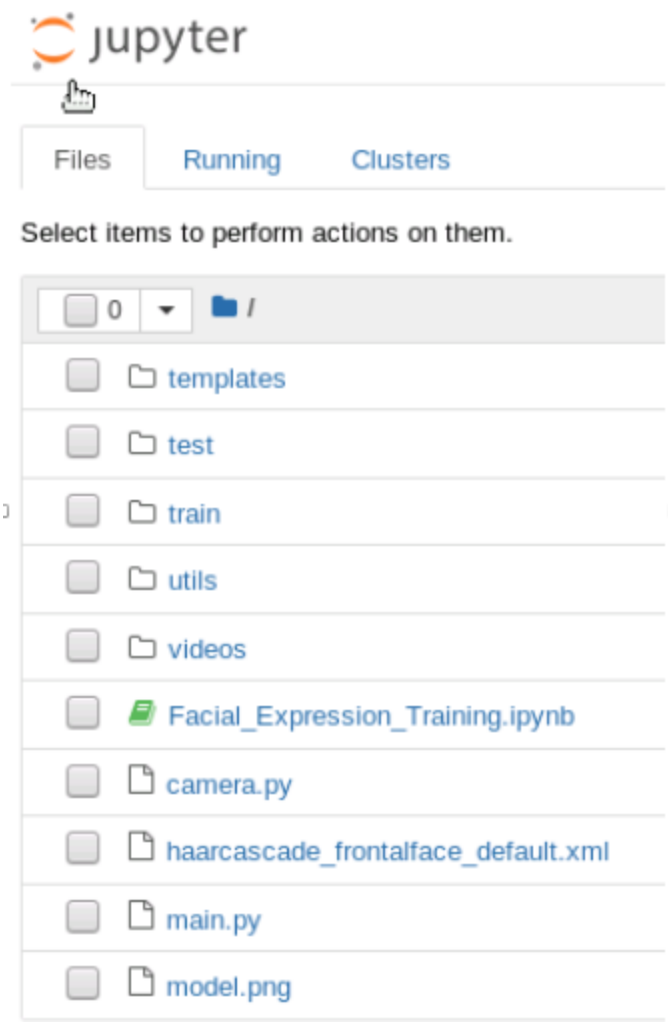
Completed Jupyter Notebook and Scripts

This .zip file contains all the code and data used in the project. It includes the FER 2013 dataset, completed Jupyter notebook for training, the Flask app to serve predictions, and other utility scripts. Please feel free to modify any and all aspects of the code to suit your needs.

Project.zip

Once you have downloaded and extracted Project.zip, make sure to install dependencies using pipenv with the provided Pipfile and execute all commands using pipenv. Also, please make sure to add the correct path to the video file in camera.py on line 11. Next, to install pipenv, the dependencies, and run the main.py file, execute the following commands from your terminal or command prompt, making sure to add the right paths where necessary:

- cd \path\to\Project\
- pip install pipenv
- pipenv install
- pipenv run python3 main.py


jupyter
Files
Running
Clusters
Select items to perform actions on them.
0
/
templates
test
train
utils
videos
Facial_Expression_Training.ipynb
camera.py
haarcascade_frontalface_default.xml
main.py
model.png

Facial Expression Recognition with Keras

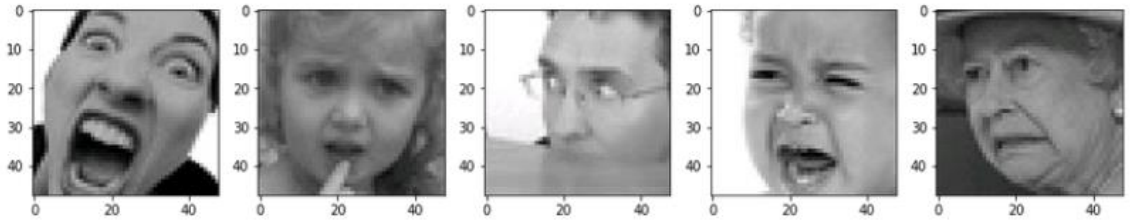
Task 1: Import Libraries

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 PlotLossesTensorFlowKeras
import tensorflow as tf
print("Tensorflow version:", tf.__version__)
```

