# Configuration Manual

MSc Research Project
Data Analytics

Tiago Leonel do Nascimento
Student ID: 19143486

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Tiago Leonel do Nascimento |
| **Student ID:** | 19143486 |
| **Programme:** | Data Analytics |
| **Year:** | 2022 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Jorge Basilio |
| **Submission Due Date:** | 15/08/2022 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 1822 |
| **Page Count:** | 15 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | |
| **Date:** | 18th September 2022 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies). | ☐ |
| **Attach a Moodle submission receipt of the online project submission**, to each project (including multiple copies). | ☐ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | ☐ |

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Tiago Leonel do Nascimento
19143486

## 1 Introduction

The specifications for the A Comparative Study of Pixel Values and Landmark Detection Features to Solve for Facial Emotion Recognition research are described in this configuration manual. It covers hardware and system specifications that should be considered minimum requirements for the study results replication. The following sections cover all phases of the implementation process and the overall assessment of the research work.

## 2 System Specification

### 2.1 Configuration of Hardware

A M1 MacBook pro 8GB was used for the entire study. Figure 1 shows the specific information regarding Device, Chip, Memory and OS also listed below.

- **Device**: MacBook Pro (13-inch, M1, 2020)
- **Chip**: Apple M1
- **Memory**: 8 GB

**MacOS Information**


Figure 1: Device Specification

## 2.2 Software Configuration

Google Colaboratory (Colab) was used for the entire duration of the study. Due to the Mac M1 have a very laborious process to install TensorFlow which can be easily accessed by utilizing Colab online. This meant that a combination of Google Drive and Google Colab had to be used to importing the datasets used. Figure 2 displays the landing page for google colab[1]

Software Requirements below:

- **OS**: macOS Monterey v 12.5
- Google Account to access Google Drive to store files.
- Google Basic Subscription Plan for 100mb of drive storage (some datasets were over the free storage capacity.).
- Same google account can be used to access Google Colab.
- Google Colab Pro+ subscription (enables access to higher RAM and GPU, TPU.).

**Google Colab**

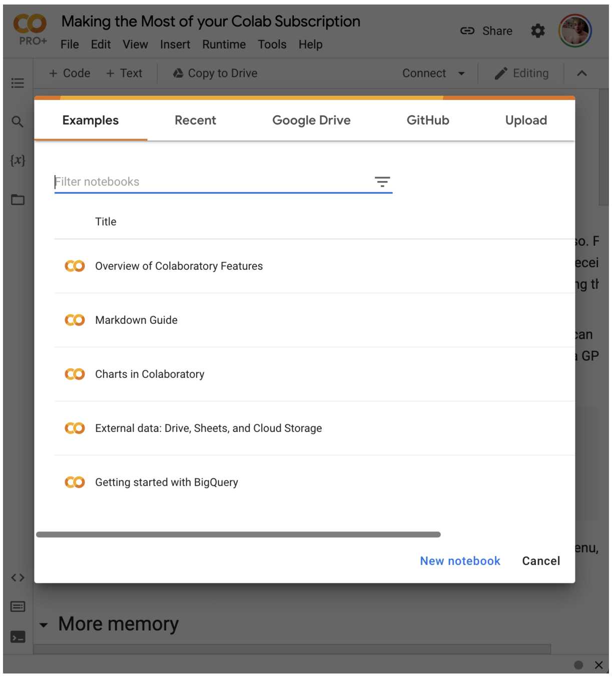



Figure 2: Colab Landing Page

As features extracted were numerous, particularly for 224 x 224 pixel images in datasets that were augmented resulting in dataframes with large number of columns memory crashes. At times the data was over 35k rows and 150k columns. This mostly happened during the experimentation phase of the project and it is not part of the needed steps to complete the study but as RAM was crashing at around 25gb+ a decision was reached to upgrade to Colab Pro+ which unlocked 54gb of RAM as per Figure 3.

GPU was also available and used during the CNN creation. Figure 4 shows the configuration in Google Colab. Figure 5 displays how to confirm that GPU is available and how to select High RAM.

