# ESAPR

# Emotional State Analyser for Psychological Response

Project Report

*Submitted by*

(Roll No: **12619001010**, Reg No: 029238 of 2019-2020)
(Roll No: **12619001087**, Reg No: 031543 of 2019-2020)
(Roll No: **12619001036**, Reg No: 013597 of 2019-2020)

*Under the Supervision of*

**Prof. Mohuya Byabartta Kar**

Department Of Computer Science And Engineering


## HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA


# HERITAGE INSTITUTE OF TECHNOLOGY

# KOLKATA

## BONAFIDE CERTIFICATE

Certified that this Project "**EMOTIONAL STATE ANALYSER FOR PSYCHOLOGICAL RESPONSE (ESPAR)**" is the bonafide work of "**Aditya Roshan Jha**", "**Anushka Das**" and "**Moonmoon Mondal**". Who carried out project work under my supervision.

**SIGNATURE**

**HEAD OF THE DEPARTMENT**

Computer Sci. & Enggi.

Heritage Institute Of Technology

**SIGNATURE**

**PROJECT GUIDE**

Computer Sci. & Enggi.

Heritage Institute Of Technology

**SIGNATURE EXAMINER(S)**

# Acknowledgement

The authors remember with gratitude the constant guidance, suggestions and encouragement forwarded by several respected persons. We would like to convey my heartfelt gratitude to **Prof. Dr. Mohuya Byabartta Kar** for her tremendous support and assistance in the completion of my project. It was in the 7th semester final project report submission and evaluations that we received valuable feedback from **Prof. Dr. Sabhyasachi Banerjee.** The completion of the project would not have been possible without their help and insights.

The authors would like to convey their gratitude to all the professors in the Bachelor Of Technology course in the Computer Science and Engineering department of Heritage Institute Of Technology for excellent teaching that helped to partially complete the project smoothly.

Finally the authors wishes to express a heartfelt gratitude to our team members, who worked tirelessly to complete the project in the time bound manner with the great hard work and ingenuity.

Aditya Roshan Jha

Anushka Das

Moonmoon Mondal

# ABSTRACT

This project have the ultimate main to reduce psychological burden that people of these time face. At the end of the project we would have a working application platform which would be available as subsytem in the form of Web and Android (if not iOS). This is an emotion processing application which is going to identify the emotion of the user and will therefore might help to recommend list of songs, web-articles as well as videos on the basis of it. The objective is to design an efficient machine learning application by using the images of the user's facial expressions. This application first takes images of the user's face and the text messages written by the user and then applies sentiment analysis to generate the overall reaction of the user using the platform that how the assess the performance on the sentimental basis.We are exploring state of the art models in multimodal emotion recognition. We have chosen to explore textual, sound and video inputs and develop an ensemble model that gathers the information from all these sources and displays it in a clear and interpretable way.

We are naming this project in acronym of **ESAPR** (Emotional State Analyser for Pschyological Response).

# Index

*This page is*

*intentionally*

*left blank.*

# Introduction

Recognizing facial expression of human is of significance in certain domains. Based on the expression of humans we can predict the type of content which affects a human and gain insights about their face expressions when exposed to a certain content. The human facial expression consists of seven states which are angry, disgust, fear, happy, neutral, sad and surprise. Each of these facial expressions have unique features and are important in classifying them. Sentiment analysis, also referred to as opinion mining, is an approach to natural language processing (NLP) that identifies the emotional tone behind a body of text. This is a popular way for organizations to determine and categorize opinions about a product, service, or idea. It involves the use of data mining, machine learning (ML) and artificial intelligence (AI) to mine text for sentiment and subjective information.

Multimodal Emotion Recognition Multimodal Emotion Recognition is a relatively new discipline that aims to include text inputs, as well as sound and video. This field has been rising with the development of social networks that gave researchers access to a vast amount of data. Recent studies have been exploring potential metrics to measure the coherence between emotions from the different channels.

# Requirement Specifications

## Hardware

- ESAPR should capture video from the camera source, that is front camera of Mobile Device and the webcam of the PC and display the resulting findings in the form of videos, music and web-articles.
- Graphic Card of at least 8GB, RAM of at least 16GB, at least Ryzen 5 3500U or Intel i5 9$^{th}$ Gen required for the overall system to run on the local machine. As if deployed this requirement would be eliminated altogether.
- It would be great if the it is used as micro-service while being deployed on the cloud based server.

## Software

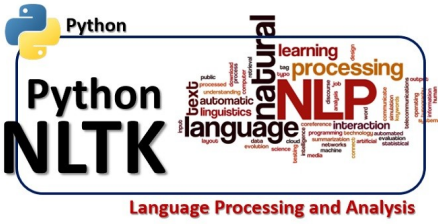
- ESAPR should be able to perform all this in the realtime. Even though the computing power of the different devices might not be good enough.
- ESAPR will use internet and cloud based approach to achieve the required results, at the same time not possibly creating hindrance to other software systems in the device.
- Some of the libraries and framework used might be legacy system, for the beta version of it, support for the legacy versions is required.

## Tools And Frameworks used till now










# Problem Definition

We are trying to provide definitions of affective computing and multimodal sentiment analysis in the context of our project. Those definitions may vary depending on the context.

**Affective Computing** Affective computing is a field of Machine Learning and Computer Science that studies the recognition and the processing of human affects.

**Multimodal Emotion Recognition** Multimodal Emotion Recognition is a relatively new discipline that aims to include text inputs, as well as sound and video. This field has been rising with the development of social networks that gave researchers access to a vast amount of data. Recent studies have been exploring potential metrics to measure the coherence between emotions from the different channels. We are going to explore several categorical targets depending on the input considered.

We are going to explore several categorical targets depending on the input considered.

| **Data Type** | **Categorical Target** |
|---|---|
| Textual | Openness, Conscientiousness, Extraversion, Agreeableness, Neuroticism |
| Sound | Happy, Sad, Angry, Fearful, Surprise, Neutral and Disgust |
| Video | Happy, Sad, Angry, Fearful, Surprise, Neutral and Disgust |

For the text inputs, we are going to focus on the so-called Big Five, widely used in personality surveys.

# Methodology

Our aim is to develop a model able to provide real time sentiment analysis with a visual user interface using Tensorow.js technology.

Therefore, we have decided to separate two types of inputs :

1. Textual input, such as answers to questions that would be asked to a person from the platform

2. Video input from a live webcam or stored from an MP4 or WAV file, from which we split the audio and the images.

# Data Sources

We have chosen to diversify the data sources we used depending on the type of data considered. All data sets used are free of charge and can be directly downloaded.

- For the text input, we are using the Stream-of-consciousness dataset that was gathered in a study by Pennebaker and King [1999]. It consists of a total of 2,468 daily writing submissions from 34 psychology students (29 women and 5 men whose ages ranged from 18 to 67 with a mean of 26.4). The writing submissions were in the form of a course unrated assignment. For each assignment, students were expected to write a minimum of 20 minutes per day about a specific topic. The data was collected during a 2-week summer course between 1993 to 1996. Each student completed their daily writing for 10 consecutive days. Students' personality scores were assessed by answering the Big Five Inventory (BFI) [John et al., 1991]. The BFI is a 44-item self-report questionnaire that provides a score for each of the five personality traits. Each item consists of short phrases and is rated using a 5-point scale that ranges from 1 (disagree strongly) to 5 (agree strongly). An instance in the data source consists of an ID, the actual essay, and five classification labels of the Big Five personality traits. Labels were originally in the form of either yes ('y') or no ('n') to indicate scoring high or low for a given trait.

- For audio data sets, we are using the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS). This database contains 7356 files (total size: 24.8 GB). The database contains 24 professional actors (12 female, 12 male), vocalizing two lexically-matched statements in a neutral North American accent. Speech includes calm, happy, sad, angry, fearful, surprise, and disgust expressions, and song contains calm, happy, sad, angry, and fearful emotions. Each expression is produced at two levels of emotional intensity(normal, strong), with an additional neutral expression. All conditions are avail-able in three modality formats: Audio-only (16bit, 48kHz .wav), Audio-Video(720p H.264, AAC 48kHz, .mp4), and Video-only (no sound)." https://zenodo.org/record/1188976#.XCx-tc9KhQI

- For the video data sets, we are using the popular FER2013 Kaggle Challenge data set. The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The data set remains quite challenging to use, since there are empty pictures, or wrongly classified images. https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

# Speech Emotion Recognition

The aim of this section is to explore speech emotion recognition techniques from an audio recording or live speech. Although in the website of our project you will only have the option for going all out live.

## Data

The data set used for training is the Ryerson Audio-Visual Database of Emotional Speech and Song: https://zenodo.org/record/1188976#.XA48aC17Q1J

RAVDESS contains 24 professional actors (12 female, 12 male), vocalizing two lexically-matched statements in a neutral North American accent. Speech includes calm, happy, sad, angry, fearful, surprise, and disgust expressions, and song contains calm, happy, sad, angry, and fearful emotions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression.

| **RAVDESS** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Emotions | Happy | Sad | Angry | Scared | Dis-gusted | Sur-prised | Neutral | **Total** |
| Man | *96* | *96* | *96* | *96* | *96* | *96* | *96* | **672** |
| Woman | *96* | *96* | *96* | *96* | *96* | *96* | *96* | **672** |
| **Total** | *192* | *192* | *192* | *192* | *192* | *192* | *192* | **1344** |

## Requirements

```
Python : 3.6.5
Scipy : 1.1.0
Scikit-learn : 0.20.1
Tensorflow : 1.12.0
Keras : 2.2.4
Numpy : 1.15.4
Librosa : 0.6.3
Pyaudio : 0.2.11
Ffmpeg : 4.0.2
```

## Notebooks

Notebooks provided for this subsystem.

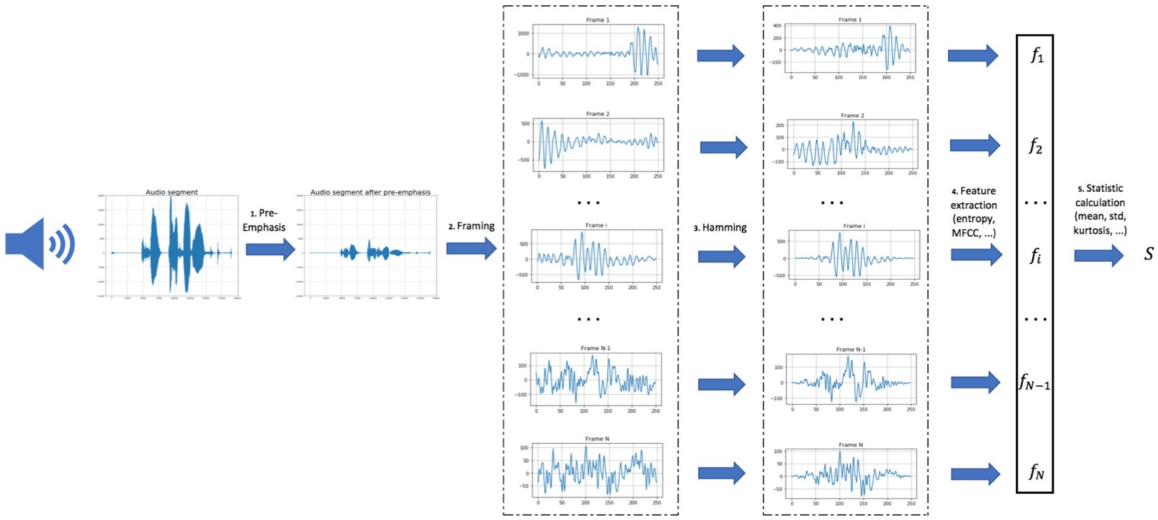
- `01 - Preprocessing[SVM].ipynb` : Signal preprocessing and feature extraction from time and frequency domain (global statistics) to train SVM classifier.
- `02 - Train [SVM].ipynb` : Implementation and training of SVM classifier for Speech Emotion Recognition

- `01 - Preprocessing[CNN-LSTM].ipynb` : Signal preprocessing and log-mel-spectrogram extraction to train TimeDistributed CNNs
- `02 - Train [CNN-LSTM].ipynb` : Implementation and training of TimeDistributed CNNs classifier for Speech Emotion Recognition

# Models

## SVM

Classical approach for Speech Emotion Recognition consists in applying a series of filters on the audio signal and partitioning it into several windows (fixed size and time-step). Then, features from time domain (Zero Crossing Rate, Energy and Entropy of Energy) and frequency domain (Spectral entropy, centroid, spread, flux, rolloff and MFCCs) are extracted for each frame. We compute then the first derivatives of each of those features to capture frame to frame changes in the signal. Finally, we calculate the following global statistics on these features: *mean, median, standard deviation, kurtosis, skewness, 1% percentile, 99% percentile, min, max* and *range* and train a simple SVM classifier with rbf kernel to predict the emotion detected in the voice.




### *SVM classification pipeline:*

1.Voice recording

2.Audio signal discretization

3.Apply pre-emphasis filter

4.Framing using a rolling window

5.Apply Hamming filter

6.Feature extraction

7.Compute global statistics

8.Make a prediction using our pre-trained model

# *01 – Preprocessing[SVM] : Signal Preprocessing*

## I. Context

The aim of this notebook is to set up all speech emotion recognition preprocessing and audio features extraction.

### *Audio features:*

The complete list of the implemented short-term features is presented below:

- **Zero Crossing Rate**: The rate of sign-changes of the signal during the duration of a particular frame.
- **Energy**: The sum of squares of the signal values, normalized by the respective frame length.
- **Entropy of Energy**: The entropy of sub-frames' normalized energies. It can be interpreted as a measure of abrupt changes.
- **Spectral Centroid**: The center of gravity of the spectrum.
- **Sprectral Spread**: The second central moment of the spectrum.
- **Spectral Entropy**: Entropy of the normalized spectral energies for a set of sub-frames.
- **Spectral Flux**: The squared difference between the normalized magnitudes of the spectra of the two successive frames.
- **Spectral Rolloff**: The frequency below which 90% of the magnitude distribution of the spectrum is concentrated.
- **MFCCS**: Mel Frequency Cepstral Coefficients form a cepstral representation where the frequency bands are not linear but distributed according to the mel-scale.

Global Statistics are then computed on upper features:

- **mean, std, med, kurt, skew, q1, q99, min, max and range**

## II. General import

```
### General imports ###
from glob import glob
import os
import pickle
import itertools
import numpy as np
```

```
### Audio preprocessing imports ###
from AudioLibrary.AudioSignal import *
from AudioLibrary.AudioFeatures import *
```

### III. Set labels

```
# RAVDESS Database
label_dict_ravdess = {'02': 'NEU', '03':'HAP', '04':'SAD', '05':'ANG', '06':'FEA', '07':'DIS', '08':'SUR'}

# Set audio files labels
def set_label_ravdess(audio_file, gender_differentiation):
    label = label_dict_ravdess.get(audio_file[6:-16])
    if gender_differentiation == True:
        if int(audio_file[18:-4])%2 == 0: # Female
            label = 'f_' + label
        if int(audio_file[18:-4])%2 == 1: # Male
            label = 'm_' + label
    return label
```

### IV. Import audio files

```
# Start feature extraction
print("Import Data: START")

# Audio file path and names
file_path = '../Datas/RAVDESS/'
file_names = os.listdir(file_path)

# Initialize signal and labels list
signal = []
labels = []

# Sample rate (44.1 kHz)
sample_rate = 44100

# Compute global statistics features for all audio file
for audio_index, audio_file in enumerate(file_names):

    # Select audio file
    if audio_file[6:-16] in label_dict_ravdess.keys():

        # Read audio file
        signal.append(AudioSignal(sample_rate, filename=file_path + audio_file))
```

```
    # Set label
    labels.append(set_label_ravdess(audio_file, True))

    # Print running...
    if (audio_index % 100 == 0):
      print("Import Data: RUNNING ... {} files".format(audio_index))

# Cast labels to array
labels = np.asarray(labels).ravel()

# Stop feature extraction
print("Import Data: END \n")
print("Number of audio files imported: {}".format(labels.shape[0]))
```

```
Import Data: START
Import Data: RUNNING ... 0 files
Import Data: RUNNING ... 200 files
Import Data: RUNNING ... 300 files
Import Data: RUNNING ... 400 files
Import Data: RUNNING ... 500 files
Import Data: RUNNING ... 600 files
Import Data: RUNNING ... 700 files
Import Data: RUNNING ... 800 files
Import Data: RUNNING ... 900 files
Import Data: RUNNING ... 1000 files
Import Data: RUNNING ... 1100 files
Import Data: RUNNING ... 1200 files
Import Data: RUNNING ... 1300 files
Import Data: RUNNING ... 1400 files
Import Data: END


Number of audio files imported: 1344
```

## V. Audio features extraction

```
# Audio features extraction function
def global_feature_statistics(y, win_size=0.025, win_step=0.01, nb_mfcc=12, mel_filter=40,
                              stats = ['mean', 'std', 'med', 'kurt', 'skew', 'q1', 'q99', 'min', 'max', 'range'],
                              features_list =  ['zcr', 'energy', 'energy_entropy', 'spectral_centroid',
'spectral_spread', 'spectral_entropy', 'spectral_flux', 'sprectral_rolloff', 'mfcc']):

  # Extract features
```

```
    audio_features = AudioFeatures(y, win_size, win_step)
    features, features_names = audio_features.global_feature_extraction(stats=stats,
features_list=features_list)
    return features

# Features extraction parameters
sample_rate = 16000 # Sample rate (16.0 kHz)
win_size = 0.025    # Short term window size (25 msec)
win_step = 0.01     # Short term window step (10 msec)
nb_mfcc = 12        # Number of MFCCs coefficients (12)
nb_filter = 40      # Number of filter banks (40)
stats = ['mean', 'std', 'med', 'kurt', 'skew', 'q1', 'q99', 'min', 'max', 'range'] # Global statistics
features_list =  ['zcr', 'energy', 'energy_entropy', 'spectral_centroid', 'spectral_spread', # Audio
features
                  'spectral_entropy', 'spectral_flux', 'sprectral_rolloff', 'mfcc']

# Start feature extraction
print("Feature extraction: START")

# Compute global feature statistics for all audio file
features = np.asarray(list(map(global_feature_statistics, signal)))

# Stop feature extraction
print("Feature extraction: END!")
```

```
Feature extraction: START
Feature extraction: END!
```

### VI. Save as

```
# Save DataFrame to pickle
pickle.dump([features, labels], open("../Datas/Pickle/[RAVDESS][HAP-SAD-NEU-ANG-FEA-DIS-
SUR][GLOBAL_STATS].p", 'wb'))
```

## *02 - Train [SVM].ipynb : SVM Classifier*

### I. Context

The aim of this notebook is to set up all speech emotion recognition preprocessing and audio features extraction.

#### *Audio features:*

The complete list of the implemented short-term features is presented below:

- **Zero Crossing Rate**: The rate of sign-changes of the signal during the duration of a particular frame.
- **Energy**: The sum of squares of the signal values, normalized by the respective frame length.
- **Entropy of Energy**: The entropy of sub-frames' normalized energies. It can be interpreted as a measure of abrupt changes.
- **Spectral Centroid**: The center of gravity of the spectrum.
- **Sprectral Spread**: The second central moment of the spectrum.
- **Spectral Entropy**: Entropy of the normalized spectral energies for a set of sub-frames.
- **Spectral Flux**: The squared difference between the normalized magnitudes of the spectra of the two successive frames.
- **Spectral Rolloff**: The frequency below which 90% of the magnitude distribution of the spectrum is concentrated.
- **MFCCS**: Mel Frequency Cepstral Coefficients form a cepstral representation where the frequency bands are not linear but distributed according to the mel-scale.

Global Statistics are then computed on upper features:

- **mean, std, med, kurt, skew, q1, q99, min, max and range**

## II. General import

```
### General imports ###
from glob import glob
import os
import pickle
import itertools
import pandas as pd
import numpy as np

### Warning import ###
import warnings
warnings.filterwarnings('ignore')

### Graph imports ###
import matplotlib.pyplot as plt

### Sklearn imports ###
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

```
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

### II. Import data

```
# Load datas from pickle
[features, labels] = pickle.load(open("../Datas/Pickle/[RAVDESS][HAP-SAD-NEU-ANG-FEA-DIS-SUR][GLOBAL_STATS].p", "rb"))
```

### III. Train and test data set

```
# Build Train and test dataset
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
random_state=123)

# Encode Label from categorical to numerical
lb = LabelEncoder()
lb.fit(y_train)
y_train, y_test = lb.transform(y_train), lb.transform(y_test)
```

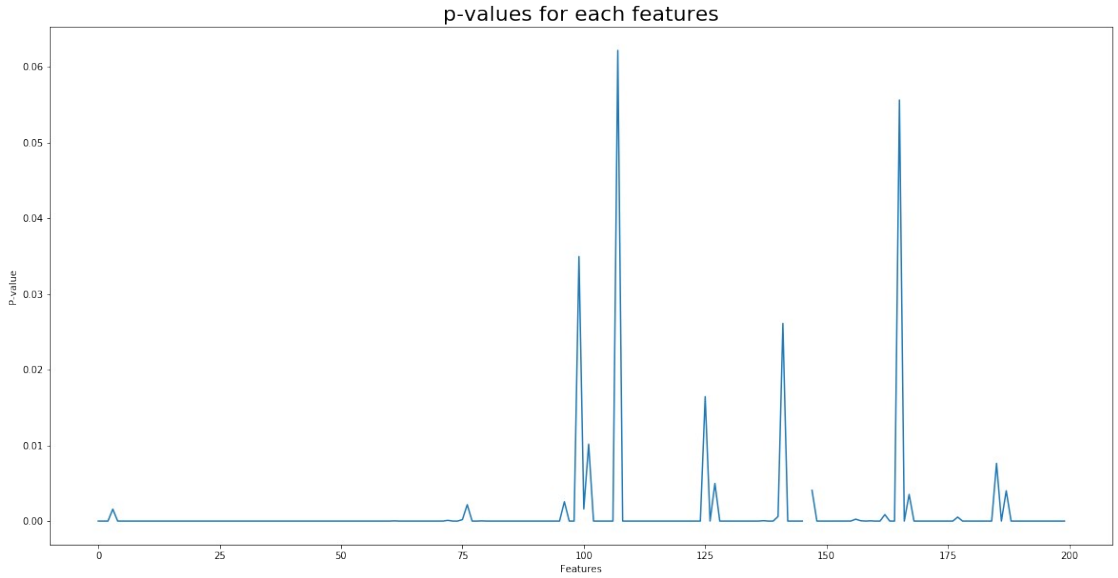
### IV. Scale features

```
# Scale train and test dataset
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

### V. Feature selection

```
# k-highest scores analysis on features
Kbest = SelectKBest(k="all")
selected_features = Kbest.fit(X_train, y_train)

# Plot P-values
plt.figure(figsize=(20, 10))
plt.plot(selected_features.pvalues_)
plt.title("p-values for each features", fontsize=22)
plt.xlabel("Features")
plt.ylabel("P-value")
plt.show()
```

```
# Display Comment
alpha = 0.01
print("Number of p-values > à 1% : {}".format(np.sum(selected_features.pvalues_ > alpha)))
```

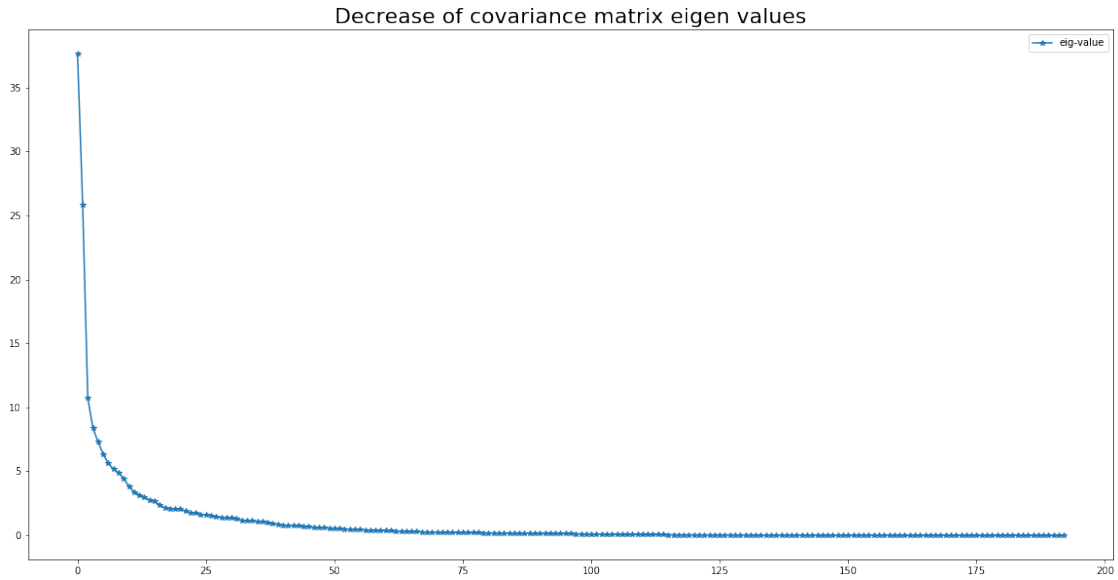



```
Number of p-values > à 1% : 6
```

```
# Remove non-significant features
X_train = X_train[:,np.where(selected_features.pvalues_ < alpha)[0]]
X_test = X_test[:,np.where(selected_features.pvalues_ < alpha)[0]]
```

## VI. Feature dimension reduction

```
# Covariance matrix
cov = pd.DataFrame(X_train).cov()

# Eigen values of covariance matrix
eig = np.linalg.svd(cov)[1]

# Plot eigen graph
fig = plt.figure(figsize=(20, 10))
plt.title('Decrease of covariance matrix eigen values', fontsize = 22)
plt.plot(eig, '-*', label = "eig-value")
plt.legend(loc = 'upper right')
plt.show()
```




```
# Initialize PCA
pca = PCA(n_components=140)

# Apply PCA on train and test set
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

## VII. Cross-Validation and hyperparameter tuning

```
# Set C and Gamma parameters list
G_list = [0.001, 0.005, 0.01]
C_list = [1, 2, 3, 4, 5, 7, 10, 20, 50]

# Set the parameters for cross-validation
parameters = [{'kernel': ['rbf'], 'C': C_list, 'gamma': G_list}]

# Initialize SVM model
model = SVC(decision_function_shape='ovr')

# Cross Validation
cv = GridSearchCV(model, parameters, cv=3, verbose=0, n_jobs=-1).fit(X_train, y_train)

# Print Best parameters
print("Best parameters set found on train set:")
print(cv.best_params_)
```

```
/anaconda3/envs/msbgd/lib/python3.6/site-packages/sklearn/model_selection/_search.py:841:
DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0.22
```

```
and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)

Best parameters set found on train set:
{'C': 3, 'gamma': 0.005, 'kernel': 'rbf'}
```

## VIII. Best model prediction

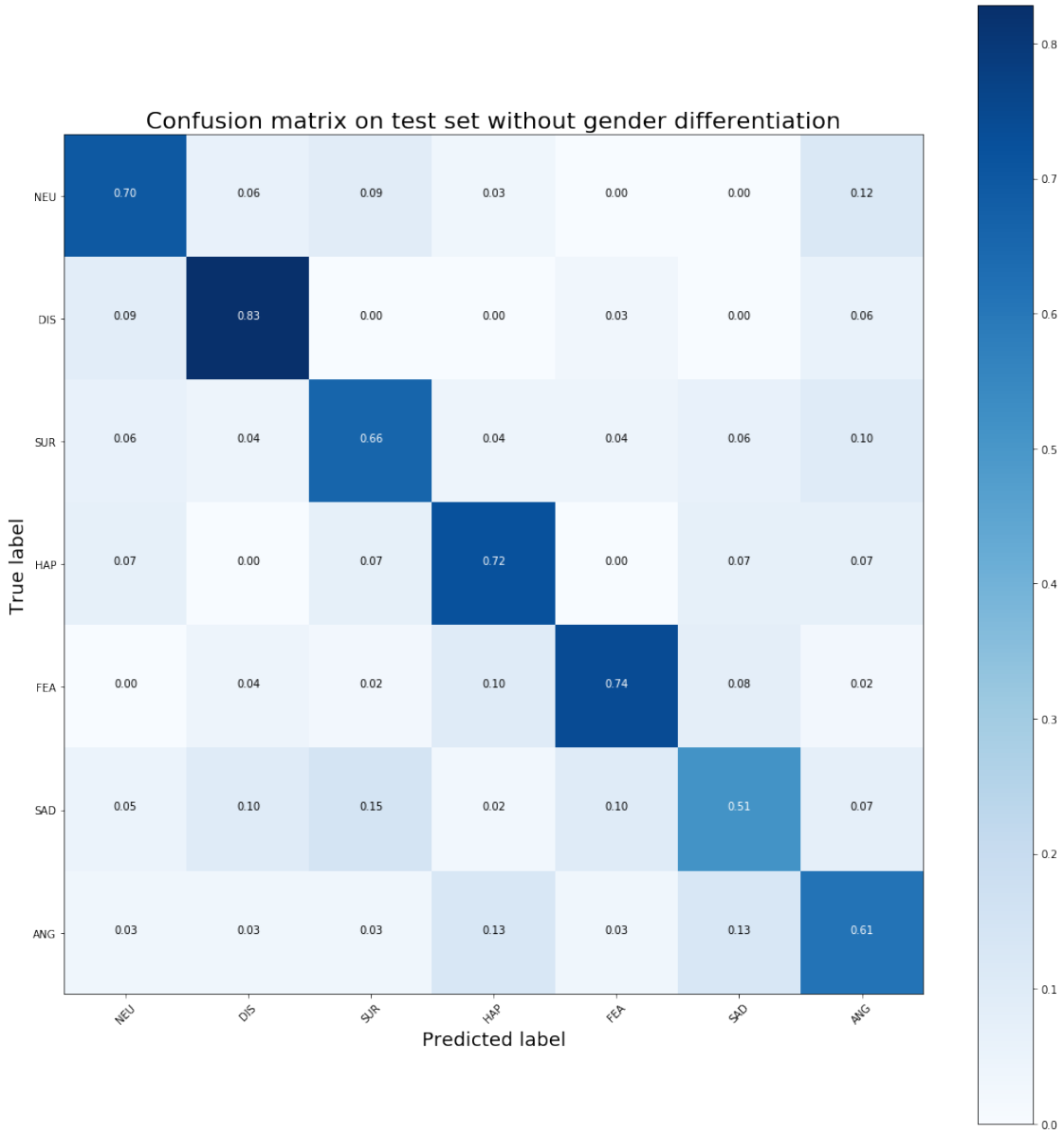
```
# Confusion matrix plot function
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=22)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label', fontsize=18)
    plt.xlabel('Predicted label', fontsize=18)
    plt.tight_layout()

# Fit best mode
model = SVC(kernel='rbf', C=3, gamma=0.005, decision_function_shape='ovr').fit(X_train, y_train)

# Prediction
pred = model.predict(X_test)

# Score
```

```
score = model.score(X_test, y_test)

# Reverse label encoder
pred = (lb.inverse_transform((pred.astype(int).flatten())))
actual = (lb.inverse_transform((y_test.astype(int).flatten())))

# Build dataFrame
df_pred = pd.DataFrame({'Actual': actual, 'Prediction': pred})

# Print Score
print('Accuracy Score on test dataset: {}%'.format(np.round(100 * score,2)))

# Compute confusion matrix
confusion = confusion_matrix(actual, pred)

# Plot non-normalized confusion matrix
plt.figure(figsize=(15, 15))
plot_confusion_matrix(confusion, classes=set(actual),normalize=True,
                      title='Confusion matrix on train set with gender differentiation')
```

```
Accuracy Score on test dataset: 66.91%
```

```
# Compute prediction without gender differentation
PRED = list(map(lambda i:i[2:], pred))
ACTUAL = list(map(lambda i:i[2:], actual))

# Compute related prediction score
SCORE = accuracy_score(ACTUAL, PRED)

# Print Score
print('Accuracy Score on test dataset: {}%'.format(np.round(100 * SCORE,2)))

# Compute confusion matrix
confusion = confusion_matrix(ACTUAL, PRED)

# Plot non-normalized confusion matrix
plt.figure(figsize=(15, 15))
plot_confusion_matrix(confusion, classes=set(ACTUAL),normalize=True,
                      title='Confusion matrix on test set without gender differentiation')
```

```
Accuracy Score on test dataset: 68.03%
```



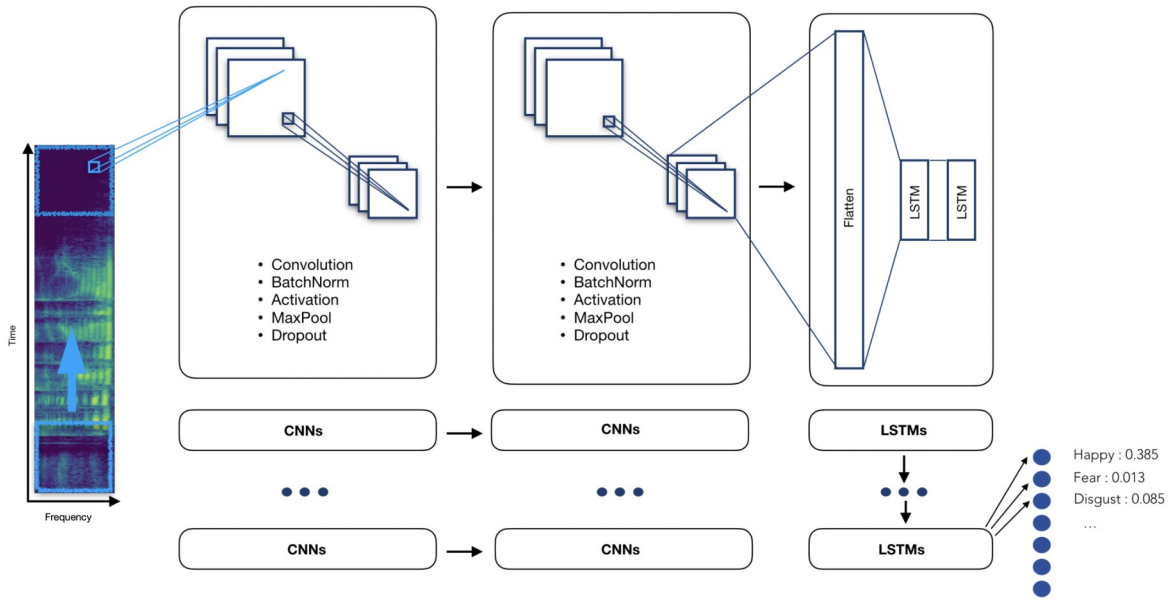

## IX. Save model

```
# save the model to local
pickle.dump(model, open('../Model/MODEL_CLASSIFIER.p', 'wb'))

# Save label encoder
pickle.dump(lb, open("../Model/MODEL_ENCODER.p", "wb"))

# Save PCA
pickle.dump(pca, open("../Model/MODEL_PCA.p", "wb"))

# Save MEAN and STD of each features
MEAN = features.mean(axis=0)
STD = features.std(axis=0)
```

```
pickle.dump([MEAN, STD], open("../Model/MODEL_SCALER.p", "wb"))

# Save feature parameters
stats = ['mean', 'std', 'kurt', 'skew', 'q1', 'q99']
features_list = ['zcr', 'energy', 'energy_entropy', 'spectral_centroid', 'spectral_spread',
'spectral_entropy', 'spectral_flux', 'sprectral_rolloff']
win_step = 0.01
win_size = 0.025
nb_mfcc = 12
diff = 0
PCA = True
DICO = {'stats':stats, 'features_list':features_list, 'win_size':win_size, 'win_step':win_step,
'nb_mfcc':nb_mfcc, 'diff':diff, 'PCA':PCA}
pickle.dump(DICO, open("../Model/MODEL_PARAM.p", "wb"))
```

## TimeDistributed CNNs

The main idea of a Time Distributed Convolutional Neural Network is to apply a rolling window (fixed size and time-step) all along the log-mel-spectrogram. Each of these windows will be the entry of a convolutional neural network, composed by four Local Feature Learning Blocks (LFLBs) and the output of each of these convolutional networks will be fed into a recurrent neural network composed by 2 cells LSTM (Long Short Term Memory) to learn the long-term contextual dependencies. Finally, a fully connected layer with *softmax* activation is used to predict the emotion detected in the voice.




## *TimeDistributed CNNs pipeline:*

1. Voice recording
2. Audio signal discretization
3. Log-mel-spectrogram extraction
4. Split spectrogram with a rolling window
5. Make a prediction using our pre-trained model

## *01 – Preprocessing[CNN-LSTM].ipynb : Signal Preprocessing*

### I. Context

The aim of this notebook is to set up all speech emotion recognition preprocessing for the time distributed ConvNet.

The signal preprocessing include :

- Signal discretization
- Audio data augmentation
- Log-mel-spectrogram extraction
- Time distributed framing
- Build train and test data set

### II. General import

```
### General imports ###
import os
from glob import glob
import pickle
import itertools
import numpy as np
from scipy.stats import zscore
from sklearn.model_selection import train_test_split

### Graph imports ###
import matplotlib.pyplot as plt
from PIL import Image

### Audio import ###
import librosa
import IPython
from IPython.display import Audio
```

### III. Set labels

```
# RAVDESS Database
label_dict_ravdess = {'02': 'NEU', '03':'HAP', '04':'SAD', '05':'ANG', '06':'FEA', '07':'DIS', '08':'SUR'}

# Set audio files labels
def set_label_ravdess(audio_file, gender_differentiation):
    label = label_dict_ravdess.get(audio_file[6:-16])
    if gender_differentiation == True:
        if int(audio_file[18:-4])%2 == 0: # Female
            label = 'f_' + label
        if int(audio_file[18:-4])%2 == 1: # Male
            label = 'm_' + label
    return label
```

IV. Import audio files

```
# Start feature extraction
print("Import Data: START")

# Audio file path and names
file_path = '../Datas/RAVDESS/'
file_names = os.listdir(file_path)

# Initialize features and labels list
signal = []
```

```
labels = []

# Sample rate (16.0 kHz)
sample_rate = 16000

# Max pad lenght (3.0 sec)
max_pad_len = 49100

# Compute spectogram for all audio file
for audio_index, audio_file in enumerate(file_names):

   if audio_file[6:-16] in list(label_dict_ravdess.keys()):

      # Read audio file
      y, sr = librosa.core.load(file_path + audio_file, sr=sample_rate, offset=0.5)

      # Z-normalization
      y = zscore(y)

      # Padding or truncated signal
      if len(y) < max_pad_len:
         y_padded = np.zeros(max_pad_len)
         y_padded[:len(y)] = y
         y = y_padded
      elif len(y) > max_pad_len:
         y = np.asarray(y[:max_pad_len])

      # Add to signal list
      signal.append(y)

      # Set label
      labels.append(set_label_ravdess(audio_file, False))

      # Print running...
      if (audio_index % 100 == 0):
         print("Import Data: RUNNING ... {} files".format(audio_index))

# Cast labels to array
labels = np.asarray(labels).ravel()

# Stop feature extraction
print("Import Data: END \n")
print("Number of audio files imported: {}".format(labels.shape[0]))
```

```
# Select one random audio file
random_idx = np.random.randint(len(labels))
random_label = labels[random_idx]
random_signal = signal[random_idx]
random_filename = file_names[random_idx]

# Plot signal wave
plt.figure(figsize=(10,5))
plt.plot(np.arange(len(random_signal))/float(sample_rate), random_signal)
plt.xlim((np.arange(len(random_signal))/float(sample_rate))[0],
(np.arange(len(random_signal))/float(sample_rate))[-1])
plt.xlabel('Time (s)', fontsize=16)
plt.ylabel('Amplitude (dB)', fontsize=16)
plt.title("Signal wave of file '{}' with label {}".format(random_filename, random_label),
fontsize=18)
plt.show()

# Play audio file
print("Audio file '{}':".format(random_filename))
Audio(random_signal, rate=sample_rate)
```

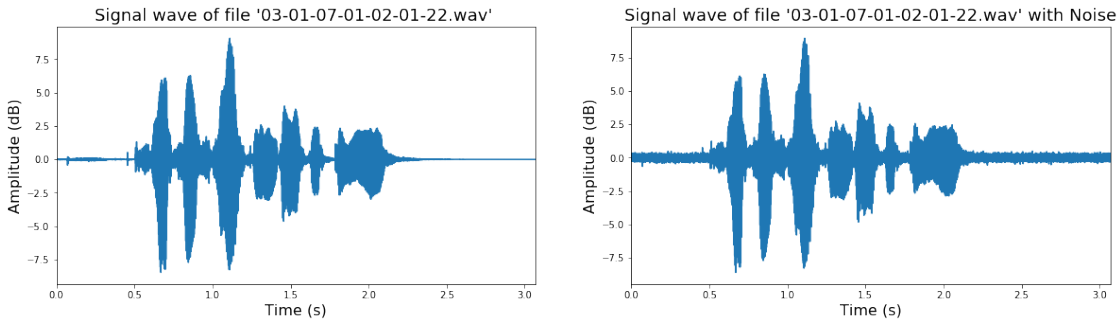
## V. Audio data augmentation

```
# Number of augmented data
nb_augmented = 2

# Function to add noise to a signals with a desired Signal Noise ratio (SNR)
def noisy_signal(signal, snr_low=15, snr_high=30, nb_augmented=2):

    # Signal length
    signal_len = len(signal)

    # Generate White noise
    noise = np.random.normal(size=(nb_augmented, signal_len))

    # Compute signal and noise power
    s_power = np.sum((signal / (2.0 ** 15)) ** 2) / signal_len
    n_power = np.sum((noise / (2.0 ** 15)) ** 2, axis=1) / signal_len

    # Random SNR: Uniform [15, 30]
    snr = np.random.randint(snr_low, snr_high)

    # Compute K coeff for each noise
    K = np.sqrt((s_power / n_power) * 10 ** (- snr / 10))
```

```
    K = np.ones((signal_len, nb_augmented)) * K

    # Generate noisy signal
    return signal + K.T * noise

# Generate noisy signals from signal list
print("Data Augmentation: START")
augmented_signal = list(map(noisy_signal, signal))
print("Data Augmentation: END!")
```

```
Data Augmentation: START
Data Augmentation: END!
```

```
# Plot signal wave
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.plot(np.arange(len(random_signal))/float(sample_rate), random_signal)
plt.xlim((np.arange(len(random_signal))/float(sample_rate))[0],
(np.arange(len(random_signal))/float(sample_rate))[-1])
plt.xlabel('Time (s)', fontsize=16)
plt.ylabel('Amplitude (dB)', fontsize=16)
plt.title("Signal wave of file '{}' ".format(random_filename), fontsize=18)

# Plot signal wave with noise
plt.subplot(1,2,2)
plt.plot(np.arange(len(random_signal))/float(sample_rate), augmented_signal[random_idx][0])
plt.xlim((np.arange(len(random_signal))/float(sample_rate))[0],
(np.arange(len(random_signal))/float(sample_rate))[-1])
plt.xlabel('Time (s)', fontsize=16)
plt.ylabel('Amplitude (dB)', fontsize=16)
plt.title("Signal wave of file '{}' with Noise".format(random_filename), fontsize=18)
plt.show()

# Play audio file
print("Audio file '{}':".format(random_filename))
IPython.display.display(Audio(random_signal, rate=sample_rate))

# Play same audio file with noise
print("Audio file '{}' with noise:".format(random_filename))
IPython.display.display(Audio(augmented_signal[random_idx][0], rate=sample_rate))
```

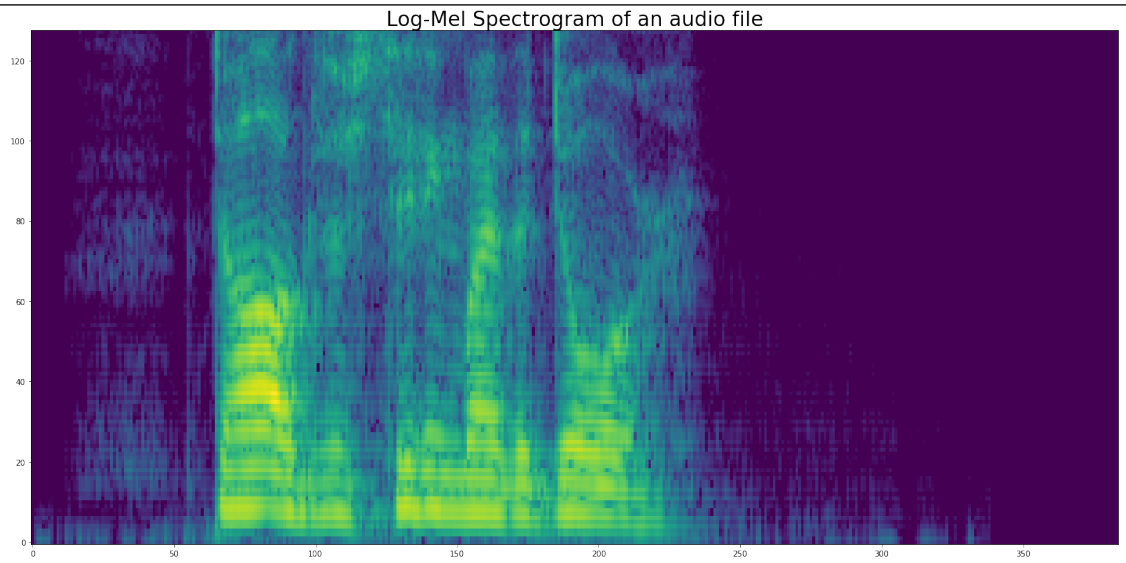



```
Audio file '03-01-07-01-02-01-22.wav':
<IPython.lib.display.Audio object>
Audio file '03-01-07-01-02-01-22.wav' with noise:
<IPython.lib.display.Audio object>
```

## VI. Feature extraction

```
def mel_spectrogram(y, sr=16000, n_fft=512, win_length=256, hop_length=128,
window='hamming', n_mels=128, fmax=4000):

    # Compute spectogram
    mel_spect = np.abs(librosa.stft(y, n_fft=n_fft, window=window, win_length=win_length,
hop_length=hop_length)) ** 2

    # Compute mel spectrogram
    mel_spect = librosa.feature.melspectrogram(S=mel_spect, sr=sr, n_mels=n_mels, fmax=fmax)

    # Compute log-mel spectrogram
    mel_spect = librosa.power_to_db(mel_spect, ref=np.max)

    return mel_spect

# Start feature extraction
print("Feature extraction: START")

# Compute spectogram for all audio file
mel_spect = np.asarray(list(map(mel_spectrogram, signal)))
augmented_mel_spect = [np.asarray(list(map(mel_spectrogram, augmented_signal[i]))) for i in
range(len(augmented_signal))]

# Stop feature extraction
print("Feature extraction: END!")

Feature extraction: START
```

```
Feature extraction: END!

# Plot one random Spectogram
plt.figure(figsize=(20, 10))
plt.imshow(mel_spect[np.random.randint(len(mel_spect))], origin='lower', aspect='auto',
cmap='viridis')
plt.title('Log-Mel Spectrogram of an audio file', fontsize=26)
plt.tight_layout()
plt.show()
```




## VII. Train and test set

```
# Build Train and test dataset
MEL_SPECT_train, MEL_SPECT_test, AUG_MEL_SPECT_train, AUG_MEL_SPECT_test, label_train,
label_test = train_test_split(mel_spect, augmented_mel_spect, labels, test_size=0.2)

# Build augmented labels and train
aug_label_train = np.asarray(list(itertools.chain.from_iterable([[label] * nb_augmented for label
in label_train])))
AUG_MEL_SPECT_train = np.asarray(list(itertools.chain.from_iterable(AUG_MEL_SPECT_train)))

# Concatenate original and augmented
X_train = np.concatenate((MEL_SPECT_train, AUG_MEL_SPECT_train))
y_train = np.concatenate((label_train, aug_label_train))

# Build test set
X_test = MEL_SPECT_test
y_test = label_test

# Delete
del MEL_SPECT_train, AUG_MEL_SPECT_train, label_train, aug_label_train,
```

```
AUG_MEL_SPECT_test, MEL_SPECT_test, label_test
del mel_spect, augmented_mel_spect, labels
```

### VIII. Time distributed framing

```
# Time distributed parameters
win_ts = 128
hop_ts = 64

# Split spectrogram into frames
def frame(x, win_step=128, win_size=64):
    nb_frames = 1 + int((x.shape[2] - win_size) / win_step)
    frames = np.zeros((x.shape[0], nb_frames, x.shape[1], win_size)).astype(np.float32)
    for t in range(nb_frames):
        frames[:,t,:,:] = np.copy(x[:,:,(t * win_step):(t * win_step + win_size)]).astype(np.float32)
    return frames

# Frame for TimeDistributed model
X_train = frame(X_train, hop_ts, win_ts)
X_test = frame(X_test, hop_ts, win_ts)
```

### IX. Save as

```
# Save Train and test set
pickle.dump(X_train.astype(np.float16), open('../Datas/Pickle/RAVDESS/DIS/[RAVDESS][MEL_SPECT][X_train].p', 'wb'))
pickle.dump(y_train, open('../Datas/Pickle/RAVDESS/DIS/[RAVDESS][MEL_SPECT][y_train].p', 'wb'))
pickle.dump(X_test.astype(np.float16), open('../Datas/Pickle/RAVDESS/DIS/[RAVDESS][MEL_SPECT][X_test].p', 'wb'))
pickle.dump(y_test, open('../Datas/Pickle/RAVDESS/DIS/[RAVDESS][MEL_SPECT][y_test].p', 'wb'))
```

## *02 - Train [CNN-LSTM].ipynb : Time Distributed ConvNet*

### I. Context

In this project I build a prevision model using deep learning combining **CNN** and **LSTM** to detect a person's emotions (HAPPY, SAD, FEAR, ANGRY, DISGUST, SURPRISE, NEUTRAL) just by their voice.