```
Tensorflow version: 2.1.0
```

Task 2: Plot Sample Image

In [3]:
```
utils.datasets.fer.plot_example_images(plt).show()
```



In [5]:
```
for expression in os.listdir("train/"):
    print(str(len(os.listdir("train/" + expression))) + " " + expression + " images")
```

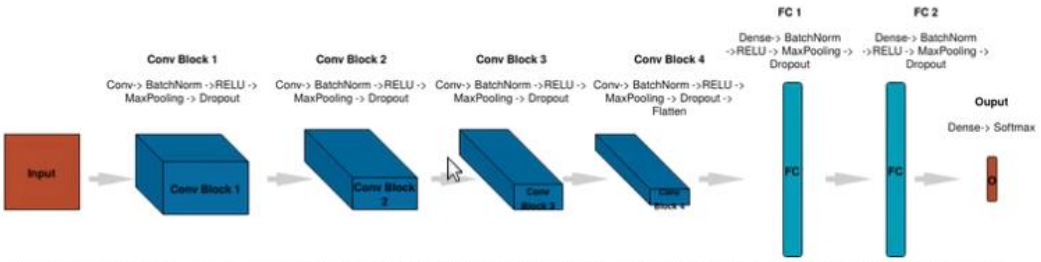
```
3171 surprise images
7215 happy images
4965 neutral images
3995 angry images
4830 sad images
436 disgust images
4097 fear images
```

Task 3: Generate Training and Validation Batches

In [6]:
```
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_train.flow_from_directory("test/",
                                                    target_size=(img_size, img_size),
                                                    color_mode='grayscale',
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=True)
```

```
Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

Task 4: Create CNN Model



Inspired by Goodfellow, I.J., et.al. (2013). Challenged in representation learning: A report of three machine learning contests. *Neural Networks*, 64, 59-63. doi:10.1016/j.neunet.2014.09.005

Task 4: Create CNN Model

```
![](model.png)
Inspired by Goodfellow, I.J., et.al. (2013). Challenged in representation learning: A report of
three machine learning contests. *Neural Networks*, 64, 59-63.
[doi:10.1016/j.neunet.2014.09.005](https://arxiv.org/pdf/1307.0414.pdf)
```

In [9]:
```
model = Sequential()

# 1 - conv
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))

# 2 - conv 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))

# 3 - conv 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))
```

```
# 4 - conv 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))

model.add(Flatten())

model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

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()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 48, 48, 64)        640
_________________________________________________________________
batch_normalization_4 (Batch (None, 48, 48, 64)        256
_________________________________________________________________
activation_5 (Activation)    (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 128)       204928
_________________________________________________________________
batch_normalization_5 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
activation_6 (Activation)    (None, 24, 24, 128)       0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
dropout_5 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
```

conv2d_6 (Conv2D)	(None, 12, 12, 512)	590336
batch_normalization_6 (Batch	(None, 12, 12, 512)	2048
activation_7 (Activation)	(None, 12, 12, 512)	0

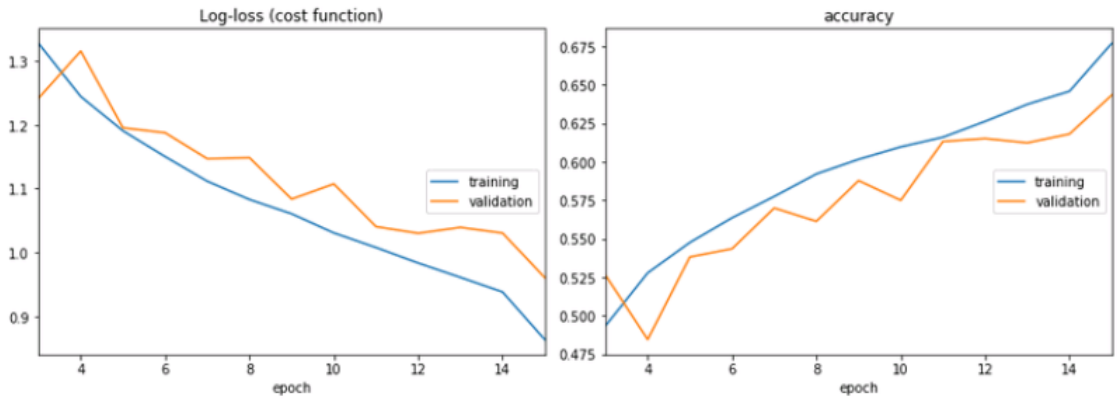
oken=4a7fa964736fc6eda836c240f6afdd481a5d359156d2d359