[1]Google Colaboratory: `https://colab.research.google.com/`

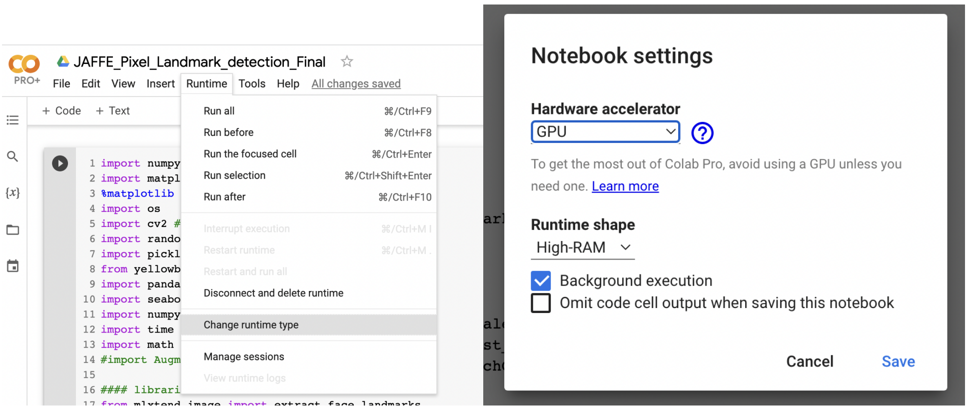
## Google Colab Memory

```python
### extra memory

from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')
```

```
Your runtime has 54.8 gigabytes of available RAM

You are using a high-RAM runtime!
```

Figure 3: Colab Memory

## How to Change Runtime to GPU




(a) Selecting Runtime (b) Selecting GPU

Figure 4: Selecting GPU

## Google Colab GPU

```python
## checking if connected to GPU for faster times

gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Wed Aug 10 22:54:19 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   36C    P0    32W / 250W |   1501MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Figure 5: Colab's GPU details

# 3 Database Sources

Many different datasets were requested to be part of this study. Due to time limitations and resources constraints two datasets were selected to the final report.

1. The posed dataset Japanese Female Facial Expression (JAFFE) created by Lyons et al. consists of 10 Japanese female subjects displaying the 6 basic facial emotions plus neutral emotion. There are 213 images in total in 256 by 256 pixes grayscale. Images are in the Tiff format with no compression. Images were created with excellent lighting and background. This dataset has over 25 years and is utilized in many different papers (Lyons; 2021).

2. The unposed dataset Static Facial Expressions in the Wild (SFEW) (Dhall et al.; 2012) was selected to contrast the JAFFE dateset for having "in-the-wild" type of images. The dataset contains 700 images and 68 subjects. The images were collected from different movies and display a variety of genders, occlusions, different illumination and poses, being referred as close to real life environment by the authors.

# 4 Libraries Used, Image Importing and Train/Test Split

This section looks into libraries used and how to point to the right directory and import the images and extract Features. The code is the same for both datasets the only thing that changes is the directory pointing at either JAFFE or SFEW folders.

## 4.1 Libraries Used

Figure 6 shows all the used libraries to execute the code. Please note that Augmentor library is commented out as it required to be "!pip installed" every time the google colab notebook was switched between GPU and normal RAM. Once the process of Data Augmentation was completed with Augmentor it was commented out to avoid this step.

## 4.2 Loading the data

The first step after loading the libraries is to point at where the folder with the dataset chosen is. Figure 7 displays the code.

## 4.3 Data Exploration and First Look at Landmarks

The following code is a loop that goes into each of the emotion folders, grabs one image, converts from BGR to RGB (more details in the paper), resizes it to 224 x 224 pixels, labels it based on the name of the folder. Extract 68 landmark features from the face and displays both the resulting image created from the points and the superimposed image above the original face. The code then breaks and jumps into the next folder until all named folders have been "visited". Figure 8 shows the code utilized and Figure 9 some of the resulting emotions and faces.

## Imported Libraries

```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import os
import cv2 ##computer vision library
import random
import pickle
from yellowbrick.target import ClassBalance
import pandas as pd
import seaborn as sn
import numpy as np
import time
import math
#import Augmentor

#### libraries for landmark facial detection
from mlxtend.image import extract_face_landmarks

#### From Sklearn
from sklearn import svm
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn import neighbors
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import tree
from sklearn.decomposition import PCA
from sklearn.ensemble import VotingClassifier
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import RandomizedSearchCV


## Tensorflow
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Figure 6: All Imported Libraries

## Path and Emotion Folder Names

```python
PATH = "/content/drive/MyDrive/Masters/jaffedbase manual"
CATEGORIES = 'Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise' ### 0, 1, 2, 3, 4, 5, 6
```