### *Audio features:*

- **Log-mel-spectrogram**: the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency

## II. General Imports

```
### General imports ###
import os
from glob import glob
import pickle
import numpy as np

### Plot imports ###
from IPython.display import Image
import matplotlib.pyplot as plt

### Time Distributed ConvNet imports ###
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Dropout, Activation, TimeDistributed,
concatenate
from tensorflow.keras.layers import Conv2D, MaxPooling2D, AveragePooling2D,
BatchNormalization, LeakyReLU, Flatten
from tensorflow.keras.layers import LSTM
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras import backend as K
from keras.utils import np_utils
from keras.utils import plot_model
from sklearn.preprocessing import LabelEncoder

### Warning ###
import warnings
warnings.filterwarnings('ignore')
```

```
Using TensorFlow backend.
```

```
# Connect Colab to google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

### III. Import datas

```
# RAVDESS mel-Spectrogram
X_train = pickle.load(open('drive/My Drive/SpeechEmotionRecognition/DIS/[RAVDESS]
[MEL_SPECT][X_train].p', 'rb'))
y_train = pickle.load(open('drive/My Drive/SpeechEmotionRecognition/DIS/[RAVDESS]
[MEL_SPECT][y_train].p', 'rb'))
y_test = pickle.load(open('drive/My Drive/SpeechEmotionRecognition/DIS/[RAVDESS]
[MEL_SPECT][y_test].p', 'rb'))
X_test = pickle.load(open('drive/My Drive/SpeechEmotionRecognition/DIS/[RAVDESS]
[MEL_SPECT][X_test].p', 'rb'))
```

### IV. Encode label

```
# Encode Label from categorical to numerical
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(np.ravel(y_train)))
y_test = np_utils.to_categorical(lb.transform(np.ravel(y_test)))
```

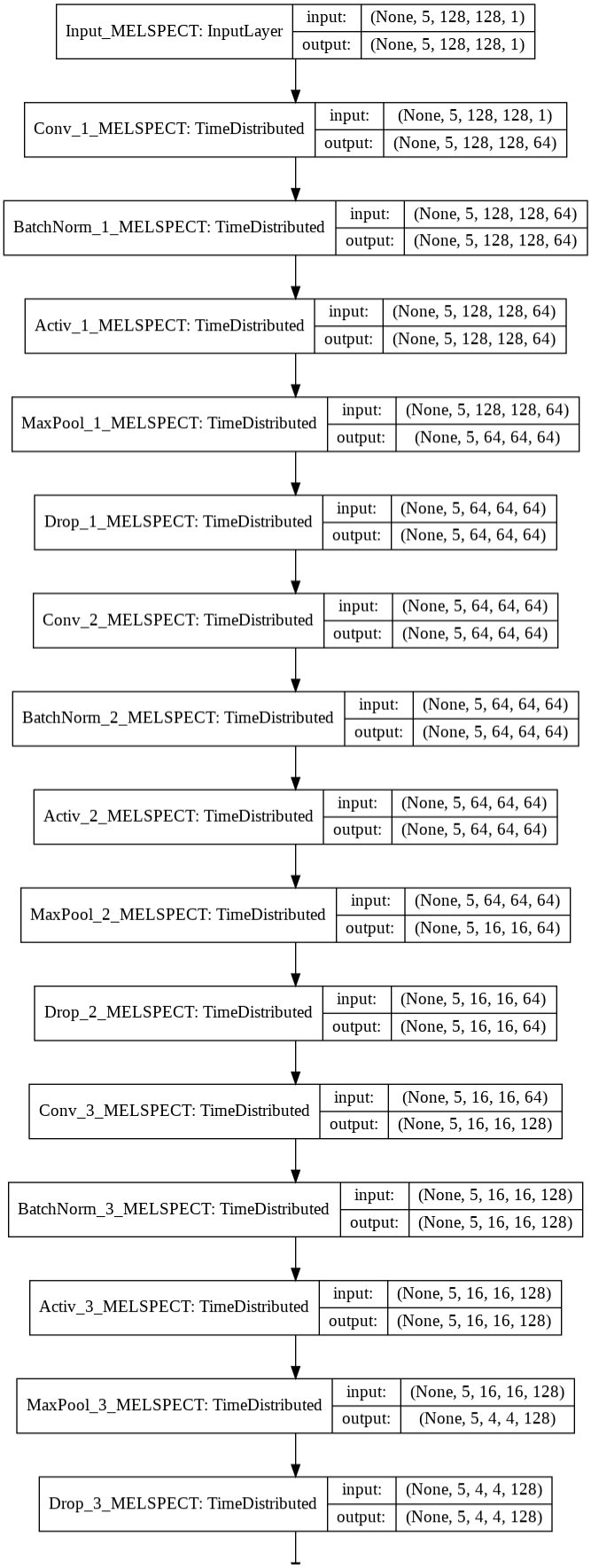
### V. Reshape train and test set

```
# Reshape for convolution
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1] , X_train.shape[2], X_train.shape[3],
1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1] , X_test.shape[2], X_test.shape[3], 1)
```

### VI. Time Distributed ConvNet model

```
K.clear_session()

# Define two sets of inputs: MFCC and FBANK
input_y = Input(shape=X_train.shape[1:], name='Input_MELSPECT')

## First LFLB (local feature learning block)
y = TimeDistributed(Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='same'),
name='Conv_1_MELSPECT')(input_y)
y = TimeDistributed(BatchNormalization(), name='BatchNorm_1_MELSPECT')(y)
y = TimeDistributed(Activation('elu'), name='Activ_1_MELSPECT')(y)
y = TimeDistributed(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'),
name='MaxPool_1_MELSPECT')(y)
y = TimeDistributed(Dropout(0.2), name='Drop_1_MELSPECT')(y)

## Second LFLB (local feature learning block)
```

```
y = TimeDistributed(Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='same'),
name='Conv_2_MELSPECT')(y)
y = TimeDistributed(BatchNormalization(), name='BatchNorm_2_MELSPECT')(y)
y = TimeDistributed(Activation('elu'), name='Activ_2_MELSPECT')(y)
y = TimeDistributed(MaxPooling2D(pool_size=(4, 4), strides=(4, 4), padding='same'),
name='MaxPool_2_MELSPECT')(y)
y = TimeDistributed(Dropout(0.2), name='Drop_2_MELSPECT')(y)

## Second LFLB (local feature learning block)
y = TimeDistributed(Conv2D(128, kernel_size=(3, 3), strides=(1, 1), padding='same'),
name='Conv_3_MELSPECT')(y)
y = TimeDistributed(BatchNormalization(), name='BatchNorm_3_MELSPECT')(y)
y = TimeDistributed(Activation('elu'), name='Activ_3_MELSPECT')(y)
y = TimeDistributed(MaxPooling2D(pool_size=(4, 4), strides=(4, 4), padding='same'),
name='MaxPool_3_MELSPECT')(y)
y = TimeDistributed(Dropout(0.2), name='Drop_3_MELSPECT')(y)

## Second LFLB (local feature learning block)
y = TimeDistributed(Conv2D(128, kernel_size=(3, 3), strides=(1, 1), padding='same'),
name='Conv_4_MELSPECT')(y)
y = TimeDistributed(BatchNormalization(), name='BatchNorm_4_MELSPECT')(y)
y = TimeDistributed(Activation('elu'), name='Activ_4_MELSPECT')(y)
y = TimeDistributed(MaxPooling2D(pool_size=(4, 4), strides=(4, 4), padding='same'),
name='MaxPool_4_MELSPECT')(y)
y = TimeDistributed(Dropout(0.2), name='Drop_4_MELSPECT')(y)

## Flat
y = TimeDistributed(Flatten(), name='Flat_MELSPECT')(y)

# Apply 2 LSTM layer and one FC
y = LSTM(256, return_sequences=False, dropout=0.2, name='LSTM_1')(y)
y = Dense(y_train.shape[1], activation='softmax', name='FC')(y)

# Build final model
model = Model(inputs=input_y, outputs=y)

# Plot model graph
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
Image(retina=True, filename='model.png')
```


Input_MELSPECT: InputLayer
input: (None, 5, 128, 128, 1)
output: (None, 5, 128, 128, 1)
Conv_1_MELSPECT: TimeDistributed
input: (None, 5, 128, 128, 1)
output: (None, 5, 128, 128, 64)
BatchNorm_1_MELSPECT: TimeDistributed
input: (None, 5, 128, 128, 64)
output: (None, 5, 128, 128, 64)
Activ_1_MELSPECT: TimeDistributed
input: (None, 5, 128, 128, 64)
output: (None, 5, 128, 128, 64)
MaxPool_1_MELSPECT: TimeDistributed
input: (None, 5, 128, 128, 64)
output: (None, 5, 64, 64, 64)
Drop_1_MELSPECT: TimeDistributed
input: (None, 5, 64, 64, 64)
output: (None, 5, 64, 64, 64)
Conv_2_MELSPECT: TimeDistributed
input: (None, 5, 64, 64, 64)
output: (None, 5, 64, 64, 64)
BatchNorm_2_MELSPECT: TimeDistributed
input: (None, 5, 64, 64, 64)
output: (None, 5, 64, 64, 64)
Activ_2_MELSPECT: TimeDistributed
input: (None, 5, 64, 64, 64)
output: (None, 5, 64, 64, 64)
MaxPool_2_MELSPECT: TimeDistributed
input: (None, 5, 64, 64, 64)
output: (None, 5, 16, 16, 64)
Drop_2_MELSPECT: TimeDistributed
input: (None, 5, 16, 16, 64)
output: (None, 5, 16, 16, 64)
Conv_3_MELSPECT: TimeDistributed
input: (None, 5, 16, 16, 64)
output: (None, 5, 16, 16, 128)
BatchNorm_3_MELSPECT: TimeDistributed
input: (None, 5, 16, 16, 128)
output: (None, 5, 16, 16, 128)
Activ_3_MELSPECT: TimeDistributed
input: (None, 5, 16, 16, 128)
output: (None, 5, 16, 16, 128)
MaxPool_3_MELSPECT: TimeDistributed
input: (None, 5, 16, 16, 128)
output: (None, 5, 4, 4, 128)
Drop_3_MELSPECT: TimeDistributed
input: (None, 5, 4, 4, 128)
output: (None, 5, 4, 4, 128)

```
# Compile model
model.compile(optimizer=SGD(lr=0.01, decay=1e-6, momentum=0.8),
loss='categorical_crossentropy', metrics=['accuracy'])

# Save best model
best_model_save = ModelCheckpoint('drive/My Drive/SpeechEmotionRecognition/[CNN-
LSTM]Model.hdf5', save_best_only=True, monitor='val_acc', mode='max')

# Early stopping
early_stopping = EarlyStopping(monitor='val_acc', patience=30, verbose=1, mode='max')

# Fit model
history = model.fit(X_train, y_train, batch_size=64, epochs=100, validation_data=(X_test, y_test),
callbacks=[early_stopping, best_model_save])
```

```
Train on 3225 samples, validate on 269 samples


Epoch 1/100
3225/3225 [==============================] - 37s 11ms/sample - loss: 1.9524 - acc: 0.1696 -
val_loss: 2.1361 - val_acc: 0.1450
Epoch 2/100
3225/3225 [==============================] - 30s 9ms/sample - loss: 1.8779 - acc: 0.2214 -
val_loss: 2.1104 - val_acc: 0.1487
Epoch 3/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.8024 - acc: 0.2766 -
val_loss: 1.8661 - val_acc: 0.2193
Epoch 4/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.7169 - acc: 0.3104 -
val_loss: 1.7818 - val_acc: 0.2788
Epoch 5/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.6603 - acc: 0.3265 -
val_loss: 1.6949 - val_acc: 0.3383
Epoch 6/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.6419 - acc: 0.3417 -
val_loss: 1.6796 - val_acc: 0.3383
Epoch 7/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.6104 - acc: 0.3510 -
val_loss: 1.5527 - val_acc: 0.4015
Epoch 8/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.5826 - acc: 0.3535 -
val_loss: 1.5243 - val_acc: 0.4349
Epoch 9/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.5501 - acc: 0.3913 -
val_loss: 1.5301 - val_acc: 0.4089
```

```
Epoch 10/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.5461 - acc: 0.3749 - val_loss: 1.4289 - val_acc: 0.4796
Epoch 11/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.4926 - acc: 0.4121 - val_loss: 1.4147 - val_acc: 0.4721
Epoch 12/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.4811 - acc: 0.4164 - val_loss: 1.3705 - val_acc: 0.5056
Epoch 13/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.4332 - acc: 0.4412 - val_loss: 1.3821 - val_acc: 0.4796
Epoch 14/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.4155 - acc: 0.4484 - val_loss: 2.4937 - val_acc: 0.2082
Epoch 15/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.4054 - acc: 0.4592 - val_loss: 1.5515 - val_acc: 0.3717
Epoch 16/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.3854 - acc: 0.4546 - val_loss: 1.6477 - val_acc: 0.3569
Epoch 17/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.3472 - acc: 0.4766 - val_loss: 1.3990 - val_acc: 0.4387
Epoch 18/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.3309 - acc: 0.4828 - val_loss: 1.4600 - val_acc: 0.4424
Epoch 19/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.2941 - acc: 0.4995 - val_loss: 1.3891 - val_acc: 0.4721
Epoch 20/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.2626 - acc: 0.5172 - val_loss: 1.5036 - val_acc: 0.4238
Epoch 21/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.2516 - acc: 0.5178 - val_loss: 1.6935 - val_acc: 0.3643
Epoch 22/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.2306 - acc: 0.5181 - val_loss: 1.4394 - val_acc: 0.4535
Epoch 23/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.2173 - acc: 0.5392 - val_loss: 3.4323 - val_acc: 0.2045
Epoch 24/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.1985 - acc: 0.5457 - val_loss: 2.8250 - val_acc: 0.2379
```

```
Epoch 25/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.1653 - acc: 0.5634 - val_loss: 1.9887 - val_acc: 0.3383
Epoch 26/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.1478 - acc: 0.5690 - val_loss: 1.5294 - val_acc: 0.4424
Epoch 27/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.1271 - acc: 0.5671 - val_loss: 1.2171 - val_acc: 0.5316
Epoch 28/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.1025 - acc: 0.5811 - val_loss: 1.7088 - val_acc: 0.3903
Epoch 29/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.0751 - acc: 0.5873 - val_loss: 2.3175 - val_acc: 0.3532
Epoch 30/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.0716 - acc: 0.5870 - val_loss: 2.0873 - val_acc: 0.3903
Epoch 31/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.0254 - acc: 0.6096 - val_loss: 1.2083 - val_acc: 0.5428
Epoch 32/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.0450 - acc: 0.5904 - val_loss: 1.7128 - val_acc: 0.4275
Epoch 33/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 1.0025 - acc: 0.6127 - val_loss: 1.5275 - val_acc: 0.4610
Epoch 34/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.9902 - acc: 0.6233 - val_loss: 1.4497 - val_acc: 0.4870
Epoch 35/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.9712 - acc: 0.6341 - val_loss: 1.7275 - val_acc: 0.4424
Epoch 36/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.9540 - acc: 0.6363 - val_loss: 1.8800 - val_acc: 0.4089
Epoch 37/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.9230 - acc: 0.6391 - val_loss: 1.4873 - val_acc: 0.4944
Epoch 38/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.9472 - acc: 0.6301 - val_loss: 1.3540 - val_acc: 0.4870
Epoch 39/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.9102 - acc: 0.6636 - val_loss: 1.1557 - val_acc: 0.5428
```

```
Epoch 40/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.9058 - acc: 0.6620 - val_loss: 1.0370 - val_acc: 0.6208
Epoch 41/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.8923 - acc: 0.6540 - val_loss: 1.5213 - val_acc: 0.4870
Epoch 42/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.8910 - acc: 0.6546 - val_loss: 1.4529 - val_acc: 0.5130
Epoch 43/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.8721 - acc: 0.6629 - val_loss: 2.1284 - val_acc: 0.4089
Epoch 44/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.8613 - acc: 0.6747 - val_loss: 1.3738 - val_acc: 0.4944
Epoch 45/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.8357 - acc: 0.6797 - val_loss: 1.3848 - val_acc: 0.4944
Epoch 46/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.8435 - acc: 0.6890 - val_loss: 1.4735 - val_acc: 0.5204
Epoch 47/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.8248 - acc: 0.6884 - val_loss: 1.2802 - val_acc: 0.5279
Epoch 48/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7974 - acc: 0.7011 - val_loss: 1.0331 - val_acc: 0.6283
Epoch 49/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7802 - acc: 0.6974 - val_loss: 1.1580 - val_acc: 0.5688
Epoch 50/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7834 - acc: 0.7011 - val_loss: 1.8445 - val_acc: 0.4387
Epoch 51/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7599 - acc: 0.7126 - val_loss: 1.0996 - val_acc: 0.5948
Epoch 52/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7470 - acc: 0.7098 - val_loss: 1.4826 - val_acc: 0.5353
Epoch 53/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7485 - acc: 0.7122 - val_loss: 1.1952 - val_acc: 0.5911
Epoch 54/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7298 - acc: 0.7216 - val_loss: 1.2548 - val_acc: 0.5651
```

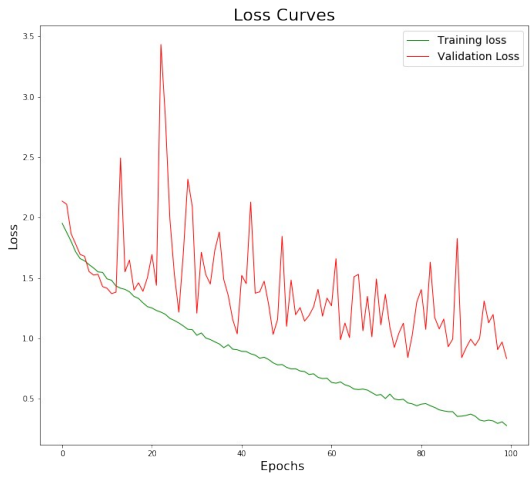
```
Epoch 55/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7251 - acc: 0.7281 -
val_loss: 1.1428 - val_acc: 0.5948
Epoch 56/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7008 - acc: 0.7343 -
val_loss: 1.1895 - val_acc: 0.5613
Epoch 57/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.7066 - acc: 0.7395 -
val_loss: 1.2597 - val_acc: 0.5799
Epoch 58/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.6770 - acc: 0.7355 -
val_loss: 1.4058 - val_acc: 0.5390
Epoch 59/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.6671 - acc: 0.7572 -
val_loss: 1.1838 - val_acc: 0.5651
Epoch 60/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.6699 - acc: 0.7411 -
val_loss: 1.3331 - val_acc: 0.5316
Epoch 61/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.6352 - acc: 0.7569 -
val_loss: 1.2696 - val_acc: 0.5651
Epoch 62/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.6287 - acc: 0.7709 -
val_loss: 1.6598 - val_acc: 0.4981
Epoch 63/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.6392 - acc: 0.7591 -
val_loss: 0.9891 - val_acc: 0.6468
Epoch 64/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.6152 - acc: 0.7687 -
val_loss: 1.1269 - val_acc: 0.6394
Epoch 65/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.6040 - acc: 0.7684 -
val_loss: 1.0064 - val_acc: 0.6580
Epoch 66/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.5803 - acc: 0.7802 -
val_loss: 1.5092 - val_acc: 0.5651
Epoch 67/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.5761 - acc: 0.7857 -
val_loss: 1.5301 - val_acc: 0.5316
Epoch 68/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.5802 - acc: 0.7777 -
val_loss: 1.0629 - val_acc: 0.6506
Epoch 69/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.5717 - acc: 0.7935 -
val_loss: 1.3473 - val_acc: 0.5948
```

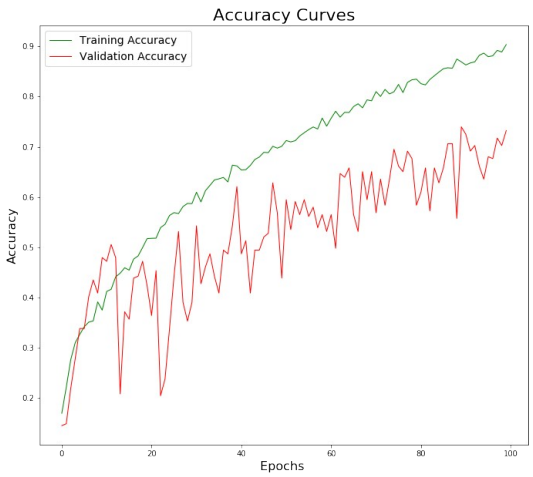
```
Epoch 70/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.5501 - acc: 0.7916 - val_loss: 1.0127 - val_acc: 0.6506
Epoch 71/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.5291 - acc: 0.8099 - val_loss: 1.4916 - val_acc: 0.5688
Epoch 72/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.5353 - acc: 0.8003 - val_loss: 1.1124 - val_acc: 0.6357
Epoch 73/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.5016 - acc: 0.8143 - val_loss: 1.3648 - val_acc: 0.5836
Epoch 74/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.5388 - acc: 0.8053 - val_loss: 1.0929 - val_acc: 0.6357
Epoch 75/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4979 - acc: 0.8096 - val_loss: 0.9247 - val_acc: 0.6952
Epoch 76/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4908 - acc: 0.8242 - val_loss: 1.0424 - val_acc: 0.6617
Epoch 77/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4958 - acc: 0.8081 - val_loss: 1.1255 - val_acc: 0.6506
Epoch 78/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4653 - acc: 0.8282 - val_loss: 0.8413 - val_acc: 0.6914
Epoch 79/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4575 - acc: 0.8335 - val_loss: 1.0298 - val_acc: 0.6766
Epoch 80/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4418 - acc: 0.8347 - val_loss: 1.3017 - val_acc: 0.5836
Epoch 81/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4556 - acc: 0.8257 - val_loss: 1.4036 - val_acc: 0.6097
Epoch 82/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4600 - acc: 0.8233 - val_loss: 1.0746 - val_acc: 0.6580
Epoch 83/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4423 - acc: 0.8341 - val_loss: 1.6310 - val_acc: 0.5725
Epoch 84/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4277 - acc: 0.8416 - val_loss: 1.1718 - val_acc: 0.6580
```

```
Epoch 85/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.4082 - acc: 0.8487 - val_loss: 1.0786 - val_acc: 0.6283
Epoch 86/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3996 - acc: 0.8555 - val_loss: 1.1600 - val_acc: 0.6580
Epoch 87/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3932 - acc: 0.8574 - val_loss: 0.9307 - val_acc: 0.7063
Epoch 88/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3929 - acc: 0.8564 - val_loss: 0.9941 - val_acc: 0.7063
Epoch 89/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3540 - acc: 0.8747 - val_loss: 1.8265 - val_acc: 0.5576
Epoch 90/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3560 - acc: 0.8691 - val_loss: 0.8405 - val_acc: 0.7398
Epoch 91/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3620 - acc: 0.8629 - val_loss: 0.9231 - val_acc: 0.7249
Epoch 92/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3731 - acc: 0.8670 - val_loss: 0.9932 - val_acc: 0.6914
Epoch 93/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3560 - acc: 0.8691 - val_loss: 0.9406 - val_acc: 0.7026
Epoch 94/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3238 - acc: 0.8822 - val_loss: 1.0011 - val_acc: 0.6617
Epoch 95/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3153 - acc: 0.8865 - val_loss: 1.3089 - val_acc: 0.6357
Epoch 96/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3227 - acc: 0.8794 - val_loss: 1.1284 - val_acc: 0.6803
Epoch 97/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3165 - acc: 0.8812 - val_loss: 1.1975 - val_acc: 0.6766
Epoch 98/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.2958 - acc: 0.8921 - val_loss: 0.9066 - val_acc: 0.7175
Epoch 99/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.3090 - acc: 0.8887 - val_loss: 0.9697 - val_acc: 0.7026
```

```
Epoch 100/100
3225/3225 [==============================] - 29s 9ms/sample - loss: 0.2774 - acc: 0.9036 - val_loss: 0.8332 - val_acc: 0.7323
```

```
# Loss Curves
plt.figure(figsize=(25, 10))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'],'-g',linewidth=1.0)
plt.plot(history.history['val_loss'],'r',linewidth=1.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=14)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=22)

# Accuracy Curves
plt.subplot(1, 2, 2)
plt.plot(history.history['acc'],'-g',linewidth=1.0)
plt.plot(history.history['val_acc'],'r',linewidth=1.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=14)
plt.xlabel('Epochs ',fontsize=16)

plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves',fontsize=22)
plt.show()
```




## VII. Save model

```
model.save('drive/My Drive/SpeechEmotionRecognition/[CNN-LSTM]M.h5')
model.save_weights('drive/My Drive/SpeechEmotionRecognition/[CNN-LSTM]W.h5')
```

## Performance

To limit overfitting during training phase, we split our data set into train (80%) and test set (20%). Following show results obtained on test set:

| Model | Accuracy |
| --- | --- |
| SVM on global statistic features | **68.3%** |
| Time distributed CNNs | **76.6%** |

# Text-based Personality Traits Recognition

In this section you will find all resources, models and Python scripts relative to text-based personality traits recognition.

Emotion recognition through text is a challenging task that goes beyond conventional sentiment analysis : instead of simply detecting neutral, positive or negative feelings from text, the goal is to identify a set of emotions characterized by a higher granularity. For instance, feelings like anger or happiness could be included in the classification. As recognizing such emotions can turn out to be complex even for the human eye, machine learning algorithms are likely to obtain mixed performances. It is important to note that nowadays, emotion recognition from facial expression tends to perform better than from textual expression. Indeed, many subtleties should be taken into account in order to perform an accurate detection of human emotions through text, context-dependency being one of the most crucial. This is the reason why using advanced natural language processing is required to obtain the best performance possible.

In the context of our study, we chose to use text mining in order not to detect regular emotions such as disgust or surprise, but to recognize personality traits based on the "Big Five" model in psychology. Even though emotion recognition and personality traits classification are two separate fields of studies based on different theoretical underpinnings, they use similar learning-based methods and literature from both areas can be interesting. The main motivation behind this choice is to offer a broader assessment to the user : as emotions can only be understood in the light of a person's own characteristics, we thought that analyzing personality traits would provide a new key to understanding emotional fluctuations. Our final

goal is to enrich the user experience and improve the quality of our analysis : any appropriate and complementary information deepening our understanding of the user's idiosyncrasies is welcome

Our main goal is to leverage on the use of statistical learning methods in order to build a tool capable of recognizing the personality traits of an individual given a text containing his answers to pre-established personal questions. Our first idea was to record a user's interview and convert the file from audio to text : in this way we would have been able to work with similar data for text, audio and video. Nevertheless, the good transcription of audio files to text requires the use of expensive APIs, and the tools available for free in the market don't provide sufficient quality. This is the reason why we chose to apply our personality traits detection model to short texts directly written by users : in this way we can easily target particular themes or questions and provide indications of the language level to use. As a result of this, we can make sure that the text data we use to perform the personality traits detection is consistent with the data used for training, and therefore ensure the highest possible quality of results.

# Data

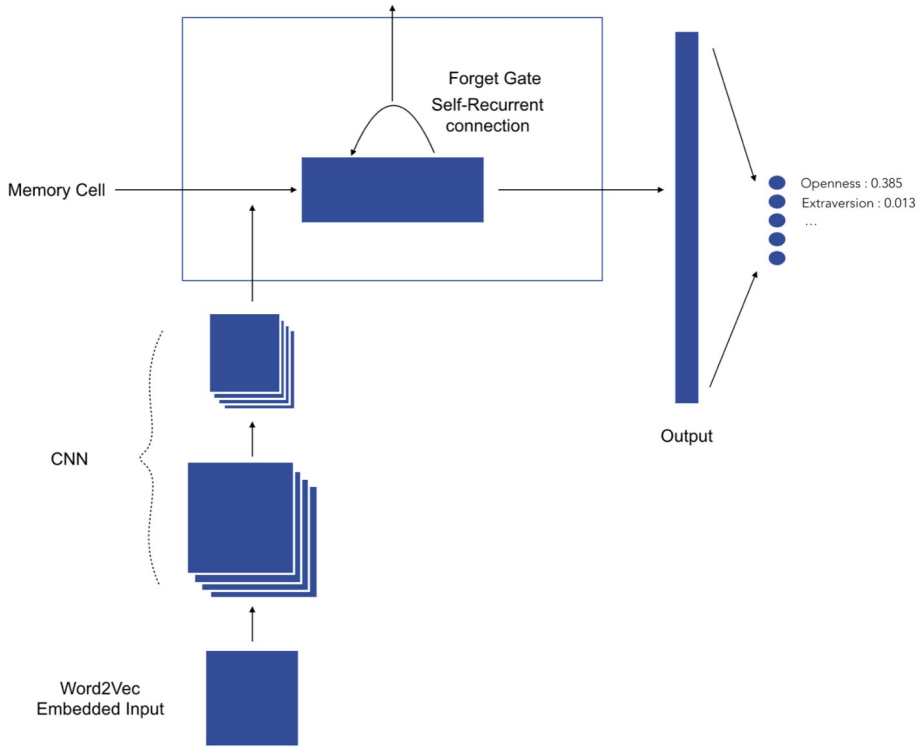
We are using data that was gathered in a study by Pennebaker and King [1999]. It consists of a total of 2,468 daily writing submissions from 34 psychology students (29 women and 5 men whose ages ranged from 18 to 67 with a mean of 26.4). The writing submissions were in the form of a course unrated assignment. For each assignment, students were expected to write a minimum of 20 minutes per day about a specific topic. The data was collected during a 2-week summer course between 1993 to 1996. Each student completed their daily writing for 10 consecutive days. Students' personality scores were assessed by answering the Big Five Inventory (BFI) [John et al., 1991]. The BFI is a 44-item self-report questionnaire that provides a score for each of the five personality traits. Each item consists of short phrases and is rated using a 5-point scale that ranges from 1 (disagree strongly) to 5 (agree strongly). An instance in the data source consists of an ID, the actual essay, and five classification labels of the Big Five personality traits. Labels were originally in the form of either yes ('y') or no ('n') to indicate scoring high or low for a given trait. It is important to note that the classification labels have been applied according to answers to a rather short self-report questionnaire : there might be a non-negligible bias in the data due to both the relative simplicity of the BFI test compared to the complexity of psychological features, and the cognitive biases preventing users from providing a perfectly accurate assessment of their own characteristics.

# Requirements

```
Python : 3.6.5

Scipy : 1.1.0

Scikit-learn : 0.20.0

Tensorflow : 1.12.0

Keras : 2.2.2

Numpy : 1.15.2

Nltk : 3.3.0
```

```
Gensim : 3.4.0
```

# Pipeline




1. The text-based personality recognition pipeline has the following structure :
2. Text data retrieving
3. Custom natural language preprocessing :
    1. Tokenization of the document
    2. Cleaning and standardization of formulations using regular expressions (for instance replacing "can't" by "cannot", "'ve" by "have")
    3. Deletion of the punctuation
    4. Lowercasing the tokens
    5. Removal of predefined stopwords (such as 'a', 'an' etc.)
    6. Application of part-of-speech tags on the remaining tokens
    7. Lemmatization of tokens using part-of-speech tags for more accuracy.
    8. Padding the sequences of tokens of each document to constrain the shape of the input vectors. The input size has been fixed to 300 : all tokens beyond this index are deleted. If the input vector has less than 300 tokens, zeros are added at the beginning of the vector in order to normalize the shape. The dimension of the padded sequence has been determine using

the characteristics of our training data. The average number of words in each essay was 652 before any preprocessing. After the standardization of formulations, and the removal of punctuation characters and stopwords, the average number of words dropped to 168 with a standard deviation of 68. In order to make sure we incorporate in our classification the right number of words without discarding too much information, we set the padding dimension to 300, which is roughly equal to the average length plus two times the standard deviation.

4. 300-dimension Word2Vec trainable embedding
5. Prediction using our pre-trained model

# Model

We have chosen a neural network architecture based on both one-dimensional convolutional neural networks and recurrent neural networks. The one-dimensional convolution layer plays a role comparable to feature extraction : it allows finding patterns in text data. The Long-Short Term Memory cell is then used in order to leverage on the sequential nature of natural language : unlike regular neural network where inputs are assumed to be independent of each other, these architectures progressively accumulate and capture information through the sequences. LSTMs have the property of selectively remembering patterns for long durations of time. Our final model first includes 3 consecutive blocks consisting of the following four layers : one-dimensional convolution layer - max pooling - spatial dropout - batch normalization. The numbers of convolution filters are respectively 128, 256 and 512 for each block, kernel size is 8, max pooling size is 2 and dropout rate is 0.3. Following the three blocks, we chose to stack 3 LSTM cells with 180 outputs each. Finally, a fully connected layer of 128 nodes is added before the last classification layer.

## Notebook

### *All_NLP.ipynb*

#### Imports

```
from nltk.corpus import movie_reviews as reviews
from sklearn.datasets import fetch_20newsgroups
from gensim.models import KeyedVectors
from gensim.models import word2vec

import wget
import numpy as np
import pandas as pd
import re
import datetime
from operator import itemgetter
from random import randint
import seaborn as sns
import matplotlib.pyplot as plt

import os
```

```
import time
import string
import dill
import pickle
import gzip

from nltk import *
from nltk import wordpunct_tokenize, WordNetLemmatizer, sent_tokenize, pos_tag
from nltk.corpus import stopwords as sw, wordnet as wn
from nltk.stem.snowball import SnowballStemmer

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.preprocessing import LabelEncoder, FunctionTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.metrics import precision_score, accuracy_score, confusion_matrix,
classification_report as clsr
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer, CountVectorizer
from sklearn.model_selection import GridSearchCV, train_test_split as tts
from sklearn.manifold import TSNE
from sklearn.multiclass import OneVsRestClassifier

import tensorflow as tf

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, Model, model_from_json
from keras.layers.normalization import BatchNormalization
from keras.layers.embeddings import Embedding
from keras.layers import Dense, LSTM, SpatialDropout1D, Activation, Conv1D, MaxPooling1D,
Input, concatenate
from keras.utils.np_utils import to_categorical
```

```
Using TensorFlow backend.
```

### Data retrieving

```
# Load from .csv file with complete dataset
data_essays = pd.read_csv('./Data/essays.csv', encoding = "ISO-8859-1")
data_essays['cEXT'] = np.where(data_essays['cEXT']=='y', 1, 0)
data_essays['cNEU'] = np.where(data_essays['cNEU']=='y', 1, 0)
```

```
data_essays['cAGR'] = np.where(data_essays['cAGR']=='y', 1, 0)
data_essays['cCON'] = np.where(data_essays['cCON']=='y', 1, 0)
data_essays['cOPN'] = np.where(data_essays['cOPN']=='y', 1, 0)
X_essays = data_essays['TEXT'].tolist()
y_essays = data_essays[['cEXT', 'cNEU', 'cAGR', 'cCON', 'cOPN']]
data_essays['text length'] = data_essays['TEXT'].apply(len)
labels = ['cEXT', 'cNEU', 'cAGR', 'cCON', 'cOPN']
X_train, X_test, y_train, y_test = tts(X_essays, y_essays, test_size=0.2)

# Train-test split to save the dataset
with open('./Data/X_train.pkl', 'wb') as f:
    pickle.dump(X_train, f)
with open('./Data/X_test.pkl', 'wb') as f:
    pickle.dump(X_test, f)
with open('./Data/y_train.pkl', 'wb') as f:
    pickle.dump(y_train, f)
with open('./Data/y_test.pkl', 'wb') as f:
    pickle.dump(y_test, f)

# Load train and test sets from pickled lists
with open('./Data/X_train.pkl', 'rb') as pickle_file:
    X_train = pickle.load(pickle_file)
with open('./Data/X_test.pkl', 'rb') as pickle_file:
    X_test = pickle.load(pickle_file)
with open('./Data/y_train.pkl', 'rb') as pickle_file:
    y_train = pickle.load(pickle_file)
with open('./Data/y_test.pkl', 'rb') as pickle_file:
    y_test = pickle.load(pickle_file)
```

**Visualization**

```
import random
class visualize:

    def __init__(self, complete_dataset, X, labels_list):
        self.data = complete_dataset
        self.X = X
        self.labels_list = labels_list

    def textlength_vs_labels_histogram(self):
        # Visualization of histograms of text length vs. label
        for label in self.labels_list:
            g = sns.FacetGrid(data=self.data, col=label)
            g.map(plt.hist, 'text length', bins=50)
        plt.show()
```

```
def textlength_vs_labels_boxplot(self):
    # Visualization of boxplots of text length vs. label
    for i, label in enumerate(self.labels_list):
        plt.figure(i)
        sns.boxplot(x=label, y='text length', data=self.data)
    plt.show()

def most_frequent_words(self):
    # Visualization of the most frequent words
    complete_corpus = ' '.join(self.X)
    words = tokenize.word_tokenize(complete_corpus)
    fdist = FreqDist(words)
    print("List of 100 most frequent words/counts")
    print(fdist.most_common(100))
    fdist.plot(40)

def most_frequent_words_preprocessed(self):
    # Visualization of the most frequent words
    if not hasattr(self, 'X_preprocess'):
        preprocessor = train(corpus = self.X).NLTKPreprocessor
        self.X_preprocess = prep.transform(self.X).tolist()
    complete_corpus = ' '.join(self.X_preprocess)
    words = tokenize.word_tokenize(complete_corpus)
    fdist = FreqDist(words)
    print("List of 100 most frequent words/counts")
    print(fdist.most_common(100))
    fdist.plot(40)

def get_corpus_statistics(self):
    # Retrieve some info on the text data
    numWords = []
    for text in self.X:
            counter = len(text.split())
            numWords.append(counter)
    numFiles = len(numWords)
    print('The total number of essays is', numFiles)
    print('The total number of words in all essays is', sum(numWords))
    print('The average number of words in each essay is', sum(numWords)/len(numWords))

def get_preprocessed_corpus_statistics(self):
    # Retrieve some info on the preprocessed text data
    if not hasattr(self, 'X_preprocess'):
        preprocessor = train(corpus = self.X).NLTKPreprocessor
        self.X_preprocess = prep.transform(self.X).tolist()
```