max_pooling2d_8 (MaxPooling2	(None, 3, 3, 512)	0
dropout_7 (Dropout)	(None, 3, 3, 512)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 256)	1179904
batch_normalization_8 (Batch	(None, 256)	1024
activation_9 (Activation)	(None, 256)	0
dropout_8 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 512)	131584
batch_normalization_9 (Batch	(None, 512)	2048
activation_10 (Activation)	(None, 512)	0
dropout_9 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 7)	3591

Total params: 4,478,727
Trainable params: 4,474,759
Non-trainable params: 3,968

Task 6: Train and Evaluate Model

In [10]:
```
epochs = 15
steps_per_epoch = train_generator.n//train_generator.batch_size
validation_steps = validation_generator.n//validation_generator.batch_size

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

callbacks = [PlotLossesTensorFlowKeras(), 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
)
```




```
Log-loss (cost function):
training   (min:    0.864, max:    1.804, cur:    0.864)
validation (min:    0.961, max:    1.885, cur:    0.961)

accuracy:
training   (min:    0.312, max:    0.677, cur:    0.677)
validation (min:    0.349, max:    0.643, cur:    0.643)

Epoch 00015: saving model to model_weights.h5
448/448 [==============================] - 26s 59ms/step - loss: 0.8636 - accuracy: 0.6769 - va
l_loss: 0.9608 - val_accuracy: 0.6433
```

Task 7: Represent Model as JSON String

In [13]:
```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

/home/rhyme/Desktop/Project/camera.py - Mousepad

File Edit Search View Document Help

```
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()
```

For your own live cam, you can change the following

self.video = cv2.VideoCapture(0)

*/home/rhyme/Desktop/Project/camera.py - Mousepad

File Edit Search View Document Help

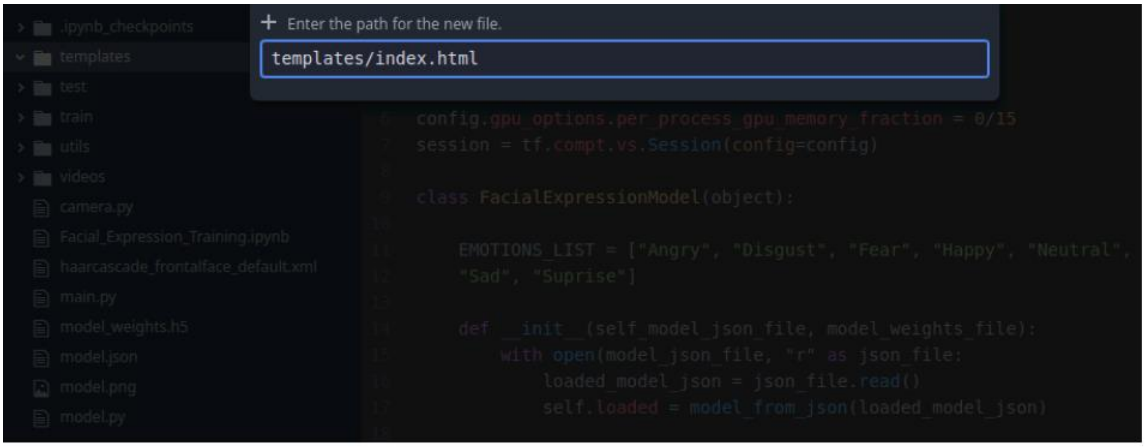
```
import cv2
from model import FacialExpressionModel
import numpy as np