Figure 7: Pointing to the Right Path

## Emotion Labels and Landmarks Code

```python
def show_landmarks(PATH):
  landmarks = []
  for x in CATEGORIES:
      path = os.path.join(PATH, x)
      for img in os.listdir(path):
          img_array = cv2.imread(os.path.join(path, img))
          img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB)
          img_new = cv2.resize(img_array, (224, 224))
          landmarks = extract_face_landmarks(img_new)
          fig = plt.figure(figsize=(15, 5))
          ax = fig.add_subplot(1, 3, 1)
          plt.title(f'Emotion: {x}')
          ax.imshow(img_new)
          ax = fig.add_subplot(1, 3, 2)
          ax.scatter(landmarks[:, 0], -landmarks[:, 1], alpha=0.8)
          ax = fig.add_subplot(1, 3, 3)
          img2 = img_new.copy()

          for p in landmarks:
              img2[p[1]-3:p[1]+3, p[0]-3:p[0]+3, :] = (255, 255, 255)
              # note that the values -3 and +3 will make the landmarks
              # overlayed on the image 6 pixels wide; depending on the
              # resolution of the face image, you may want to change
              # this value

          ax.imshow(img2)
          plt.title(f'Emotion: {x}')
          plt.show()
          break
```

Figure 8: Emotion Labels and Landmarks Code

## Emotion Labels and Landmarks Results

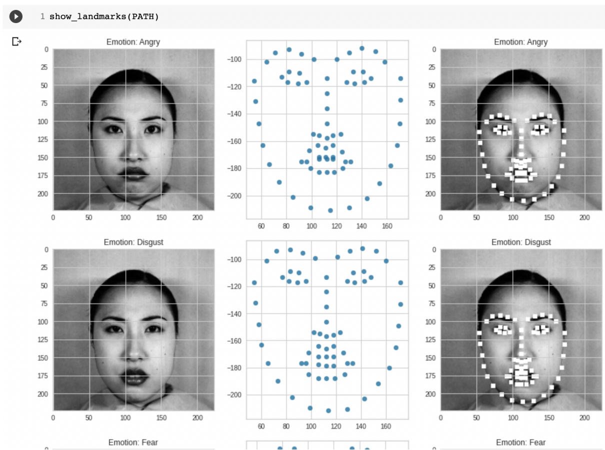



Figure 9: Emotion Labels and Landmarks Results

Variation of the code in figure 10 import the data either as Pixel Values, 68 Landmarks or the calculation of the euclidean distances between landmark points. Figure 10 shows a version of the code and the main difference is explained in the list below.

**read data Function**

```python
#### extracting array inline
IMGSIZE = 224
data = []

def read_data(PATH):
    for category in CATEGORIES:
        path = os.path.join(PATH, category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
             try:
                img_array = cv2.imread(os.path.join(path, img))
                img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB)
                img_new = cv2.resize(img_array, (IMGSIZE, IMGSIZE))
                img_new = np.array(img_new).flatten()
                data.append([img_new, class_num])
             except Exception as e:
               print(f"Image {path}/{img} causes troubles")
               continue
```

```python
read_data(PATH)
```

Figure 10: read data Function

The version of the code with the two lines below after image is resized with img_new = cv2.resize(img_array, (IMGSIZE, IMGSIZE)) collects flattened RGB pixel values.

- img_new = np.array(img_new).flatten() (*flattens the np array image after it was resized*).
- data.append([img_new, class_num]) (*appends the information that was just flattened to the data empty list*).

The following 3 lines instead of the ones above after image resize takes place (img_new = cv2.resize(img_array, (IMGSIZE, IMGSIZE))) collects information on the 68 landamrk points.

- landmarks = extract_face_landmarks(img_new) (*extracts 68 facial landmark points using the Mlxtend library*).
- landmarks = np.array(landmarks).flatten() (*flattens the landmark information*).
- data.append([landmarks, class_num]) (*appends the information that was just flattened to the data empty list*).

The following 3 lines instead of the ones above after image resize takes place (img_new = cv2.resize(img_array, (IMGSIZE, IMGSIZE))) collects information on the 68 landamrk points.

- landmarks = extract_face_landmarks(img_new) (*extracts 68 facial landmark points using the Mlxtend library*).
- distances = compute_landmark_distances(landmarks) (*utilizes the code function to calculate euclidean distance*).
- data.append([distances, class_num]) (*appends the information that was just flattened to the data empty list*).