```
        len_list = [np.count_nonzero(self.X_preprocess[i]) for i in range(len(self.X))]
        print('The average number of words in each preprocessed essay is', np.mean(len_list))
        print('The standard deviation of the number of words in each preprocessed essay is',
np.std(len_list))
        print('The average number of words in each preprocessed essay plus 2 standard deviations
is', np.mean(len_list) + 2 * np.std(len_list))

class tsne:

    def __init__(self, X, max_features = 30000, max_sentence_len = 300, embed_dim = 300,
n_elements = 100):
        self.X = X
        self.max_features =max_features
        self.max_sentence_len = max_sentence_len
        self.embed_dim = embed_dim
        self.n_elements = n_elements
        self.vectors, self.words, self.dic =  self.prepare_embedding(self.X)

    def load_google_vec(self):
        url = 'https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz'
        #wget.download(url, 'Data/GoogleNews-vectors.bin.gz')
        return KeyedVectors.load_word2vec_format(
            'Data/GoogleNews-vectors.bin.gz',
            binary=True)

    def lemmatize_token(self, token, tag):
        tag = {
            'N': wn.NOUN,
            'V': wn.VERB,
            'R': wn.ADV,
            'J': wn.ADJ
        }.get(tag[0], wn.NOUN)
        return WordNetLemmatizer().lemmatize(token, tag)


    def get_preprocessed_corpus(self, X_corpus):
        """
        Returns a preprocessed version of a full corpus (ie. tokenization and lemmatization using POS
taggs)
        """
        X = ' '.join(X_corpus)
        lemmatized_tokens = []

        # Break the document into sentences
        for sent in sent_tokenize(X):
```

```
        # Break the sentence into part of speech tagged tokens
        for token, tag in pos_tag(wordpunct_tokenize(sent)):

            # Apply preprocessing to the token
            token = token.lower()
            token = token.strip()
            token = token.strip('_')
            token = token.strip('*')

            # If punctuation or stopword, ignore token and continue
            if token in set(sw.words('english')) or all(char in set(string.punctuation) for char in
token):
                continue

            # Lemmatize the token and yield
            lemma = self.lemmatize_token(token, tag)
            lemmatized_tokens.append(lemma)

    doc = ' '.join(lemmatized_tokens)
    return doc


  def prepare_embedding(self, X):
    """
    Returns the embedding weights matrix, the word index, and the word-vector dictionnary
corresponding
    to the training corpus set of words.
    """
    # Load Word2Vec vectors
    word2vec = self.load_google_vec()

    # Fit and apply an NLTK tokenizer on the preprocessed training corpus to obtain sequences.
    tokenizer = Tokenizer(num_words=self.max_features)
    X_pad = self.get_preprocessed_corpus(X)
    tokenizer.fit_on_texts(pd.Series(X_pad))
    X_pad = tokenizer.texts_to_sequences(pd.Series(X_pad))

    # Pad the sequences
    X_pad = pad_sequences(X_pad, maxlen=self.max_sentence_len, padding='post',
truncating='post')

    # Retrieve the word index
    train_word_index = tokenizer.word_index
```

```
    # Construct the embedding weights matrix and word-vector dictionnary
    train_embedding_weights = np.zeros((len(train_word_index) + 1, self.embed_dim))
    for word, index in train_word_index.items():
      train_embedding_weights[index, :] = word2vec[word] if word in word2vec else
np.random.rand(self.embed_dim)
    word_vector_dict = dict(zip(pd.Series(list(train_word_index.keys())),
                    pd.Series(list(train_word_index.keys())).apply(
                      lambda x: train_embedding_weights[train_word_index[x]])))
    return train_embedding_weights, train_word_index, word_vector_dict


  def plot(self):
    labels = []
    tokens = []

    l_bound = 0
    u_bound = len(self.words)
    step = int(len(self.words)/self.n_elements)

    #for index in range(l_bound,u_bound, step):
    for index in random.sample(range(l_bound,u_bound), self.n_elements):
      tokens.append(self.vectors[index])
      labels.append(self.words[index])

    tsne_model = TSNE(perplexity=40, n_components=2, init='pca', n_iter=2500,
random_state=23)
    new_values = tsne_model.fit_transform(tokens)

    xx = []
    yy = []
    for value in new_values:
      xx.append(value[0])
      yy.append(value[1])

    plt.figure(figsize=(16, 16))
    for i in range(len(xx)):
      plt.scatter(xx[i],yy[i])
      plt.annotate(labels[i],
            xy=(xx[i], yy[i]),
            xytext=(5, 2),
            textcoords='offset points',
            ha='right',
            va='bottom')
    plt.show()
```

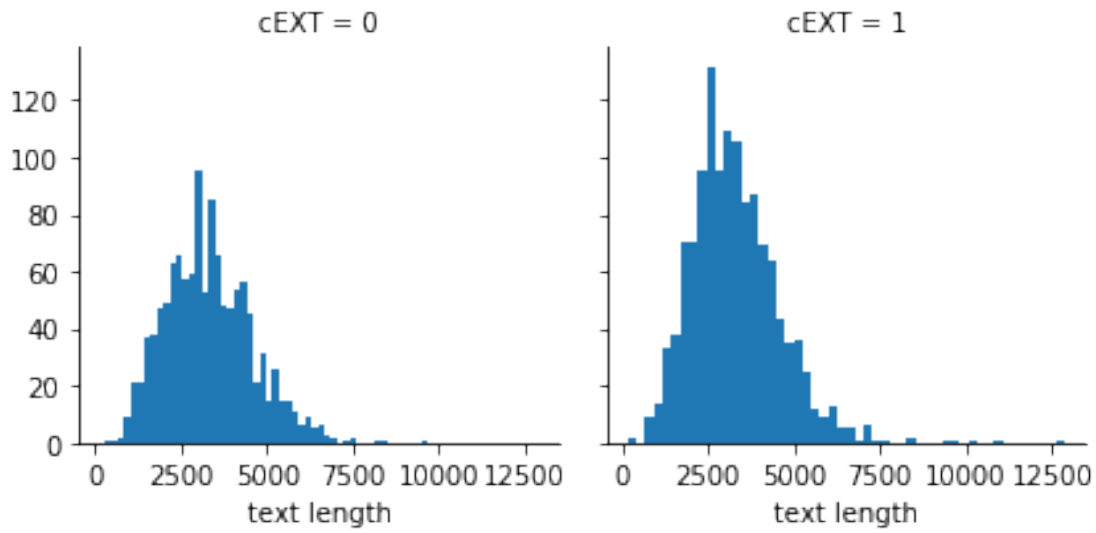
```
class NLTKPreprocessor(BaseEstimator, TransformerMixin):
    """
    Transforms input data by using NLTK tokenization, POS tagging, lemmatization and vectorization.
    """

    def __init__(self, corpus, max_sentence_len = 300, stopwords=None, punct=None,
lower=True, strip=True):
        """
        Instantiates the preprocessor.
        """
        self.lower = lower
        self.strip = strip
        self.stopwords = set(stopwords) if stopwords else set(sw.words('english'))
        self.punct = set(punct) if punct else set(string.punctuation)
        self.lemmatizer = WordNetLemmatizer()
        self.corpus = corpus
        self.max_sentence_len = max_sentence_len

    def fit(self, X, y=None):
        """
        Fit simply returns self.
        """
        return self

    def inverse_transform(self, X):
        """
        No inverse transformation.
        """
        return X

    def transform(self, X):
        """
        Actually runs the preprocessing on each document.
        """
        output = np.array([(self.tokenize(doc)) for doc in X])
        return output

    def tokenize(self, document):
        """
        Returns a normalized, lemmatized list of tokens from a document by
        applying segmentation, tokenization, and part of speech tagging.
        Uses the part of speech tags to look up the lemma in WordNet, and returns the lowercase
        version of all the words, removing stopwords and punctuation.
        """
```

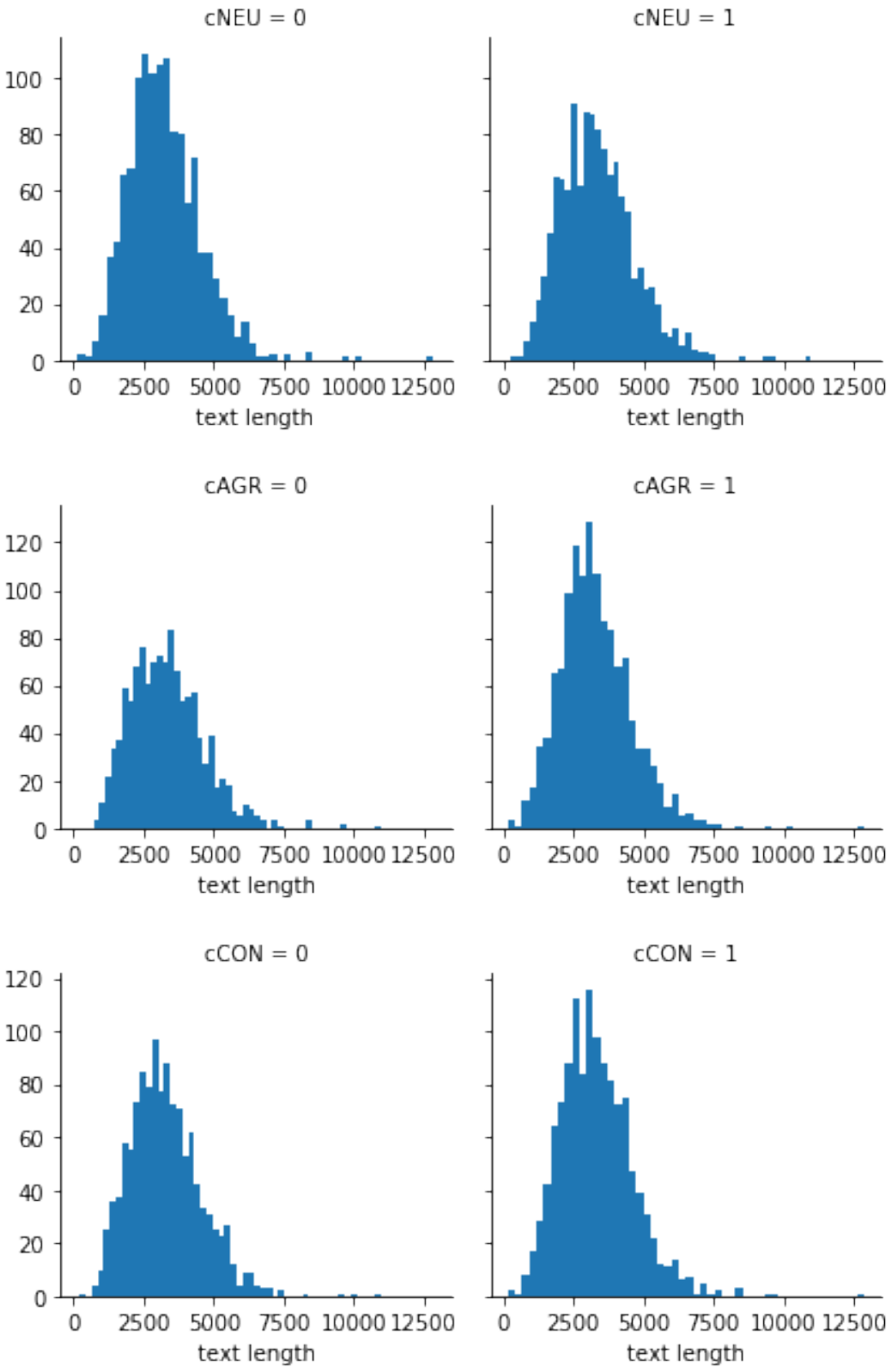
```
    lemmatized_tokens = []

    # Clean the text
    document = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", document)
    document = re.sub(r"what's", "what is ", document)
    document = re.sub(r"\'s", " ", document)
    document = re.sub(r"\'ve", " have ", document)
    document = re.sub(r"can't", "cannot ", document)
    document = re.sub(r"n't", " not ", document)
    document = re.sub(r"i'm", "i am ", document)
    document = re.sub(r"\'re", " are ", document)
    document = re.sub(r"\'d", " would ", document)
    document = re.sub(r"\'ll", " will ", document)
    document = re.sub(r"(\d+)(k)", r"\g<1>000", document)

    # Break the document into sentences
    for sent in sent_tokenize(document):

        # Break the sentence into part of speech tagged tokens
        for token, tag in pos_tag(wordpunct_tokenize(sent)):

            # Apply preprocessing to the token
            token = token.lower() if self.lower else token
            token = token.strip() if self.strip else token
            token = token.strip('_') if self.strip else token
            token = token.strip('*') if self.strip else token

            # If punctuation or stopword, ignore token and continue
            if token in self.stopwords or all(char in self.punct for char in token):
                continue

            # Lemmatize the token
            lemma = self.lemmatize(token, tag)
            lemmatized_tokens.append(lemma)

    doc = ' '.join(lemmatized_tokens)
    tokenized_document = self.vectorize(np.array(doc)[np.newaxis])
    return tokenized_document


def vectorize(self, doc):
    """
    Returns a vectorized padded version of sequences.
    """
    save_path = "Data/padding.pickle"
```

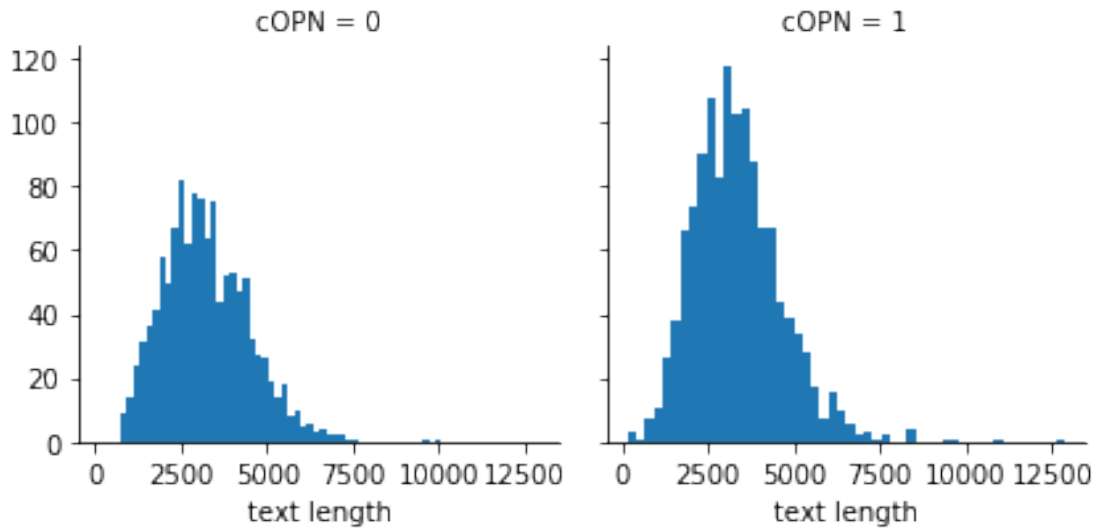
```
        with open(save_path, 'rb') as f:
            tokenizer = pickle.load(f)
        doc_pad = tokenizer.texts_to_sequences(doc)
        doc_pad = pad_sequences(doc_pad, padding='pre', truncating='pre',
maxlen=self.max_sentence_len)
        return np.squeeze(doc_pad)

    def lemmatize(self, token, tag):
        """
        Converts the Penn Treebank tag to a WordNet POS tag, then uses that
        tag to perform WordNet lemmatization.
        """
        tag = {
            'N': wn.NOUN,
            'V': wn.VERB,
            'R': wn.ADV,
            'J': wn.ADJ
        }.get(tag[0], wn.NOUN)

        return self.lemmatizer.lemmatize(token, tag)
```

**Histograms of text length distrbution for the different labels**

```
viz = visualize(data_essays, X_essays, labels)
viz.textlength_vs_labels_histogram()
```

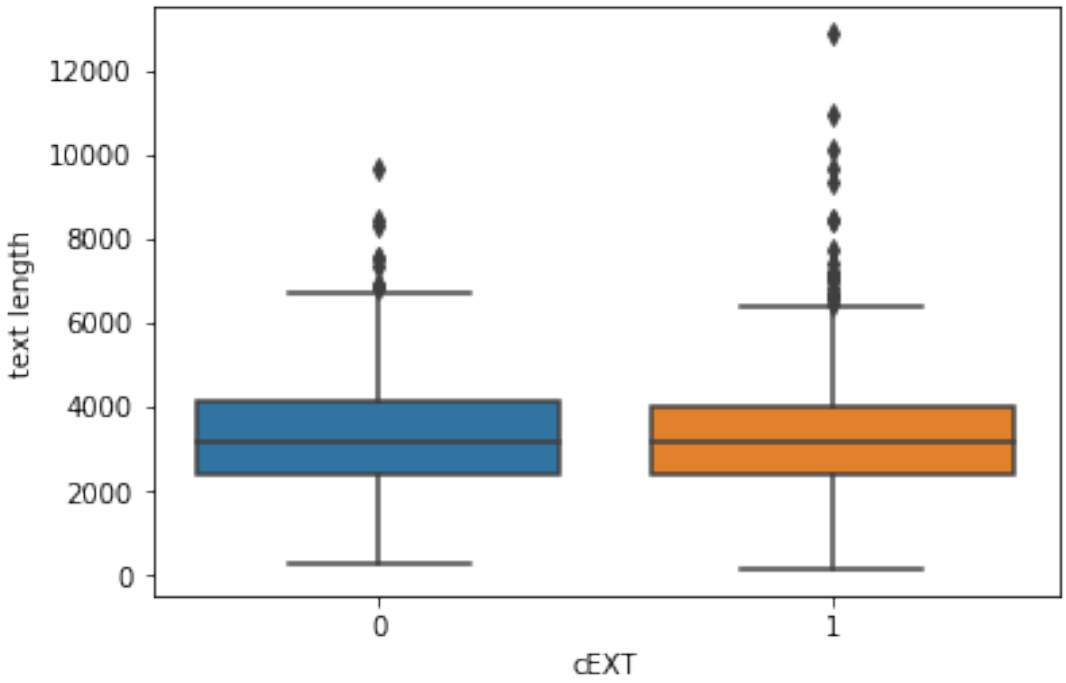


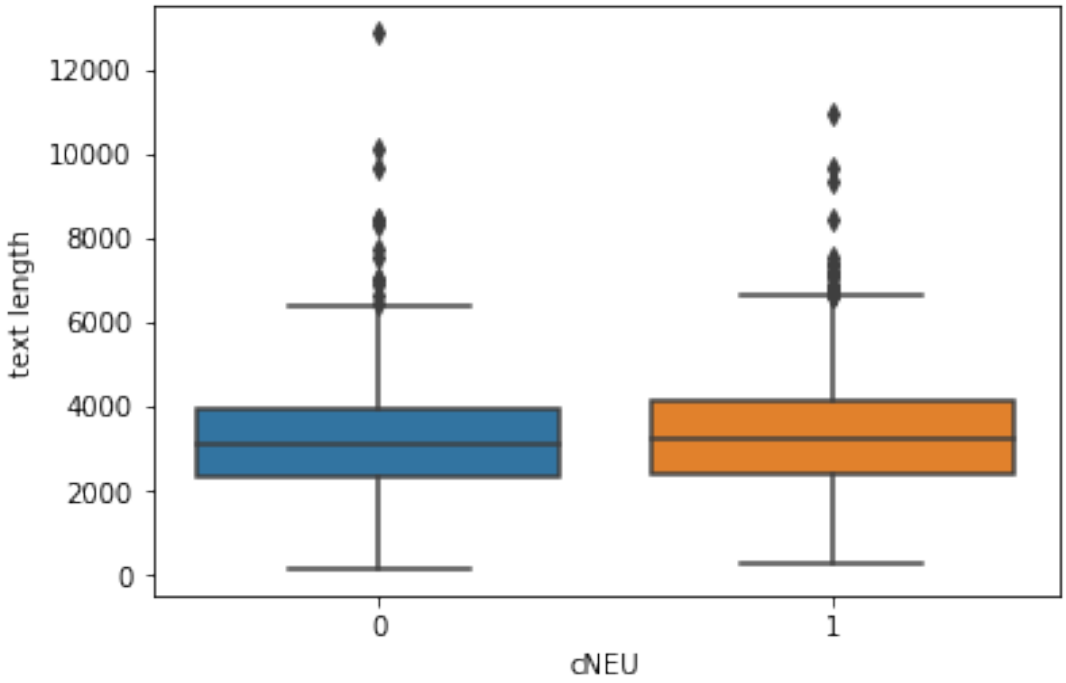

cNEU = 0
cNEU = 1
cAGR = 0
cAGR = 1
cCON = 0
cCON = 1
text length
0
2500
5000
7500
10000
12500
0
20
40
60
80
100
120

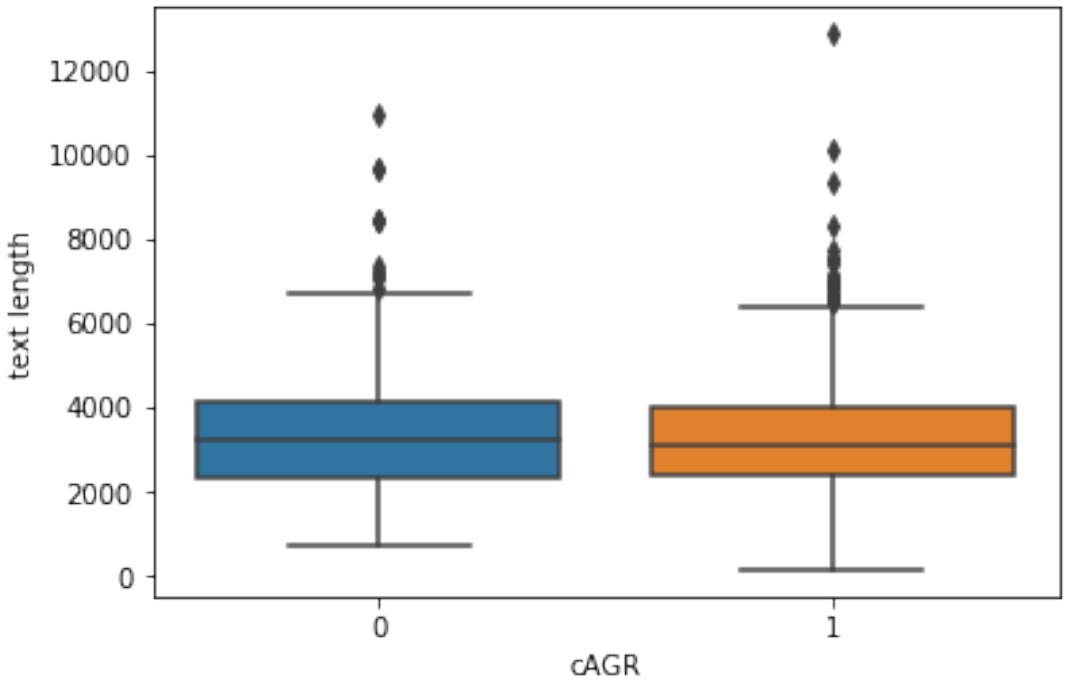


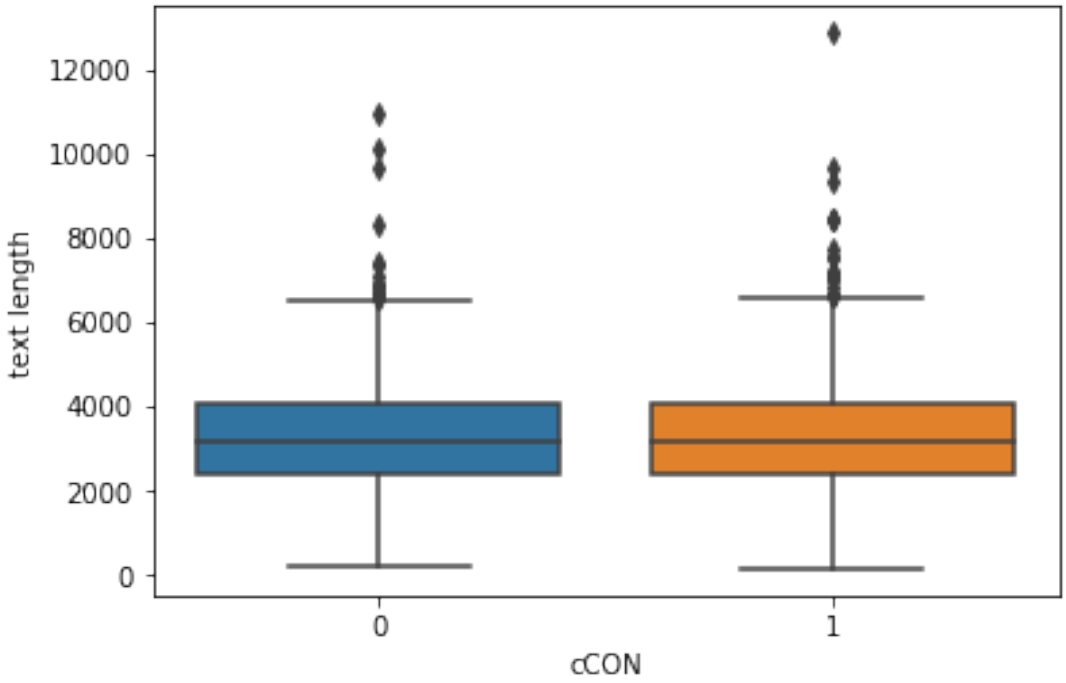

**Boxplots of text length distrbution for the different labels**

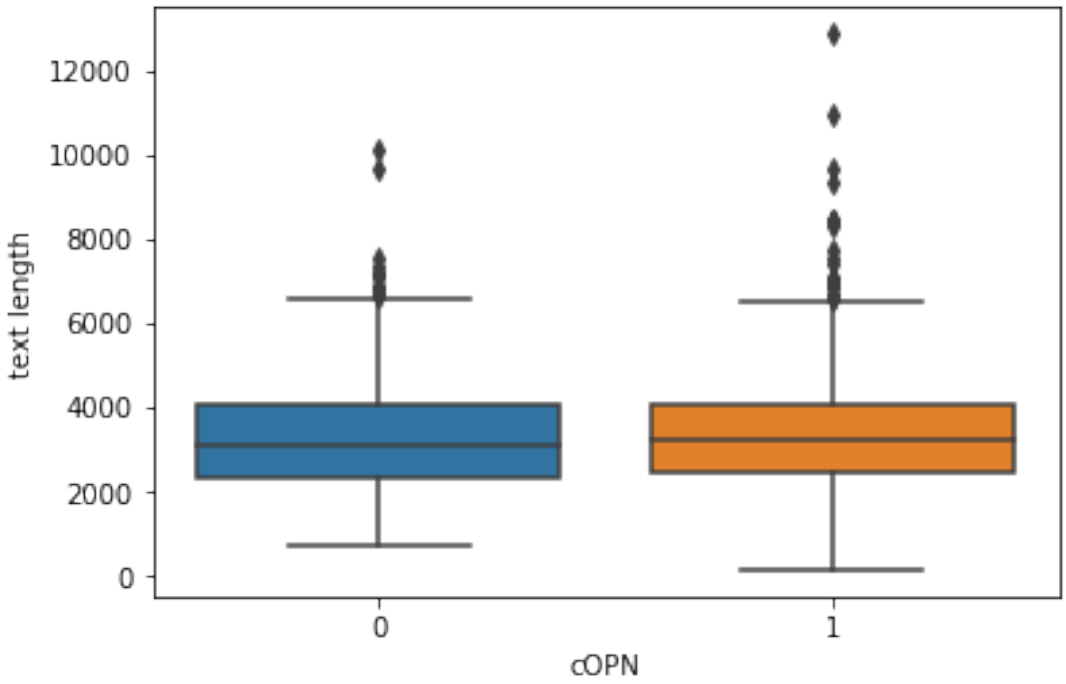
```
viz.textlength_vs_labels_boxplot()
```




text length
12000
10000
8000
6000
4000
2000
0
0
1
cNEU


text length
12000
10000
8000
6000
4000
2000
0
0
1
cAGR

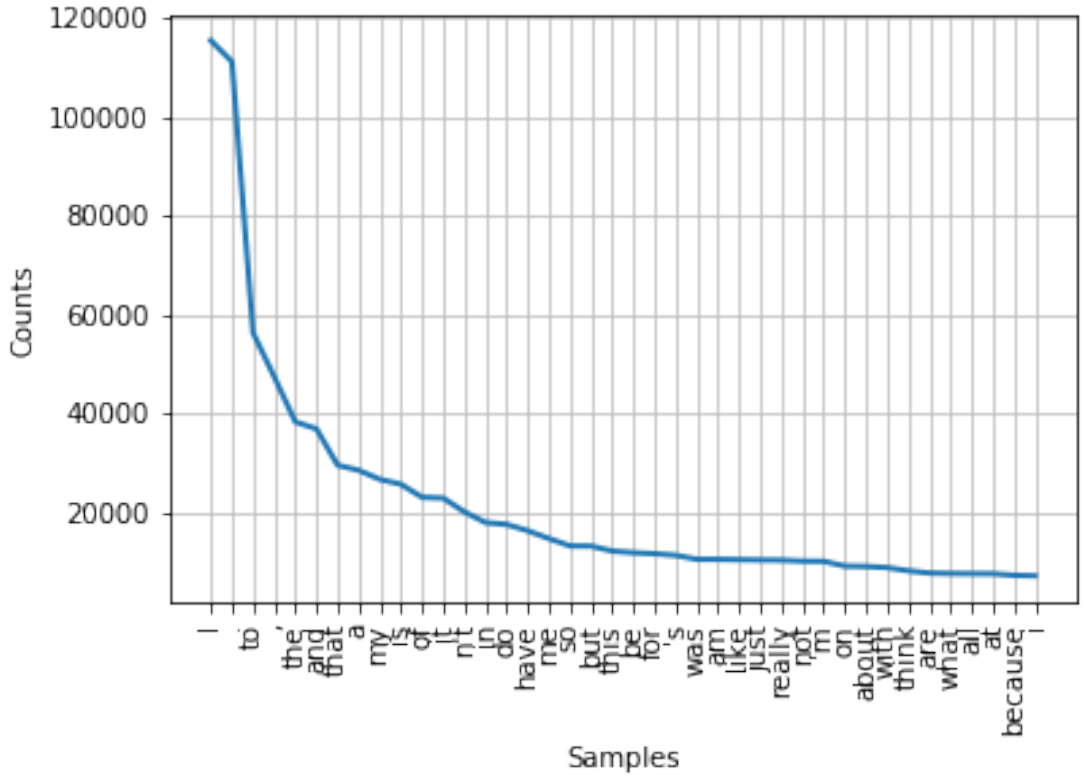



**Most frequent words in the corpus**

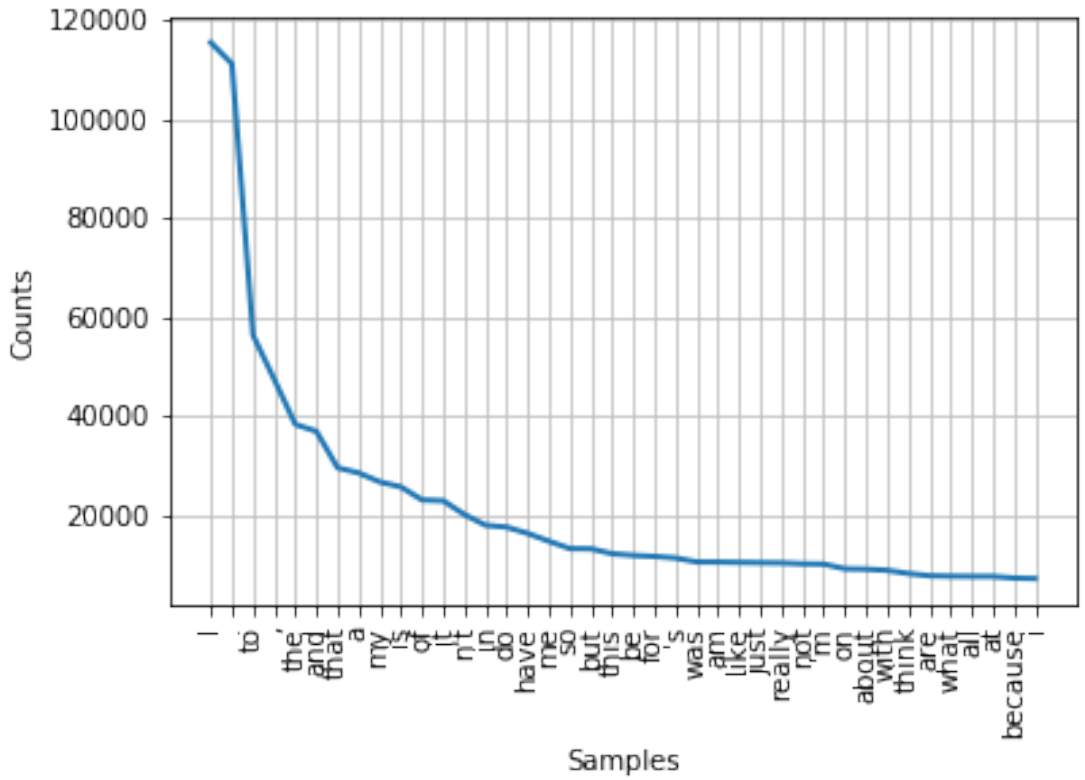
| viz.most_frequent_words() |
|---|
| List of 100 most frequent words/counts<br>[('I', 115468), ('.', 111178), ('to', 56263), (',', 47355), ('the', 38232), ('and', 36810), ('that', 29456), ('a', 28408), ('my', 26580), ('is', 25576), ('of', 22939), ('it', 22781), ("n't", 19996), ('in', 17828), ('do', 17448), |

```
('have', 16166), ('me', 14588), ('so', 13099), ('but', 13060), ('this', 12054), ('be', 11724), ('for', 11520),
("'s", 11198), ('was', 10392), ('am', 10378), ('like', 10308), ('just', 10250), ('really', 10207), ('not',
10015), ("'m", 9973), ('on', 9015), ('about', 8941), ('with', 8708), ('think', 8061), ('are', 7602), ('what',
7517), ('all', 7475), ('at', 7469), ('because', 7144), ('i', 7047), ('know', 6959), ('get', 6875), ('he', 6605),
('now', 6154), ('would', 6077), ('you', 6013), ('if', 6001), ('time', 5966), ('out', 5923), ('they', 5905),
('up', 5743), ('or', 5733), ('going', 5621), ('go', 5576), ('she', 5556), ('?', 5539), ('want', 5483), ('will',
5420), ('can', 5276), ('!', 4959), ('as', 4939), ('people', 4898), ('her', 4879), ('when', 4867), ('we', 4836),
('much', 4702), ('It', 4626), ('one', 4465), ('how', 4308), ('feel', 4239), ('there', 4218), ('him', 3988),
('good', 3964), ('here', 3837), ('more', 3810), ('some', 3762), ('had', 3759), ('from', 3684), ('need',
3580), ('been', 3560), ('right', 3428), ('them', 3346), ('did', 3323), ('ca', 3261), ('too', 3213), ('has',
3200), ('could', 3131), ('things', 3117), ('My', 3075), ('well', 3012), ('school', 3010), ('class', 2983),
('wonder', 2959), ('see', 2882), ('should', 2836), ('guess', 2811), ('friends', 2804), ('back', 2684),
('something', 2657), ('very', 2648)]
```




**Most frequent words in the preprocessed corpus**

```
viz.most_frequent_words_preprocessed()
```




**Statistics on the text corpus**

```
viz.get_corpus_statistics()

The total number of essays is 2467
The total number of words in all essays is 1608813
The average number of words in each essay is 652.1333603567085

viz.get_preprocessed_corpus_statistics()

The average number of words in each preprocessed essay is 167.26509931090393
The standard deviation of the number of words in each preprocessed essay is 62.85333564580388
The average number of words in each preprocessed essay plus 2 standard deviations is
292.9717706025117

viz_tsne = tsne(X_essays, max_features = 30000, max_sentence_len = 300, embed_dim = 300,
n_elements = 100)
viz_tsne.plot()
```

**Train models**

***Train Keras model***

```
class train:

  def __init__(self, corpus):
```

```
        self.max_sentence_len = 300
        self.max_features = 300
        self.embed_dim = 300
        self.lstm_out = 180
        self.dropout_lstm = 0.3
        self.recurrent_dropout_lstm = 0.3
        self.dropout = 0.3
        self.conv_nfilters = 128
        self.conv_kernel_size = 8
        self.max_pool_size = 2
        self.NLTKPreprocessor = self.NLTKPreprocessor(corpus)
        #self.MyRNNTransformer = self.MyRNNTransformer()


    class NLTKPreprocessor(BaseEstimator, TransformerMixin):
        """
        Transforms input data by using NLTK tokenization, POS tagging, lemmatization and
vectorization.
        """

        def __init__(self, corpus, max_sentence_len = 300, stopwords=None, punct=None,
lower=True, strip=True):
            """
            Instantiates the preprocessor.
            """
            self.lower = lower
            self.strip = strip
            self.stopwords = set(stopwords) if stopwords else set(sw.words('english'))
            self.punct = set(punct) if punct else set(string.punctuation)
            self.lemmatizer = WordNetLemmatizer()
            self.corpus = corpus
            self.max_sentence_len = max_sentence_len

        def fit(self, X, y=None):
            """
            Fit simply returns self.
            """
            return self

        def inverse_transform(self, X):
            """
            No inverse transformation.
            """
            return X
```

```
    def transform(self, X):
        """
        Actually runs the preprocessing on each document.
        """
        output = np.array([(self.tokenize(doc)) for doc in X])
        return output

    def tokenize(self, document):
        """
        Returns a normalized, lemmatized list of tokens from a document by
        applying segmentation, tokenization, and part of speech tagging.
        Uses the part of speech tags to look up the lemma in WordNet, and returns the lowercase
        version of all the words, removing stopwords and punctuation.
        """
        lemmatized_tokens = []

        # Clean the text
        document = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", document)
        document = re.sub(r"what's", "what is ", document)
        document = re.sub(r"\'s", " ", document)
        document = re.sub(r"\'ve", " have ", document)
        document = re.sub(r"can't", "cannot ", document)
        document = re.sub(r"n't", " not ", document)
        document = re.sub(r"i'm", "i am ", document)
        document = re.sub(r"\'re", " are ", document)
        document = re.sub(r"\'d", " would ", document)
        document = re.sub(r"\'ll", " will ", document)
        document = re.sub(r"(\d+)(k)", r"\g<1>000", document)

        # Break the document into sentences
        for sent in sent_tokenize(document):

            # Break the sentence into part of speech tagged tokens
            for token, tag in pos_tag(wordpunct_tokenize(sent)):

                # Apply preprocessing to the token
                token = token.lower() if self.lower else token
                token = token.strip() if self.strip else token
                token = token.strip('_') if self.strip else token
                token = token.strip('*') if self.strip else token

                # If punctuation or stopword, ignore token and continue
                if token in self.stopwords or all(char in self.punct for char in token):
                    continue
```

```
            # Lemmatize the token
            lemma = self.lemmatize(token, tag)
            lemmatized_tokens.append(lemma)

        doc = ' '.join(lemmatized_tokens)
        tokenized_document = self.vectorize(np.array(doc)[np.newaxis])
        return tokenized_document


    def vectorize(self, doc):
        """
        Returns a vectorized padded version of sequences.
        """
        save_path = "./Data/padding.pickle"
        with open(save_path, 'rb') as f:
            tokenizer = pickle.load(f)
        doc_pad = tokenizer.texts_to_sequences(doc)
        doc_pad = pad_sequences(doc_pad, padding='pre', truncating='pre',
maxlen=self.max_sentence_len)
        return np.squeeze(doc_pad)

    def lemmatize(self, token, tag):
        """
        Converts the Penn Treebank tag to a WordNet POS tag, then uses that
        tag to perform WordNet lemmatization.
        """
        tag = {
            'N': wn.NOUN,
            'V': wn.VERB,
            'R': wn.ADV,
            'J': wn.ADJ
        }.get(tag[0], wn.NOUN)

        return self.lemmatizer.lemmatize(token, tag)


class MyRNNTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer allowing our Keras model to be included in our pipeline
    """
    def __init__(self, classifier):
        self.classifier = classifier

    def fit(self, X, y):
        batch_size = 32
```

```
        num_epochs = 135
        batch_size = batch_size
        epochs = num_epochs
        self.classifier.fit(X, y, epochs=epochs, batch_size=batch_size, verbose=2)
        return self

    def transform(self, X):
        self.pred = self.classifier.predict(X)
        self.classes = [[0 if el < 0.2 else 1 for el in item] for item in self.pred]
        return self.classes


  def multiclass_accuracy(self,predictions, target):
    "Returns the multiclass accuracy of the classifier's predictions"
    score = []
    for j in range(0, 5):
        count = 0
        for i in range(len(predictions)):
            if predictions[i][j] == target[i][j]:
                count += 1
        score.append(count / len(predictions))
    return score


  def load_google_vec(self):
    url = 'https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz'
    #wget.download(url, 'Data/GoogleNews-vectors.bin.gz')
    return KeyedVectors.load_word2vec_format(
        'Data/GoogleNews-vectors.bin.gz',
        binary=True)

  def lemmatize_token(self, token, tag):
    tag = {
        'N': wn.NOUN,
        'V': wn.VERB,
        'R': wn.ADV,
        'J': wn.ADJ
    }.get(tag[0], wn.NOUN)
    return WordNetLemmatizer().lemmatize(token, tag)


  def get_preprocessed_corpus(self, X_corpus):
    """
    Returns a preprocessed version of a full corpus (ie. tokenization and lemmatization using POS
taggs)
```

```
        """
        X = ' '.join(X_corpus)
        lemmatized_tokens = []

        # Break the document into sentences
        for sent in sent_tokenize(X):

            # Break the sentence into part of speech tagged tokens
            for token, tag in pos_tag(wordpunct_tokenize(sent)):

                # Apply preprocessing to the token
                token = token.lower()
                token = token.strip()
                token = token.strip('_')
                token = token.strip('*')

                # If punctuation or stopword, ignore token and continue
                if token in set(sw.words('english')) or all(char in set(string.punctuation) for char in
token):
                    continue

                # Lemmatize the token and yield
                lemma = self.lemmatize_token(token, tag)
                lemmatized_tokens.append(lemma)

        doc = ' '.join(lemmatized_tokens)
        return doc


    def prepare_embedding(self, X):
        """
        Returns the embedding weights matrix, the word index, and the word-vector dictionnary
corresponding
        to the training corpus set of words.
        """
        # Load Word2Vec vectors
        word2vec = self.load_google_vec()

        # Fit and apply an NLTK tokenizer on the preprocessed training corpus to obtain sequences.
        tokenizer = Tokenizer(num_words=self.max_features)
        X_pad = self.get_preprocessed_corpus(X)
        tokenizer.fit_on_texts(pd.Series(X_pad))
        X_pad = tokenizer.texts_to_sequences(pd.Series(X_pad))