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(0)

    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()
```

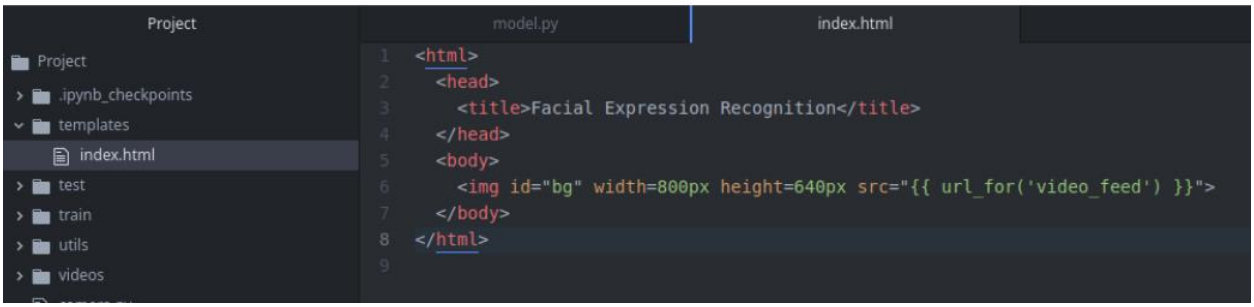
/home/rhyme/Desktop/Project/main.py - Mousepad

File Edit Search View Document Help

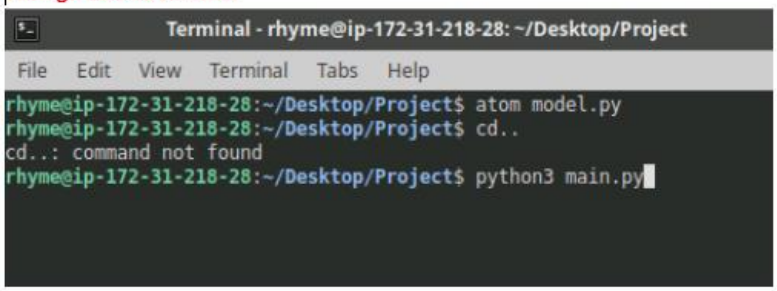
```
from flask import Flask, render_template, Response
from camera import VideoCamera