**Function to calculate euclidean distances**

```python
def compute_landmark_distances(landmarks):
  """
  Computing the distance of landmarks from each other
  :param landmarks: np.array in format [[x, y], [x, y]]
  :return: list of distances
  """
  dists = []
  X = landmarks[:, 0]
  Y = landmarks[:, 1]
  for i in range(len(X)):
    for j in range(len(X)):
      x1 = X[i]
      x2 = X[j]
      y1 = Y[i]
      y2 = Y[j]
  # computing the euclidean distance
      dist = math.sqrt(math.pow((x2 - x1), 2) + math.pow((y2 - y1), 2))
      dists.append(dist / 60)
  return dists
```

Figure 11: Function to calculate euclidean distances

Additional function to calculate euclidean distance between points can be seen in figure 11.

Once data is read, it needs to be shuffled, random shuffle is used, first setting the seed so results can be replicated, second shuffling the data and finally assigning the data to X and y variables and changing from list to np.array. For pixel values, the values can be normalised by dividing X = X/255.0. After everything is ready a pickle file is created so that in future data reading step doesn't have to be repeated. Figure 12 shows it in details.

**reading, shuffling and saving pickle file**

```python
read_data(PATH)
```

```python
random.seed(42)
random.shuffle(data)
```

```python
X = []
y = []


for features, label in data:
    X.append(features)
    y.append(label)


X = np.array(X)
X = X/255.0
y = np.array(y)
```

```python
X.shape
```

```
(1394, 150528)
```

```python
#saving the features to a pickle file
with open('/content/drive/MyDrive/Masters/Sfew/Train/X_pickle_RGB.pickle', 'wb') as f:
    pickle.dump(X, f)

with open('/content/drive/MyDrive/Masters/Sfew/Train/y_pickle_RGB.pickle', 'wb') as f:
    pickle.dump(y, f)
```

Figure 12: reading, shuffling and saving pickle file

## 4.4 Looking for missing values

Transforming the data from np.array to dataframe allows us to look into the data better and see if any values are null or 0.0000 where landmark detection could not read faces. This didn't happen for JAFFE dataset or for extraction of pixel values but happened for SFEW over 26k times with data augmented and normal dataset. Figure 13 shows how the table looks like when there's no missing data. Figure 14 shows the table with missing data.

There's no need to try to trust your vision only as a simple code df = df.loc[ (df.iloc[:, 0:4623] == 0).all(axis=1)] removes the missing values and the table is ready to be saved

### No missing data

```
### Checking for missing values

df = pd.DataFrame(X)
df['Original_label'] = y
```

```
df
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 4615 | 4616 | 4617 | 4618 | 4619 | 4620 | 4621 | 46 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.250555 | 0.501110 | 0.769560 | 1.013794 | 1.229386 | 1.426924 | 1.641476 | 1.794126 | 1.890914 | ... | 0.190029 | 0.235702 | 0.000000 | 0.100000 | 0.183333 | 0.333750 | 0.183333 | 0.1000 |
| **1** | 0.0 | 0.250555 | 0.520950 | 0.756637 | 1.000278 | 1.224972 | 1.461069 | 1.677051 | 1.830604 | 1.952562 | ... | 0.177169 | 0.166667 | 0.066667 | 0.120185 | 0.195078 | 0.366667 | 0.200000 | 0.1000 |
| **2** | 0.0 | 0.250555 | 0.485913 | 0.723610 | 0.954521 | 1.181336 | 1.377296 | 1.557954 | 1.719577 | 1.833106 | ... | 0.166667 | 0.184089 | 0.037268 | 0.105409 | 0.186339 | 0.417000 | 0.200000 | 0.1013 |
| **3** | 0.0 | 0.235702 | 0.454911 | 0.693221 | 0.905692 | 1.122621 | 1.313181 | 1.515292 | 1.666667 | 1.791957 | ... | 0.142400 | 0.216667 | 0.133333 | 0.177169 | 0.263523 | 0.380058 | 0.217307 | 0.1178 |
| **4** | 0.0 | 0.233928 | 0.469338 | 0.709656 | 0.941925 | 1.170233 | 1.389744 | 1.580436 | 1.728840 | 1.807776 | ... | 0.134371 | 0.190029 | 0.184089 | 0.195078 | 0.250000 | 0.368932 | 0.166667 | 0.0833 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **206** | 0.0 | 0.267187 | 0.537484 | 0.791974 | 1.055804 | 1.291532 | 1.538488 | 1.774511 | 1.932255 | 2.044573 | ... | 0.164992 | 0.153659 | 0.116667 | 0.166667 | 0.226691 | 0.372678 | 0.184089 | 0.1000 |
| **207** | 0.0 | 0.300000 | 0.603692 | 0.895979 | 1.184389 | 1.465530 | 1.767374 | 2.005686 | 2.158253 | 2.230159 | ... | 0.169967 | 0.333333 | 0.400347 | 0.416667 | 0.440328 | 0.471699 | 0.216667 | 0.1178 |
| **208** | 0.0 | 0.217307 | 0.438432 | 0.660387 | 0.889444 | 1.085766 | 1.284631 | 1.468654 | 1.608657 | 1.724658 | ... | 0.120185 | 0.174005 | 0.066667 | 0.097183 | 0.179505 | 0.403113 | 0.183333 | 0.0833 |
| **209** | 0.0 | 0.250555 | 0.504425 | 0.759020 | 0.987140 | 1.213352 | 1.424001 | 1.595131 | 1.738454 | 1.848122 | ... | 0.177169 | 0.150923 | 0.016667 | 0.083333 | 0.184089 | 0.334996 | 0.183333 | 0.0833 |
| **210** | 0.0 | 0.266667 | 0.533594 | 0.801561 | 1.058432 | 1.294433 | 1.539120 | 1.770515 | 1.924765 | 2.030462 | ... | 0.188562 | 0.166667 | 0.066667 | 0.097183 | 0.195078 | 0.350000 | 0.183333 | 0.0849 |

211 rows × 4625 columns

Figure 13: No missing data

### Missing data