        # Pad the sequences
```

```
    X_pad = pad_sequences(X_pad, maxlen=self.max_sentence_len, padding='post',
truncating='post')

    # Retrieve the word index
    train_word_index = tokenizer.word_index

    # Construct the embedding weights matrix and word-vector dictionnary
    train_embedding_weights = np.zeros((len(train_word_index) + 1, self.embed_dim))
    for word, index in train_word_index.items():
      train_embedding_weights[index, :] = word2vec[word] if word in word2vec else
np.random.rand(self.embed_dim)
    word_vector_dict = dict(zip(pd.Series(list(train_word_index.keys())),
                        pd.Series(list(train_word_index.keys())).apply(
                          lambda x: train_embedding_weights[train_word_index[x]])))
    return train_embedding_weights, train_word_index, word_vector_dict


  def run(self, X, y, model_name=None, pretrained_weights_path = None,
pretrained_model_path = None, verbose=True):
    """
    Builds a classifer for the given list of documents and targets

    """

    def build(classifier, X, y, embedding_dict, corpus):
      """
      Inner build function that builds a pipeline including a preprocessor and a classifier.
      """
      model = Pipeline([
        ('preprocessor', self.NLTKPreprocessor),
        ('classifier', classifier)
      ])
      return model.fit(X, y)

    # Label encode the targets
    y_trans = y

    # Prepare the embedding
    train_embedding_weights, train_word_index, wv_dict = self.prepare_embedding(X)

    # Begin evaluation
    if verbose: print("Building for evaluation")
    indices = range(len(y))

    # Keras model definition
```

```
    Input_words = Input(shape=(300,), name='input1')
    x = Embedding(len(train_word_index) + 1, self.embed_dim,
weights=[train_embedding_weights],
                  input_length=self.max_sentence_len, trainable=True)(Input_words)
    # classifier.add(Embedding(30000, 300,input_length = 350))
    x = Conv1D(filters=self.conv_nfilters, kernel_size= self.conv_kernel_size, padding='same',
activation='relu')(x)
    x = MaxPooling1D(pool_size=self.max_pool_size)(x)
    x = SpatialDropout1D(self.dropout)(x)
    x = BatchNormalization()(x)
    x = Conv1D(filters=(self.conv_nfilters)*2, kernel_size= self.conv_kernel_size, padding='same',
activation='relu')(x)
    x = MaxPooling1D(pool_size=self.max_pool_size)(x)
    x = SpatialDropout1D(self.dropout)(x)
    x = BatchNormalization()(x)
    x = Conv1D(filters=(self.conv_nfilters)*3, kernel_size= self.conv_kernel_size, padding='same',
activation='relu')(x)
    x = MaxPooling1D(pool_size=self.max_pool_size)(x)
    x = SpatialDropout1D(self.dropout)(x)
    x = BatchNormalization()(x)
    x = LSTM(self.lstm_out, return_sequences=True, dropout=self.dropout_lstm,
recurrent_dropout=self.recurrent_dropout_lstm)(x)
    x = LSTM(self.lstm_out, return_sequences=True, dropout=self.dropout_lstm,
recurrent_dropout=self.recurrent_dropout_lstm)(x)
    x = LSTM(self.lstm_out, dropout=self.dropout_lstm,
recurrent_dropout=self.recurrent_dropout_lstm)(x)
    x = Dense(128, activation='softmax')(x)
    out = Dense(5, activation='softmax')(x)
    classifier = Model(inputs=Input_words, outputs=[out])
    classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    print(classifier.summary())

    # Loading pretrained model for transfer learning
    if pretrained_weights_path and pretrained_model_path:
      json_file = open(pretrained_model_path, 'r')
      classifier = model_from_json(json_file.read())
      classifier.load_weights(pretrained_weights_path)
      classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
      json_file.close()
      model = build(self.MyRNNTransformer(classifier), X, y_trans, wv_dict, corpus=X)

    # Train on the whole set from scratch
    if verbose:
      print("Building complete model and saving ...")
      model= build(self.MyRNNTransformer(classifier), X, y_trans, wv_dict, corpus=X)
```

```
    # Save the model
    if model_name:
        outpath = './Models/'
        classifier.save_weights(outpath + model_name + '.h5')
        with open(outpath + model_name + '.json', 'w') as json_file:
            json_file.write(classifier.to_json())
        print("Model written out to {}".format(model_name))
    else:
        print('Please provide model name for saving')

    return model


model = train(X_train).run(X_train, y_train, "Personality_traits_NN")
```

```
Building for evaluation
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input1 (InputLayer)          (None, 300)               0
_________________________________________________________________
embedding_15 (Embedding)     (None, 300, 300)          6704100
_________________________________________________________________
conv1d_38 (Conv1D)           (None, 300, 128)          307328
_________________________________________________________________
max_pooling1d_38 (MaxPooling (None, 150, 128)          0
_________________________________________________________________
spatial_dropout1d_38 (Spatia (None, 150, 128)          0
_________________________________________________________________
batch_normalization_38 (Batc (None, 150, 128)          512
_________________________________________________________________
conv1d_39 (Conv1D)           (None, 150, 256)          262400
_________________________________________________________________
max_pooling1d_39 (MaxPooling (None, 75, 256)           0
_________________________________________________________________
spatial_dropout1d_39 (Spatia (None, 75, 256)           0
_________________________________________________________________
batch_normalization_39 (Batc (None, 75, 256)           1024
_________________________________________________________________
conv1d_40 (Conv1D)           (None, 75, 384)           786816
_________________________________________________________________
max_pooling1d_40 (MaxPooling (None, 37, 384)           0
_________________________________________________________________
spatial_dropout1d_40 (Spatia (None, 37, 384)           0
```

```
_________________________________________________________________
batch_normalization_40 (Batc (None, 37, 384)           1536
_________________________________________________________________
lstm_38 (LSTM)               (None, 37, 180)           406800
_________________________________________________________________
lstm_39 (LSTM)               (None, 37, 180)           259920
_________________________________________________________________
lstm_40 (LSTM)               (None, 180)               259920
_________________________________________________________________
dense_24 (Dense)             (None, 128)               23168
_________________________________________________________________
dense_25 (Dense)             (None, 5)                 645
=================================================================
Total params: 9,014,169
Trainable params: 9,012,633
Non-trainable params: 1,536
_________________________________________________________________
```

```
None
Building complete model and saving ...
Epoch 1/135
 - 43s - loss: 4.1201 - acc: 0.0851
Epoch 2/135
 - 31s - loss: 4.1199 - acc: 0.1049
Epoch 3/135
 - 31s - loss: 4.1149 - acc: 0.0806
Epoch 4/135
 - 31s - loss: 4.1034 - acc: 0.0720
Epoch 5/135
 - 31s - loss: 4.0920 - acc: 0.0715
Epoch 6/135
 - 31s - loss: 4.0858 - acc: 0.0725
                  :

                  :

                  :
Epoch 26/135
 - 31s - loss: 3.9464 - acc: 0.0563
Epoch 27/135
 - 31s - loss: 3.9639 - acc: 0.0644
Epoch 28/135
 - 31s - loss: 3.9539 - acc: 0.0542
Epoch 29/135
 - 31s - loss: 3.9504 - acc: 0.0588
Epoch 30/135
 - 31s - loss: 3.9384 - acc: 0.0684
```

```
Epoch 31/135
 - 31s - loss: 3.9281 - acc: 0.0740
Epoch 32/135
 - 31s - loss: 3.9266 - acc: 0.0730
                  :

                  :

                  :

Epoch 109/135
 - 31s - loss: 3.5640 - acc: 0.2255
Epoch 110/135
 - 31s - loss: 3.5813 - acc: 0.2357
                  :

                  :

                  :
 - 31s - loss: 3.4707 - acc: 0.2519
Epoch 133/135
 - 31s - loss: 3.4741 - acc: 0.2286
Epoch 134/135
 - 32s - loss: 3.4758 - acc: 0.2367
Epoch 135/135
 - 32s - loss: 3.4566 - acc: 0.2372
Model written out to Personality_traits_NN
```

***Train SVM***

```
class train_svm:

  def __init__(self, corpus):
    self.max_sentence_len = 300
    self.max_features = 300
    self.embed_dim = 300
    self.NLTKPreprocessor = self.NLTKPreprocessor(corpus)
    #self.MyRNNTransformer = self.MyRNNTransformer()


  class NLTKPreprocessor(BaseEstimator, TransformerMixin):
    """
    Transforms input data by using NLTK tokenization, POS tagging, lemmatization and
vectorization.
    """

    def __init__(self, corpus, max_sentence_len = 300, stopwords=None, punct=None,
```

```
lower=True, strip=True):
        """
        Instantiates the preprocessor.
        """
        self.lower = lower
        self.strip = strip
        self.stopwords = set(stopwords) if stopwords else set(sw.words('english'))
        self.punct = set(punct) if punct else set(string.punctuation)
        self.lemmatizer = WordNetLemmatizer()
        self.corpus = corpus
        self.max_sentence_len = max_sentence_len

    def fit(self, X, y=None):
        """
        Fit simply returns self.
        """
        return self

    def inverse_transform(self, X):
        """
        No inverse transformation.
        """
        return X

    def transform(self, X):
        """
        Actually runs the preprocessing on each document.
        """
        output = np.array([(self.tokenize(doc)) for doc in X])
        return output

    def tokenize(self, document):
        """
        Returns a normalized, lemmatized list of tokens from a document by
        applying segmentation, tokenization, and part of speech tagging.
        Uses the part of speech tags to look up the lemma in WordNet, and returns the lowercase
        version of all the words, removing stopwords and punctuation.
        """
        lemmatized_tokens = []

        # Clean the text
        document = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", document)
        document = re.sub(r"what's", "what is ", document)
        document = re.sub(r"\'s", " ", document)
        document = re.sub(r"\'ve", " have ", document)
```

```
        document = re.sub(r"can't", "cannot ", document)
        document = re.sub(r"n't", " not ", document)
        document = re.sub(r"i'm", "i am ", document)
        document = re.sub(r"\'re", " are ", document)
        document = re.sub(r"\'d", " would ", document)
        document = re.sub(r"\'ll", " will ", document)
        document = re.sub(r"(\d+)(k)", r"\g<1>000", document)

        # Break the document into sentences
        for sent in sent_tokenize(document):

            # Break the sentence into part of speech tagged tokens
            for token, tag in pos_tag(wordpunct_tokenize(sent)):

                # Apply preprocessing to the token
                token = token.lower() if self.lower else token
                token = token.strip() if self.strip else token
                token = token.strip('_') if self.strip else token
                token = token.strip('*') if self.strip else token

                # If punctuation or stopword, ignore token and continue
                if token in self.stopwords or all(char in self.punct for char in token):
                    continue

                # Lemmatize the token
                lemma = self.lemmatize(token, tag)
                lemmatized_tokens.append(lemma)

        doc = ' '.join(lemmatized_tokens)
        tokenized_document = self.vectorize(np.array(doc)[np.newaxis])
        return tokenized_document


    def vectorize(self, doc):
        """
        Returns a vectorized padded version of sequences.
        """
        save_path = "./Data/padding.pickle"
        with open(save_path, 'rb') as f:
            tokenizer = pickle.load(f)
        doc_pad = tokenizer.texts_to_sequences(doc)
        doc_pad = pad_sequences(doc_pad, padding='pre', truncating='pre',
maxlen=self.max_sentence_len)
        return np.squeeze(doc_pad)
```

```
    def lemmatize(self, token, tag):
        """
        Converts the Penn Treebank tag to a WordNet POS tag, then uses that
        tag to perform WordNet lemmatization.
        """
        tag = {
            'N': wn.NOUN,
            'V': wn.VERB,
            'R': wn.ADV,
            'J': wn.ADJ
        }.get(tag[0], wn.NOUN)

        return self.lemmatizer.lemmatize(token, tag)


class MyRNNTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer allowing our Keras model to be included in our pipeline
    """
    def __init__(self, classifier):
        self.classifier = classifier

    def fit(self, X, y):
        batch_size = 32
        num_epochs = 35
        batch_size = batch_size
        epochs = num_epochs
        self.classifier.fit(X, y, epochs=epochs, batch_size=batch_size, verbose=2)
        return self

    def transform(self, X):
        self.pred = self.classifier.predict_proba(X)
        self.classes = [[0 if el < 0.2 else 1 for el in item] for item in self.pred]
        return self.classes


class TfidfEmbeddingVectorizer(object):
    def __init__(self, word2vec):
        self.word2vec = word2vec
        self.word2weight = None
        self.dim = len(word2vec.values())

    def fit(self, X, y):
        tfidf = TfidfVectorizer(analyzer=lambda x: x)
        tfidf.fit(X)
```

```
        # if a word was never seen - it must be at least as infrequent
        # as any of the known words. So the default idf is the max of
        # known idf's
        max_idf = max(tfidf.idf_)
        self.word2weight = defaultdict(
          lambda: max_idf,
          [(w, tfidf.idf_[i]) for w, i in tfidf.vocabulary_.items()])

        return self

    def transform(self, X):
        return np.array([
            np.mean([self.word2vec[w] * self.word2weight[w]
                     for w in words if w in self.word2vec] or
                    [np.zeros(self.dim)], axis=0)
            for words in X
        ])


def identity(self, arg):
    """
    Simple identity function works as a passthrough.
    """
    return arg


def reshape_a_feature_column(self, series):
    return np.reshape(np.asarray(series), (len(series), 1))


def pipelinize_feature(self, function, active=True):
    def list_comprehend_a_function(list_or_series, active=True):
        if active:
            processed = [function(i) for i in list_or_series]
            processed = self.reshape_a_feature_column(processed)
            return processed
        else:
            return self.reshape_a_feature_column(np.zeros(len(list_or_series)))


def get_text_length(self, text):
    return len(text)

def load_google_vec(self):
    #url = 'https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz'
```

```
        #wget.download(url, 'Data/GoogleNews-vectors.bin.gz')
        return KeyedVectors.load_word2vec_format(
            'Data/GoogleNews-vectors.bin.gz',
            binary=True)


    def lemmatize_token(self, token, tag):
        tag = {
            'N': wn.NOUN,
            'V': wn.VERB,
            'R': wn.ADV,
            'J': wn.ADJ
        }.get(tag[0], wn.NOUN)
        return WordNetLemmatizer().lemmatize(token, tag)


    def get_preprocessed_corpus(self, X_corpus):
        """
        Returns a preprocessed version of a full corpus (ie. tokenization and lemmatization using POS
taggs)
        """
        X = ' '.join(X_corpus)
        lemmatized_tokens = []

        # Break the document into sentences
        for sent in sent_tokenize(X):

            # Break the sentence into part of speech tagged tokens
            for token, tag in pos_tag(wordpunct_tokenize(sent)):

                # Apply preprocessing to the token
                token = token.lower()
                token = token.strip()
                token = token.strip('_')
                token = token.strip('*')

                # If punctuation or stopword, ignore token and continue
                if token in set(sw.words('english')) or all(char in set(string.punctuation) for char in
token):
                    continue

                # Lemmatize the token and yield
                lemma = self.lemmatize_token(token, tag)
                lemmatized_tokens.append(lemma)
```

```
        doc = ' '.join(lemmatized_tokens)
        return doc


    def prepare_embedding(self, X):
        """
        Returns the embedding weights matrix, the word index, and the word-vector dictionnary corresponding
        to the training corpus set of words.
        """
        # Load Word2Vec vectors
        word2vec = self.load_google_vec()

        # Fit and apply an NLTK tokenizer on the preprocessed training corpus to obtain sequences.
        tokenizer = Tokenizer(num_words=self.max_features)
        X_pad = self.get_preprocessed_corpus(X)
        tokenizer.fit_on_texts(pd.Series(X_pad))
        X_pad = tokenizer.texts_to_sequences(pd.Series(X_pad))

        # Pad the sequences
        X_pad = pad_sequences(X_pad, maxlen=self.max_sentence_len, padding='post', truncating='post')

        # Retrieve the word index
        train_word_index = tokenizer.word_index

        # Construct the embedding weights matrix and word-vector dictionnary
        train_embedding_weights = np.zeros((len(train_word_index) + 1, self.embed_dim))
        for word, index in train_word_index.items():
            train_embedding_weights[index, :] = word2vec[word] if word in word2vec else np.random.rand(self.embed_dim)
        word_vector_dict = dict(zip(pd.Series(list(train_word_index.keys())),
                                    pd.Series(list(train_word_index.keys())).apply(
                                        lambda x: train_embedding_weights[train_word_index[x]])))
        return train_embedding_weights, train_word_index, word_vector_dict



    def multiclass_accuracy(self,predictions, target):
        "Returns the multiclass accuracy of the classifier's predictions"
        score = []
        for j in range(0, 5):
            count = 0
            for i in range(len(predictions)):
                if predictions[i][j] == target[i][j]:
```

```
                count += 1
            score.append(count / len(predictions))
        return score


    def run(self,X, y, classifier=SGDClassifier, model_name=None,
                    verbose=True):
        """
        Builds a classifer for the given list of documents and targets
        """
        def build(classifier, X, y, embedding_dict, corpus):
            """
            Inner build function that builds a single model.
            """
            classifier = OneVsRestClassifier(classifier(loss = "modified_huber", alpha=0.00000001),
n_jobs=-1)
            model = Pipeline([
                ('preprocessor', self.NLTKPreprocessor),
                ("wordVectz", self.TfidfEmbeddingVectorizer(embedding_dict)),
                ('clf', classifier)
            ])
            return model.fit(X, y)

        y_trans = y

        # Prepare the embedding
        train_embedding_weights, train_word_index, wv_dict = self.prepare_embedding(X)

        # Begin evaluation
        if verbose: print("Building complete model and saving ...")
        model = build(classifier, X, y_trans, wv_dict, corpus=X)

        # Save the model
        if model_name:
            outpath = 'Models/'
            with open(outpath + model_name, 'wb') as f:
                dill.dump(model, f)
            print("Model written out to {}".format(model_name))

        return model

model = train_svm(X_train).run(X_train, y_train, model_name = "Personality_traits_SVM")
```

```
Building complete model and saving ...
Model written out to Personality_traits_SVM
```

### Test models

#### *Test Keras model*

```
class test:

   def __init__(self):
      self.max_sentence_len = 300
      self.max_features = 300
      self.embed_dim = 300
      self.NLTKPreprocessor = self.NLTKPreprocessor()
      #self.MyRNNTransformer = self.MyRNNTransformer()


   class NLTKPreprocessor(BaseEstimator, TransformerMixin):
      """
      Transforms input data by using NLTK tokenization, POS tagging, lemmatization and
vectorization.
      """

      def __init__(self, max_sentence_len = 300, stopwords=None, punct=None, lower=True,
strip=True):
         """
         Instantiates the preprocessor.
         """
         self.lower = lower
         self.strip = strip
         self.stopwords = set(stopwords) if stopwords else set(sw.words('english'))
         self.punct = set(punct) if punct else set(string.punctuation)
         self.lemmatizer = WordNetLemmatizer()
         self.max_sentence_len = max_sentence_len

      def fit(self, X, y=None):
         """
         Fit simply returns self.
         """
         return self

      def inverse_transform(self, X):
         """
         No inverse transformation.
         """
         return X

      def transform(self, X):
```

```
        """
        Actually runs the preprocessing on each document.
        """
        output = np.array([(self.tokenize(doc)) for doc in X])
        return output

    def tokenize(self, document):
        """
        Returns a normalized, lemmatized list of tokens from a document by
        applying segmentation, tokenization, and part of speech tagging.
        Uses the part of speech tags to look up the lemma in WordNet, and returns the lowercase
        version of all the words, removing stopwords and punctuation.
        """
        lemmatized_tokens = []

        # Clean the text
        document = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", document)
        document = re.sub(r"what's", "what is ", document)
        document = re.sub(r"\'s", " ", document)
        document = re.sub(r"\'ve", " have ", document)
        document = re.sub(r"can't", "cannot ", document)
        document = re.sub(r"n't", " not ", document)
        document = re.sub(r"i'm", "i am ", document)
        document = re.sub(r"\'re", " are ", document)
        document = re.sub(r"\'d", " would ", document)
        document = re.sub(r"\'ll", " will ", document)
        document = re.sub(r"(\d+)(k)", r"\g<1>000", document)

        # Break the document into sentences
        for sent in sent_tokenize(document):

            # Break the sentence into part of speech tagged tokens
            for token, tag in pos_tag(wordpunct_tokenize(sent)):

                # Apply preprocessing to the token
                token = token.lower() if self.lower else token
                token = token.strip() if self.strip else token
                token = token.strip('_') if self.strip else token
                token = token.strip('*') if self.strip else token

                # If punctuation or stopword, ignore token and continue
                if token in self.stopwords or all(char in self.punct for char in token):
                    continue

                # Lemmatize the token
```

```
                lemma = self.lemmatize(token, tag)
                lemmatized_tokens.append(lemma)

        doc = ' '.join(lemmatized_tokens)
        tokenized_document = self.vectorize(np.array(doc)[np.newaxis])
        return tokenized_document


    def vectorize(self, doc):
        """
        Returns a vectorized padded version of sequences.
        """
        save_path = "./Data/padding.pickle"
        with open(save_path, 'rb') as f:
            tokenizer = pickle.load(f)
        doc_pad = tokenizer.texts_to_sequences(doc)
        doc_pad = pad_sequences(doc_pad, padding='pre', truncating='pre',
maxlen=self.max_sentence_len)
        return np.squeeze(doc_pad)

    def lemmatize(self, token, tag):
        """
        Converts the Penn Treebank tag to a WordNet POS tag, then uses that
        tag to perform WordNet lemmatization.
        """
        tag = {
            'N': wn.NOUN,
            'V': wn.VERB,
            'R': wn.ADV,
            'J': wn.ADJ
        }.get(tag[0], wn.NOUN)

        return self.lemmatizer.lemmatize(token, tag)


class MyRNNTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer allowing our Keras model to be included in our pipeline
    """
    def __init__(self, classifier):
        self.classifier = classifier

    def fit(self, X, y):
        batch_size = 32
        num_epochs = 35
```

```
        batch_size = batch_size
        epochs = num_epochs
        self.classifier.fit(X, y, epochs=epochs, batch_size=batch_size, verbose=2)
        return self

    def transform(self, X):
        self.pred = self.classifier.predict(X)
        self.classes = [[0 if el < 0.2 else 1 for el in item] for item in self.pred]
        return self.pred

def multiclass_accuracy(self,predictions, target):
    "Returns the multiclass accuracy of the classifier's predictions"
    score = []
    for j in range(0, 5):
        count = 0
        for i in range(len(predictions)):
            if predictions[i][j] == target[i][j]:
                count += 1
        score.append(count / len(predictions))
    return score


def run(self, X, y, model_name):
    """
    Returns the predictions from the pipeline including our NLTKPreprocessor and Keras classifier.
    """
    def build(classifier):
        """
        Inner build function that builds a pipeline including a preprocessor and a classifier.
        """
        model = Pipeline([
                ('preprocessor', self.NLTKPreprocessor),
                ('classifier', classifier)
            ])
        return model

    save_path = './Models/'
    json_file = open(save_path + model_name + '.json', 'r')
    classifier = model_from_json(json_file.read())
    classifier.load_weights(save_path + model_name + '.h5')
    classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    json_file.close()
    model = build(self.MyRNNTransformer(classifier))
    y_pred = model.transform(X)
    y_pred_classes = [[0 if el < 0.2 else 1 for el in item] for item in y_pred]
```

```
        print(self.multiclass_accuracy(y.values.tolist(), y_pred_classes))

        return y_pred
pred = test().run(X_test, y_test, model_name = "Personality_traits_NN")
```

```
[0.6437246963562753, 0.771255060728745, 0.7125506072874493,
0.8117408906882592, 0.8502024291497976]
```

***Test SVM***

```
class test_svm:

    def __init__(self):
        self.max_sentence_len = 300
        self.NLTKPreprocessor = self.NLTKPreprocessor()
        #self.MyRNNTransformer = self.MyRNNTransformer()


    class NLTKPreprocessor(BaseEstimator, TransformerMixin):
        """
        Transforms input data by using NLTK tokenization, POS tagging, lemmatization and
vectorization.
        """

        def __init__(self, max_sentence_len = 300, stopwords=None, punct=None, lower=True,
strip=True):
            """
            Instantiates the preprocessor.
            """
            self.lower = lower
            self.strip = strip
            self.stopwords = set(stopwords) if stopwords else set(sw.words('english'))
            self.punct = set(punct) if punct else set(string.punctuation)
            self.lemmatizer = WordNetLemmatizer()
            self.max_sentence_len = max_sentence_len

        def fit(self, X, y=None):
            """
            Fit simply returns self.
            """
            return self

        def inverse_transform(self, X):
            """
```

```
        No inverse transformation.
        """
        return X

    def transform(self, X):
        """
        Actually runs the preprocessing on each document.
        """
        output = np.array([(self.tokenize(doc)) for doc in X])
        return output

    def tokenize(self, document):
        """
        Returns a normalized, lemmatized list of tokens from a document by
        applying segmentation, tokenization, and part of speech tagging.
        Uses the part of speech tags to look up the lemma in WordNet, and returns the lowercase
        version of all the words, removing stopwords and punctuation.
        """
        lemmatized_tokens = []

        # Clean the text
        document = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", document)
        document = re.sub(r"what's", "what is ", document)
        document = re.sub(r"\'s", " ", document)
        document = re.sub(r"\'ve", " have ", document)
        document = re.sub(r"can't", "cannot ", document)
        document = re.sub(r"n't", " not ", document)
        document = re.sub(r"i'm", "i am ", document)
        document = re.sub(r"\'re", " are ", document)
        document = re.sub(r"\'d", " would ", document)
        document = re.sub(r"\'ll", " will ", document)
        document = re.sub(r"(\d+)(k)", r"\g<1>000", document)

        # Break the document into sentences
        for sent in sent_tokenize(document):

            # Break the sentence into part of speech tagged tokens
            for token, tag in pos_tag(wordpunct_tokenize(sent)):

                # Apply preprocessing to the token
                token = token.lower() if self.lower else token
                token = token.strip() if self.strip else token
                token = token.strip('_') if self.strip else token
                token = token.strip('*') if self.strip else token
```

```
            # If punctuation or stopword, ignore token and continue
            if token in self.stopwords or all(char in self.punct for char in token):
                continue

            # Lemmatize the token
            lemma = self.lemmatize(token, tag)
            lemmatized_tokens.append(lemma)

        doc = ' '.join(lemmatized_tokens)
        tokenized_document = self.vectorize(np.array(doc)[np.newaxis])
        return tokenized_document


    def vectorize(self, doc):
        """
        Returns a vectorized padded version of sequences.
        """
        save_path = "./Data/padding.pickle"
        with open(save_path, 'rb') as f:
            tokenizer = pickle.load(f)
        doc_pad = tokenizer.texts_to_sequences(doc)
        doc_pad = pad_sequences(doc_pad, padding='pre', truncating='pre',
maxlen=self.max_sentence_len)
        return np.squeeze(doc_pad)

    def lemmatize(self, token, tag):
        """
        Converts the Penn Treebank tag to a WordNet POS tag, then uses that
        tag to perform WordNet lemmatization.
        """
        tag = {
            'N': wn.NOUN,
            'V': wn.VERB,
            'R': wn.ADV,
            'J': wn.ADJ
        }.get(tag[0], wn.NOUN)

        return self.lemmatizer.lemmatize(token, tag)


class MyRNNTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer allowing our Keras model to be included in our pipeline
    """
    def __init__(self, classifier):
```

```
        self.classifier = classifier

    def fit(self, X, y):
        batch_size = 32
        num_epochs = 35
        batch_size = batch_size
        epochs = num_epochs
        self.classifier.fit(X, y, epochs=epochs, batch_size=batch_size, verbose=2)
        return self

    def transform(self, X):
        self.pred = self.classifier.predict_proba(X)
        self.classes = [[0 if el < 0.2 else 1 for el in item] for item in self.pred]
        return self.pred


def identity(self, arg):
    """
    Simple identity function works as a passthrough.
    """
    return arg


def reshape_a_feature_column(self, series):
    return np.reshape(np.asarray(series), (len(series), 1))


def pipelinize_feature(self, function, active=True):
    def list_comprehend_a_function(list_or_series, active=True):
        if active:
            processed = [function(i) for i in list_or_series]
            processed = self.reshape_a_feature_column(processed)
            return processed
        else:
            return self.reshape_a_feature_column(np.zeros(len(list_or_series)))


def get_text_length(self, text):
    return len(text)


def multiclass_accuracy(self,predictions, target):
    "Returns the multiclass accuracy of the classifier's predictions"
    score = []
    for j in range(0, 5):
```

```
        count = 0
        for i in range(len(predictions)):
            if predictions[i][j] == target[i][j]:
                count += 1
        score.append(count / len(predictions))
    return score


  def run(self, X, y, model_name):
    """
    Returns the predictions from the pipeline including our NLTKPreprocessor and SVM classifier.
    """
    save_path = "./Models/"
    with open(save_path + model_name, 'rb') as f:
        model = dill.load(f)
    y_pred = model.predict_proba(X)
    y_pred_classes = [[0 if el < 0.2 else 1 for el in item] for item in y_pred]
    print(self.multiclass_accuracy(y.values.tolist(), y_pred_classes))

    return y_pred

pred = test_svm().run(X_test, y_test, "Personality_traits_SVM")
```

```
[0.5121457489878543, 0.5020242914979757, 0.5465587044534413,
0.5040485829959515, 0.4959514170040486]
```

**Predict single outputs**

***Predict with Keras model***

```
class predict:

  def __init__(self):
    self.max_sentence_len = 300
    self.max_features = 300
    self.embed_dim = 300
    self.NLTKPreprocessor = self.NLTKPreprocessor()


  class NLTKPreprocessor(BaseEstimator, TransformerMixin):
    """
    Transforms input data by using NLTK tokenization, POS tagging, lemmatization and
vectorization.
    """

    def __init__(self, max_sentence_len = 300, stopwords=None, punct=None, lower=True,
```

```
strip=True):
        """
        Instantiates the preprocessor.
        """
        self.lower = lower
        self.strip = strip
        self.stopwords = set(stopwords) if stopwords else set(sw.words('english'))
        self.punct = set(punct) if punct else set(string.punctuation)
        self.lemmatizer = WordNetLemmatizer()
        self.max_sentence_len = max_sentence_len

    def fit(self, X, y=None):
        """
        Fit simply returns self.
        """
        return self

    def inverse_transform(self, X):
        """
        No inverse transformation.
        """
        return X

    def transform(self, X):
        """
        Actually runs the preprocessing on each document.
        """
        output = np.array([(self.tokenize(doc)) for doc in X])
        return output

    def tokenize(self, document):
        """
        Returns a normalized, lemmatized list of tokens from a document by
        applying segmentation, tokenization, and part of speech tagging.
        Uses the part of speech tags to look up the lemma in WordNet, and returns the lowercase
        version of all the words, removing stopwords and punctuation.
        """
        lemmatized_tokens = []

        # Clean the text
        document = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", document)
        document = re.sub(r"what's", "what is ", document)
        document = re.sub(r"\'s", " ", document)
        document = re.sub(r"\'ve", " have ", document)
        document = re.sub(r"can't", "cannot ", document)
```

```
        document = re.sub(r"n't", " not ", document)
        document = re.sub(r"i'm", "i am ", document)
        document = re.sub(r"\'re", " are ", document)
        document = re.sub(r"\'d", " would ", document)
        document = re.sub(r"\'ll", " will ", document)
        document = re.sub(r"(\d+)(k)", r"\g<1>000", document)

        # Break the document into sentences
        for sent in sent_tokenize(document):

            # Break the sentence into part of speech tagged tokens
            for token, tag in pos_tag(wordpunct_tokenize(sent)):

                # Apply preprocessing to the token
                token = token.lower() if self.lower else token
                token = token.strip() if self.strip else token
                token = token.strip('_') if self.strip else token
                token = token.strip('*') if self.strip else token

                # If punctuation or stopword, ignore token and continue
                if token in self.stopwords or all(char in self.punct for char in token):
                    continue

                # Lemmatize the token
                lemma = self.lemmatize(token, tag)
                lemmatized_tokens.append(lemma)

        doc = ' '.join(lemmatized_tokens)
        tokenized_document = self.vectorize(np.array(doc)[np.newaxis])
        return tokenized_document


    def vectorize(self, doc):
        """
        Returns a vectorized padded version of sequences.
        """
        save_path = "./Data/padding.pickle"
        with open(save_path, 'rb') as f:
            tokenizer = pickle.load(f)
        doc_pad = tokenizer.texts_to_sequences(doc)
        doc_pad = pad_sequences(doc_pad, padding='pre', truncating='pre',
maxlen=self.max_sentence_len)
        return np.squeeze(doc_pad)

    def lemmatize(self, token, tag):
```

```
        """
        Converts the Penn Treebank tag to a WordNet POS tag, then uses that
        tag to perform WordNet lemmatization.
        """
        tag = {
            'N': wn.NOUN,
            'V': wn.VERB,
            'R': wn.ADV,
            'J': wn.ADJ
        }.get(tag[0], wn.NOUN)

        return self.lemmatizer.lemmatize(token, tag)


class MyRNNTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer allowing our Keras model to be included in our pipeline
    """
    def __init__(self, classifier):
        self.classifier = classifier

    def fit(self, X, y):
        batch_size = 32
        num_epochs = 35
        batch_size = batch_size
        epochs = num_epochs
        self.classifier.fit(X, y, epochs=epochs, batch_size=batch_size, verbose=2)
        return self

    def transform(self, X):
        self.pred = self.classifier.predict(X)
        self.classes = [[0 if el < 0.2 else 1 for el in item] for item in self.pred]
        return self.pred


def run(self, X, model_name):
    """
    Returns the predictions from the pipeline including our NLTKPreprocessor and Keras classifier.
    """
    def build(classifier):
        """
        Inner build function that builds a pipeline including a preprocessor and a classifier.
        """
        model = Pipeline([
            ('preprocessor', self.NLTKPreprocessor),
```

```
            ('classifier', classifier)
          ])
        return model

    save_path = './Models/'
    json_file = open(save_path + model_name + '.json', 'r')
    classifier = model_from_json(json_file.read())
    classifier.load_weights(save_path + model_name + '.h5')
    classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    json_file.close()
    model = build(self.MyRNNTransformer(classifier))
    y_pred = model.transform([X])

    return y_pred

pred = predict().run(X_test[0], model_name = "Personality_traits_NN")
```

```
['I hate this type of assignment, the kind of work that tries to force a
thought into one\'s head. Thoughts should come naturally, and when the
mind is prompted, then nothing comes to it. Well, as long as I\'m
supposed to be thinking, let\'s think about why I\'m doing this. It\'s to
get a good grade. Why do I do anything. I always seem to respond in a
manner in which the answer seems acceptable to everyone else. Maybe I\'m
doing this to please everyone else: my instructor, my parents, myself. So
why do I do everything else that I do?  Why do I devote so much time to
working out and exercise?  Is it because I want to fool myself into
living longer?  Is it to mold my body in such a way as to attract others
and get their attentions. I always say that I just don\'t want to become
weak, that when I find someone to protect, I want to be able to protect
them. But if I am always desparately trying to find this person that
would fill the current void in my life, why is it I push everyone away?
My friends, my peers, my elders?  I don\'t do what everyone says is "fun.
"  I don\'t hang around others for long, I don\'t go to parties, and I
rarely impart what I truly feel. It seems as if I\'ve been hiding half my
life, hiding from others and from myself. Inside, I scream to myself, "go
out have fun meet new people you boring dolt!"  and in my mind I\'m
always prepared to do so, but when the moment of truth comes about, I
never act out my true intentions, either being intimidated or
discouraged. I remember when I went to that UT freshman orientation. My
friend said when you get to the dance, just go up to people, introduce
yourself, and ask for a dance. And he\'s a loser. But it worked for him,
so I tried it myself. The first time, rejected. The second time,
rejected. The third time, rejected, and even I know that 3 strikes is
out. I get so sick of hearing excuses like oh I\'m too tired or I don\'t
feel like it. Yeah right. Then why the hell are you here at the dance?
Just give me a straight and vehement no and I\'ll be on my way, not even
```

```
to think twice about the encounter. Now on strike three, I had another
reason for retiring that night. That girl, the way she looked, the way
she acted, it was deja vu. I went to another country with a girl like
that, stood by her side, opened my heart to her, and offered it to her.
She led me on, to believe that for once, I had an opportunity for love,
that it was my time in the spotlight, and with the most beautiful girl I
have ever been with. Visually stunning, patient but forceful, and gentle,
someone to listen. I thought I finally found her, and felt as though I
was riding above clouds. Then I just came down hard with the rain. I\'ve
never been the same afterwards, because everything was seen in a
different light. This third girl at the dance, the rudeness didn\'t get
to me, but the memories of heartbreak was just too much to handle. I
asked Tonia out to the dance once, and she came. But we never danced. I
asked and she ignored. Just like This time. Love. I\'m too young to know
its true meanings, and too inexperienced to have any justification in
even corrolating it with the words I or me. I\'m always looking for
"love" for companionship without even really knowing what to look for. It
can\'t just be the person that gives you that indescribable burning
feeling within. Too many times has that happened. I say to myself, you
know what love is. You loved Tonia, because all you ever did was argue,
yet you could stand it. She broke your heart, lied to you, worked behind
your back, carried you up and dropped you, and yet you still thought of
her. Hell, you still think of her. You\'ve been talking to her as a
friend and the feeling has never left you. Isn\'t that enough to be love?
It\'s as close as I\'ve gotten, I\'m sure, but what about prom. To have
danced with someone I\'ve never known before, to talk, and to, after so
long, have fun. When I held her in my arms, and her hair brushed my face
as the slow songs played through, what was that feeling. Was it a feeling
of completeness, of safety that as long as with her that nothing can go
wrong?  What was that?  I go through sleepless nights thinking of
Christie sometimes, someone who I never gave a second thought to until
that moment I told her goodbye. The moment in which the swelling in my
throat caught even me by surprise. But am I looking for love or just
someone. I don\'t want to end up like my uncle, having married a woman
only because she threatened to commit suicide. Is he happy?  Am I happy
now?  no, I\'m not. ']
```

### *Predict with SVM*

```
class predict_svm:

  def __init__(self):
    self.max_sentence_len = 300
    self.NLTKPreprocessor = self.NLTKPreprocessor()
    #self.MyRNNTransformer = self.MyRNNTransformer()
```

```
class NLTKPreprocessor(BaseEstimator, TransformerMixin):
    """
    Transforms input data by using NLTK tokenization, POS tagging, lemmatization and
vectorization.
    """

    def __init__(self, max_sentence_len = 300, stopwords=None, punct=None, lower=True,
strip=True):
        """
        Instantiates the preprocessor.
        """
        self.lower = lower
        self.strip = strip
        self.stopwords = set(stopwords) if stopwords else set(sw.words('english'))
        self.punct = set(punct) if punct else set(string.punctuation)
        self.lemmatizer = WordNetLemmatizer()
        self.max_sentence_len = max_sentence_len

    def fit(self, X, y=None):
        """
        Fit simply returns self.
        """
        return self

    def inverse_transform(self, X):
        """
        No inverse transformation.
        """
        return X

    def transform(self, X):
        """
        Actually runs the preprocessing on each document.
        """
        output = np.array([(self.tokenize(doc)) for doc in X])
        return output

    def tokenize(self, document):
        """
        Returns a normalized, lemmatized list of tokens from a document by
        applying segmentation, tokenization, and part of speech tagging.
        Uses the part of speech tags to look up the lemma in WordNet, and returns the lowercase
        version of all the words, removing stopwords and punctuation.
```

```
        """
        lemmatized_tokens = []

        # Clean the text
        document = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", document)
        document = re.sub(r"what's", "what is ", document)
        document = re.sub(r"\'s", " ", document)
        document = re.sub(r"\'ve", " have ", document)
        document = re.sub(r"can't", "cannot ", document)
        document = re.sub(r"n't", " not ", document)
        document = re.sub(r"i'm", "i am ", document)
        document = re.sub(r"\'re", " are ", document)
        document = re.sub(r"\'d", " would ", document)
        document = re.sub(r"\'ll", " will ", document)
        document = re.sub(r"(\d+)(k)", r"\g<1>000", document)

        # Break the document into sentences
        for sent in sent_tokenize(document):

            # Break the sentence into part of speech tagged tokens
            for token, tag in pos_tag(wordpunct_tokenize(sent)):

                # Apply preprocessing to the token
                token = token.lower() if self.lower else token
                token = token.strip() if self.strip else token
                token = token.strip('_') if self.strip else token
                token = token.strip('*') if self.strip else token

                # If punctuation or stopword, ignore token and continue
                if token in self.stopwords or all(char in self.punct for char in token):
                    continue

                # Lemmatize the token
                lemma = self.lemmatize(token, tag)
                lemmatized_tokens.append(lemma)

        doc = ' '.join(lemmatized_tokens)
        tokenized_document = self.vectorize(np.array(doc)[np.newaxis])
        return tokenized_document


    def vectorize(self, doc):
        """
        Returns a vectorized padded version of sequences.
        """
```

```
        save_path = "./Data/padding.pickle"
        with open(save_path, 'rb') as f:
            tokenizer = pickle.load(f)
        doc_pad = tokenizer.texts_to_sequences(doc)
        doc_pad = pad_sequences(doc_pad, padding='pre', truncating='pre',
maxlen=self.max_sentence_len)
        return np.squeeze(doc_pad)

    def lemmatize(self, token, tag):
        """
        Converts the Penn Treebank tag to a WordNet POS tag, then uses that
        tag to perform WordNet lemmatization.
        """
        tag = {
            'N': wn.NOUN,
            'V': wn.VERB,
            'R': wn.ADV,
            'J': wn.ADJ
        }.get(tag[0], wn.NOUN)

        return self.lemmatizer.lemmatize(token, tag)


class MyRNNTransformer(BaseEstimator, TransformerMixin):
    """
    Transformer allowing our Keras model to be included in our pipeline
    """
    def __init__(self, classifier):
        self.classifier = classifier

    def fit(self, X, y):
        batch_size = 32
        num_epochs = 35
        batch_size = batch_size
        epochs = num_epochs
        self.classifier.fit(X, y, epochs=epochs, batch_size=batch_size, verbose=2)
        return self

    def transform(self, X):
        self.pred = self.classifier.predict_proba(X)
        self.classes = [[0 if el < 0.2 else 1 for el in item] for item in self.pred]
        return self.pred


def identity(self, arg):
```

```
    """
    Simple identity function works as a passthrough.
    """
    return arg


def reshape_a_feature_column(self, series):
    return np.reshape(np.asarray(series), (len(series), 1))


def pipelinize_feature(self, function, active=True):
    def list_comprehend_a_function(list_or_series, active=True):
        if active:
            processed = [function(i) for i in list_or_series]
            processed = self.reshape_a_feature_column(processed)
            return processed
        else:
            return self.reshape_a_feature_column(np.zeros(len(list_or_series)))


def get_text_length(self, text):
    return len(text)


def multiclass_accuracy(self,predictions, target):
    "Returns the multiclass accuracy of the classifier's predictions"
    score = []
    for j in range(0, 5):
        count = 0
        for i in range(len(predictions)):
            if predictions[i][j] == target[i][j]:
                count += 1
        score.append(count / len(predictions))
    return score


def run(self, X, model_name):
    """
    Returns the predictions from the pipeline including our NLTKPreprocessor and SVM classifier.
    """
    save_path = "./Models/"
    with open(save_path + model_name, 'rb') as f:
        model = dill.load(f)
    y_pred = model.predict_proba([X])
```