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)
```

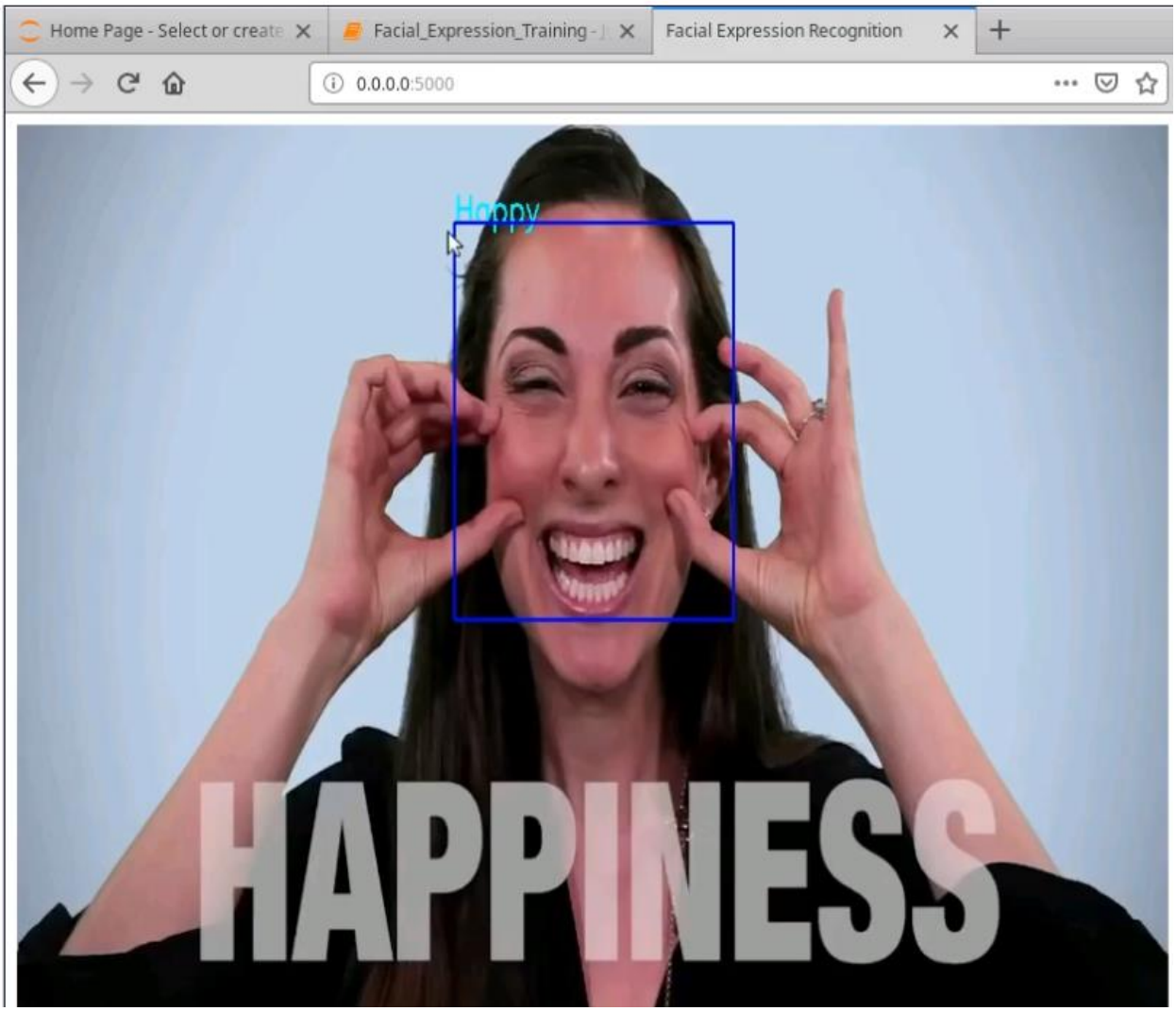
Open terminal
Type: atom model.py


Enter the path for the new file.
templates/index.html
.ipynb_checkpoints
templates
test
train
utils
videos
camera.py
Facial_Expression_Training.ipynb
haarcascade_frontalface_default.xml
main.py
model_weights.h5
model.json
model.png
model.py
config.gpu_options.per_process_gpu_memory_fraction = 0.15
session = tf.compat.v1.Session(config=config)
class FacialExpressionModel(object):
EMOTIONS_LIST = ["Angry", "Disgust", "Fear", "Happy", "Neutral",
"Sad", "Suprise"]
def __init__(self, model_json_file, model_weights_file):
with open(model_json_file, "r") as json_file:
loaded_model_json = json_file.read()
self.loaded = model_from_json(loaded_model_json)


Project
Project
.ipynb_checkpoints
templates
index.html
test
train
utils
videos
model.py
index.html