```
df
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 4615 | 4616 | 4617 | 4618 | 4619 | 4620 | 4621 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **1** | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **2** | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **3** | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **4** | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **1389** | 0.0 | 0.117851 | 0.217307 | 0.320590 | 0.396162 | 0.479873 | 0.573973 | 0.643126 | 0.690008 | 0.719761 | ... | 0.074536 | 0.101379 | 0.016667 | 0.023570 | 0.052705 | 0.089753 | 0.052705 | 0.0 |
| **1390** | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **1391** | 0.0 | 0.084984 | 0.183333 | 0.266667 | 0.351584 | 0.454911 | 0.529675 | 0.579751 | 0.626276 | 0.674949 | ... | 0.060093 | 0.037268 | 0.000000 | 0.037268 | 0.068718 | 0.153659 | 0.068718 | 0.0 |
| **1392** | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **1393** | 0.0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |

1394 rows × 4625 columns

Figure 14: missing data

as X again now without unwanted 0 values. After all is confirmed and pickled file saved it can be loaded using the code displayed in figure 15.

### Reading pickle file

```
#load data as needed if starting a new colab or starting work separately than

with open('/content/drive/MyDrive/Masters/jaffedbase manual/pickle_landmark_dist.pickle', 'rb') as f:
    X = pickle.load(f)


with open('/content/drive/MyDrive/Masters/jaffedbase manual/y_pickle_landmark_dist.pickle', 'rb') as f:
    y = pickle.load(f)
```

Figure 15: reading pickle file

## 4.5 Creating Train and Test Split

The code in Figure 16 splits X and y into train and test data that can be used to train a model.

# 5 Training a ML model

This section follows the code in the colab book named "Machine Learning Techniques". Once data is assigned to the X_train and X_test, y_train and y_test it is ready to be used with the different ML models.

Baseline models are run as per Figure 17.

figure 18 shows an example of the grid search parameters.

Follow the code lines running both baseline models and grid search. Some of the grid search combination took above 11 hours to run.

## Train and Test Split

Creating X_train/y_train/X_Test/y_test

```
# create train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, shuffle=True, random_state=42)

### edited from code https://www.thepythoncode.com/article/building-a-speech-emotion-recognizer-using-sklearn

# details
# n samples in training data
print("[+] Number of training samples:", X_train.shape[0])
# n samples in testing data
print("[+] Number of testing samples:", X_test.shape[0])
```

Figure 16: Creating Train and Test Split

## Baseline model

KNN

```
## model = nbrs.fit(X_train, y_train) ## normalised non-lda
nbrs = KNeighborsClassifier(n_neighbors=5)

y_pred = nbrs.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points : %d"
      % (X_test.shape[0], (y_test != y_pred).sum()))
```

```
Number of mislabeled points out of a total 53 points : 29
```

```
print("Overal Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("\n")

confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'], margins = False)

sn.heatmap(confusion_matrix, annot=True, fmt="d")
plt.show()

print(classification_report(y_test,y_pred))
```

```
Overal Accuracy: 0.4528301886792453
```