```
    return y_pred

pred = predict_svm().run(X_test[4], model_name = "Personality_traits_SVM")
```

## Performance

We tried different baseline models in order to assess the performance of our final architecture. Here are the accuracies of the different models.

| Model | EXT | NEU | AGR | CON | OPN |
|---|---|---|---|---|---|
| TF-IDF + MNB | 45.34 | 45.11 | 45.24 | 45.31 | 45.12 |
| TF-IDF + SVM | 45.78 | 45.91 | 45.41 | 45.54 | 45.56 |
| Word2Vec + MNB | 45.02 | 46.01 | 46.34 | 46.38 | 45.97 |
| Word2Vec + SVM | 46.18 | 48.21 | 49.65 | 49.97 | 50.07 |
| Word2Vec (TF-IDF averaging) + MNB | 45.87 | 44.99 | 45.38 | 44.21 | 44.84 |
| Word2Vec (TF-IDF averaging) + SVM | 46.01 | 46.19 | 47.56 | 48.11 | 48.89 |
| Word2Vec + NN (LSTM) | 51.98 | 50.01 | 51.57 | 51.11 | 50.51 |
| Word2Vec + NN (CONV + LSTM) | **55.07** | **50.17** | **54.57** | **53.23** | **53.84** |

# Facial Emotion Recognition

The aim of this section is to explore facial emotion recognition techniques from a live webcam video stream.

## Video Processing

### Pipeline

The video processing pipeline was built the following way :

1. Launch the webcam
2. Identify the face by Histogram of Oriented Gradients

3. Zoom on the face
4. Dimension the face to 48 * 48 pixels
5. Make a prediction on the face using our pre-trained model
6. Also identify the number of blinks on the facial landmarks on each picture

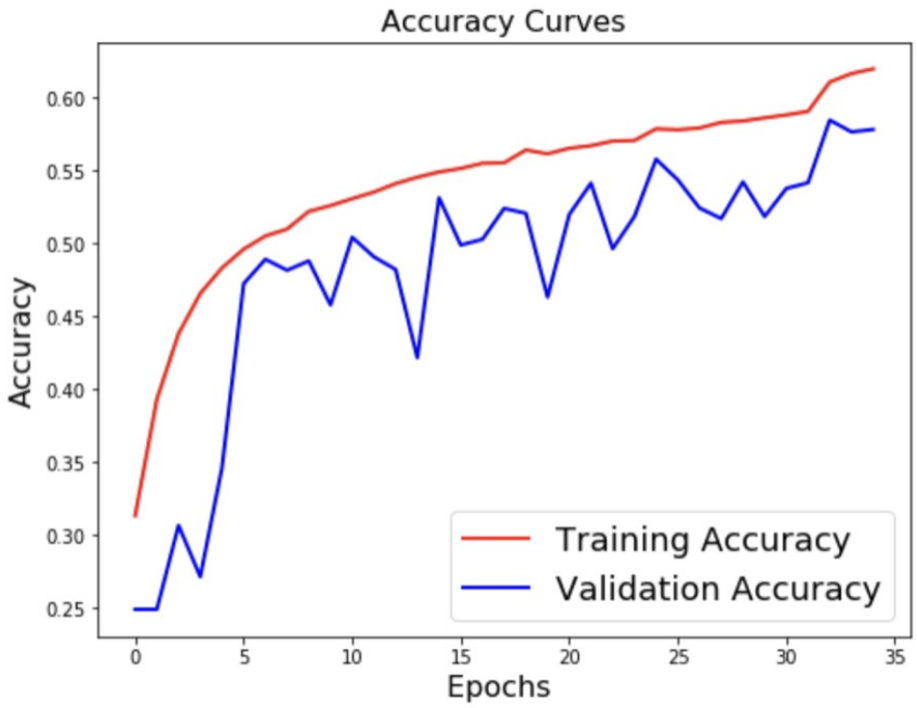
## Model

The model we have chosen is an XCeption model, since it outperformed the other approaches we developed so far. We tuned the model with :

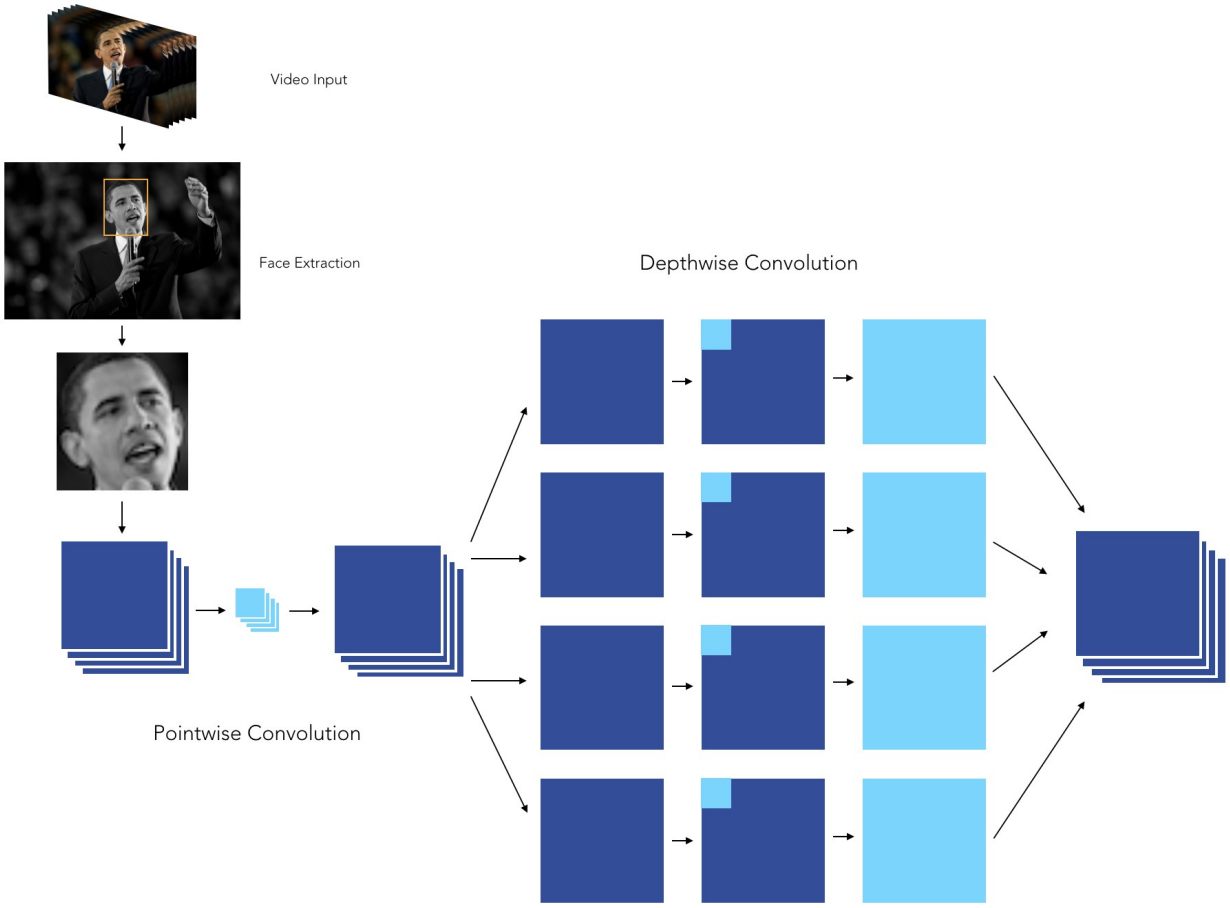
- Data augmentation
- Early stopping
- Decreasing learning rate on plateau
- L2-Regularization
- Class weight balancing
- And kept the best model

As you might have understood, the aim was to limit overfitting as much as possible in order to obtain a robust model.

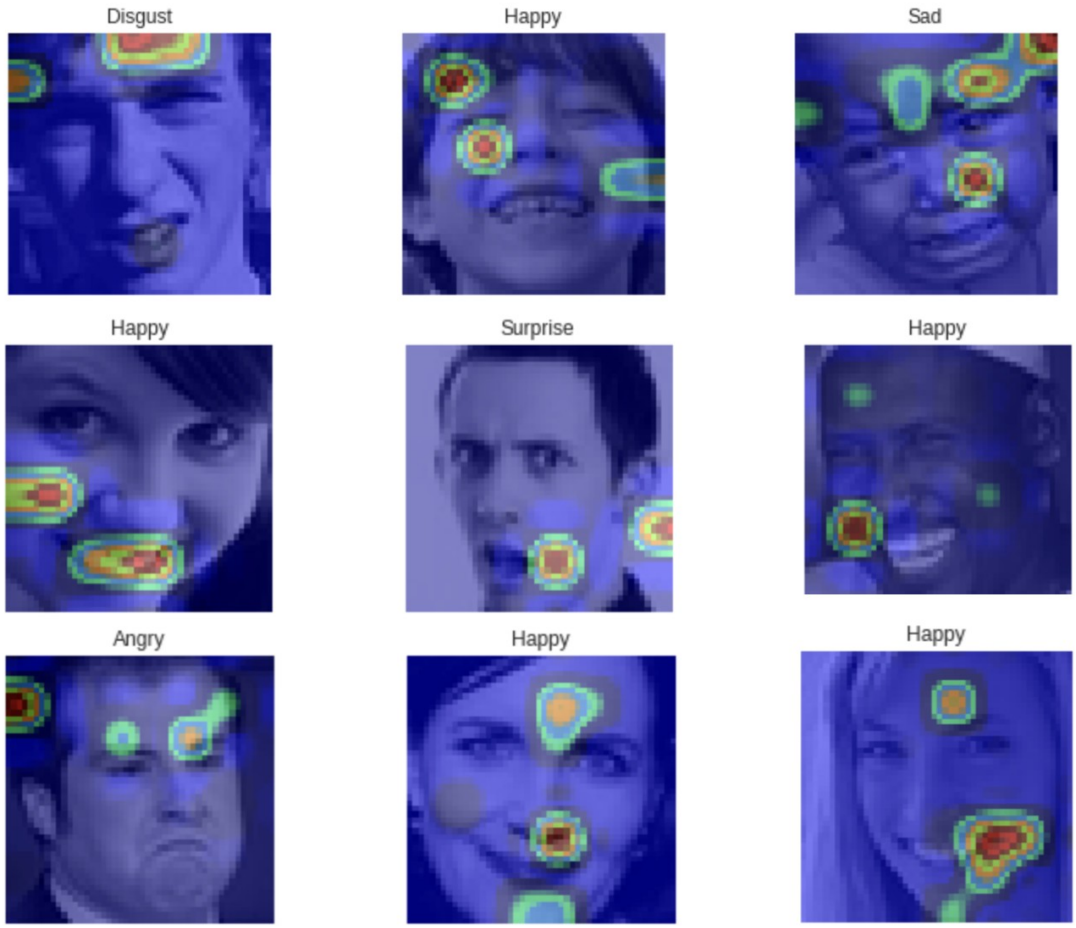
- To know more on how we prevented overfitting, check this article : https://maelfabien.github.io/deeplearning/regu/
- To know more on the XCeption model, check this article : https://maelfabien.github.io/deeplearning/xception/




The XCeption architecture is based on DepthWise Separable convolutions that allow to train much fewer parameters, and therefore reduce training time on Colab's GPUs to less than 90 minutes.




When it comes to applying CNNs in real life application, being able to explain the results is a great challenge. We can indeed plot class activation maps, which display the pixels that have been activated by the last convolution layer. We notice how the pixels are being activated differently depending on the emotion being labeled. The happiness seems to depend on the pixels linked to the eyes and mouth, whereas the sadness or the anger seem for example to be more related to the eyebrows.




# Notebooks

Among the notebooks, the role of each notebook is the following :

- `01-Pre-Processing.ipynb` : Transform the initial CSV file into train and test data sets
- `02-HOG_Features.ipynb` : A manual extraction of features (Histograms of Oriented Gradients, Landmarks) and SVM
- `03-Pre-Processing-EmotionalDAN.ipynb` : An implementation of Deep Alignment Networks to extract features
- `04-LGBM.ipynb` : Use of classical Boosting techniques on top on flatenned image or auto-encoded image
- `05-Simple_Arch.ipynb` : A simple Deep Learning Architecture
- `06-Inception.ipynb` : An implementation of the Inception Architecture
- `07-Xception.ipynb` : An implementation of the Xception Architecture
- `08-DeXpression.ipynb` : An implementation of the DeXpression Architecture
- `09-Prediction.ipynb` : Live Webcam prediction of the model
- `10-Hybrid.ipynb` : A hybrid deep learning model taking both the HOG/Landmarks model and the image

Instead of laying all the Notebooks one by one in front of you, it would be better if you could better able to see all in one go. So here is everything.

# I. Context

The models explored include :

- Manual filters
- Deep Learning Architectures
- DenseNet Inspired Architectures

This model will be combined with voice emotion recongition as well as psychological traits extracted from text inputs, and should provide a benchmark and a deep analysis of both verbal and non-verbal insights for candidates seeking for a job and their performance during an interview.

# II. General imports

```
### General imports ###
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from time import time
from time import sleep
import re
import os
import argparse
from collections import OrderedDict
import matplotlib.animation as animation

### Image processing ###
from scipy.ndimage import zoom
from scipy.spatial import distance
import imutils
from scipy import ndimage
import cv2
import dlib
from __future__ import division
from imutils import face_utils

### CNN models ###
import keras
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array,
load_img
from keras.callbacks import TensorBoard
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D, SeparableConv2D
```

```
from keras.utils import np_utils
from keras.regularizers import l2#, activity_l2
from keras.optimizers import SGD, RMSprop
from keras.utils import to_categorical
from keras.layers.normalization import BatchNormalization
from keras import models
from keras.utils.vis_utils import plot_model
from keras.layers import Input, GlobalAveragePooling2D
from keras.models import Model
from tensorflow.keras import layers

### Build SVM models ###
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import svm

### Same trained models ###
import h5py
from keras.models import model_from_json
import pickle
```

## III. Import datas

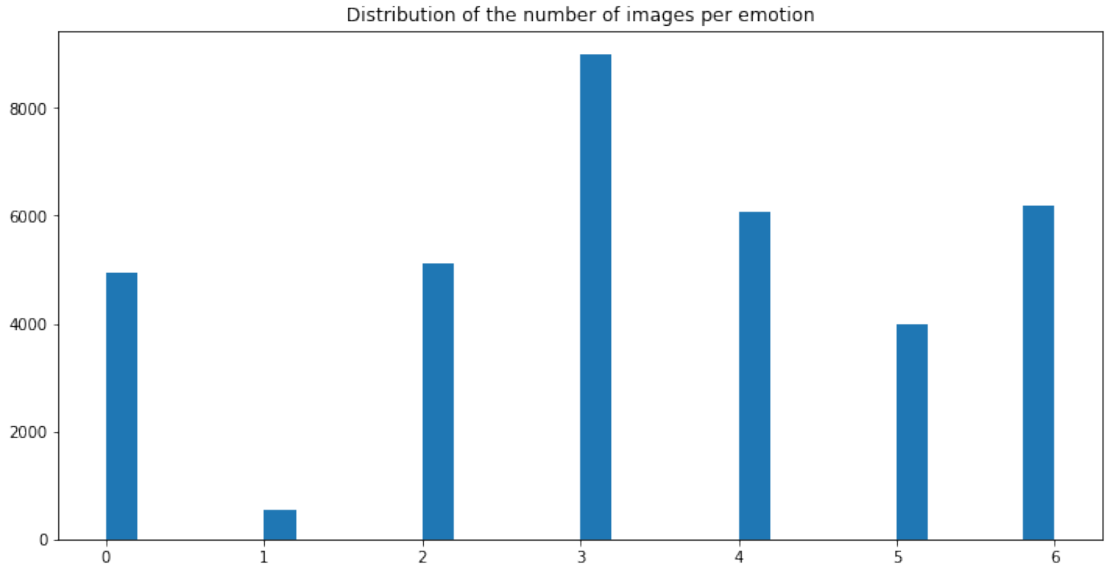
```
path = '/Users/maelfabien/filrouge_pole_emploi/Video/'
local_path = '/Users/maelfabien/Desktop/LocalDB/Videos/'

pd.options.mode.chained_assignment = None  # default='warn'  #to suppress
SettingWithCopyWarning

#Reading the dataset
dataset = pd.read_csv(local_path + 'fer2013.csv')

#Obtaining train data where usage is "Training"
train = dataset[dataset["Usage"] == "Training"]

#Obtaining test data where usage is "PublicTest"
test = dataset[dataset["Usage"] == "PublicTest"]

#Converting " " separated pixel values to list
train['pixels'] = train['pixels'].apply(lambda image_px : np.fromstring(image_px, sep = ' '))
test['pixels'] = test['pixels'].apply(lambda image_px : np.fromstring(image_px, sep = ' '))

dataset.head()
```

```
   emotion                                                pixels     Usage
0        0  70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...  Training
1        0  151 150 147 155 148 133 111 140 170 174 182 15...  Training
2        2  231 212 156 164 174 138 161 173 182 200 106 38...  Training
3        4  24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...  Training
4        6  4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...  Training
```

```
dataset[dataset['emotion'] == 1].head()
```

```
     emotion                                                pixels     Usage
299        1  126 126 129 120 110 168 174 172 173 174 170 15...  Training
388        1  89 55 24 40 43 48 53 55 59 41 33 31 22 32 42 4...  Training
416        1  204 195 181 131 50 50 57 56 66 98 138 161 173 ...  Training
473        1  14 11 13 12 41 95 113 112 111 122 132 137 142 ...  Training
533        1  18 25 49 75 89 97 100 100 101 103 105 107 107 ...  Training
```

```
plt.figure(figsize=(12,6))
plt.hist(dataset['emotion'], bins=30)
plt.title("Distribution of the number of images per emotion")
plt.show()
```




```
train.shape
```

```
(28709, 3)
```

```
test.shape
```

```
(3589, 3)
```

## IV. Create the data set

```
shape_x = 48
shape_y = 48

X_train = train.iloc[:, 1].values
y_train = train.iloc[:, 0].values
X_test = test.iloc[:, 1].values
y_test = test.iloc[:, 0].values

#np.vstack stack arrays in sequence vertically (picking element row wise)
X_train = np.vstack(X_train)
X_test = np.vstack(X_test)

#Reshape X_train, y_train,X_test,y_test in desired formats
X_train = np.reshape(X_train, (X_train.shape[0],48,48,1))
y_train = np.reshape(y_train, (y_train.shape[0],1))
X_test = np.reshape(X_test, (X_test.shape[0],48,48,1))
y_test = np.reshape(y_test, (y_test.shape[0],1))

print("Shape of X_train and y_train is " + str(X_train.shape) +" and " + str(y_train.shape) +"
respectively.")
print("Shape of X_test and y_test is " + str(X_test.shape) +" and " + str(y_test.shape) +"
respectively.")
```

```
Shape of X_train and y_train is (28709, 48, 48, 1) and (28709, 1) respectively.
Shape of X_test and y_test is (3589, 48, 48, 1) and (3589, 1) respectively.
```

```
# Change to float datatype
train_data = X_train.astype('float32')
test_data = X_test.astype('float32')

# Scale the data to lie between 0 to 1
train_data /= 255
test_data /= 255

# Change the labels from integer to categorical data
train_labels_one_hot = to_categorical(y_train)
test_labels_one_hot = to_categorical(y_test)
```

## V. Define the number of classes

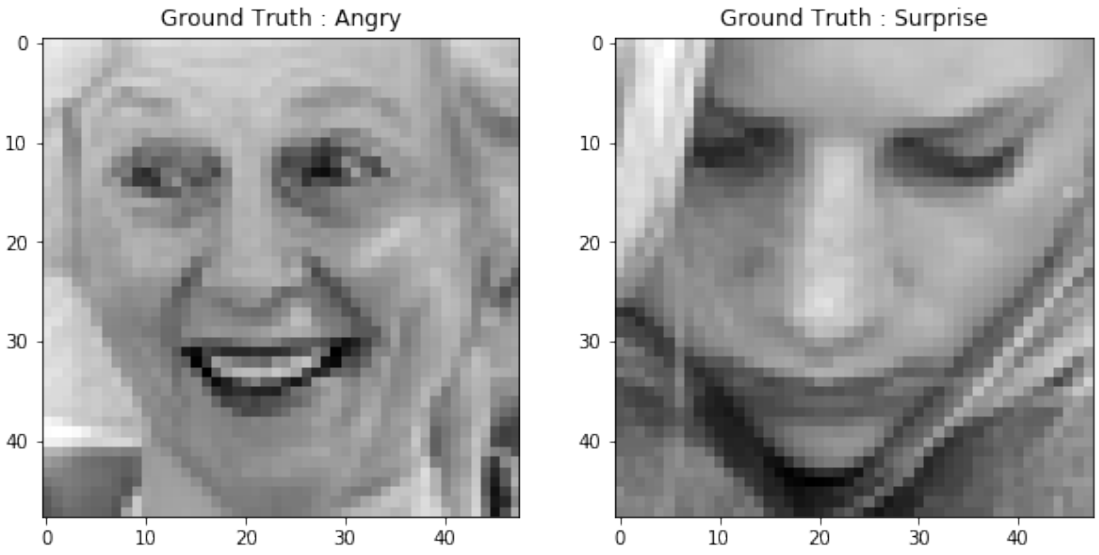
```
# Find the unique numbers from the train labels
classes = np.unique(y_train)
nClasses = len(classes)
```

```
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)

# Find the shape of input images and create the variable input_shape
nRows,nCols,nDims = X_train.shape[1:]
input_shape = (nRows, nCols, nDims)
```

```
Total number of outputs :  7
Output classes :  [0 1 2 3 4 5 6]
```

```
#Defining labels
def get_label(argument):
    labels = {0:'Angry', 1:'Disgust', 2:'Fear', 3:'Happy', 4:'Sad' , 5:'Surprise', 6:'Neutral'}
    return(labels.get(argument, "Invalid emotion"))

plt.figure(figsize=[10,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(np.squeeze(X_train[25,:,:], axis = 2), cmap='gray')
plt.title("Ground Truth : {}".format(get_label(int(y_train[0]))))

# Display the first image in testing data
plt.subplot(122)
plt.imshow(np.squeeze(X_test[26,:,:], axis = 2), cmap='gray')
plt.title("Ground Truth : {}".format(get_label(int(y_test[1500]))))
```

```
Text(0.5, 1.0, 'Ground Truth : Surprise')
```



## VI. Detect Faces

First of all, we need to detect the faces inside an image. This will allow us to :

- focus on the region of the face
- stop the prediction if no face is recognized.

To do so, we use OpenCV faceCascade classifier. Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

The term "Cascade" comes from the fact that when a window is explored and no face edge is identified, the region is left apart and we move on to the next one using Adaboost classifier. This makes the overall process very efficient.

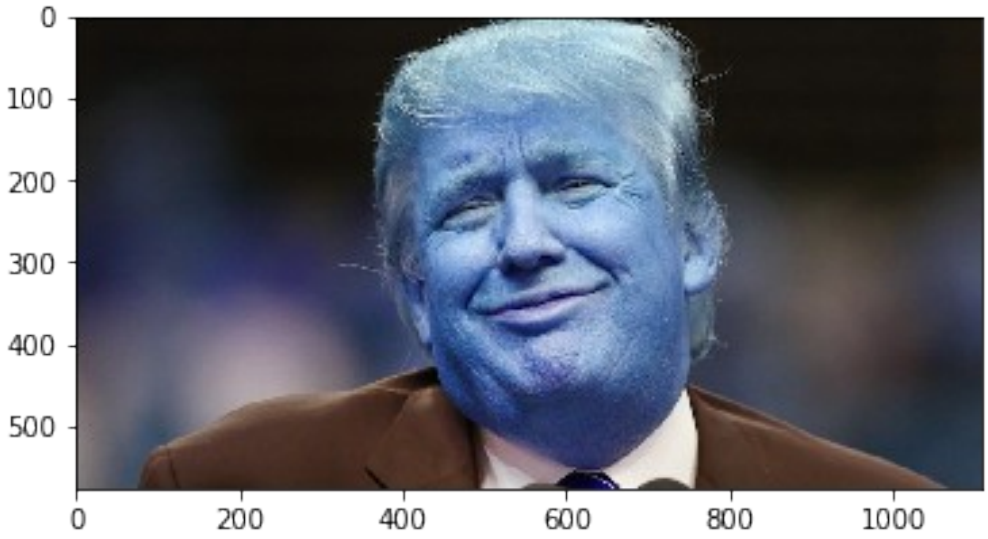
```
def detect_face(frame):

    #Cascade classifier pre-trained model
    cascPath =
'/usr/local/lib/python3.7/site-packages/cv2/data/haarcascade_frontalface_default.xml'
    faceCascade = cv2.CascadeClassifier(cascPath)

    #BGR -> Gray conversion
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    #Cascade MultiScale classifier
    detected_faces = faceCascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=6,
                                        minSize=(shape_x, shape_y),
                                        flags=cv2.CASCADE_SCALE_IMAGE)
    coord = []

    for x, y, w, h in detected_faces :
        if w > 100 :
            sub_img=frame[y:y+h,x:x+w]
            #cv2.rectangle(frame,(x,y),(x+w,y+h),(0, 255,255),1)
            coord.append([x,y,w,h])

    return gray, detected_faces, coord

#Extraire les features faciales
def extract_face_features(faces, offset_coefficients=(0.075, 0.05)):
    gray = faces[0]
    detected_face = faces[1]

    new_face = []
```

```
  for det in detected_face :
    #Region dans laquelle la face est détectée
    x, y, w, h = det
    #X et y correspondent à la conversion en gris par gray, et w, h correspondent à la
hauteur/largeur

    #Offset coefficient, np.floor takes the lowest integer (delete border of the image)
    horizontal_offset = np.int(np.floor(offset_coefficients[0] * w))
    vertical_offset = np.int(np.floor(offset_coefficients[1] * h))

    #gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    #gray transforme l'image
    extracted_face = gray[y+vertical_offset:y+h, x+horizontal_offset:x-horizontal_offset+w]

    #Zoom sur la face extraite
    new_extracted_face = zoom(extracted_face, (shape_x / extracted_face.shape[0],shape_y /
extracted_face.shape[1]))
    #cast type float
    new_extracted_face = new_extracted_face.astype(np.float32)
    #scale
    new_extracted_face /= float(new_extracted_face.max())
    #print(new_extracted_face)

    new_face.append(new_extracted_face)

  return new_face
```

Initial picture :

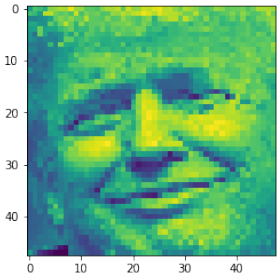
```
trump = '/Users/maelfabien/filrouge_pole_emploi/Video/test_samples/trump.jpg'
trump_face = cv2.imread(trump, cv2.COLOR_BGR2RGB)
plt.imshow(trump_face)
```

```
<matplotlib.image.AxesImage at 0x1a32e6e6a0>
```




Extracted face :

```
face = extract_face_features(detect_face(trump_face))[0]
plt.imshow(face)
```




```
<matplotlib.image.AxesImage at 0x1a32ec6dd8>
```

## VII. Deep Learning Model architectures

### *1. A first simple model*

```
def createModel():

    #Model Initialization
    model = Sequential()

    #Adding Input Layer
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=input_shape))

    #Adding more layers
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
```

```
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    #Flattening
    model.add(Flatten())

    #Adding fully connected layer
    model.add(Dense(512, activation='relu'))

    model.add(Dropout(0.6))

    #Adding Output Layer
    model.add(Dense(nClasses, activation='softmax'))

    return model
```

### *2. Prevent Overfitting*

```
def createModel2():

    #Model Initialization
    model = Sequential()

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())

    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))

    #Flattening
    model.add(Flatten())
```

```
    #Adding fully connected layer
    model.add(Dense(512, activation='relu'))

    #Adding Output Layer
    model.add(Dense(nClasses, activation='softmax'))

    return model
```

### *3. Go Deeper*

```
def createModel3():

    #Model Initialization
    model = Sequential()

    model.add(Conv2D(20, (3, 3), padding='same', activation='relu', input_shape=input_shape))
    model.add(Conv2D(30, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Conv2D(40, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(50, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Conv2D(60, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(70, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(80, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(90, (3, 3), padding='same', activation='relu'))

    #Flattening
    model.add(Flatten())

    #Adding fully connected layer
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(512, activation='relu'))

    #Adding Output Layer
```

```
    model.add(Dense(nClasses, activation='softmax'))

    return model
```

### *4. Build Model*

```
model = createModel3()
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 20)        200
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 48, 48, 30)        5430
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 30)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 30)        120
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 30)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 40)        10840
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 50)        18050
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 50)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 50)        200
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 50)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 60)        27060
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 70)        37870
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 70)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 70)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 80)          50480
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 6, 6, 90)          64890
```

```
_________________________________________________________________
flatten_1 (Flatten)          (None, 3240)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              3241000
_________________________________________________________________
dense_2 (Dense)              (None, 512)               512512
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 3591
=================================================================
Total params: 3,972,243
Trainable params: 3,972,083
Non-trainable params: 160
_________________________________________________________________
```

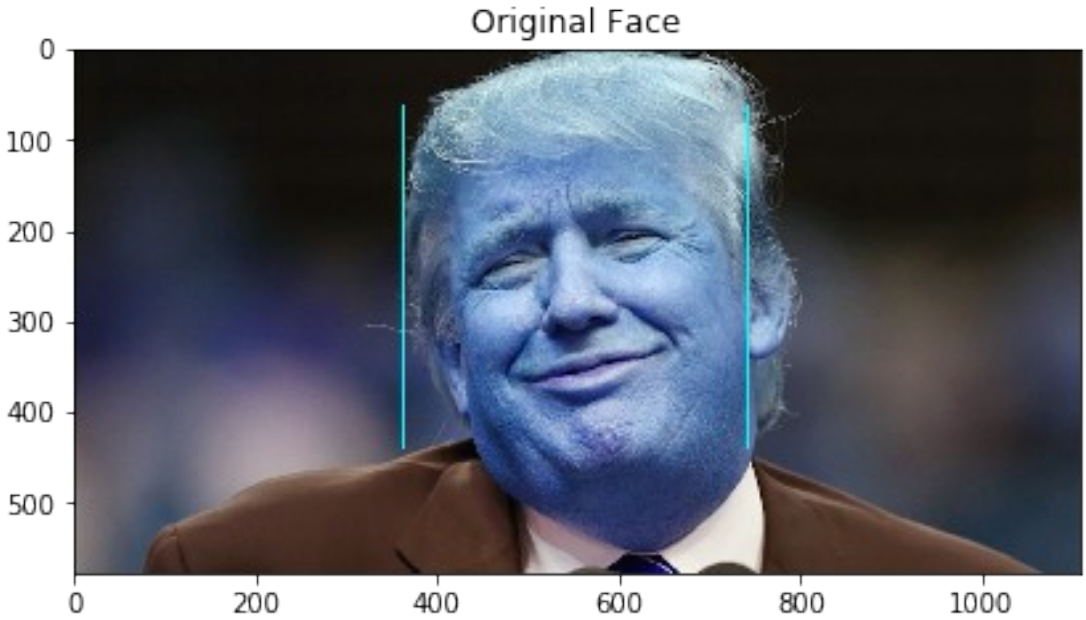
And visualize the model architecture :

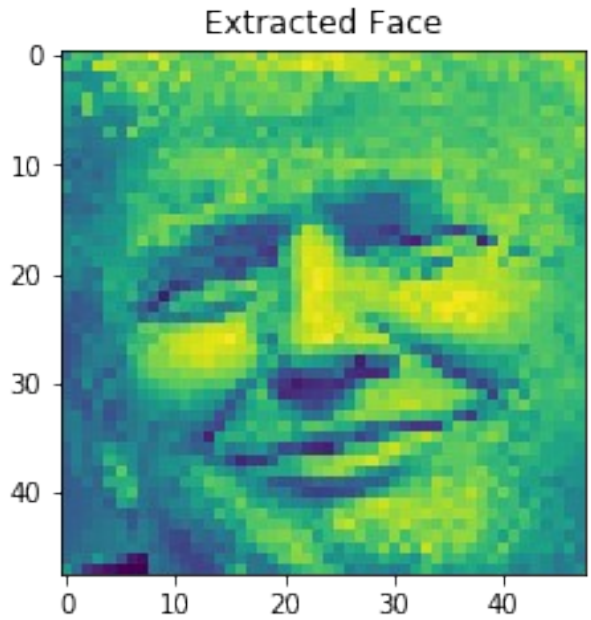
```
plot_model(model, to_file='model_images/model_plot.png', show_shapes=True,
show_layer_names=True)
```

## VIII. Visualize layers and output

```
layer_outputs = [layer.output for layer in model.layers[:12]]
# Extracts the outputs of the top 12 layers
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)

layer_names = []
for layer in model.layers[:12]:
    layer_names.append(layer.name) # Names of the layers, so you can have them as part of your
plot


images_per_row = 16

trump = '/Users/maelfabien/filrouge_pole_emploi/Video/test_samples/trump.jpg'
trump_face = cv2.imread(trump)
face = extract_face_features(detect_face(trump_face))[0]

to_predict = np.reshape(face.flatten(), (1,48,48,1))
res = model.predict(to_predict)
activations = activation_model.predict(to_predict)

plt.figure(figsize=(12,8))

plt.subplot(211)
plt.title("Original Face")
plt.imshow(trump_face)

plt.subplot(212)
```

```
plt.title("Extracted Face")
plt.imshow(face)

plt.show()
```

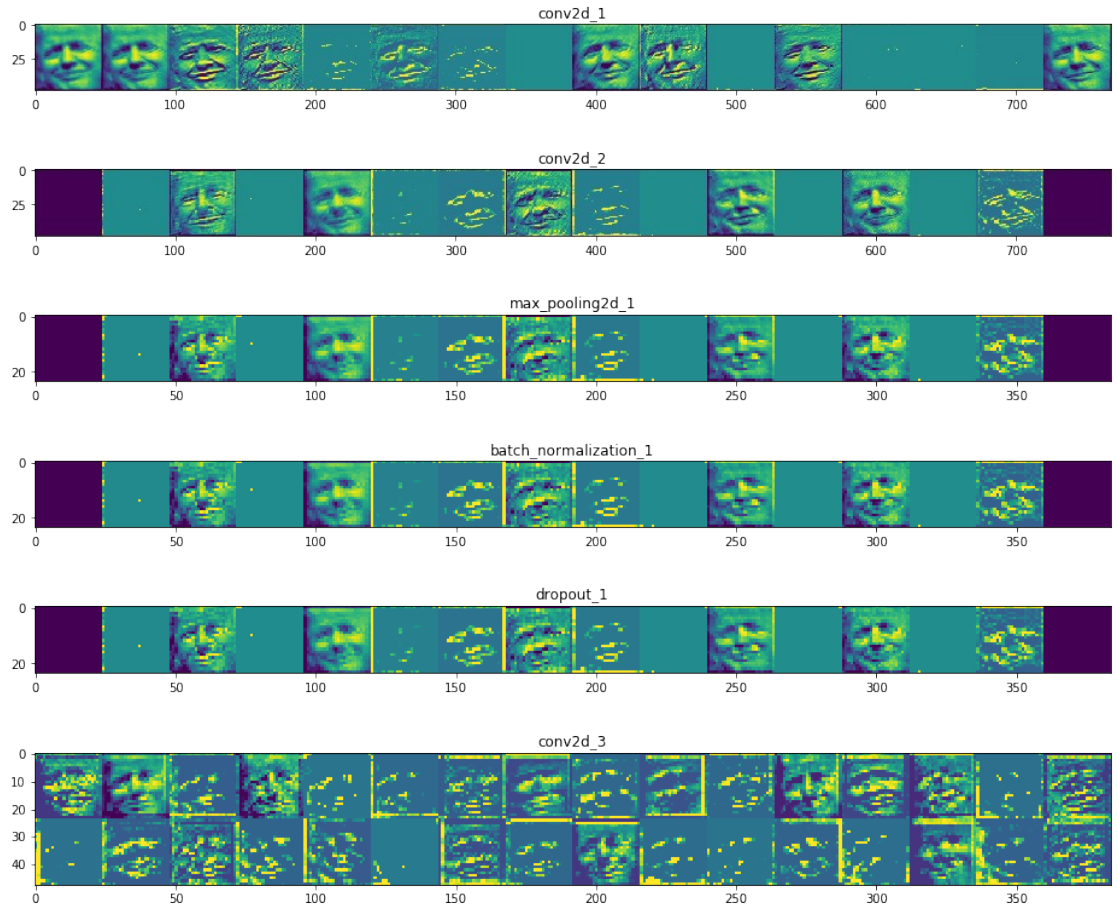


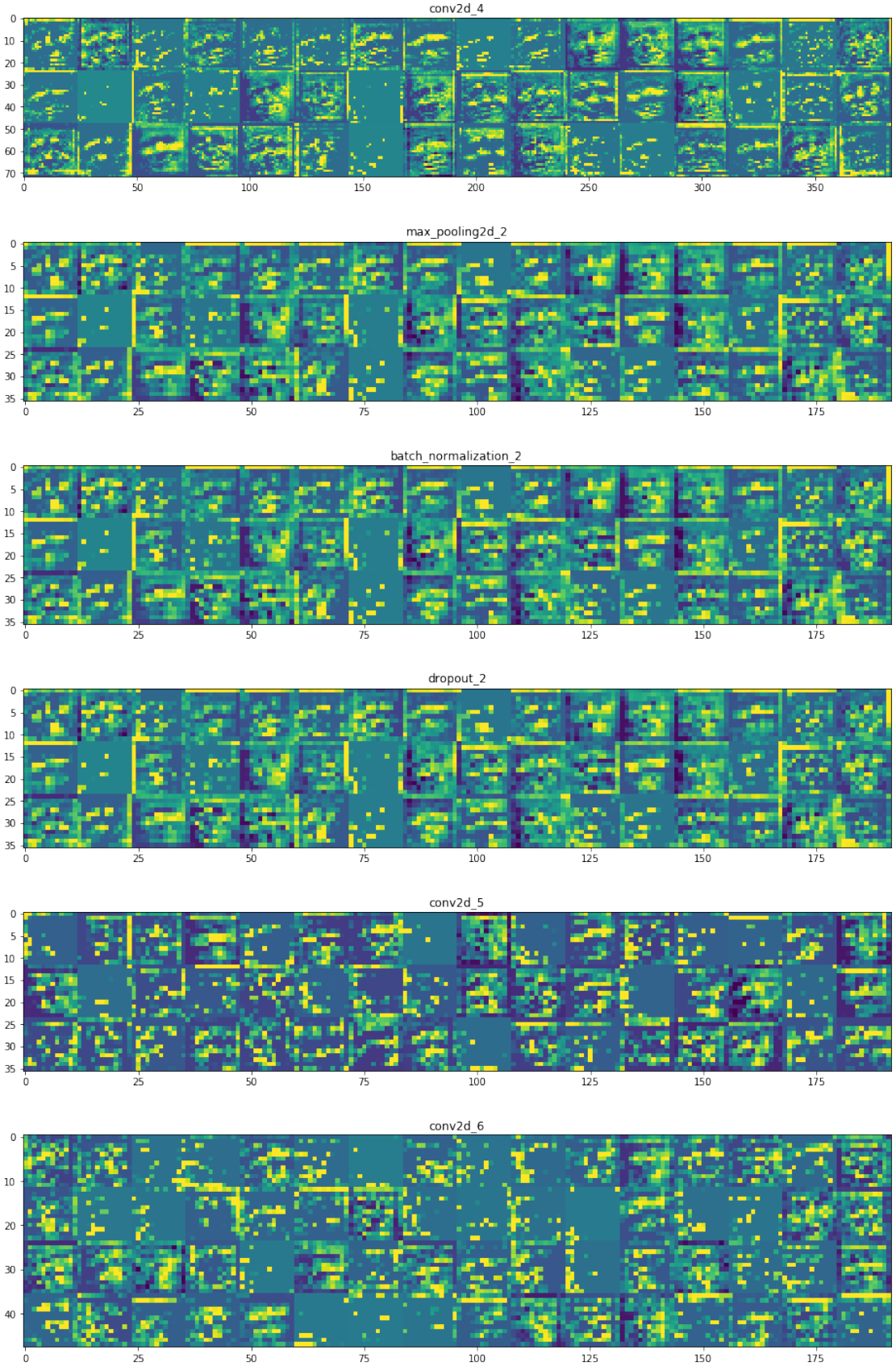




```
for layer_name, layer_activation in zip(layer_names, activations): # Displays the feature maps
    n_features = layer_activation.shape[-1] # Number of features in the feature map
    size = layer_activation.shape[1] #The feature map has shape (1, size, size, n_features).
    n_cols = n_features // images_per_row # Tiles the activation channels in this matrix
    display_grid = np.zeros((size * n_cols, images_per_row * size))
    for col in range(n_cols): # Tiles each filter into a big horizontal grid
```

```
    for row in range(images_per_row):
        channel_image = layer_activation[0,:, :,col * images_per_row + row]
        channel_image -= channel_image.mean() # Post-processes the feature to make it visually
palatable
        channel_image /= channel_image.std()
        channel_image *= 64
        channel_image += 128
        channel_image = np.clip(channel_image, 0, 255).astype('uint8')
        display_grid[col * size : (col + 1) * size, # Displays the grid
                     row * size : (row + 1) * size] = channel_image
  scale = 1. / size
  plt.figure(figsize=(scale * display_grid.shape[1],
                      scale * display_grid.shape[0]))
  plt.title(layer_name)
  plt.grid(False)
  plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:10:
RuntimeWarning: invalid value encountered in true_divide
  # Remove the CWD from sys.path while we load stuff.
```




conv2d_4
max_pooling2d_2
batch_normalization_2
dropout_2
conv2d_5
conv2d_6

## IX. Create and train the model

To prevent overfitting, we can do data augmentation.

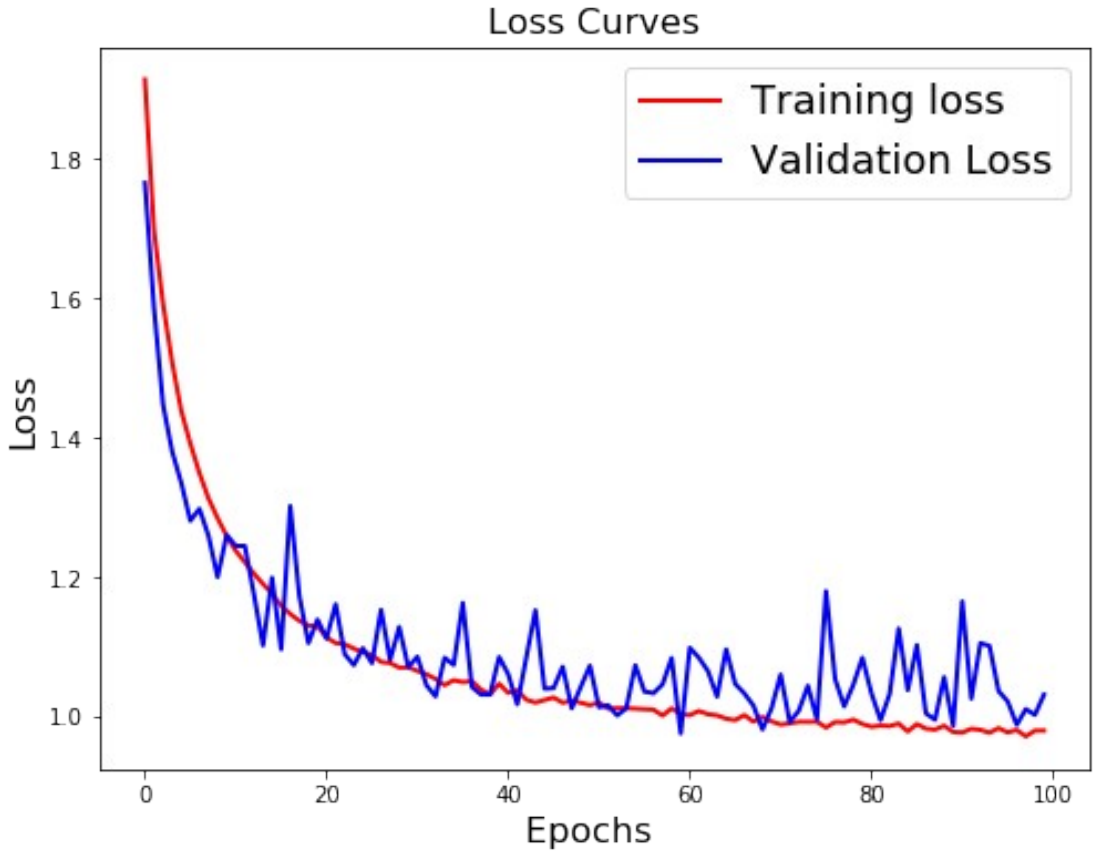
```
datagen = ImageDataGenerator(
      zoom_range=0.2,        # randomly zoom into images
      rotation_range=10,     # randomly rotate images in the range (degrees, 0 to 180)
      width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
      height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
      horizontal_flip=True,   # randomly flip images
      vertical_flip=False)    # randomly flip images