<html>
<head>
<title>Facial Expression Recognition</title>
</head>
<body>
<img id="bg" width=800px height=640px src="{{ url_for('video_feed') }}">
</body>
</html>

Go back to Jupyter, go to Kernel and select restart
Then go back to terminal



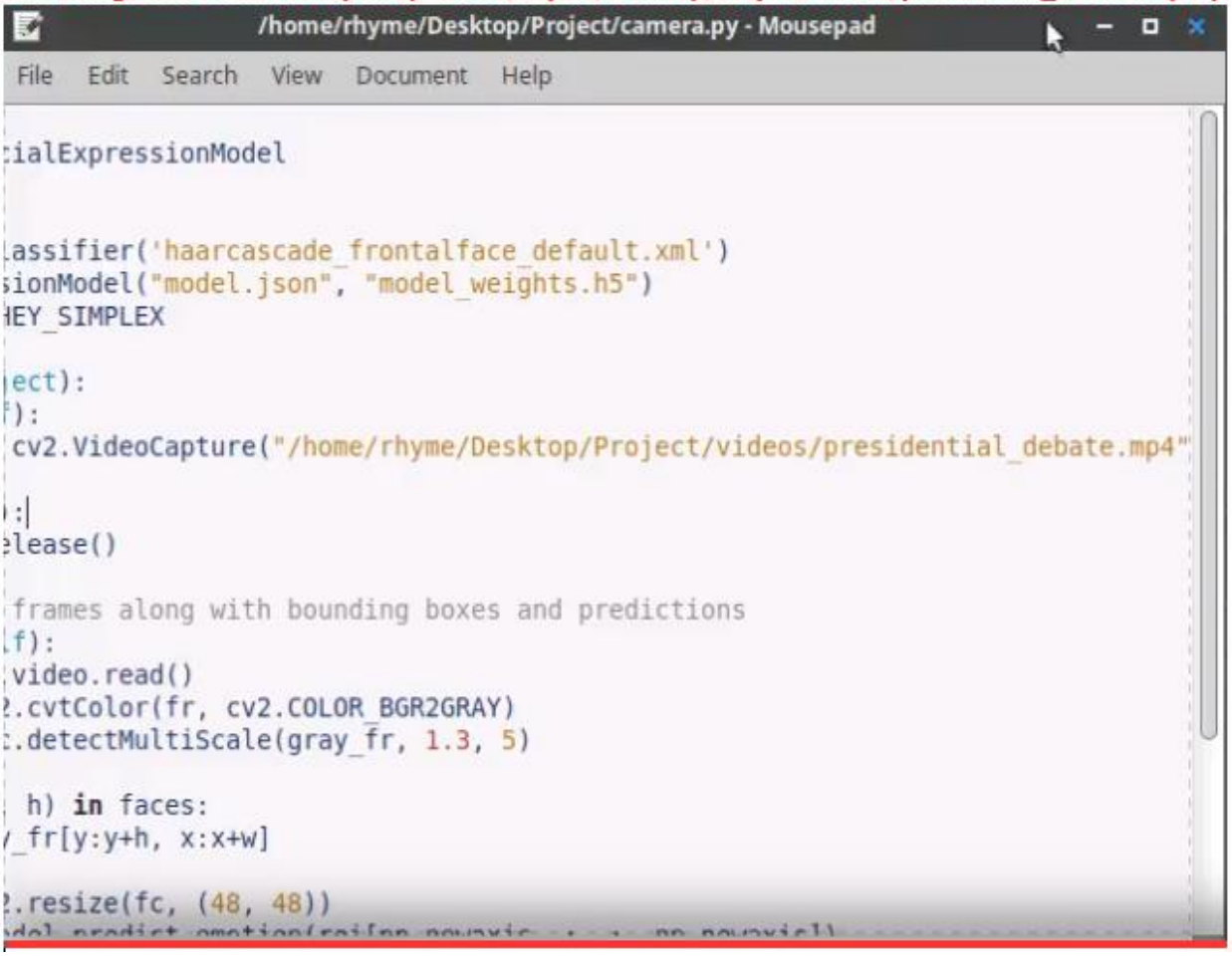



0.0.0.0:5000
Angry
ANGER


0.0.0.0:5000
Surprise
SURPRISE


7 UNIVERSAL
FACIAL
EXPRESSIONS

Change the location of file to be analyzed:
Just change it in cv2.VideoCapture(/"home/rhyme/Desktop/Project/videos/presidential_debate.mp4")




Home Page - Select or create
Facial_Expression_Training -
Facial Expression Recognition
0.0.0.0:5000
Sad
The Guardian
The Washington Post
NBC NEWS
MSNBC
LIVE
MSNBC
DEMOCRATIC DEBATE


Home Page - Select or create
Facial_Expression_Training -
Facial Expression Recognition
0.0.0.0:5000
MSNBC/Washington Post
The Guardian
Neutral
NBC NEWS
MSNBC
LIVE
MSNBC
DEMOCRATIC DEBATE