Figure 17: Baseline model

## Grid search parameters

```
## code edited from the original - https://medium.com/@erikgreenj/k-neighbors-classifier-with-gridsearchcv-basics-3c445ddeb657

grid_params = {
    'n_neighbors': [3,5,11,15,19,23,25,33],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
```

```
## code edited from the original - https://medium.com/@erikgreenj/k-neighbors-classifier-with-gridsearchcv-basics-3c445ddeb657
gs = GridSearchCV(
    KNeighborsClassifier(),
    grid_params,
    verbose = 1,
    cv = 10,
    n_jobs = -1
)
```

```
gs_results = gs.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 32 candidates, totalling 320 fits
```

```
gs_results.best_estimator_
```

```
KNeighborsClassifier(metric='manhattan', n_neighbors=3, weights='distance')
```

Figure 18: Grid search parameters

### 5.1 Data Transformation

Two types of data transformation were used to reduce dimensionality in the data LDA transformation and PCA. Once new X_train and X_test variables were created using PCA or LDA methods the same models were ran again, with the new variables, so that results could be compared.

# 6 Data Augmentation

Data augmentation was used to increase the number of image examples to train the model. Both a library called Augmentor and a Keras approach were used at different times in the process of the code creation. Augmentor generated images in a folder called output in the different emotion classes. These were then manually moved and reorganised but as Mlxtend library struggled to find faces in the data this step was not used in the main project data. Augmentor steps can be seen in figure 19.

**Augmentor Data Augmentation**

Data Augmentation

```
p = Augmentor.Pipeline("/content/drive/MyDrive/Masters/Sfew/Train/Disgust") #disgust increasing from 66 to 5k examples

Initialised with 66 image(s) found.
Output directory set to /content/drive/MyDrive/Masters/Sfew/Train/Disgust/output.
```

```
p.random_distortion(probability=1, grid_width=2, grid_height=2, magnitude=4)
p.flip_left_right(probability=1)
p.rotate(probability=1, max_left_rotation=4, max_right_rotation=4)
p.crop_random(probability=1, percentage_area=0.95)
```

```
p.sample(5000)

Processing <PIL.Image.Image image mode=RGB size=684x547 at 0x7F8B05451390>: 100%|██████████| 5000/5000 [04:12<00:00, 19.84 Samples/s]
```

```
p = Augmentor.Pipeline("/content/drive/MyDrive/Masters/Sfew/Train/Angry") #Angry increasing from 178 to 5k examples

Initialised with 178 image(s) found.
Output directory set to /content/drive/MyDrive/Masters/Sfew/Train/Angry/output.
```

```
p.random_distortion(probability=1, grid_width=2, grid_height=2, magnitude=4)
p.flip_left_right(probability=1)
p.rotate(probability=1, max_left_rotation=4, max_right_rotation=4)
p.crop_random(probability=1, percentage_area=0.95)
```

```
p.sample(5000)

Processing <PIL.Image.Image image mode=RGB size=684x547 at 0x7F8B8BD59690>: 100%|██████████| 5000/5000 [04:40<00:00, 17.85 Samples/s]
```

Figure 19: Augmentor Data Augmentation

Figure 20 shows the data augmentation that was used in the creation of the CNN. Although results did not increase with the data augmentation so this step is not present in the main accuracy of the CNN model.

**CNN Test Data**

```
### data augmentation


from tensorflow.keras import layers

data_augmentation = keras.Sequential([layers.experimental.preprocessing.RandomFlip("horizontal",
                                                                                    input_shape= (224,224,3)),
                                      #layers.experimental.preprocessing.RandomZoom(0.1),
                                      layers.experimental.preprocessing.RandomRotation(0.1),
])
```

Figure 20: CNN Test Data

# 7 CNN Setup

To import data for the CNN with the right dimension the code is almost the same as the pixel value feature extraction code but without the .flatten() line as displayed by figure 21.

**Importing data for CNN**

```python
PATH = "/content/drive/MyDrive/Masters/jaffedbase manual"
CATEGORIES = 'Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise' ### 0, 1, 2, 3, 4, 5, 6
```

```python
#### extracting array inline
IMGSIZE = 224
data = []

def read_data():
    for x in CATEGORIES:
        path = os.path.join(PATH, x)
        class_num = CATEGORIES.index(x)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, img))
                img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB)
                img_new = cv2.resize(img_array, (IMGSIZE, IMGSIZE))
                data.append([img_new,class_num])
            except Exception as e:
                pass
```

```python
read_data()
```

```python
random.shuffle(data)

X = []
y = []

for features, label in data:
    X.append(features)
    y.append(label)

X = np.array(X)
y = np.array(y)
```

```python
X.shape
```

Figure 21: Importing data for CNN

Run the code to create the function that plots the history of the model training as seen in figure 22. Followed by the code with the CNN layers in figure 23. and to compile the model in figure 24.