#Creating 2nd model and training(fitting)

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

batch_size = 256
epochs = 100

# Fit the model on the batches generated by datagen.flow().
history = model.fit_generator(
   datagen.flow(train_data, train_labels_one_hot, batch_size=batch_size),
   steps_per_epoch=int(np.ceil(train_data.shape[0] / float(batch_size))),
   epochs = epochs,
   validation_data=(test_data, test_labels_one_hot)
)
```

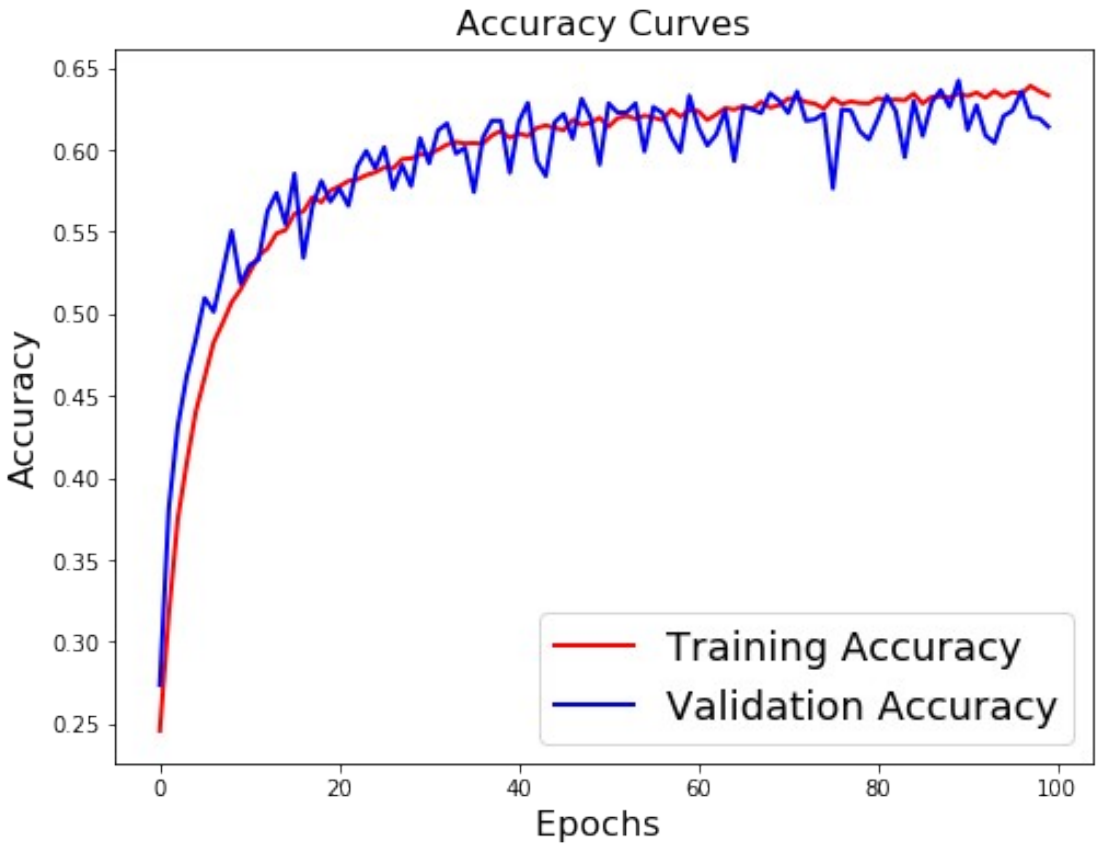
## X. Evaluate the model

```
#Plotting accuracy and loss curves for 2nd model

# Loss Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=2.0)
plt.plot(history.history['val_loss'],'b',linewidth=2.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)

# Accuracy Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['acc'],'r',linewidth=2.0)
plt.plot(history.history['val_acc'],'b',linewidth=2.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
```

```
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves',fontsize=16)
```

```
Text(0.5, 1.0, 'Accuracy Curves')
```






## XI. Save and re-open the model

```
#save the model weights
json_string = model.to_json()
model.save_weights(local_path + 'savedmodels/model_3.h5')
open(local_path + 'savedmodels/model_3.json', 'w').write(json_string)
#model.save_weights(local_path + 'savedmodels/Emotion_Face_Detection_Model.h5')
```

8372

```
with open(local_path + 'savedmodels/model_3.json','r') as f:
    json = f.read()
model = model_from_json(json)

model.load_weights(local_path + 'savedmodels/model_3.h5')
print("Loaded model from disk")
```

Loaded model from disk

## XII. Making a prediction on an image

```
hanks = '/Users/maelfabien/filrouge_pole_emploi/Video/test_samples/hanks_vs.jpg'
```

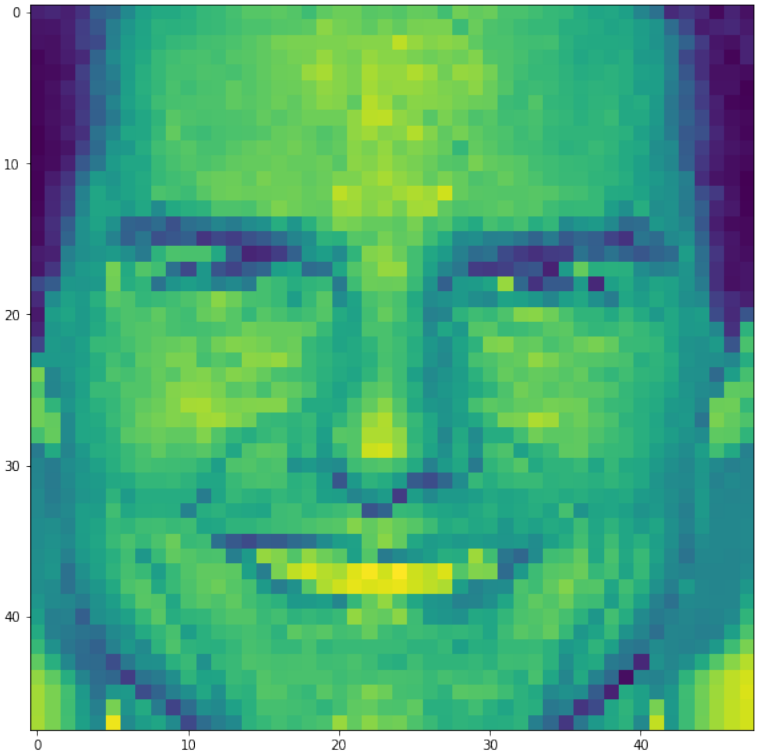
```
hanks_face = cv2.imread(hanks)

plt.imshow(cv2.cvtColor(hanks_face, cv2.COLOR_BGR2RGB))
```

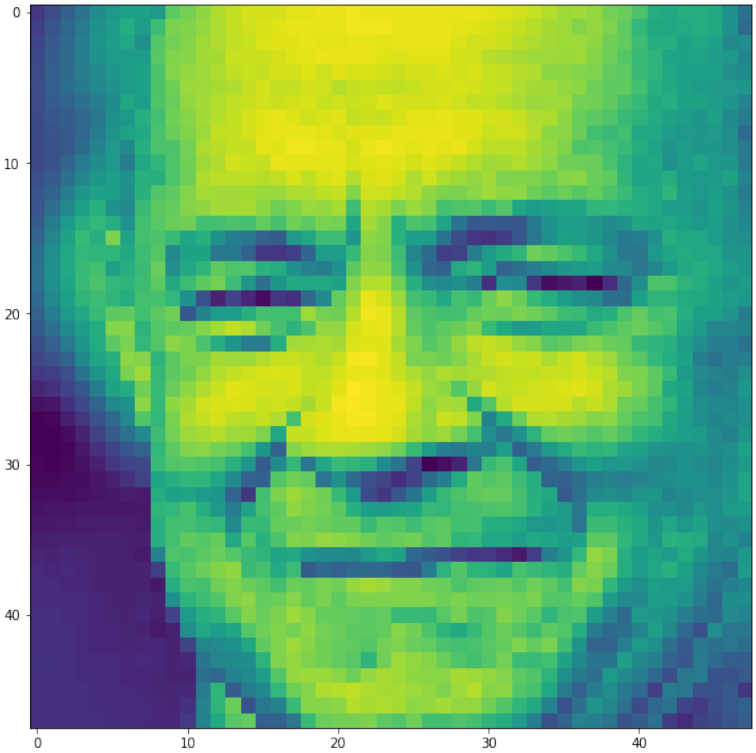
```
<matplotlib.image.AxesImage at 0x1a3b7ee358>
```


```
plt.figure(figsize=(12,12))
plt.imshow(detect_face(hanks_face)[0])
plt.show()
```


```
for face in extract_face_features(detect_face(hanks_face)) :
    plt.figure(figsize=(10,10))
```

```
plt.imshow(face)
plt.show()
```




```
for face in extract_face_features(detect_face(hanks_face)) :
```

```
to_predict = np.reshape(face.flatten(), (1,48,48,1))
res = model.predict(to_predict)
result_num = np.argmax(res)
print(result_num)
```

```
3
3
```

This corresponds to the Happy Labels which is a good prediction.

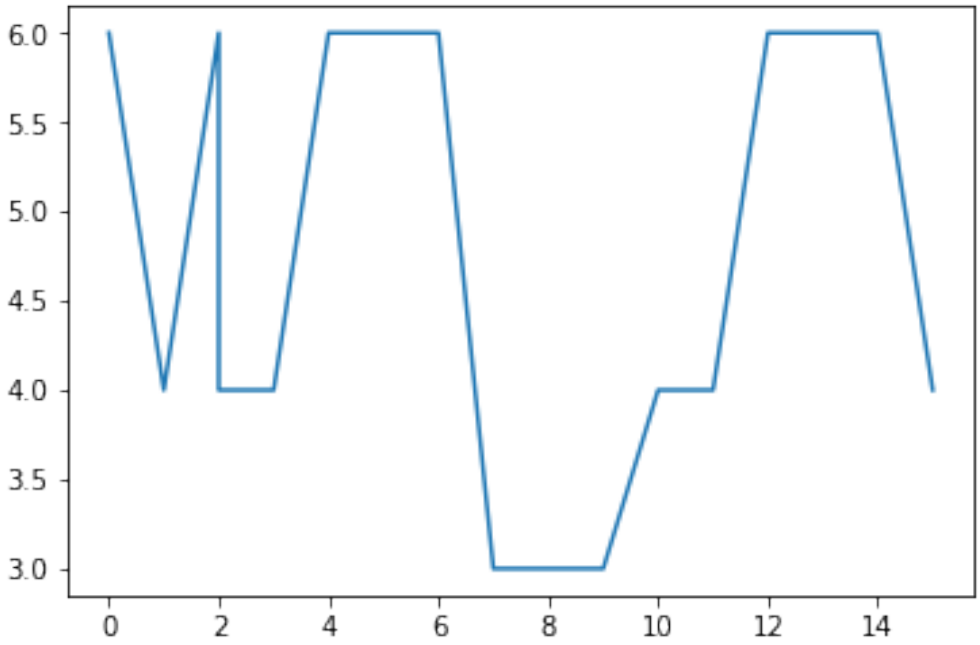
## XIII. Making live predictions from Webcam

```
#Lancer la capture video
video_capture = cv2.VideoCapture(0)
font = cv2.FONT_HERSHEY_SIMPLEX

emotion = []

xs = []
ys = []

i = 0

while True:
    # Capture frame-by-frame
    # sleep(0.8)
    ret, frame = video_capture.read()

    face_index = 0
    gray, detected_faces, coord = detect_face(frame)

    try :
        for face in detected_faces :
            face = extract_face_features(gray, face)
            face = np.reshape(face.flatten(), (1,48,48,1))
            x,y,w,h = detect_face(frame)[2][face_index]

            #if w > 200 :
            #Dessiner rectangle autour de la tête
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

            # predict smile
            prediction = model.predict(face)
            prediction_result = np.argmax(prediction)
```

```
        cv2.putText(frame, "Angry : " + str(round(prediction[0][0],3)),(10,30),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 0)
        cv2.putText(frame, "Disgust : " + str(round(prediction[0][1],3)),(10,50),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 0)
        cv2.putText(frame, "Fear : " + str(round(prediction[0][2],3)),(10,70),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)
        cv2.putText(frame, "Happy : " + str(round(prediction[0][3],3)),(10,90),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)
        cv2.putText(frame, "Sad : " + str(round(prediction[0][4],3)),(10,110),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)
        cv2.putText(frame, "Surprise : " + str(round(prediction[0][5],3)),(10,130),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)
        cv2.putText(frame, "Neutral : " + str(round(prediction[0][6],3)),(10,150),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)

        # draw extracted face in the top right corner
        #frame[face_index * shape_x: (face_index + 1) * shape_x, -1 * shape_y - 1:-1, :] =
cv2.cvtColor(extracted_face * 255, cv2.COLOR_GRAY2RGB)

        # annotate main image with a label
        if prediction_result == 0 :
           cv2.putText(frame, "Angry",(x,y), cv2.FONT_HERSHEY_SIMPLEX, 2, 155, 10)
        elif prediction_result == 1 :
           cv2.putText(frame, "Disgust",(x,y), cv2.FONT_HERSHEY_SIMPLEX, 2, 155, 10)
        elif prediction_result == 2 :
           cv2.putText(frame, "Fear",(x,y), cv2.FONT_HERSHEY_SIMPLEX, 2, 155, 10)
        elif prediction_result == 3 :
           cv2.putText(frame, "Happy",(x,y), cv2.FONT_HERSHEY_SIMPLEX, 2, 155, 10)
        elif prediction_result == 4 :
           cv2.putText(frame, "Sad",(x,y), cv2.FONT_HERSHEY_SIMPLEX, 2, 155, 10)
        elif prediction_result == 5 :
           cv2.putText(frame, "Surprise",(x,y), cv2.FONT_HERSHEY_SIMPLEX, 2, 155, 10)
        else :
           cv2.putText(frame, "Neutral",(x,y), cv2.FONT_HERSHEY_SIMPLEX, 2, 155, 10)

    #emotion.append(prediction_result)
    #xs.append(i)
    #ys.append(prediction_result)

    #line1, = ax.plot(i, prediction_result, 'b-')
    # line1.set_ydata(prediction_result)
    # fig.canvas.draw()

    #print(prediction_result)
```

```
            face_index += 1
        i = i + 1
    except :
        continue
    # Display the resulting frame
    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break


# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()

#%matplotlib qt
#plt.bar(xs, ys)
```

```
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)

def animate(i) :
    graph_data = emotion
    xs = []
    ys = []
    for emo in graph_data:
        xs.append(emo[0])
        ys.append(emo[1])
    ax1.clear()
    ax1.plot(xs,ys)

ani = animation.FuncAnimation(fig, animate, interval=1000)
plt.show()
```




## XIV. Enhanced Visualization

This basic step is now woring properly and results are quite satisfying. There are lots of sources of improvements we'll try to implement over time :

- add features from manually selected filters (e.g Gabor filters)
- take into account the frequency of eye blinks
- take into account the symetry of the keypoints on a face
- display all the keypoints of the face
- align the face by scaling of the facial features
- add emojis translating the emotion

### *a. Frequency of eye blink*

```
def eye_aspect_ratio(eye):
        A = distance.euclidean(eye[1], eye[5])
        B = distance.euclidean(eye[2], eye[4])
        C = distance.euclidean(eye[0], eye[3])
        ear = (A + B) / (2.0 * C)
        return ear

thresh = 0.25
frame_check = 20
face_detect = dlib.get_frontal_face_detector()
predictor_landmarks =
dlib.shape_predictor("/Users/maelfabien/Desktop/LocalDB/Videos/landmarks/
shape_predictor_68_face_landmarks.dat")
```

```
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

### *b. Detect Keypoints to plot them*

```
(nStart, nEnd) = face_utils.FACIAL_LANDMARKS_IDXS["nose"]
(mStart, mEnd) = face_utils.FACIAL_LANDMARKS_IDXS["mouth"]
(jStart, jEnd) = face_utils.FACIAL_LANDMARKS_IDXS["jaw"]

(eblStart, eblEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eyebrow"]
(ebrStart, ebrEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eyebrow"]
```

### *c. Face Alignment*

```
desiredLeftEye=(0.35, 0.35)

def align(gray, rect):
    # convert the landmark (x, y)-coordinates to a NumPy array
    shape = predictor(gray, rect)
    shape = shape_to_np(shape)

    # extract the left and right eye (x, y)-coordinates
    (lStart, lEnd) = FACIAL_LANDMARKS_IDXS["left_eye"]
    (rStart, rEnd) = FACIAL_LANDMARKS_IDXS["right_eye"]
    leftEyePts = shape[lStart:lEnd]
    rightEyePts = shape[rStart:rEnd]

    # compute the center of mass for each eye
    leftEyeCenter = leftEyePts.mean(axis=0).astype("int")
    rightEyeCenter = rightEyePts.mean(axis=0).astype("int")

    # compute the angle between the eye centroids
    dY = rightEyeCenter[1] - leftEyeCenter[1]
    dX = rightEyeCenter[0] - leftEyeCenter[0]
    angle = np.degrees(np.arctan2(dY, dX)) - 180

    # compute the desired right eye x-coordinate based on the
    # desired x-coordinate of the left eye
    desiredRightEyeX = 1.0 - desiredLeftEye[0]

    # determine the scale of the new resulting image by taking
    # the ratio of the distance between eyes in the *current*
    # image to the ratio of distance between eyes in the
```

```
    # *desired* image
    dist = np.sqrt((dX ** 2) + (dY ** 2))
    desiredDist = (desiredRightEyeX - desiredLeftEye[0])
    desiredDist *= self.desiredFaceWidth
    scale = desiredDist / dist

    # compute center (x, y)-coordinates (i.e., the median point)
    # between the two eyes in the input image
    eyesCenter = ((leftEyeCenter[0] + rightEyeCenter[0]) // 2,
        (leftEyeCenter[1] + rightEyeCenter[1]) // 2)

    # grab the rotation matrix for rotating and scaling the face
    M = cv2.getRotationMatrix2D(eyesCenter, angle, scale)

    # update the translation component of the matrix
    tX = self.desiredFaceWidth * 0.5
    tY = self.desiredFaceHeight * self.desiredLeftEye[1]
    M[0, 2] += (tX - eyesCenter[0])
    M[1, 2] += (tY - eyesCenter[1])

    # apply the affine transformation
    (w, h) = (self.desiredFaceWidth, self.desiredFaceHeight)
    #output = cv2.warpAffine(image, M, (w, h), flags=cv2.INTER_CUBIC)
    output = cv2.warpAffine(gray, M, (w, h), flags=cv2.INTER_CUBIC)
    # return the aligned face
    return output
```

### *d. Final Prediction*

```
#Lancer la capture video
video_capture = cv2.VideoCapture(0)
flag = 0
j = 1

while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    face_index = 0

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rects = face_detect(gray, 1)
    #gray, detected_faces, coord = detect_face(frame)

    for (i, rect) in enumerate(rects):
```

```
    shape = predictor_landmarks(gray, rect)
    shape = face_utils.shape_to_np(shape)

    (x, y, w, h) = face_utils.rect_to_bb(rect)
    face = gray[y:y+h,x:x+w]

    #Zoom sur la face extraite
    face = zoom(face, (shape_x / face.shape[0],shape_y / face.shape[1]))
    #cast type float
    face = face.astype(np.float32)
    #scale
    face /= float(face.max())
    face = np.reshape(face.flatten(), (1, 48, 48, 1))
    prediction = model.predict(face)
    prediction_result = np.argmax(prediction)

    # Rectangle around the face
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv2.putText(frame, "Face #{}".format(i + 1), (x - 10, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (0, 255, 0), 2)

    for (j, k) in shape:
        cv2.circle(frame, (j, k), 1, (0, 0, 255), -1)

     # 12. Add prediction probabilities
    cv2.putText(frame, "Emotional report : ",(40,120), cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 0)
    cv2.putText(frame, "Angry : " + str(round(prediction[0][0],3)),(40,140),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 0)
    cv2.putText(frame, "Disgust : " + str(round(prediction[0][1],3)),(40,160),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 0)
    cv2.putText(frame, "Fear : " + str(round(prediction[0][2],3)),(40,180),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)
    cv2.putText(frame, "Happy : " + str(round(prediction[0][3],3)),(40,200),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)
    cv2.putText(frame, "Sad : " + str(round(prediction[0][4],3)),(40,220),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)
    cv2.putText(frame, "Surprise : " + str(round(prediction[0][5],3)),(40,240),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)
    cv2.putText(frame, "Neutral : " + str(round(prediction[0][6],3)),(40,260),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 1)

    # draw extracted face in the top right corner
    #frame[face_index * shape_x: (face_index + 1) * shape_x, -1 * shape_y - 1:-1, :] =
```

```
cv2.cvtColor(face * 255, cv2.COLOR_GRAY2RGB)

    # 13. Annotate main image with a label
    if prediction_result == 0 :
      cv2.putText(frame, "Happy",(x+w-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    elif prediction_result == 1 :
      cv2.putText(frame, "Happy",(x+w-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    elif prediction_result == 2 :
      cv2.putText(frame, "Happy",(x+w-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    elif prediction_result == 3 :
      cv2.putText(frame, "Happy",(x+w-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    elif prediction_result == 4 :
      cv2.putText(frame, "Happy",(x+w-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    elif prediction_result == 5 :
      cv2.putText(frame, "Happy",(x+w-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    else :
      cv2.putText(frame, "Happy",(x+w-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)


    # 5. Eye Detection and Blink Count
    leftEye = shape[lStart:lEnd]
    rightEye = shape[rStart:rEnd]

    # Compute Eye Aspect Ratio
    leftEAR = eye_aspect_ratio(leftEye)
    rightEAR = eye_aspect_ratio(rightEye)
    ear = (leftEAR + rightEAR) / 2.0

    # And plot its contours
    leftEyeHull = cv2.convexHull(leftEye)
    rightEyeHull = cv2.convexHull(rightEye)
    cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
    cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

    # Compute total blinks and frequency
    if ear < thresh:
      flag += 1
      #cv2.putText(frame, "Blink", (10, 200), cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 0)

    cv2.putText(frame, "Total blinks : " + str(flag), (40, 70), cv2.FONT_HERSHEY_SIMPLEX, 1, 155,
1)
    #cv2.putText(frame, "Blink Frequency : " + str(int(flag/j)), (40, 220),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 155, 0)

    # 6. Detect Nose
```

```
        nose = shape[nStart:nEnd]
        noseHull = cv2.convexHull(nose)
        cv2.drawContours(frame, [noseHull], -1, (0, 255, 0), 1)

        # 7. Detect Mouth
        mouth = shape[mStart:mEnd]
        mouthHull = cv2.convexHull(mouth)
        cv2.drawContours(frame, [mouthHull], -1, (0, 255, 0), 1)

        # 8. Detect Jaw
        jaw = shape[jStart:jEnd]
        jawHull = cv2.convexHull(jaw)
        cv2.drawContours(frame, [jawHull], -1, (0, 255, 0), 1)

        # 9. Detect Eyebrows
        ebr = shape[ebrStart:ebrEnd]
        ebrHull = cv2.convexHull(ebr)
        cv2.drawContours(frame, [ebrHull], -1, (0, 255, 0), 1)
        ebl = shape[eblStart:eblEnd]
        eblHull = cv2.convexHull(ebl)
        cv2.drawContours(frame, [eblHull], -1, (0, 255, 0), 1)

    cv2.putText(frame,'Number of Faces : ' + str(i+1),(40, 40), cv2.FONT_HERSHEY_SIMPLEX, 1,
155, 1)
    j = j + 1
    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

## XV. Deeper CNN Models

### *Inception*

Architecture Inspired by : https://ieeexplore.ieee.org/document/7477450

```
input_img = Input(shape=(shape_x, shape_y, 1))

layer1 = Conv2D(10, (3, 3), padding='same', activation='relu')(input_img)
layer1_2 = Conv2D(20, (3, 3), padding='same', activation='relu')(layer1)
```

```
layer2 = MaxPooling2D(pool_size=(3, 3))(layer1_2)
bn1 = BatchNormalization()(layer2)

layer3 = Conv2D(30, (3, 3), padding='same', activation='relu')(bn1)
layer3_2 = Conv2D(40, (3, 3), padding='same', activation='relu')(layer3)
layer4 = MaxPooling2D(pool_size=(3, 3))(layer3_2)
bn2 = BatchNormalization()(layer4)

layer5 = Conv2D(50, (3, 3), padding='same', activation='relu')(bn2)
layer5_2 = Conv2D(60, (3, 3), padding='same', activation='relu')(layer5)
layer6 = MaxPooling2D(pool_size=(3, 3))(layer5_2)
bn3 = BatchNormalization()(layer6)

Conv11 = Conv2D(1, (1, 1), padding='same', activation='relu')(bn3)
Conv33 = Conv2D(4, (3, 3), padding='same', activation='relu')(bn3)
Conv332 = Conv2D(1, (3, 3), padding='same', activation='relu')(Conv33)
Conv55 = Conv2D(4, (5, 5), padding='same', activation='relu')(bn3)
Conv552 = Conv2D(1, (3, 3), padding='same', activation='relu')(Conv55)
#Pool33 = MaxPooling2D(pool_size=(3, 3))(bn3)
#ConvPool1 = Conv2D(4, (1, 1), padding='same', activation='relu')(Pool33)

intermediate1 = keras.layers.concatenate([Conv11, Conv332, Conv552], axis=1)

Conv2_11 = Conv2D(1, (1, 1), padding='same', activation='relu')(intermediate1)
Conv2_33 = Conv2D(4, (3, 3), padding='same', activation='relu')(intermediate1)
Conv2_332 = Conv2D(1, (3, 3), padding='same', activation='relu')(Conv2_33)
Conv2_55 = Conv2D(4, (5, 5), padding='same', activation='relu')(intermediate1)
Conv2_552 = Conv2D(1, (3, 3), padding='same', activation='relu')(Conv2_55)
#Pool2_33 = MaxPooling2D(pool_size=(3, 3))(intermediate1)
#ConvPool2 = Conv2D(4, (1, 1), padding='same', activation='relu')(Pool2_33)

intermediate2 = keras.layers.concatenate([Conv2_11, Conv2_332, Conv2_552], axis=1)

Conv3_11 = Conv2D(1, (1, 1), padding='same', activation='relu')(intermediate2)
Conv3_33 = Conv2D(4, (3, 3), padding='same', activation='relu')(intermediate2)
Conv3_332 = Conv2D(1, (3, 3), padding='same', activation='relu')(Conv3_33)
Conv3_55 = Conv2D(4, (5, 5), padding='same', activation='relu')(intermediate2)
Conv3_552 = Conv2D(1, (3, 3), padding='same', activation='relu')(Conv3_55)
#Pool3_33 = MaxPooling2D(pool_size=(3, 3))(intermediate2)
#ConvPool3 = Conv2D(4, (1, 1), padding='same', activation='relu')(Pool3_33)

intermediate3 = keras.layers.concatenate([Conv3_11, Conv3_332, Conv3_552], axis=1)

#Pool4 = MaxPooling2D(pool_size=(3, 3))(intermediate3)
```

```
Flat = Flatten()(intermediate3)

Dense1 = Dense(25, activation='relu')(Flat)
Dense2 = Dense(15, activation='relu')(Dense1)
Dense3 = Dense(7, activation='softmax')(Dense2)

model = Model([input_img], Dense3)

plot_model(model, to_file='model_images/model_plot_3.png', show_shapes=True,
show_layer_names=True)


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

batch_size = 128
epochs = 150

# Fit the model on the batches generated by datagen.flow().

datagen = ImageDataGenerator(
      zoom_range=0.2,        # randomly zoom into images
      rotation_range=10,     # randomly rotate images in the range (degrees, 0 to 180)
      width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
      height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
      horizontal_flip=True,    # randomly flip images
      vertical_flip=False)    # randomly flip images

history = model.fit_generator(
   datagen.flow(train_data, train_labels_one_hot, batch_size=batch_size),
   steps_per_epoch=int(np.ceil(train_data.shape[0] / float(batch_size))),
   epochs = epochs,
   validation_data=(test_data, test_labels_one_hot)
)
```

```
Epoch 1/150
225/225 [==============================] - 134s 598ms/step - loss: 1.7556 - acc: 0.2817 -
val_loss: 1.9348 - val_acc: 0.2014
Epoch 2/150
225/225 [==============================] - 129s 574ms/step - loss: 1.5606 - acc: 0.3659 -
val_loss: 1.4868 - val_acc: 0.4015
Epoch 3/150
225/225 [==============================] - 139s 619ms/step - loss: 1.4446 - acc: 0.4140 -
val_loss: 1.4238 - val_acc: 0.4218

Epoch 36/150
225/225 [==============================] - 156s 692ms/step - loss: 1.0394 - acc: 0.6105 -
```

```
val_loss: 1.1415 - val_acc: 0.5804
Epoch 37/150
225/225 [==============================] - 154s 685ms/step - loss: 1.0388 - acc: 0.6097 -
val_loss: 1.1102 - val_acc: 0.5910
Epoch 38/150
225/225 [==============================] - 156s 693ms/step - loss: 1.0337 - acc: 0.6128 -
val_loss: 1.1569 - val_acc: 0.5667
Epoch 39/150
225/225 [==============================] - 154s 685ms/step - loss: 1.0273 - acc: 0.6158 -
val_loss: 1.0716 - val_acc: 0.5965
Epoch 40/150
225/225 [==============================] - 155s 691ms/step - loss: 1.0311 - acc: 0.6139 -
val_loss: 1.1158 - val_acc: 0.5885
Epoch 41/150
225/225 [==============================] - 156s 693ms/step - loss: 1.0257 - acc: 0.6167 -
val_loss: 1.1021 - val_acc: 0.5913
Epoch 42/150
225/225 [==============================] - 154s 685ms/step - loss: 1.0212 - acc: 0.6185 -
val_loss: 1.1384 - val_acc: 0.5857
Epoch 43/150
225/225 [==============================] - 158s 704ms/step - loss: 1.0218 - acc: 0.6159 -
val_loss: 1.1309 - val_acc: 0.5809
Epoch 44/150
225/225 [==============================] - 155s 690ms/step - loss: 1.0177 - acc: 0.6182 -
val_loss: 1.1030 - val_acc: 0.5893
Epoch 45/150
225/225 [==============================] - 157s 698ms/step - loss: 1.0178 - acc: 0.6192 -
val_loss: 1.0963 - val_acc: 0.5924
Epoch 46/150
225/225 [==============================] - 155s 689ms/step - loss: 1.0175 - acc: 0.6195 -
val_loss: 1.0777 - val_acc: 0.5985
Epoch 47/150
225/225 [==============================] - 157s 698ms/step - loss: 1.0137 - acc: 0.6202 -
val_loss: 1.0925 - val_acc: 0.5988
Epoch 48/150
225/225 [==============================] - 158s 702ms/step - loss: 1.0094 - acc: 0.6208 -
val_loss: 1.1073 - val_acc: 0.5860
Epoch 49/150
225/225 [==============================] - 155s 688ms/step - loss: 1.0088 - acc: 0.6238 -
val_loss: 1.1263 - val_acc: 0.5832
Epoch 50/150
225/225 [==============================] - 155s 689ms/step - loss: 1.0059 - acc: 0.6245 -
val_loss: 1.1144 - val_acc: 0.5801
Epoch 51/150
225/225 [==============================] - 157s 700ms/step - loss: 1.0076 - acc: 0.6244 -
```

```
val_loss: 1.0908 - val_acc: 0.6007
Epoch 52/150
225/225 [==============================] - 157s 700ms/step - loss: 0.9971 - acc: 0.6290 -
val_loss: 1.0990 - val_acc: 0.5893
Epoch 53/150
225/225 [==============================] - 158s 700ms/step - loss: 1.0035 - acc: 0.6272 -
val_loss: 1.0983 - val_acc: 0.5926
Epoch 54/150
225/225 [==============================] - 156s 692ms/step - loss: 1.0055 - acc: 0.6207 -
val_loss: 1.1331 - val_acc: 0.5793
Epoch 55/150
225/225 [==============================] - 155s 689ms/step - loss: 0.9987 - acc: 0.6259 -
val_loss: 1.1338 - val_acc: 0.5695
Epoch 56/150
225/225 [==============================] - 154s 684ms/step - loss: 0.9969 - acc: 0.6271 -
val_loss: 1.0826 - val_acc: 0.5982
Epoch 57/150
225/225 [==============================] - 157s 699ms/step - loss: 0.9937 - acc: 0.6311 -
val_loss: 1.1364 - val_acc: 0.5854
Epoch 58/150
225/225 [==============================] - 156s 695ms/step - loss: 0.9951 - acc: 0.6282 -
val_loss: 1.0738 - val_acc: 0.6024
Epoch 59/150
225/225 [==============================] - 157s 697ms/step - loss: 0.9895 - acc: 0.6319 -
val_loss: 1.0769 - val_acc: 0.5977
Epoch 60/150
225/225 [==============================] - 154s 685ms/step - loss: 0.9919 - acc: 0.6292 -
val_loss: 1.0715 - val_acc: 0.6052
Epoch 61/150
225/225 [==============================] - 153s 682ms/step - loss: 0.9912 - acc: 0.6300 -
val_loss: 1.0614 - val_acc: 0.6088
Epoch 62/150
225/225 [==============================] - 156s 693ms/step - loss: 0.9897 - acc: 0.6297 -
val_loss: 1.0907 - val_acc: 0.5985
Epoch 63/150
225/225 [==============================] - 155s 688ms/step - loss: 0.9866 - acc: 0.6318 -
val_loss: 1.1008 - val_acc: 0.5929
Epoch 64/150
225/225 [==============================] - 153s 682ms/step - loss: 0.9813 - acc: 0.6322 -
val_loss: 1.1126 - val_acc: 0.5907
Epoch 65/150
225/225 [==============================] - 154s 686ms/step - loss: 0.9862 - acc: 0.6311 -
val_loss: 1.1110 - val_acc: 0.5893
Epoch 66/150
225/225 [==============================] - 155s 689ms/step - loss: 0.9792 - acc: 0.6332 -
```

```
val_loss: 1.0974 - val_acc: 0.5988
Epoch 67/150
225/225 [==============================] - 156s 693ms/step - loss: 0.9843 - acc: 0.6322 -
val_loss: 1.1040 - val_acc: 0.5896
Epoch 68/150
225/225 [==============================] - 155s 689ms/step - loss: 0.9770 - acc: 0.6329 -
val_loss: 1.1221 - val_acc: 0.5915
Epoch 69/150
225/225 [==============================] - 156s 692ms/step - loss: 0.9779 - acc: 0.6345 -
val_loss: 1.1036 - val_acc: 0.5857
Epoch 70/150
225/225 [==============================] - 155s 687ms/step - loss: 0.9771 - acc: 0.6353 -
val_loss: 1.1055 - val_acc: 0.5871
Epoch 71/150
225/225 [==============================] - 152s 678ms/step - loss: 0.9777 - acc: 0.6346 -
val_loss: 1.0578 - val_acc: 0.6030
Epoch 72/150
225/225 [==============================] - 155s 687ms/step - loss: 0.9723 - acc: 0.6367 -
val_loss: 1.1546 - val_acc: 0.5670
Epoch 73/150
225/225 [==============================] - 154s 684ms/step - loss: 0.9733 - acc: 0.6357 -
val_loss: 1.0501 - val_acc: 0.6116
Epoch 74/150
225/225 [==============================] - 154s 682ms/step - loss: 0.9707 - acc: 0.6366 -
val_loss: 1.0691 - val_acc: 0.6060
Epoch 75/150
225/225 [==============================] - 156s 692ms/step - loss: 0.9671 - acc: 0.6418 -
val_loss: 1.0547 - val_acc: 0.6135
Epoch 76/150
225/225 [==============================] - 155s 689ms/step - loss: 0.9669 - acc: 0.6354 -
val_loss: 1.0804 - val_acc: 0.5946
Epoch 77/150
225/225 [==============================] - 155s 689ms/step - loss: 0.9676 - acc: 0.6383 -
val_loss: 1.1031 - val_acc: 0.5924
Epoch 78/150
225/225 [==============================] - 156s 695ms/step - loss: 0.9732 - acc: 0.6360 -
val_loss: 1.0534 - val_acc: 0.6113
Epoch 79/150
225/225 [==============================] - 153s 680ms/step - loss: 0.9667 - acc: 0.6394 -
val_loss: 1.0980 - val_acc: 0.6057
Epoch 80/150
225/225 [==============================] - 153s 678ms/step - loss: 0.9672 - acc: 0.6405 -
val_loss: 1.0432 - val_acc: 0.6183
Epoch 81/150
225/225 [==============================] - 156s 691ms/step - loss: 0.9651 - acc: 0.6404 -
```