**CNN History**

```python
def plot_history(history):
    fig, axs = plt.subplots(2)

    # create accuracy sublpot
    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="val accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy eval")

    # create error sublpot
    axs[1].plot(history.history["loss"], label="train error")
    axs[1].plot(history.history["val_loss"], label="val error")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Error eval")

    plt.show()
```

Figure 22: CNN history

The next lines of code create the prediction function that will use the test data to test the model with data that hasn't been seen by the model yet. It runs the entire test folder through the model and lets you pick a random image to test individually. As seen in figure 25.

# 8 Model Ensemble

A model ensemble made of the best performing models of this study was created in an attempt to increase the maximum achieved accuracy. This ensemble was successful and increased accuracy from 87% to 89%. Image 26 shows the details of the ensemble.

## CNN Layers

```python
model = tf.keras.models.Sequential([ #data_augmentation,
                                    tf.keras.layers.Conv2D(16,(3,3), 1, activation='relu',input_shape = (224,224,3)), ### 1st conv layer
                                    tf.keras.layers.MaxPool2D(),

                                    ## 2nd conv layer
                                    tf.keras.layers.Conv2D(32,(3,3), 1, activation='relu'),
                                    tf.keras.layers.MaxPool2D(),
                                    tf.keras.layers.BatchNormalization(),

                                    ## 5th conv layer
                                    tf.keras.layers.Conv2D(16,(3,3), 1, activation='relu'),
                                    tf.keras.layers.MaxPool2D(),
                                    keras.layers.Dropout(0.2),
                                    tf.keras.layers.BatchNormalization(),

                                    ### flatten output and feed it into dense layer
                                    tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(512, activation='relu'),

                                    ## output layer
                                    tf.keras.layers.Dense(7, activation='softmax')
                                    ])
```

Figure 23: CNN Layers

## CNN Compiler

```python
# compile model
# optimiser = keras.optimizers.Adam(learning_rate=0.001)

# model.compile(optimizer=optimiser,
#               loss='categorical_crossentropy',
#               metrics=['accuracy'])
# print("Creating EarlyStopping Callback ...")
# early_stopping_callback = EarlyStopping(monitor='val_loss',patience=5)

model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001),loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), metrics=['accuracy'])
early_stopping_callback = EarlyStopping(monitor='val_loss',patience=5)

model.summary()
```

```
Model: "sequential_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_45 (Conv2D)          (None, 222, 222, 16)      448

 max_pooling2d_45 (MaxPoolin  (None, 111, 111, 16)     0
 g2D)

 conv2d_46 (Conv2D)          (None, 109, 109, 32)      4640

 max_pooling2d_46 (MaxPoolin  (None, 54, 54, 32)       0
 g2D)

 batch_normalization_22 (Bat  (None, 54, 54, 32)       128
 chNormalization)

 conv2d_47 (Conv2D)          (None, 52, 52, 16)        4624

 max_pooling2d_47 (MaxPoolin  (None, 26, 26, 16)       0
 g2D)

 dropout_18 (Dropout)        (None, 26, 26, 16)        0

 batch_normalization_23 (Bat  (None, 26, 26, 16)       64
 chNormalization)

 flatten_15 (Flatten)        (None, 10816)             0

 dense_34 (Dense)            (None, 512)               5538304

 dense_35 (Dense)            (None, 7)                 3591

=================================================================
Total params: 5,551,799
Trainable params: 5,551,703
Non-trainable params: 96
_________________________________________________________________
```

Figure 24: CNN Compiler

## CNN Test Data

```python
[ ]
def predict(model, X, y):
    # add a dimension to input data for sample - model.predict() expects a 4d array in this case
    X = X[np.newaxis, ...] # array shape (new dimention, 130, 13, 1)

    # perform prediction
    prediction = model.predict(X)

    # get index with max value
    predicted_index = np.argmax(prediction, axis=1)

    print("Target: {}, Predicted label: {}".format(y, predicted_index))
```

```python
### here we evaluate the model using the test dataset,
###this files have not been seen by the DL model

test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)

# pick a sample to predict from the test set (30 chosen at random)
X_to_predict = X_test[5]
y_to_predict = y_test[5]

# predict sample
predict(model, X_to_predict, y_to_predict)
```