```
val_loss: 1.1233 - val_acc: 0.5929
Epoch 82/150
225/225 [==============================] - 154s 683ms/step - loss: 0.9621 - acc: 0.6418 -
val_loss: 1.0881 - val_acc: 0.5988
Epoch 83/150
225/225 [==============================] - 155s 690ms/step - loss: 0.9613 - acc: 0.6419 -
val_loss: 1.1330 - val_acc: 0.5790
Epoch 84/150
225/225 [==============================] - 155s 687ms/step - loss: 0.9604 - acc: 0.6426 -
val_loss: 1.0548 - val_acc: 0.6108
Epoch 85/150
225/225 [==============================] - 154s 685ms/step - loss: 0.9558 - acc: 0.6420 -
val_loss: 1.0802 - val_acc: 0.6108
Epoch 86/150
225/225 [==============================] - 156s 691ms/step - loss: 0.9610 - acc: 0.6446 -
val_loss: 1.0552 - val_acc: 0.6135
Epoch 87/150
225/225 [==============================] - 155s 690ms/step - loss: 0.9590 - acc: 0.6429 -
val_loss: 1.0674 - val_acc: 0.6063
Epoch 88/150
225/225 [==============================] - 156s 692ms/step - loss: 0.9586 - acc: 0.6409 -
val_loss: 1.0949 - val_acc: 0.6038
Epoch 89/150
225/225 [==============================] - 158s 700ms/step - loss: 0.9579 - acc: 0.6423 -
val_loss: 1.0755 - val_acc: 0.6063
Epoch 90/150
225/225 [==============================] - 157s 697ms/step - loss: 0.9539 - acc: 0.6441 -
val_loss: 1.0927 - val_acc: 0.6016
Epoch 91/150
225/225 [==============================] - 156s 695ms/step - loss: 0.9529 - acc: 0.6460 -
val_loss: 1.0896 - val_acc: 0.5924
Epoch 92/150
225/225 [==============================] - 156s 692ms/step - loss: 0.9526 - acc: 0.6465 -
val_loss: 1.0743 - val_acc: 0.6057
Epoch 93/150
225/225 [==============================] - 157s 696ms/step - loss: 0.9454 - acc: 0.6505 -
val_loss: 1.0684 - val_acc: 0.6149
Epoch 94/150
225/225 [==============================] - 157s 697ms/step - loss: 0.9510 - acc: 0.6486 -
val_loss: 1.0928 - val_acc: 0.5957
Epoch 95/150
225/225 [==============================] - 155s 688ms/step - loss: 0.9518 - acc: 0.6463 -
val_loss: 1.0699 - val_acc: 0.6013
Epoch 96/150
225/225 [==============================] - 157s 696ms/step - loss: 0.9527 - acc: 0.6473 -
```

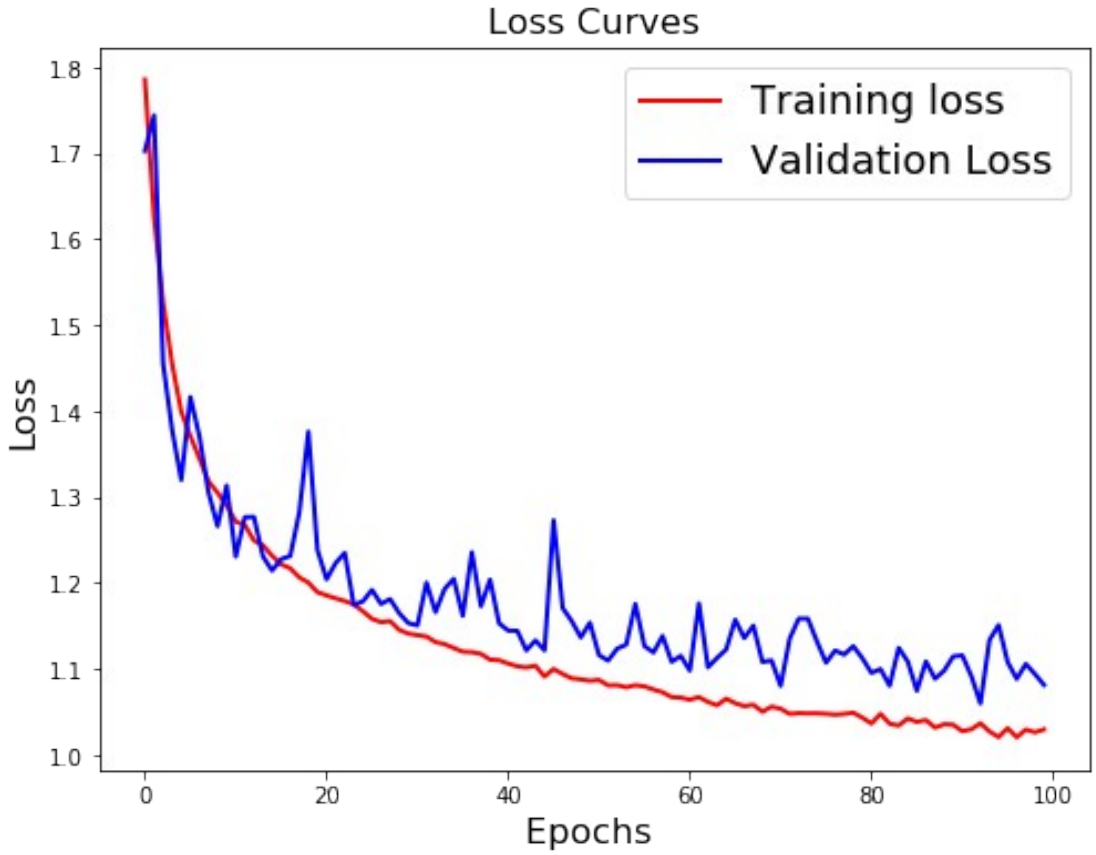
```
val_loss: 1.0690 - val_acc: 0.6032
Epoch 97/150
225/225 [==============================] - 157s 699ms/step - loss: 0.9455 - acc: 0.6488 -
val_loss: 1.0707 - val_acc: 0.6085
Epoch 98/150
225/225 [==============================] - 157s 699ms/step - loss: 0.9480 - acc: 0.6494 -
val_loss: 1.0812 - val_acc: 0.5993
Epoch 99/150
225/225 [==============================] - 154s 683ms/step - loss: 0.9480 - acc: 0.6480 -
val_loss: 1.0736 - val_acc: 0.5985
Epoch 100/150
225/225 [==============================] - 157s 700ms/step - loss: 0.9418 - acc: 0.6502 -
val_loss: 1.0584 - val_acc: 0.6102
Epoch 101/150
225/225 [==============================] - 158s 703ms/step - loss: 0.9397 - acc: 0.6479 -
val_loss: 1.1139 - val_acc: 0.5921
Epoch 102/150
225/225 [==============================] - 156s 693ms/step - loss: 0.9436 - acc: 0.6466 -
val_loss: 1.0500 - val_acc: 0.6155
Epoch 103/150
225/225 [==============================] - 156s 691ms/step - loss: 0.9427 - acc: 0.6512 -
val_loss: 1.0581 - val_acc: 0.6160
Epoch 104/150
225/225 [==============================] - 157s 698ms/step - loss: 0.9405 - acc: 0.6477 -
val_loss: 1.0617 - val_acc: 0.6147
Epoch 105/150
225/225 [==============================] - 158s 702ms/step - loss: 0.9379 - acc: 0.6508 -
val_loss: 1.1032 - val_acc: 0.6021
Epoch 106/150
225/225 [==============================] - 154s 685ms/step - loss: 0.9394 - acc: 0.6512 -
val_loss: 1.0379 - val_acc: 0.6283
Epoch 107/150
225/225 [==============================] - 152s 673ms/step - loss: 0.9346 - acc: 0.6560 -
val_loss: 1.0356 - val_acc: 0.6289
Epoch 108/150
225/225 [==============================] - 156s 694ms/step - loss: 0.9406 - acc: 0.6484 -
val_loss: 1.1560 - val_acc: 0.5899
Epoch 109/150
225/225 [==============================] - 156s 692ms/step - loss: 0.9450 - acc: 0.6495 -
val_loss: 1.0720 - val_acc: 0.6186
Epoch 110/150
225/225 [==============================] - 157s 697ms/step - loss: 0.9407 - acc: 0.6518 -
val_loss: 1.0537 - val_acc: 0.6124
Epoch 111/150
225/225 [==============================] - 156s 695ms/step - loss: 0.9405 - acc: 0.6509 -
```

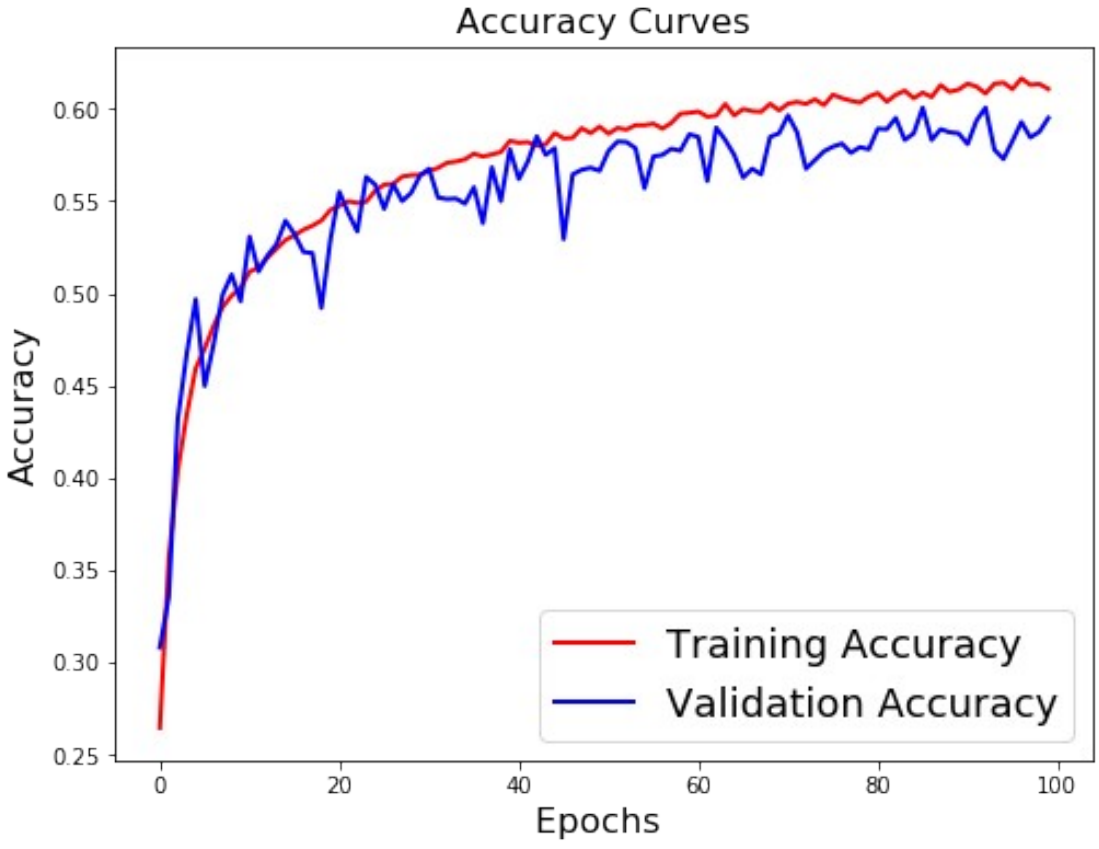
```
val_loss: 1.0441 - val_acc: 0.6177
Epoch 112/150
225/225 [==============================] - 156s 694ms/step - loss: 0.9373 - acc: 0.6518 -
val_loss: 1.0697 - val_acc: 0.6147
Epoch 113/150
225/225 [==============================] - 157s 696ms/step - loss: 0.9363 - acc: 0.6523 -
val_loss: 1.0771 - val_acc: 0.6094
Epoch 114/150
225/225 [==============================] - 156s 694ms/step - loss: 0.9339 - acc: 0.6530 -
val_loss: 1.0902 - val_acc: 0.6138
Epoch 115/150
225/225 [==============================] - 157s 700ms/step - loss: 0.9323 - acc: 0.6549 -
val_loss: 1.0701 - val_acc: 0.6088
Epoch 116/150
225/225 [==============================] - 156s 693ms/step - loss: 0.9383 - acc: 0.6502 -
val_loss: 1.0734 - val_acc: 0.6077
Epoch 117/150
225/225 [==============================] - 156s 693ms/step - loss: 0.9312 - acc: 0.6544 -
val_loss: 1.0966 - val_acc: 0.6013
Epoch 118/150
225/225 [==============================] - 155s 687ms/step - loss: 0.9384 - acc: 0.6518 -
val_loss: 1.0692 - val_acc: 0.6152

                                                :

                                                :

                                                :
Epoch 120/150
225/225 [==============================] - 156s 695ms/step - loss: 0.9332 - acc: 0.6545 -
val_loss: 1.0403 - val_acc: 0.6194
Epoch 121/150
225/225 [==============================] - 153s 678ms/step - loss: 0.9284 - acc: 0.6571 -
val_loss: 1.0729 - val_acc: 0.6124
Epoch 122/150
225/225 [==============================] - 154s 686ms/step - loss: 0.9331 - acc: 0.6522 -
val_loss: 1.0679 - val_acc: 0.6130
Epoch 123/150
225/225 [==============================] - 154s 684ms/step - loss: 0.9321 - acc: 0.6538 -
Epoch 128/150
225/225 [==============================] - 156s 691ms/step - loss: 0.9215 - acc: 0.6569 -
val_loss: 1.0529 - val_acc: 0.6241
Epoch 129/150
225/225 [==============================] - 156s 694ms/step - loss: 0.9219 - acc: 0.6548 -
val_loss: 1.0369 - val_acc: 0.6233
```

```
                                                :
                                                :
                                                :
225/225 [==============================] - 156s 693ms/step - loss: 0.9246 - acc: 0.6564 -
val_loss: 1.0564 - val_acc: 0.6208
Epoch 149/150
225/225 [==============================] - 152s 676ms/step - loss: 0.9203 - acc: 0.6573 -
val_loss: 1.0591 - val_acc: 0.6191
Epoch 150/150
225/225 [==============================] - 152s 677ms/step - loss: 0.9194 - acc: 0.6579 -
val_loss: 1.0600 - val_acc: 0.6166
```

```
#Plotting accuracy and loss curves for 2nd model
# Loss Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=2.0)
plt.plot(history.history['val_loss'],'b',linewidth=2.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)

# Accuracy Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['acc'],'r',linewidth=2.0)
plt.plot(history.history['val_acc'],'b',linewidth=2.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves',fontsize=16)
```

```
Text(0.5, 1.0, 'Accuracy Curves')
```


Loss Curves
Training loss
Validation Loss
Loss
Epochs
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0
20
40
60
80
100


Accuracy Curves
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0
20
40
60
80
100

```
#save the model weights
json_string = model.to_json()
model.save_weights(local_path + 'savedmodels/model_deep_2.h5')
open(local_path + 'savedmodels/model_deep_2.json', 'w').write(json_string)
#model.save_weights(local_path + 'savedmodels/Emotion_Face_Detection_Model.h5')

17443

with open(local_path + 'savedmodels/model_deep_2.json','r') as f:
    json = f.read()
model = model_from_json(json)

model.load_weights(local_path + 'savedmodels/model_deep_2.h5')
print("Loaded model from disk")
```

## *X-Ception*

```
def entry_flow(inputs) :

    x = Conv2D(32, 3, strides = 2, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(64,3,padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    previous_block_activation = x

    for size in [128, 256, 728] :

        x = Activation('relu')(x)
        x = SeparableConv2D(size, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x = Activation('relu')(x)
        x = SeparableConv2D(size, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x = MaxPooling2D(3, strides=2, padding='same')(x)

        residual = Conv2D(size, 1, strides=2, padding='same')(previous_block_activation)

        x = keras.layers.Add()([x, residual])
        previous_block_activation = x
```

```
    return x
def middle_flow(x, num_blocks=8) :

    previous_block_activation = x

    for _ in range(num_blocks) :

        x = Activation('relu')(x)
        x = SeparableConv2D(728, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x = Activation('relu')(x)
        x = SeparableConv2D(728, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x = Activation('relu')(x)
        x = SeparableConv2D(728, 3, padding='same')(x)
        x = BatchNormalization()(x)

        x = keras.layers.Add()([x, previous_block_activation])
        previous_block_activation = x

    return x
def exit_flow(x, num_classes=7) :

    previous_block_activation = x

    x = Activation('relu')(x)
    x = SeparableConv2D(728, 3, padding='same')(x)
    x = BatchNormalization()(x)

    x = Activation('relu')(x)
    x = SeparableConv2D(1024, 3, padding='same')(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D(3, strides=2, padding='same')(x)

    residual = Conv2D(1024, 1, strides=2, padding='same')(previous_block_activation)
    x = keras.layers.Add()([x, residual])

    x = Activation('relu')(x)
    x = SeparableConv2D(728, 3, padding='same')(x)
    x = BatchNormalization()(x)
```

```
    x = Activation('relu')(x)
    x = SeparableConv2D(1024, 3, padding='same')(x)
    x = BatchNormalization()(x)

    x = GlobalAveragePooling2D()(x)
    x = Dense(num_classes, activation='softmax')(x)

    return x

inputs = Input(shape=(shape_x, shape_y, 1))
outputs = exit_flow(middle_flow(entry_flow(inputs)))

xception = Model(inputs, outputs)

plot_model(xception, to_file='model_images/model_plot_4.png', show_shapes=True,
show_layer_names=True)


xception.summary()
```

```
__________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected
to
==================================================================================
input_12 (InputLayer)           (None, 48, 48, 1)    0


__________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 24, 24, 32)   320
input_12[0][0]

__________________________________________________________________________________
batch_normalization_73 (BatchNo (None, 24, 24, 32)   128
conv2d_28[0][0]

__________________________________________________________________________________
activation_74 (Activation)      (None, 24, 24, 32)   0
batch_normalization_73[0][0]

__________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 24, 24, 64)   18496
activation_74[0][0]

__________________________________________________________________________________
batch_normalization_74 (BatchNo (None, 24, 24, 64)   256
conv2d_29[0][0]

__________________________________________________________________________________
activation_75 (Activation)      (None, 24, 24, 64)   0
batch_normalization_74[0][0]

__________________________________________________________________________________
activation_76 (Activation)      (None, 24, 24, 64)   0
activation_75[0][0]
```

```
____________________________________________________________________________________________
separable_conv2d_59 (SeparableC (None, 24, 24, 128)  8896
activation_76[0][0]
____________________________________________________________________________________________
batch_normalization_75 (BatchNo (None, 24, 24, 128)  512
separable_conv2d_59[0][0]
____________________________________________________________________________________________
activation_77 (Activation)      (None, 24, 24, 128)  0
batch_normalization_75[0][0]
____________________________________________________________________________________________
separable_conv2d_60 (SeparableC (None, 24, 24, 128)  17664
activation_77[0][0]
____________________________________________________________________________________________
batch_normalization_76 (BatchNo (None, 24, 24, 128)  512
separable_conv2d_60[0][0]
____________________________________________________________________________________________
max_pooling2d_16 (MaxPooling2D) (None, 12, 12, 128)  0
batch_normalization_76[0][0]
____________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 12, 12, 128)  8320
activation_75[0][0]
____________________________________________________________________________________________
add_21 (Add)                    (None, 12, 12, 128)  0
max_pooling2d_16[0][0]

conv2d_30[0][0]
____________________________________________________________________________________________
activation_78 (Activation)      (None, 12, 12, 128)  0           add_21[0]
[0]
____________________________________________________________________________________________
separable_conv2d_61 (SeparableC (None, 12, 12, 256)  34176
activation_78[0][0]
____________________________________________________________________________________________
batch_normalization_77 (BatchNo (None, 12, 12, 256)  1024
separable_conv2d_61[0][0]
____________________________________________________________________________________________
activation_79 (Activation)      (None, 12, 12, 256)  0
batch_normalization_77[0][0]
____________________________________________________________________________________________
separable_conv2d_62 (SeparableC (None, 12, 12, 256)  68096
activation_79[0][0]
____________________________________________________________________________________________
batch_normalization_78 (BatchNo (None, 12, 12, 256)  1024
separable_conv2d_62[0][0]
____________________________________________________________________________________________
max_pooling2d_17 (MaxPooling2D) (None, 6, 6, 256)    0
batch_normalization_78[0][0]
```

```
____________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 6, 6, 256)    33024       add_21[0]
[0]
____________________________________________________________________________________________
add_22 (Add)                    (None, 6, 6, 256)    0
max_pooling2d_17[0][0]

conv2d_31[0][0]
____________________________________________________________________________________________
activation_80 (Activation)      (None, 6, 6, 256)    0           add_22[0]
[0]
____________________________________________________________________________________________
separable_conv2d_63 (SeparableC (None, 6, 6, 728)    189400
activation_80[0][0]
____________________________________________________________________________________________
batch_normalization_79 (BatchNo (None, 6, 6, 728)    2912
separable_conv2d_63[0][0]
____________________________________________________________________________________________
activation_81 (Activation)      (None, 6, 6, 728)    0
batch_normalization_79[0][0]
____________________________________________________________________________________________
separable_conv2d_64 (SeparableC (None, 6, 6, 728)    537264
activation_81[0][0]
____________________________________________________________________________________________
batch_normalization_80 (BatchNo (None, 6, 6, 728)    2912
separable_conv2d_64[0][0]
____________________________________________________________________________________________
max_pooling2d_18 (MaxPooling2D) (None, 3, 3, 728)    0
batch_normalization_80[0][0]
____________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 3, 3, 728)    187096      add_22[0]
[0]
____________________________________________________________________________________________
add_23 (Add)                    (None, 3, 3, 728)    0
max_pooling2d_18[0][0]

conv2d_32[0][0]
____________________________________________________________________________________________
activation_82 (Activation)      (None, 3, 3, 728)    0           add_23[0]
[0]
____________________________________________________________________________________________
separable_conv2d_65 (SeparableC (None, 3, 3, 728)    537264
activation_82[0][0]
____________________________________________________________________________________________
batch_normalization_81 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_65[0][0]
____________________________________________________________________________________________
```

```
activation_83 (Activation)      (None, 3, 3, 728)    0
batch_normalization_81[0][0]
__________________________________________________________________________________________
separable_conv2d_66 (SeparableC (None, 3, 3, 728)    537264
activation_83[0][0]
__________________________________________________________________________________________
batch_normalization_82 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_66[0][0]
__________________________________________________________________________________________
activation_84 (Activation)      (None, 3, 3, 728)    0
batch_normalization_82[0][0]
__________________________________________________________________________________________
separable_conv2d_67 (SeparableC (None, 3, 3, 728)    537264
activation_84[0][0]
__________________________________________________________________________________________
batch_normalization_83 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_67[0][0]
__________________________________________________________________________________________
add_24 (Add)                    (None, 3, 3, 728)    0
batch_normalization_83[0][0]
                                                                    add_23[0]
[0]
__________________________________________________________________________________________
activation_85 (Activation)      (None, 3, 3, 728)    0           add_24[0]
[0]
__________________________________________________________________________________________
separable_conv2d_68 (SeparableC (None, 3, 3, 728)    537264
activation_85[0][0]
__________________________________________________________________________________________
batch_normalization_84 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_68[0][0]
__________________________________________________________________________________________
activation_86 (Activation)      (None, 3, 3, 728)    0
batch_normalization_84[0][0]
__________________________________________________________________________________________
separable_conv2d_69 (SeparableC (None, 3, 3, 728)    537264
activation_86[0][0]
__________________________________________________________________________________________
batch_normalization_85 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_69[0][0]
__________________________________________________________________________________________
activation_87 (Activation)      (None, 3, 3, 728)    0
batch_normalization_85[0][0]
__________________________________________________________________________________________
separable_conv2d_70 (SeparableC (None, 3, 3, 728)    537264
activation_87[0][0]
__________________________________________________________________________________________
```

```
batch_normalization_86 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_70[0][0]
__________________________________________________________________________________________
add_25 (Add)                    (None, 3, 3, 728)    0
batch_normalization_86[0][0]
                                                                    add_24[0]
[0]
__________________________________________________________________________________________
activation_88 (Activation)      (None, 3, 3, 728)    0           add_25[0]
[0]
__________________________________________________________________________________________
separable_conv2d_71 (SeparableC (None, 3, 3, 728)    537264
activation_88[0][0]
__________________________________________________________________________________________
batch_normalization_87 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_71[0][0]
__________________________________________________________________________________________
activation_89 (Activation)      (None, 3, 3, 728)    0
batch_normalization_87[0][0]
__________________________________________________________________________________________
separable_conv2d_72 (SeparableC (None, 3, 3, 728)    537264
activation_89[0][0]
__________________________________________________________________________________________
batch_normalization_88 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_72[0][0]
__________________________________________________________________________________________
activation_90 (Activation)      (None, 3, 3, 728)    0
batch_normalization_88[0][0]
__________________________________________________________________________________________
separable_conv2d_73 (SeparableC (None, 3, 3, 728)    537264
activation_90[0][0]
__________________________________________________________________________________________
batch_normalization_89 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_73[0][0]
__________________________________________________________________________________________
add_26 (Add)                    (None, 3, 3, 728)    0
batch_normalization_89[0][0]
                                                                    add_25[0]
[0]
__________________________________________________________________________________________
activation_91 (Activation)      (None, 3, 3, 728)    0           add_26[0]
[0]
__________________________________________________________________________________________
separable_conv2d_74 (SeparableC (None, 3, 3, 728)    537264
activation_91[0][0]
__________________________________________________________________________________________
batch_normalization_90 (BatchNo (None, 3, 3, 728)    2912
```

```
separable_conv2d_74[0][0]
__________________________________________________________________________________________
activation_92 (Activation)      (None, 3, 3, 728)    0
batch_normalization_90[0][0]
__________________________________________________________________________________________
separable_conv2d_75 (SeparableC (None, 3, 3, 728)    537264
activation_92[0][0]
__________________________________________________________________________________________
batch_normalization_91 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_75[0][0]
__________________________________________________________________________________________
activation_93 (Activation)      (None, 3, 3, 728)    0
batch_normalization_91[0][0]
__________________________________________________________________________________________
separable_conv2d_76 (SeparableC (None, 3, 3, 728)    537264
activation_93[0][0]
__________________________________________________________________________________________
batch_normalization_92 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_76[0][0]
__________________________________________________________________________________________
add_27 (Add)                    (None, 3, 3, 728)    0
batch_normalization_92[0][0]
                                                                    add_26[0]
[0]
__________________________________________________________________________________________
activation_94 (Activation)      (None, 3, 3, 728)    0           add_27[0]
[0]
__________________________________________________________________________________________
separable_conv2d_77 (SeparableC (None, 3, 3, 728)    537264
activation_94[0][0]
__________________________________________________________________________________________
batch_normalization_93 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_77[0][0]
__________________________________________________________________________________________
activation_95 (Activation)      (None, 3, 3, 728)    0
batch_normalization_93[0][0]
__________________________________________________________________________________________
separable_conv2d_78 (SeparableC (None, 3, 3, 728)    537264
activation_95[0][0]
__________________________________________________________________________________________
batch_normalization_94 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_78[0][0]
__________________________________________________________________________________________
activation_96 (Activation)      (None, 3, 3, 728)    0
batch_normalization_94[0][0]
__________________________________________________________________________________________
separable_conv2d_79 (SeparableC (None, 3, 3, 728)    537264
```

```
activation_96[0][0]
__________________________________________________________________________________________
batch_normalization_95 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_79[0][0]
__________________________________________________________________________________________
add_28 (Add)                    (None, 3, 3, 728)    0
batch_normalization_95[0][0]
                                                                    add_27[0]
[0]
__________________________________________________________________________________________
activation_97 (Activation)      (None, 3, 3, 728)    0           add_28[0]
[0]
__________________________________________________________________________________________
separable_conv2d_80 (SeparableC (None, 3, 3, 728)    537264
activation_97[0][0]
__________________________________________________________________________________________
batch_normalization_96 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_80[0][0]
__________________________________________________________________________________________
activation_98 (Activation)      (None, 3, 3, 728)    0
batch_normalization_96[0][0]
__________________________________________________________________________________________
separable_conv2d_81 (SeparableC (None, 3, 3, 728)    537264
activation_98[0][0]
__________________________________________________________________________________________
batch_normalization_97 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_81[0][0]
__________________________________________________________________________________________
activation_99 (Activation)      (None, 3, 3, 728)    0
batch_normalization_97[0][0]
__________________________________________________________________________________________
separable_conv2d_82 (SeparableC (None, 3, 3, 728)    537264
activation_99[0][0]
__________________________________________________________________________________________
batch_normalization_98 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_82[0][0]
__________________________________________________________________________________________
add_29 (Add)                    (None, 3, 3, 728)    0
batch_normalization_98[0][0]
                                                                    add_28[0]
[0]
__________________________________________________________________________________________
activation_100 (Activation)     (None, 3, 3, 728)    0           add_29[0]
[0]
__________________________________________________________________________________________
separable_conv2d_83 (SeparableC (None, 3, 3, 728)    537264
activation_100[0][0]
```

```
____________________________________________________________________________________________
batch_normalization_99 (BatchNo (None, 3, 3, 728)    2912
separable_conv2d_83[0][0]
____________________________________________________________________________________________
activation_101 (Activation)     (None, 3, 3, 728)    0
batch_normalization_99[0][0]
____________________________________________________________________________________________
separable_conv2d_84 (SeparableC (None, 3, 3, 728)    537264
activation_101[0][0]
____________________________________________________________________________________________
batch_normalization_100 (BatchN (None, 3, 3, 728)    2912
separable_conv2d_84[0][0]
____________________________________________________________________________________________
activation_102 (Activation)     (None, 3, 3, 728)    0
batch_normalization_100[0][0]
____________________________________________________________________________________________
separable_conv2d_85 (SeparableC (None, 3, 3, 728)    537264
activation_102[0][0]
____________________________________________________________________________________________
batch_normalization_101 (BatchN (None, 3, 3, 728)    2912
separable_conv2d_85[0][0]
____________________________________________________________________________________________
add_30 (Add)                    (None, 3, 3, 728)    0
batch_normalization_101[0][0]
                                                                 add_29[0]
[0]
____________________________________________________________________________________________
activation_103 (Activation)     (None, 3, 3, 728)    0           add_30[0]
[0]
____________________________________________________________________________________________
separable_conv2d_86 (SeparableC (None, 3, 3, 728)    537264
activation_103[0][0]
____________________________________________________________________________________________
batch_normalization_102 (BatchN (None, 3, 3, 728)    2912
separable_conv2d_86[0][0]
____________________________________________________________________________________________
activation_104 (Activation)     (None, 3, 3, 728)    0
batch_normalization_102[0][0]
____________________________________________________________________________________________
separable_conv2d_87 (SeparableC (None, 3, 3, 728)    537264
activation_104[0][0]
____________________________________________________________________________________________
batch_normalization_103 (BatchN (None, 3, 3, 728)    2912
separable_conv2d_87[0][0]
____________________________________________________________________________________________
activation_105 (Activation)     (None, 3, 3, 728)    0
batch_normalization_103[0][0]
```

```
____________________________________________________________________________________________
separable_conv2d_88 (SeparableC (None, 3, 3, 728)    537264
activation_105[0][0]
____________________________________________________________________________________________
batch_normalization_104 (BatchN (None, 3, 3, 728)    2912
separable_conv2d_88[0][0]
____________________________________________________________________________________________
add_31 (Add)                    (None, 3, 3, 728)    0
batch_normalization_104[0][0]
                                                                    add_30[0]
[0]
____________________________________________________________________________________________
activation_106 (Activation)     (None, 3, 3, 728)    0           add_31[0]
[0]
____________________________________________________________________________________________
separable_conv2d_89 (SeparableC (None, 3, 3, 728)    537264
activation_106[0][0]
____________________________________________________________________________________________
batch_normalization_105 (BatchN (None, 3, 3, 728)    2912
separable_conv2d_89[0][0]
____________________________________________________________________________________________
activation_107 (Activation)     (None, 3, 3, 728)    0
batch_normalization_105[0][0]
____________________________________________________________________________________________
separable_conv2d_90 (SeparableC (None, 3, 3, 1024)   753048
activation_107[0][0]
____________________________________________________________________________________________
batch_normalization_106 (BatchN (None, 3, 3, 1024)   4096
separable_conv2d_90[0][0]
____________________________________________________________________________________________
max_pooling2d_19 (MaxPooling2D) (None, 2, 2, 1024)   0
batch_normalization_106[0][0]
____________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 2, 2, 1024)   746496      add_31[0]
[0]
____________________________________________________________________________________________
add_32 (Add)                    (None, 2, 2, 1024)   0
max_pooling2d_19[0][0]

conv2d_33[0][0]
____________________________________________________________________________________________
activation_108 (Activation)     (None, 2, 2, 1024)   0           add_32[0]
[0]
____________________________________________________________________________________________
separable_conv2d_91 (SeparableC (None, 2, 2, 728)    755416
activation_108[0][0]
____________________________________________________________________________________________
```

```
batch_normalization_107 (BatchN (None, 2, 2, 728)    2912
separable_conv2d_91[0][0]
__________________________________________________________________________________________
activation_109 (Activation)     (None, 2, 2, 728)    0
batch_normalization_107[0][0]
__________________________________________________________________________________________
separable_conv2d_92 (SeparableC (None, 2, 2, 1024)   753048
activation_109[0][0]
__________________________________________________________________________________________
batch_normalization_108 (BatchN (None, 2, 2, 1024)   4096
separable_conv2d_92[0][0]
__________________________________________________________________________________________
global_average_pooling2d_1 (Glo (None, 1024)         0
batch_normalization_108[0][0]
__________________________________________________________________________________________
dense_1 (Dense)                 (None, 7)            7175
global_average_pooling2d_1[0][0]
==========================================================================================
Total params: 17,642,719
Trainable params: 17,596,127
Non-trainable params: 46,592
__________________________________________________________________________________________
```

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

batch_size = 128
epochs = 150

datagen = ImageDataGenerator(
      zoom_range=0.2,        # randomly zoom into images
      rotation_range=10,     # randomly rotate images in the range (degrees, 0 to 180)
      width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
      height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
      horizontal_flip=True,   # randomly flip images
      vertical_flip=False)    # randomly flip images

history = xception.fit_generator(
   datagen.flow(train_data, train_labels_one_hot, batch_size=batch_size),
   steps_per_epoch=int(np.ceil(train_data.shape[0] / float(batch_size))),
   epochs = epochs,
   validation_data=(test_data, test_labels_one_hot)
)
```

```
Epoch 1/150
225/225 [==============================] - 2132s 9s/step - loss: 1.7515 -
```

```
acc: 0.2968 - val_loss: 2.3387 - val_acc: 0.2644
Epoch 2/150
225/225 [==============================] - 2083s 9s/step - loss: 1.4834 -
acc: 0.4275 - val_loss: 1.9313 - val_acc: 0.3583
Epoch 3/150
225/225 [==============================] - 2057s 9s/step - loss: 1.3495 -
acc: 0.4862 - val_loss: 2.4919 - val_acc: 0.3756
Epoch 4/150
225/225 [==============================] - 1886s 8s/step - loss: 1.2673 -
acc: 0.5203 - val_loss: 1.6898 - val_acc: 0.4196
Epoch 5/150
225/225 [==============================] - 6899s 31s/step - loss: 1.2175
- acc: 0.5407 - val_loss: 1.4270 - val_acc: 0.5054
Epoch 6/150
225/225 [==============================] - 1723s 8s/step - loss: 1.1719 -
acc: 0.5567 - val_loss: 1.7161 - val_acc: 0.4689
Epoch 7/150
225/225 [==============================] - 1888s 8s/step - loss: 1.1396 -
acc: 0.5720 - val_loss: 1.3121 - val_acc: 0.5252
Epoch 8/150
225/225 [==============================] - 1818s 8s/step - loss: 1.1138 -
acc: 0.5810 - val_loss: 2.0774 - val_acc: 0.3870
Epoch 9/150
225/225 [==============================] - 1925s 9s/step - loss: 1.0852 -
acc: 0.5896 - val_loss: 1.3122 - val_acc: 0.5230
                                :
                                :
                                :
Epoch 15/150
225/225 [==============================] - 1810s 8s/step - loss: 0.9804 -
acc: 0.6293 - val_loss: 1.1222 - val_acc: 0.5921
Epoch 16/150
225/225 [==============================] - 1818s 8s/step - loss: 0.9671 -
acc: 0.6382 - val_loss: 1.1181 - val_acc: 0.5896
Epoch 17/150
225/225 [==============================] - 1756s 8s/step - loss: 0.9532 -
acc: 0.6418 - val_loss: 1.4968 - val_acc: 0.5169
Epoch 18/150
225/225 [==============================] - 1710s 8s/step - loss: 0.9383 -
acc: 0.6488 - val_loss: 1.4305 - val_acc: 0.4873
Epoch 19/150
225/225 [==============================] - 1690s 8s/step - loss: 0.9303 -
acc: 0.6530 - val_loss: 1.2825 - val_acc: 0.5291
Epoch 20/150
225/225 [==============================] - 1677s 7s/step - loss: 0.9244 -
```

```
acc: 0.6527 - val_loss: 1.3539 - val_acc: 0.5208
Epoch 21/150
225/225 [==============================] - 1692s 8s/step - loss: 0.9048 -
acc: 0.6625 - val_loss: 1.1874 - val_acc: 0.5818
Epoch 22/150
225/225 [==============================] - 1700s 8s/step - loss: 0.8931 -
acc: 0.6650 - val_loss: 1.3070 - val_acc: 0.5531
Epoch 23/150
225/225 [==============================] - 1688s 8s/step - loss: 0.8813 -
acc: 0.6735 - val_loss: 1.0668 - val_acc: 0.6208
Epoch 24/150
225/225 [==============================] - 1698s 8s/step - loss: 0.8762 -
acc: 0.6681 - val_loss: 1.2662 - val_acc: 0.5603
Epoch 25/150
225/225 [==============================] - 1671s 7s/step - loss: 0.8584 -
acc: 0.6794 - val_loss: 1.7926 - val_acc: 0.4338
Epoch 26/150
225/225 [==============================] - 1692s 8s/step - loss: 0.8498 -
acc: 0.6803 - val_loss: 1.2420 - val_acc: 0.5690
Epoch 27/150
225/225 [==============================] - 1695s 8s/step - loss: 0.8448 -
acc: 0.6836 - val_loss: 1.1498 - val_acc: 0.6021
Epoch 28/150
225/225 [==============================] - 1697s 8s/step - loss: 0.8260 -
acc: 0.6912 - val_loss: 1.2023 - val_acc: 0.5706
Epoch 29/150
225/225 [==============================] - 1695s 8s/step - loss: 0.8188 -
acc: 0.6936 - val_loss: 1.0196 - val_acc: 0.6361
Epoch 30/150
225/225 [==============================] - 1696s 8s/step - loss: 0.8040 -
acc: 0.6984 - val_loss: 1.1013 - val_acc: 0.6099
Epoch 31/150
225/225 [==============================] - 1749s 8s/step - loss: 0.8010 -
acc: 0.6988 - val_loss: 1.6459 - val_acc: 0.4815
Epoch 32/150
225/225 [==============================] - 1894s 8s/step - loss: 0.7807 -
acc: 0.7087 - val_loss: 1.1472 - val_acc: 0.5996
Epoch 33/150
225/225 [==============================] - 2131s 9s/step - loss: 0.7760 -
acc: 0.7107 - val_loss: 1.2035 - val_acc: 0.5801
Epoch 34/150
225/225 [==============================] - 5026s 22s/step - loss: 0.7625
- acc: 0.7158 - val_loss: 1.1156 - val_acc: 0.6080
Epoch 35/150
 70/225 [========>.....................] - ETA: 38:41 - loss: 0.7274 -
acc: 0.7291
```

## XVI. Sources

- Visualization : https://github.com/JostineHo/mememoji/blob/master/data_visualization.ipynb
- State of the art Architecture : https://github.com/amineHorseman/facial-expression-recognition-using-cnn
- Eyes Tracking : https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/
- Face Alignment : https://www.pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/
- C.Pramerdorfer, and M.Kampel.Facial Expression Recognition using Con-volutional Neural Networks: State of the Art. Computer Vision Lab, TU Wien. https://arxiv.org/pdf/1612.02903.pdf
- A Brief Review of Facial Emotion Recognition Based on Visual Information : https://www.mdpi.com/1424-8220/18/2/401/pdf
- Going deeper in facial expression recognition using deep neural networks : https://ieeexplore.ieee.org/document/7477450

# Performance

The models have been trained on Google Colab using free GPUs.The set of emotions we are trying to predict are the following :

1. Happiness 2. Sadness 3. Fear 4. Disgust

5. Surprise 6. Neutral 7. Anger

| Features | Accuracy |
|---|---|
| LGBM on flat image | **--.-%** |
| LGBM on auto-encoded image | **--.-%** |
| SVM on HOG Features | **32.8%** |
| SVM on Facial Landmarks features | **46.4%** |
| SVM on Facial Landmarks and HOG features | **47.5%** |
| SVM on Sliding window Landmarks & HOG | **24.6%** |

| Features | Accuracy |
|---|---|
| Simple Deep Learning Architecture | **62.7%** |
| Inception Architecture | **59.5%** |
| Xception Architecture | **64.5%** |
| DeXpression Architecture | **--.-%** |
| Hybrid (HOG, Landmarks, Image) | **45.8%** |

# Multimodal Emotion Recognition WebApp

We analyse facial, vocal and textual emotions, using mostly deep learning based approaches. We deployed a web app using Flask .

## Procedure to Run

To use the web app :

1. Clone the project locally
2. Go in the WebApp folder
3. Run `$ pip install -r requirements.txt``
4. Launch the app by running `python main.py`
5. Go to http://127.0.0.1:5000/ (if running as the local host or else follow the link given in your terminal or command prompt)

## Getting the feedback

For both the text and the audio, a button will directly allow you to get feedback. A dashboard displays your performance compared to other candidates.

For the video, due to restrictions of Flask, we are recording the video input for 45 seconds. After this time, the image will freeze. Simply switch the URL to `/video_dash` instead of `/video1` in the URL bar to go to the dashboard.