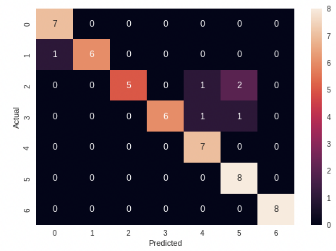
```
2/2 - 0s - loss: 0.6847 - accuracy: 0.8140 - 26ms/epoch - 13ms/step

Test accuracy: 0.8139534592628479
Target: 2, Predicted label: [2]
```

```
[ ]
```

Figure 25: CNN Test Data

**Model Ensemble**

```python
# import voting classifier
from sklearn.ensemble import VotingClassifier


# create a voting classifier with hard voting
voting_classifier_hard = VotingClassifier(
    estimators = [('rf',  RandomForestClassifier(random_state=42)),
                  ('rf2',  RandomForestClassifier(random_state=42)),
                  ('etc', ExtraTreesClassifier()),
                  ('etc2', ExtraTreesClassifier()),
                  ('svc', SVC(C=0.01, gamma=1, kernel='linear', probability=True)),
                  ('svc2', SVC(C=0.01, gamma=1, kernel='linear', probability=True))],
    voting='hard')


# make predictions with the hard voting model
voting_classifier_hard.fit(X_train, y_train)
y_pred_vch = voting_classifier_hard.predict(X_test)

confusion_matrix = pd.crosstab(y_test, y_pred_vch, rownames=['Actual'], colnames=['Predicted'], margins = False)
sn.heatmap(confusion_matrix, annot=True, fmt="d")
plt.show()
print(classification_report(y_test, y_pred_vch))
```




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.88 | 1.00 | 0.93 | 7 |
| 1 | 1.00 | 0.86 | 0.92 | 7 |
| 2 | 1.00 | 0.62 | 0.77 | 8 |
| 3 | 1.00 | 0.75 | 0.86 | 8 |
| 4 | 0.78 | 1.00 | 0.88 | 7 |
| 5 | 0.73 | 1.00 | 0.84 | 8 |
| 6 | 1.00 | 1.00 | 1.00 | 8 |
| accuracy | | | 0.89 | 53 |
| macro avg | 0.91 | 0.89 | 0.89 | 53 |
| weighted avg | 0.91 | 0.89 | 0.88 | 53 |

Figure 26: Model Ensemble

The ensemble used a repetition of the same model 3 times which increased the accuracy rather than using the algorithm just once. each repeated model builds up on each other and using hard voting helps increase accuracy.

# 9 Confusion Matrix

Confusion matrix is used to display model accuracy and F1-score information. This is shown in detail per emotional class, support and macro weighted average.

# References

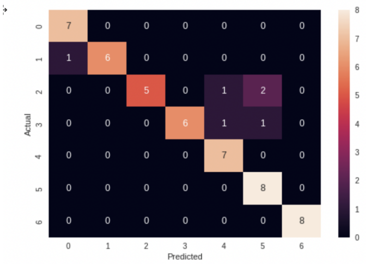

Dhall, A., Goecke, R., Lucey, S. and Gedeon, T. (2012). Collecting large, richly annotated facial-expression databases from movies, *IEEE multimedia* **19**(03): 34–41.

Lyons, M. J. (2021). ” excavating ai” re-excavated: Debunking a fallacious account of the jaffe dataset, *arXiv preprint arXiv:2107.13998* .

Lyons, M. J., Kamachi, M. and Gyoba, J. (2020). Coding facial expressions with gabor wavelets (ivc special issue), *arXiv preprint arXiv:2009.05938* .

# Confusion Matrix

```python
confusion_matrix = pd.crosstab(y_test, y_pred_vch, rownames=['Actual'], colnames=['Predicted'], margins = False)
sn.heatmap(confusion_matrix, annot=True, fmt="d")
plt.show()
print(classification_report(y_test, y_pred_vch))
```




```
              precision    recall  f1-score   support

           0       0.88      1.00      0.93         7
           1       1.00      0.86      0.92         7
           2       1.00      0.62      0.77         8
           3       1.00      0.75      0.86         8
           4       0.78      1.00      0.88         7
           5       0.73      1.00      0.84         8
           6       1.00      1.00      1.00         8

    accuracy                           0.89        53
   macro avg       0.91      0.89      0.89        53
weighted avg       0.91      0.89      0.88        53
```

Figure 27: Confusion Matrix