# Organization of Files and Folder

The organization of the project is the following :

- •Models : All the pre-trained models used by the WebApp
- •library : The Python scripts that run the emotion detection algorithms
- •static :
  - •CSS : The CSS style sheet and fixed images to display
  - •JS : The JavaScript of the app (D3.js) and the databases that store the information
- •templates : All the HTML pages of the project
- •tmp : Temporary files (i.e. an image from video interview, an audio file or a PDF)
- •main.py : The Flask page that calls the functions and redirects to HTML files

## Flask Server Program

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

### General imports ###
from __future__ import division
import numpy as np
import pandas as pd
import time
import re
import os
from collections import Counter
import altair as alt

### Flask imports
import requests
from flask import Flask, render_template, session, request, redirect, flash, Response

### Audio imports ###
from library.speech_emotion_recognition import *

### Video imports ###
from library.video_emotion_recognition import *

### Text imports ###
from library.text_emotion_recognition import *
from library.text_preprocessor import *
from nltk import *
import nltk
```

```
from tika import parser
from werkzeug.utils import secure_filename
import tempfile



#for nltk lookup error uncomment the line of code below.
#nltk.download('all')



# Flask config
app = Flask(__name__)
app.secret_key = b'(\xee\x00\xd4\xce"\xcf\xe8@\r\xde\xfc\xbdJ\x08W'
app.config['UPLOAD_FOLDER'] = '/Upload'

################################################################################
################################### INDEX ######################################
################################################################################

# Home page
@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')

################################################################################
################################### RULES ######################################
################################################################################

# Rules of the game
@app.route('/rules')
```

```
def rules():
    return render_template('rules.html')


################################################################################
############################### VIDEO INTERVIEW ################################
################################################################################


# Read the overall dataframe before the user starts to add his own data
df = pd.read_csv('static/js/db/histo.txt', sep=",")


# Video interview template
@app.route('/video', methods=['POST'])
def video() :
    # Display a warning message
    flash('You will have 45 seconds to discuss the topic mentioned above. Due to restrictions, we are not able to redirect you once the video is over. Please move your URL to /video_dash instead of /video_1 once over. You will be able to see your results then.')
    return render_template('video.html')


# Display the video flow (face, landmarks, emotion)
@app.route('/video_1', methods=['POST'])
def video_1() :
    try :
        # Response is used to display a flow of information
        return Response(gen(),mimetype='multipart/x-mixed-replace; boundary=frame')
    #return Response(stream_template('video.html', gen()))
    except :
        return None
```

```
# Dashboard
@app.route('/video_dash', methods=("POST", "GET"))
def video_dash():

    # Load personal history
    df_2 = pd.read_csv('static/js/db/histo_perso.txt')



    def emo_prop(df_2) :
        return [int(100*len(df_2[df_2.density==0])/len(df_2)),
                int(100*len(df_2[df_2.density==1])/len(df_2)),
                int(100*len(df_2[df_2.density==2])/len(df_2)),
                int(100*len(df_2[df_2.density==3])/len(df_2)),
                int(100*len(df_2[df_2.density==4])/len(df_2)),
                int(100*len(df_2[df_2.density==5])/len(df_2)),
                int(100*len(df_2[df_2.density==6])/len(df_2))]

    emotions = ["Angry", "Disgust", "Fear",  "Happy", "Sad", "Surprise", "Neutral"]
    emo_perso = {}
    emo_glob = {}

    for i in range(len(emotions)) :
        emo_perso[emotions[i]] = len(df_2[df_2.density==i])
        emo_glob[emotions[i]] = len(df[df.density==i])

    df_perso = pd.DataFrame.from_dict(emo_perso, orient='index')
    df_perso = df_perso.reset_index()
    df_perso.columns = ['EMOTION', 'VALUE']
    df_perso.to_csv('static/js/db/hist_vid_perso.txt', sep=",", index=False)
```

```
df_glob = pd.DataFrame.from_dict(emo_glob, orient='index')
df_glob = df_glob.reset_index()
df_glob.columns = ['EMOTION', 'VALUE']
df_glob.to_csv('static/js/db/hist_vid_glob.txt', sep=",", index=False)

emotion = df_2.density.mode()[0]
emotion_other = df.density.mode()[0]

def emotion_label(emotion) :
   if emotion == 0 :
      return "Angry"
   elif emotion == 1 :
      return "Disgust"
   elif emotion == 2 :
      return "Fear"
   elif emotion == 3 :
      return "Happy"
   elif emotion == 4 :
      return "Sad"
   elif emotion == 5 :
      return "Surprise"
   else :
      return "Neutral"

### Altair Plot
df_altair = pd.read_csv('static/js/db/prob.csv', header=None, index_col=None).reset_index()
df_altair.columns = ['Time', 'Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
```

```
angry = alt.Chart(df_altair).mark_line(color='orange', strokeWidth=2).encode(
  x='Time:Q',
  y='Angry:Q',
  tooltip=["Angry"]
)

disgust = alt.Chart(df_altair).mark_line(color='red', strokeWidth=2).encode(
   x='Time:Q',
   y='Disgust:Q',
   tooltip=["Disgust"])


fear = alt.Chart(df_altair).mark_line(color='green', strokeWidth=2).encode(
   x='Time:Q',
   y='Fear:Q',
   tooltip=["Fear"])


happy = alt.Chart(df_altair).mark_line(color='blue', strokeWidth=2).encode(
   x='Time:Q',
   y='Happy:Q',
   tooltip=["Happy"])


sad = alt.Chart(df_altair).mark_line(color='black', strokeWidth=2).encode(
   x='Time:Q',
   y='Sad:Q',
   tooltip=["Sad"])
```

```
    surprise = alt.Chart(df_altair).mark_line(color='pink', strokeWidth=2).encode(
        x='Time:Q',
        y='Surprise:Q',
        tooltip=["Surprise"])


    neutral = alt.Chart(df_altair).mark_line(color='brown', strokeWidth=2).encode(
        x='Time:Q',
        y='Neutral:Q',
        tooltip=["Neutral"])


    chart = (angry + disgust + fear + happy + sad + surprise + neutral).properties(
    width=1000, height=400, title='Probability of each emotion over time')


    chart.save('static/CSS/chart.html')


    return render_template('video_dash.html', emo=emotion_label(emotion), emo_other =
emotion_label(emotion_other), prob = emo_prop(df_2), prob_other = emo_prop(df))



################################################################################
############################### AUDIO INTERVIEW ################################
################################################################################


# Audio Index
@app.route('/audio_index', methods=['POST'])
```

```
def audio_index():


    # Flash message
    flash("After pressing the button above, you will have 15sec to answer the question.")


    return render_template('audio.html', display_button=False)


# Audio Recording
@app.route('/audio_recording', methods=("POST", "GET"))
def audio_recording():


    # Instanciate new SpeechEmotionRecognition object
    SER = speechEmotionRecognition()


    # Voice Recording
    rec_duration = 16 # in sec
    rec_sub_dir = os.path.join('tmp','voice_recording.wav')
    SER.voice_recording(rec_sub_dir, duration=rec_duration)


    # Send Flash message
    flash("The recording is over! You now have the opportunity to do an analysis of your emotions.
If you wish, you can also choose to record yourself again.")


    return render_template('audio.html', display_button=True)




# Audio Emotion Analysis
@app.route('/audio_dash', methods=("POST", "GET"))
def audio_dash():
```

```
    # Sub dir to speech emotion recognition model
    model_sub_dir = os.path.join('Models', 'audio.hdf5')

    # Instanciate new SpeechEmotionRecognition object
    SER = speechEmotionRecognition(model_sub_dir)

    # Voice Record sub dir
    rec_sub_dir = os.path.join('tmp','voice_recording.wav')

    # Predict emotion in voice at each time step
    step = 1 # in sec
    sample_rate = 16000 # in kHz
    emotions, timestamp = SER.predict_emotion_from_file(rec_sub_dir,
chunk_step=step*sample_rate)

    # Export predicted emotions to .txt format
    SER.prediction_to_csv(emotions, os.path.join("static/js/db", "audio_emotions.txt"),
mode='w')
    SER.prediction_to_csv(emotions, os.path.join("static/js/db", "audio_emotions_other.txt"),
mode='a')

    # Get most common emotion during the interview
    major_emotion = max(set(emotions), key=emotions.count)

    # Calculate emotion distribution
    emotion_dist = [int(100 * emotions.count(emotion) / len(emotions)) for emotion in
SER._emotion.values()]

    # Export emotion distribution to .csv format for D3JS
```

```
    df = pd.DataFrame(emotion_dist, index=SER._emotion.values(),
columns=['VALUE']).rename_axis('EMOTION')
    df.to_csv(os.path.join('static/js/db','audio_emotions_dist.txt'), sep=',')

    # Get most common emotion of other candidates
    df_other = pd.read_csv(os.path.join("static/js/db", "audio_emotions_other.txt"), sep=",")

    # Get most common emotion during the interview for other candidates
    major_emotion_other = df_other.EMOTION.mode()[0]

    # Calculate emotion distribution for other candidates
    emotion_dist_other = [int(100 * len(df_other[df_other.EMOTION==emotion]) / len(df_other))
for emotion in SER._emotion.values()]

    # Export emotion distribution to .csv format for D3JS
    df_other = pd.DataFrame(emotion_dist_other, index=SER._emotion.values(),
columns=['VALUE']).rename_axis('EMOTION')
    df_other.to_csv(os.path.join('static/js/db','audio_emotions_dist_other.txt'), sep=',')

    # Sleep
    time.sleep(0.5)

    return render_template('audio_dash.html', emo=major_emotion,
emo_other=major_emotion_other, prob=emotion_dist, prob_other=emotion_dist_other)



################################################################################
################################## TEXT INTERVIEW ##################################
################################################################################
```

```
global df_text


tempdirectory = tempfile.gettempdir()


@app.route('/text', methods=['POST'])
def text() :
   return render_template('text.html')


def get_personality(text):
   try:
      pred = predict().run(text, model_name = "Personality_traits_NN")
      return pred
   except KeyError:
      return None


def get_text_info(text):
   text = text[0]
   words = wordpunct_tokenize(text)
   common_words = FreqDist(words).most_common(100)
   counts = Counter(words)
   num_words = len(text.split())
   return common_words, num_words, counts


def preprocess_text(text):
   preprocessed_texts = NLTKPreprocessor().transform([text])
   return preprocessed_texts


@app.route('/text_1', methods=['POST'])
def text_1():
```

```
text = request.form.get('text')
traits = ['Extraversion', 'Neuroticism', 'Agreeableness', 'Conscientiousness', 'Openness']
probas = get_personality(text)[0].tolist()

df_text = pd.read_csv('static/js/db/text.txt', sep=",")
#As of pandas 2.0, append (precedently deprecated) was effectively removed.
#You need to use concat instead (for most applications)
df_new = pd.concat([df_text, pd.DataFrame([probas], columns=traits)], axis=0)
#df_new = df_text.concat(pd.DataFrame([probas], columns=traits))
df_new.to_csv('static/js/db/text.txt', sep=",", index=False)

perso = {}
perso['Extraversion'] = probas[0]
perso['Neuroticism'] = probas[1]
perso['Agreeableness'] = probas[2]
perso['Conscientiousness'] = probas[3]
perso['Openness'] = probas[4]

df_text_perso = pd.DataFrame.from_dict(perso, orient='index')
df_text_perso = df_text_perso.reset_index()
df_text_perso.columns = ['Trait', 'Value']

df_text_perso.to_csv('static/js/db/text_perso.txt', sep=',', index=False)

means = {}
means['Extraversion'] = np.mean(df_new['Extraversion'])
means['Neuroticism'] = np.mean(df_new['Neuroticism'])
means['Agreeableness'] = np.mean(df_new['Agreeableness'])
```

```
    means['Conscientiousness'] = np.mean(df_new['Conscientiousness'])
    means['Openness'] = np.mean(df_new['Openness'])


    probas_others = [np.mean(df_new['Extraversion']), np.mean(df_new['Neuroticism']),
np.mean(df_new['Agreeableness']), np.mean(df_new['Conscientiousness']),
np.mean(df_new['Openness'])]
    probas_others = [int(e*100) for e in probas_others]


    df_mean = pd.DataFrame.from_dict(means, orient='index')
    df_mean = df_mean.reset_index()
    df_mean.columns = ['Trait', 'Value']


    df_mean.to_csv('static/js/db/text_mean.txt', sep=',', index=False)
    trait_others = df_mean.loc[df_mean['Value'].idxmax()]['Trait']


    probas = [int(e*100) for e in probas]


    data_traits = zip(traits, probas)


    session['probas'] = probas
    session['text_info'] = {}
    session['text_info']["common_words"] = []
    session['text_info']["num_words"] = []


    preprocessed_text = preprocess_text(text)
    common_words, num_words, counts = get_text_info(preprocessed_text)


    session['text_info']["common_words"].append(common_words)
    session['text_info']["num_words"].append(num_words)
```

```
    trait = traits[probas.index(max(probas))]


    with open("static/js/db/words_perso.txt", "w") as d:
        d.write("WORDS,FREQ" + '\n')
        for line in counts :
            d.write(line + "," + str(counts[line]) + '\n')
        d.close()


    with open("static/js/db/words_common.txt", "a") as d:
        for line in counts :
            d.write(line + "," + str(counts[line]) + '\n')
        d.close()


    df_words_co = pd.read_csv('static/js/db/words_common.txt', sep=',', on_bad_lines='skip')
    df_words_co.FREQ = df_words_co.FREQ.apply(pd.to_numeric)
    df_words_co = df_words_co.groupby('WORDS').sum().reset_index()
    df_words_co.to_csv('static/js/db/words_common.txt', sep=",", index=False)
    common_words_others = df_words_co.sort_values(by=['FREQ'], ascending=False)['WORDS']
[:15]


    df_words_perso = pd.read_csv('static/js/db/words_perso.txt', sep=',', on_bad_lines='skip')
    common_words_perso = df_words_perso.sort_values(by=['FREQ'], ascending=False)
['WORDS'][:15]


    return render_template('text_dash.html', traits = probas, trait = trait, trait_others =
trait_others, probas_others = probas_others, num_words = num_words, common_words =
common_words_perso, common_words_others=common_words_others)


ALLOWED_EXTENSIONS = set(['pdf'])
```

```
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


@app.route('/text_pdf', methods=['POST'])
def text_pdf():
    f = request.files['file']
    f.save(secure_filename(f.filename))

    text = parser.from_file(f.filename)['content']
    traits = ['Extraversion', 'Neuroticism', 'Agreeableness', 'Conscientiousness', 'Openness']
    probas = get_personality(text)[0].tolist()

    df_text = pd.read_csv('static/js/db/text.txt', sep=",")
    df_new = pd.concat([df_text, pd.DataFrame([probas], columns=traits)], axis=0)
    #df_new = df_text.append(pd.DataFrame([probas], columns=traits))
    df_new.to_csv('static/js/db/text.txt', sep=",", index=False)

    perso = {}
    perso['Extraversion'] = probas[0]
    perso['Neuroticism'] = probas[1]
    perso['Agreeableness'] = probas[2]
    perso['Conscientiousness'] = probas[3]
    perso['Openness'] = probas[4]

    df_text_perso = pd.DataFrame.from_dict(perso, orient='index')
    df_text_perso = df_text_perso.reset_index()
    df_text_perso.columns = ['Trait', 'Value']
```

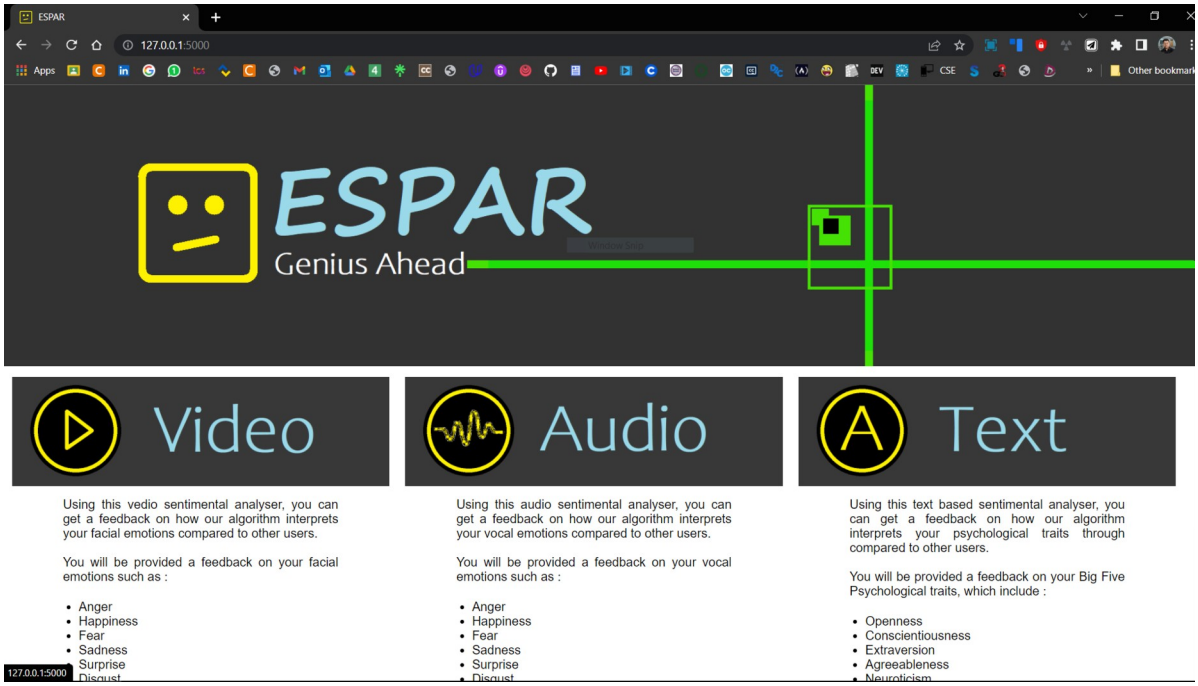
```
    df_text_perso.to_csv('static/js/db/text_perso.txt', sep=',', index=False)


    means = {}
    means['Extraversion'] = np.mean(df_new['Extraversion'])
    means['Neuroticism'] = np.mean(df_new['Neuroticism'])
    means['Agreeableness'] = np.mean(df_new['Agreeableness'])
    means['Conscientiousness'] = np.mean(df_new['Conscientiousness'])
    means['Openness'] = np.mean(df_new['Openness'])


    probas_others = [np.mean(df_new['Extraversion']), np.mean(df_new['Neuroticism']),
np.mean(df_new['Agreeableness']), np.mean(df_new['Conscientiousness']),
np.mean(df_new['Openness'])]
    probas_others = [int(e*100) for e in probas_others]


    df_mean = pd.DataFrame.from_dict(means, orient='index')
    df_mean = df_mean.reset_index()
    df_mean.columns = ['Trait', 'Value']


    df_mean.to_csv('static/js/db/text_mean.txt', sep=',', index=False)
    trait_others = df_mean.loc[df_mean['Value'].idxmax()]['Trait']


    probas = [int(e*100) for e in probas]


    data_traits = zip(traits, probas)


    session['probas'] = probas
    session['text_info'] = {}
    session['text_info']["common_words"] = []
    session['text_info']["num_words"] = []
```

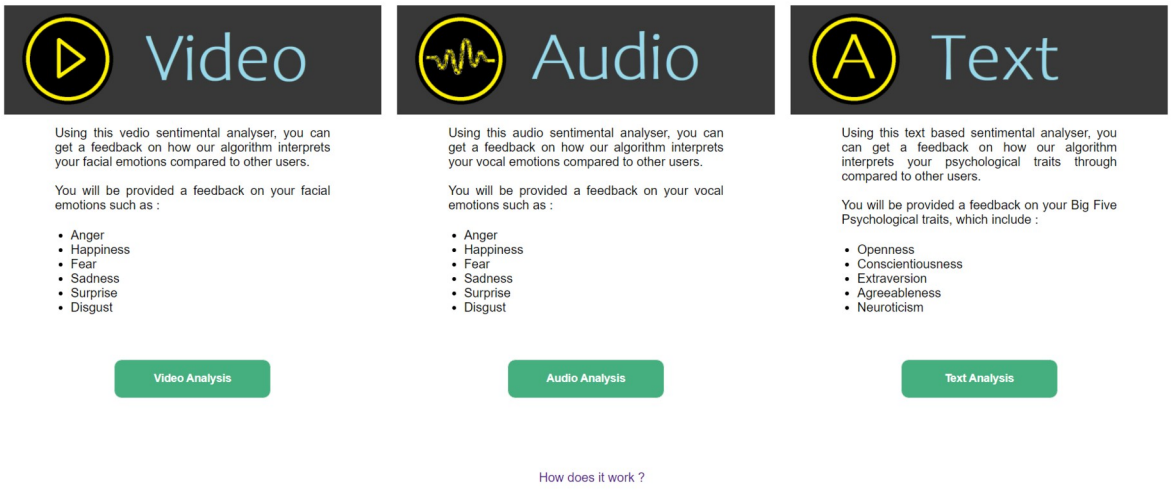
```
    preprocessed_text = preprocess_text(text)
    common_words, num_words, counts = get_text_info(preprocessed_text)

    session['text_info']["common_words"].append(common_words)
    session['text_info']["num_words"].append(num_words)

    trait = traits[probas.index(max(probas))]

    with open("static/js/db/words_perso.txt", "w") as d:
        d.write("WORDS,FREQ" + '\n')
        for line in counts :
            d.write(line + "," + str(counts[line]) + '\n')
        d.close()

    with open("static/js/db/words_common.txt", "a") as d:
        for line in counts :
            d.write(line + "," + str(counts[line]) + '\n')
        d.close()

    df_words_co = pd.read_csv('static/js/db/words_common.txt', sep=',', on_bad_lines='skip')
    df_words_co.FREQ = df_words_co.FREQ.apply(pd.to_numeric)
    df_words_co = df_words_co.groupby('WORDS').sum().reset_index()
    df_words_co.to_csv('static/js/db/words_common.txt', sep=",", index=False)
    common_words_others = df_words_co.sort_values(by=['FREQ'], ascending=False)['WORDS']
[:15]

    df_words_perso = pd.read_csv('static/js/db/words_perso.txt', sep=',', on_bad_lines='skip')
    common_words_perso = df_words_perso.sort_values(by=['FREQ'], ascending=False)
```

```
['WORDS'][:15]


    return render_template('text_dash.html', traits = probas, trait = trait, trait_others =
trait_others, probas_others = probas_others, num_words = num_words, common_words =
common_words_perso, common_words_others=common_words_others)


if __name__ == '__main__':
    #For the live server where non-local connection can access it.
    #to access this, use this ip address http://192.168.1.6:5000/
    #app.run(host="192.168.1.6", port=5000, debug=True)
    #To run this local uncomment the below line.
    app.run(debug=True)
```

# Sample Run

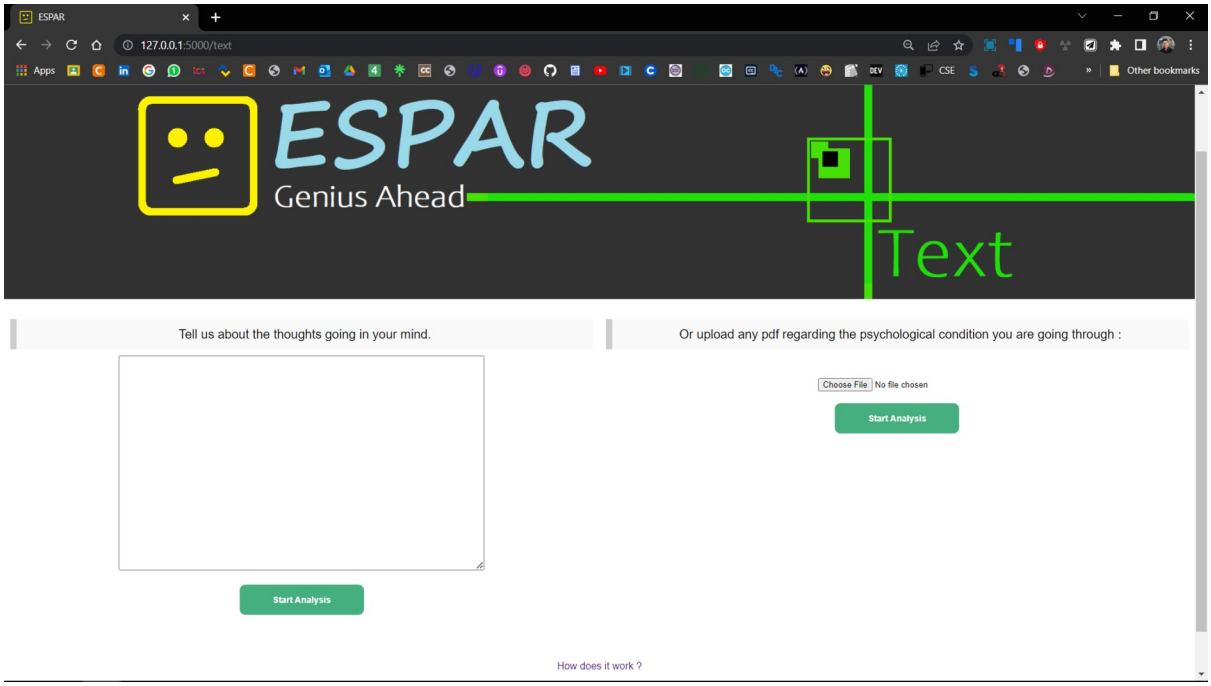
The home Screen or home web-page



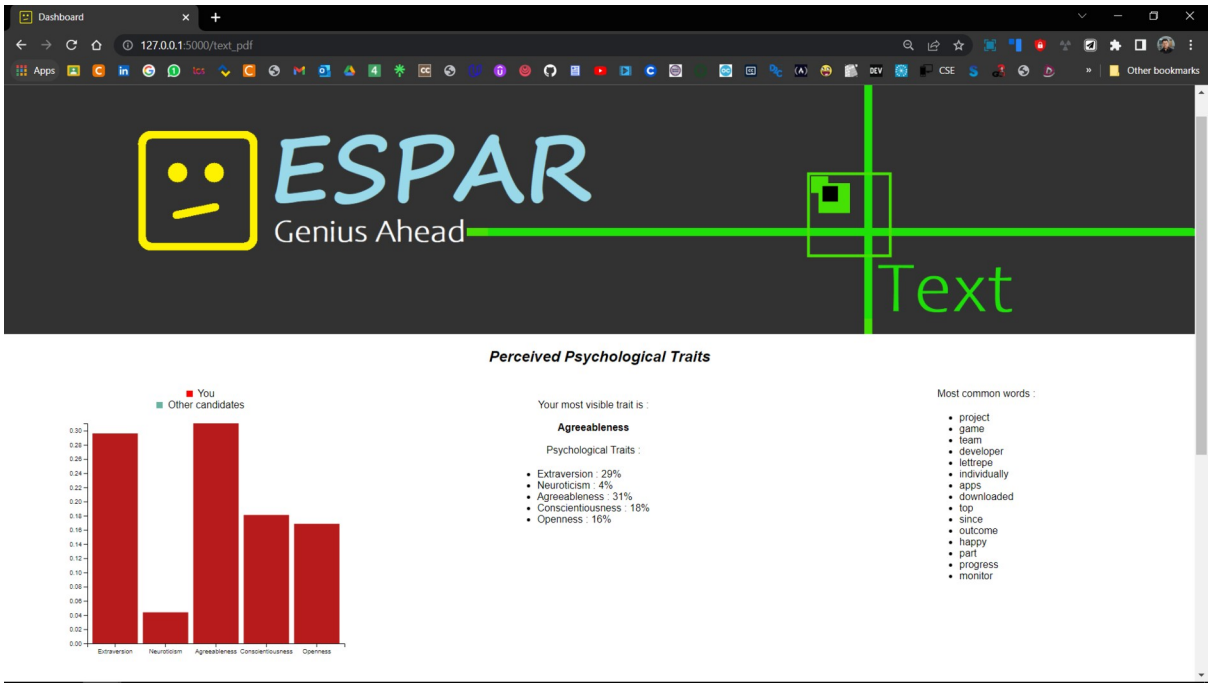



# 1. TEXT ANALYSIS

http://127.0.0.1:5000/text

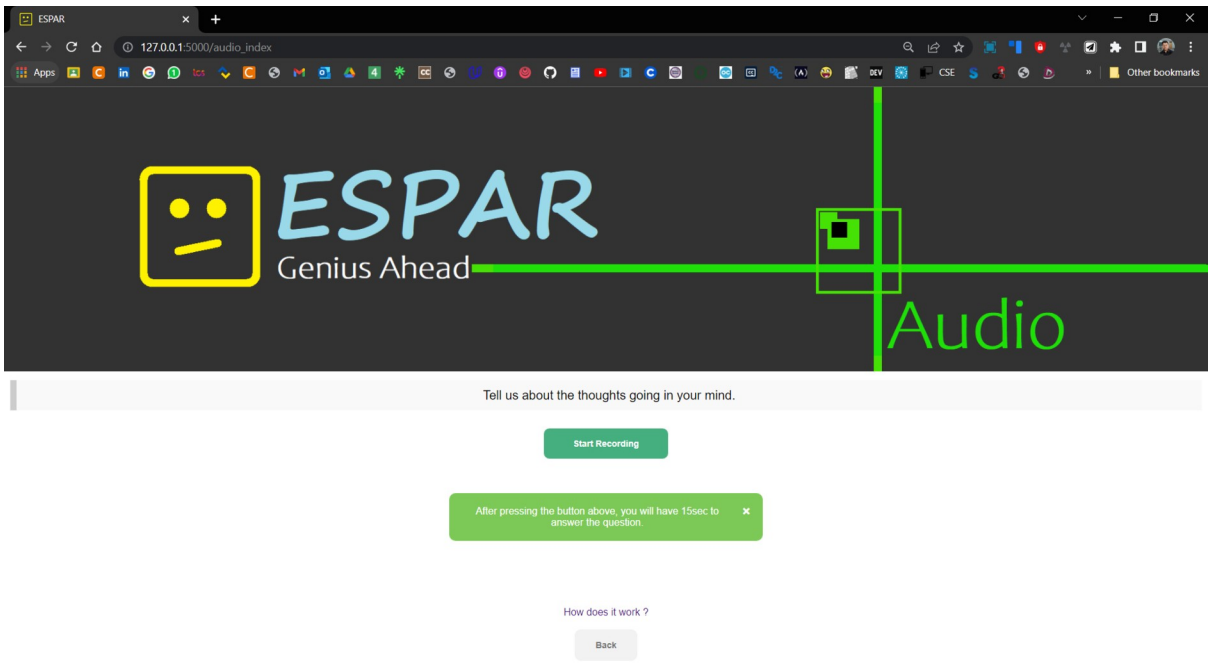


TEXT ANALYSIS : http://127.0.0.1:5000/text_pdf

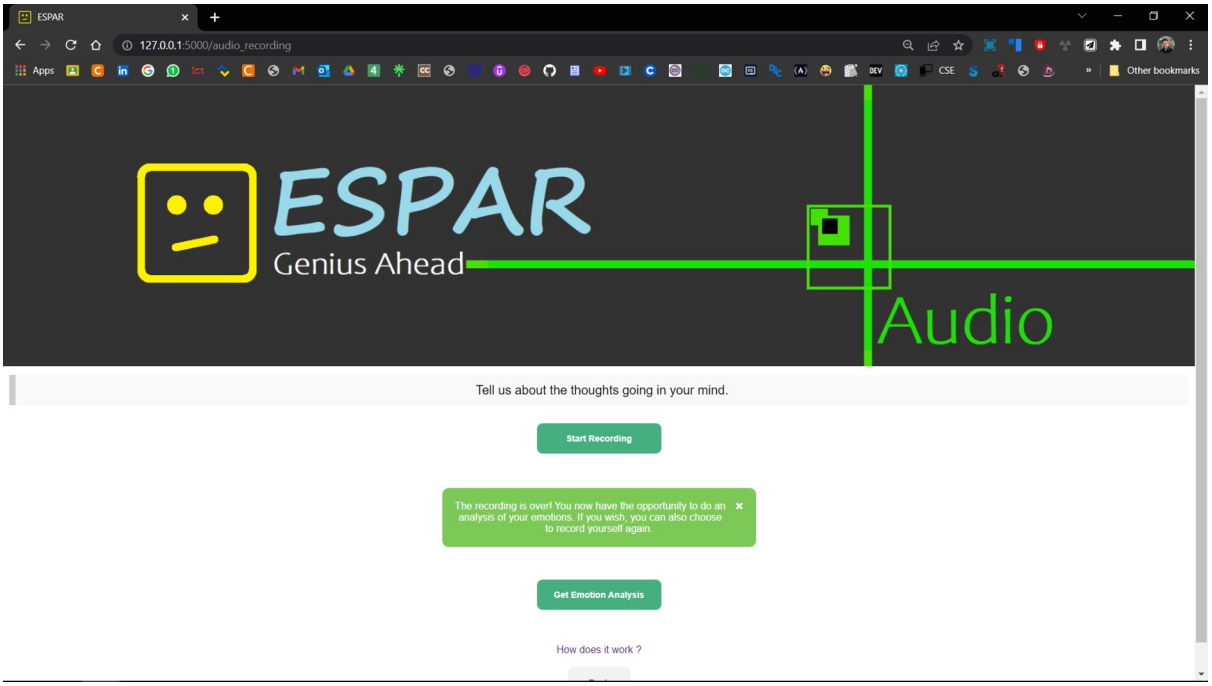



## 2. AUDIO ANALYSIS

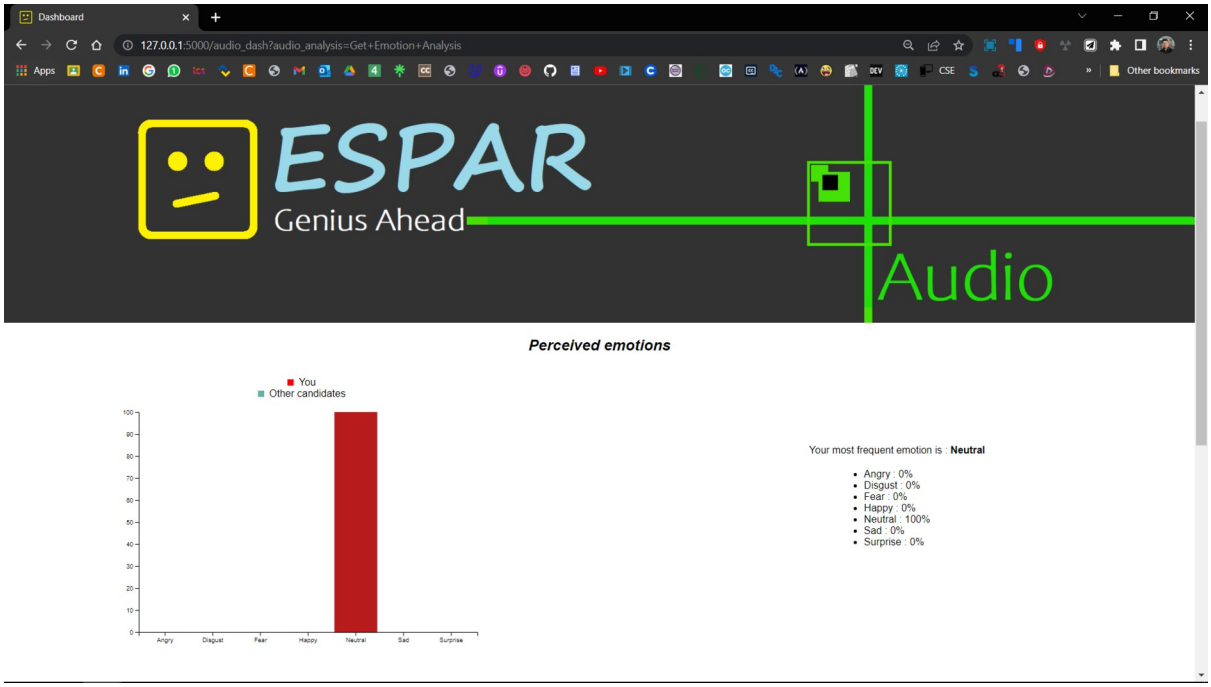
http://127.0.0.1:5000/audio_index



After the recording has started and finished : http://127.0.0.1:5000/audio_recording

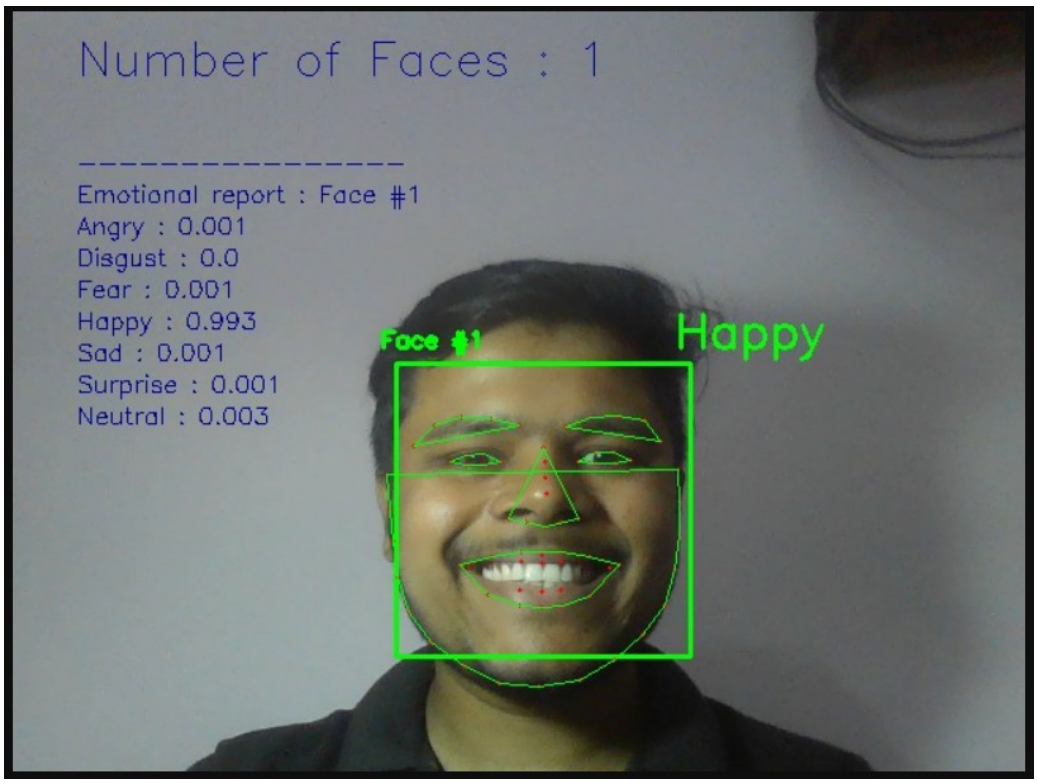


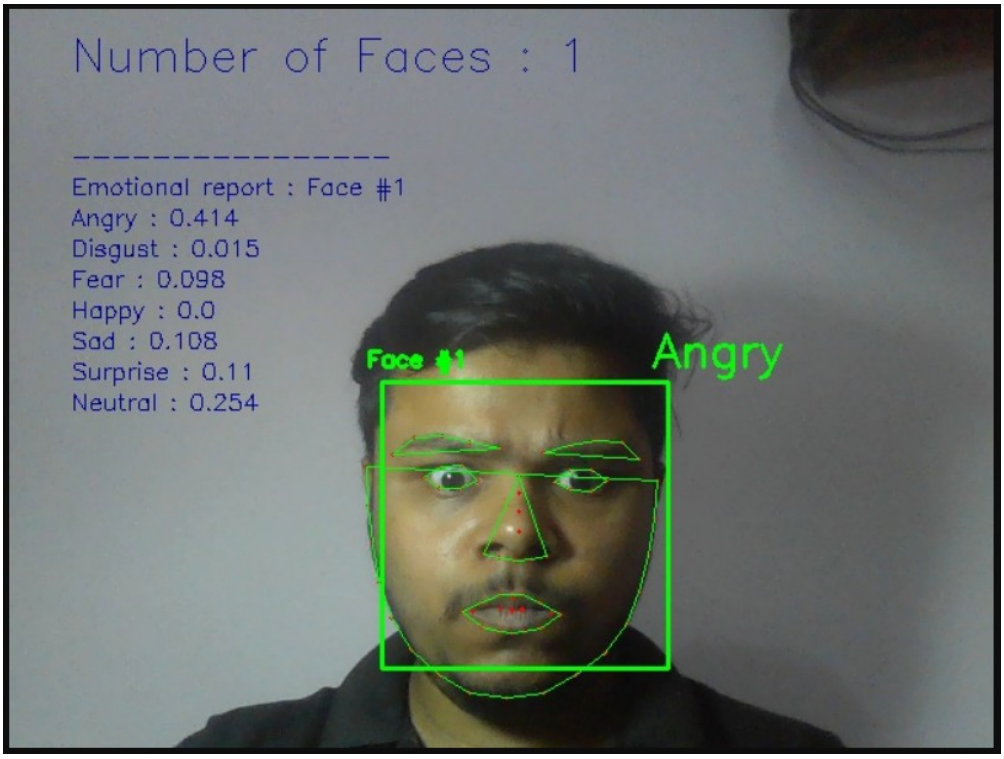

Audio Analysis Results: http://127.0.0.1:5000/audio_dash?audio_analysis=Get+Emotion+Analysis

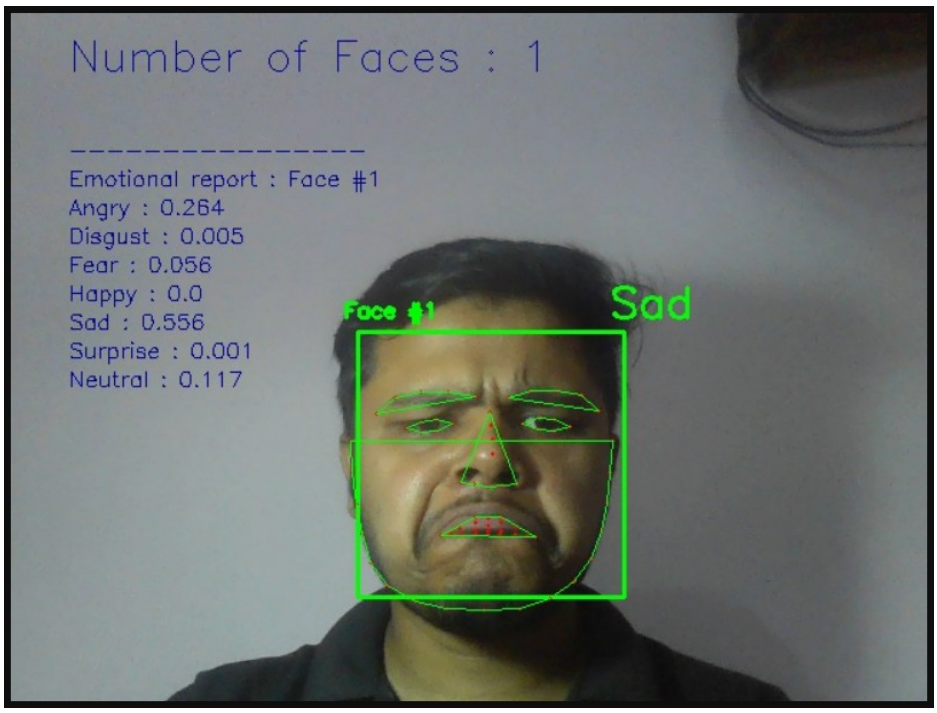


## 3. VIDEO SENTIMENTAL ANALYSIS

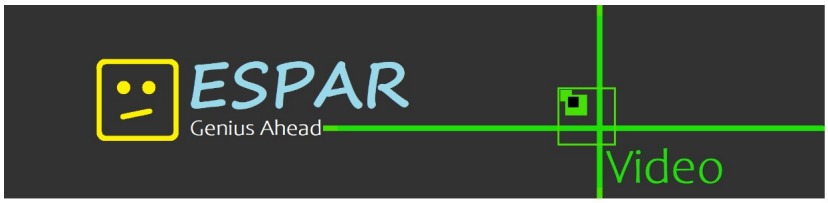
http://127.0.0.1:5000/video_1



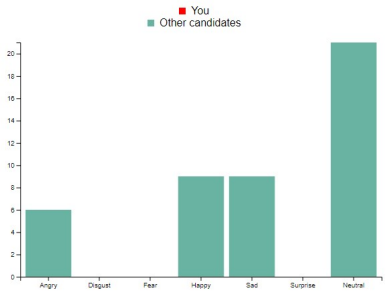




Number of Faces : 1
Emotional report : Face #1
Angry : 0.264
Disgust : 0.005
Fear : 0.056
Happy : 0.0
Sad : 0.556
Surprise : 0.001
Neutral : 0.117
Face #1
Sad

Video Analysis Results : http://127.0.0.1:5000/video_dash




***Perceived emotions***




**Facial Emotions**

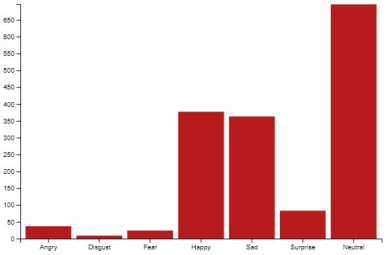
Your most frequent emotion is :

**Neutral**

- Anger : 13%
- Disgust : 0%
- Fear : 0%
- Happiness : 20%
- Sadness : 20%
- Surprise : 0%
- Neutrality : 46%

---

***Other candidates***



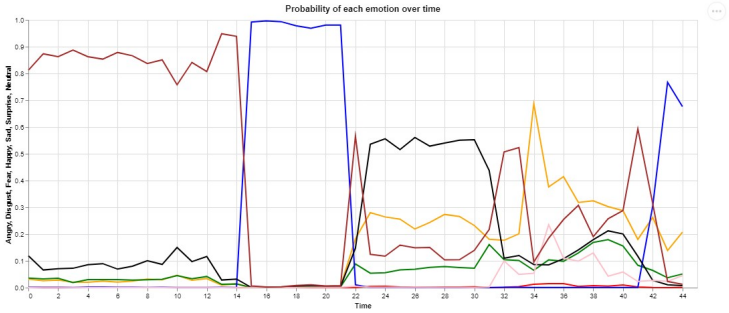

Other candidates most frequent emotion is :

**Neutral**

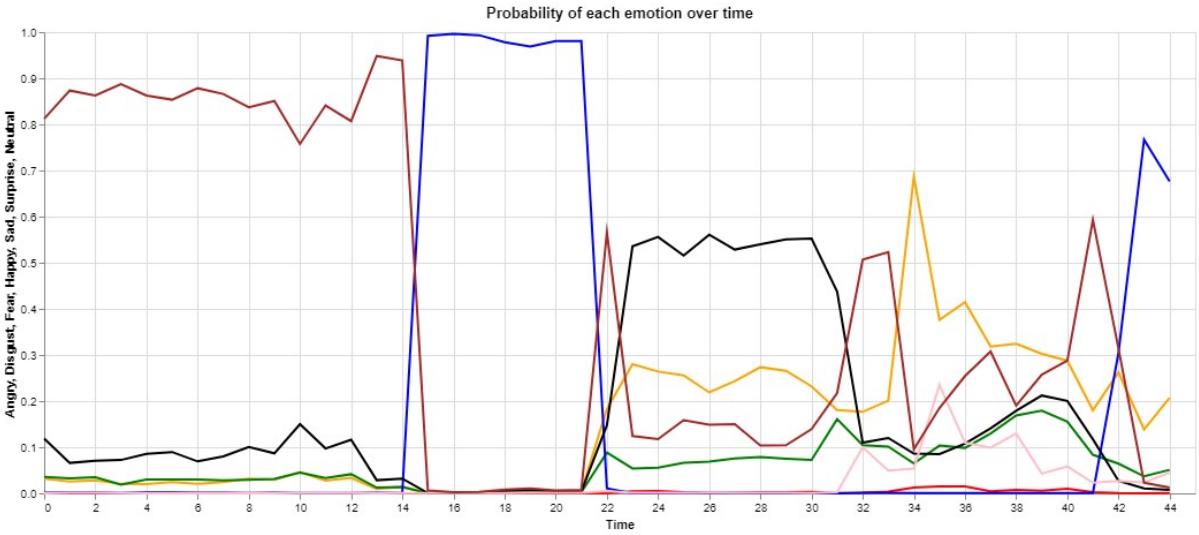
- Angry : 2%
- Disgust : 0%
- Fear : 1%
- Happy : 23%
- Neutral : 22%
- Sad : 5%
- Surprise : 43%

---

***Over time***




Back


Probability of each emotion over time
Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral
Time
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44

# References

1. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS), https://zenodo.org/record/1188976/?f=3.XAcEs5NKhQK

2. The Facial Emotion Recognition Challenge from Kaggle, https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

3. End-to-End Multimodal Emotion Recognition using Deep Neural Networks,

https://arxiv.org/pdf/1704.08619.pdf

4. Facial Expression Recognition using Convolutional Neural Networks: State of the Art, https://arxiv.org/pdf/1612.02903.pdf

5. B.Basharirad, and M.Moradhaseli. Speech emotion recognition methods: A literature review. AIP Conference Proceedings 2017. https://aip.scitation.org/doi/pdf/10.1063/1.5005438

6. L.Chen, M.Mao, Y.Xue and L.L.Cheng. Speech emotion recognition: Features and classication models. Digit. Signal Process, vol 22 Dec. 2012.

7. T.Vogt, E.Andre and J.Wagner. Automatic Recognition of Emotions from Speech: A Review of the Literature and Recommendations for Practical Realisation. Aectand Emotion in Human-Computer Interaction, 2008.

8. T.Vogt and E.Andre. Improving Automatic Emotion Recognition from Speech via Gender Dierentiation. Language Resources and Evaluation Conference, 2006.

9. T.Giannakopoulos. pyAudioAnalysis: An Open-Source Python Library for Audio Signal Analysis. Dec. 2015https://doi.org/10.1371/journal.pone.0144610

*The End*