Configuration Manual

MSc Research Project
Artificial Intelligence

Nouman Ali
Student ID: x23239221

School of Computing
National College of Ireland

Supervisor: Anh Duong Trinh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Nouman Ali……. ………………………………………………………………………………………………………
Student ID:	X23239221…………………………………………………………………
Programme:	MSC Artificial intelligence………….. **Year:** 2024………………..
Module:	Master of science in artificial intelligence ………………………………………
Lecturer:	Anh Duong Trinh ……
Submission Due Date:	12/12/2024 ……
Project Title:	Sentiment Analysis Using Text and Facial Emotions ……
Word Count:	510 ……………………………………… **Page Count:** ………14………………………………

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: Nouman ali ………

Date: 12/12/24 ………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	Y
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	**Y**
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.	**y**

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

Nouman Ali
Student ID: x23239221

1 Introduction

This configuration manual provides step-by-step instructions to set up, configure, and execute the Text-Based Sentiment Analysis and Face Image-Based Sentiment Analysis projects. These projects utilise advanced natural language processing (NLP) techniques and computer vision models for sentiment analysis.

2 System Requirements

- **Hardware Requirements**
- **Processor:** Intel Core i5 or equivalent
- **RAM:** 8 GB minimum (16 GB recommended)
- **Storage:** 256 GB SSD
- **GPU:** NVIDIA GTX 1650 or higher for training deep learning models
- **Software Requirements**
- **Operating System:** Windows 10/11 or Ubuntu 18.04 and above
- **Python Version:** 3.9
- **Additional Tools:** Anaconda (for environment management)

3 Text -Based Sentiment Analysis

3.1 Required Libraries and Dependencies

Install the following Python libraries:

pip install tensorflow keras transformers wordcloud imbalanced-learn

Import the necessary libraries

```python
# importing the necessary packages
import pandas as pd
import numpy as np
import nltk
nltk.download('vader_lexicon')
nltk.download('punkt')
nltk.download('stopwords')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import string
import re
from wordcloud import WordCloud
from sklearn.preprocessing import LabelEncoder
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from imblearn.under_sampling import NearMiss
import tensorflow as tf
import  transformers
from transformers import BertTokenizer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import keras
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential, Model
from keras.utils import to_categorical
from tensorflow.keras.layers import Embedding, Dense, Flatten, Layer, Input, Dropout, BatchNormalization
from tensorflow.keras.layers import Attention, Bidirectional, LSTM, LayerNormalization, Add, Multiply, MaxPooling2D, InputLayer
```

Figure 1: List of Necessary Libraries

4 Data Collection and Exploration

Reading the dataset files using Pandas Library.

```python
train_data = pd.read_csv('textData/train.csv',encoding='latin1');
test_data = pd.read_csv('textData/test.csv',encoding='latin1');
```

```python
train_data.shape, test_data.shape
```

```
27481, 10), (4815, 9))
```

```python
# data import
data = pd.concat([train_data,test_data])
```

Figure 2: Reading the Text Data Files

Information about the column metadata.

```python
data.info()  #checking data information
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 32296 entries, 0 to 4814
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   textID             31015 non-null  object 
 1   text               31014 non-null  object 
 2   selected_text      27480 non-null  object 
 3   sentiment          31015 non-null  object 
 4   Time of Tweet      31015 non-null  object 
 5   Age of User        31015 non-null  object 
 6   Country            31015 non-null  object 
 7   Population -2020   31015 non-null  float64
 8   Land Area (Km²)    31015 non-null  float64
 9   Density (P/Km²)    31015 non-null  float64
dtypes: float64(3), object(7)
memory usage: 2.7+ MB
```

Figure 3: Dataset information

```
data.isnull().sum()   # checking for missing data sum count for each column
```

```
textID              1281
text                1282
selected_text       4816
sentiment           1281
Time of Tweet       1281
Age of User         1281
Country             1281
Population -2020    1281
Land Area (Km²)     1281
Density (P/Km²)     1281
dtype: int64
```

Figure 4: Check for Nulls

```
data['sentiment'].unique() , len(data['sentiment'].unique() )  #checking values for column
```

```
(array(['neutral', 'negative', 'positive', nan], dtype=object), 4)
```

```
data['sentiment'].value_counts()   #checking values for column
```

```
sentiment
neutral     12548
positive     9685
negative     8782
Name: count, dtype: int64
```

```
data['Time of Tweet'].value_counts() #checking values for column
```

```
Time of Tweet
morning    10339
noon       10338
night      10338
Name: count, dtype: int64
```

```
data['Age of User'].value_counts() #checking values for column
```

```
Age of User
0-20      5171
21-30     5170
31-45     5170
46-60     5168
60-70     5168
70-100    5168
Name: count, dtype: int64
```

Figure 5: Getting Value Counts

```
data['Country'].value_counts() #checking values for column
```

```
Country
Afghanistan       169
Ecuador           169
Chile             169
China             169
Colombia          169
                 ...
Singapore         144
Slovakia          144
Slovenia          144
Solomon Islands   144
Zimbabwe          144
Name: count, Length: 195, dtype: int64
```

```
data['Population -2020'].value_counts() #checking values for column
```

```
Population -2020
3.892835e+07    169
1.764305e+07    169
1.911620e+07    169
1.439324e+09    169
5.088289e+07    169
               ...
5.850342e+06    144
5.459642e+06    144
2.078938e+06    144
6.868840e+05    144
1.486292e+07    144
Name: count, Length: 195, dtype: int64
```

Figure 6: Value Counts for Country and Population

```
data['Land Area (Km²)'].value_counts() #checking values for column
```

```
Land Area (Km²)
700.0       310
460.0       288
652860.0    169
9240.0      169
48300.0     169
           ...
48088.0     144
20140.0     144
28000.0     144
627340.0    144
386850.0    144
Name: count, Length: 193, dtype: int64
```

```
data.isnull().sum()
```

```
textID              1281
text                1282
selected_text       4816
sentiment           1281
Time of Tweet       1281
Age of User         1281
Country             1281
Population -2020    1281
Land Area (Km²)     1281
Density (P/Km²)     1281
dtype: int64
```

Figure 7: Identifying Nulls

```
data.dropna(inplace=True)     #dropping null values from the data
```

```
#checking the count for each word, uppercase characters and special characters
data['WordCount'] = [len(title.split()) for title in data['text']]
data['UppercaseCount'] = [sum(char.isupper() for char in title) for title in data['text']]
data['SpecialCount'] = [sum(char in string.punctuation for char in title) for title in data['text']]
```

Figure 8: Dropping Nulls

Generating sentiments for the text present in the dataset using Vader Sentiment Analyzer.

VADER Sentiment Analyzer

Sentiment analysis is used to find out the polarity of the text, which is positive, negative, or neutral. VADER has the advan

```
sa = SentimentIntensityAnalyzer()   # initializing sentiment intenstity analyser
```

```
score = lambda title: sa.polarity_scores(title)['compound']     # checking the compound score for the sentiment
data['scores'] = data['text'].apply(score)
# Adding user review and restuartenat star to get final sentiment
data['scores']
```

```
0         0.0000
1        -0.7437
2        -0.5994
3        -0.3595
4         0.0000
          ...
27476    0.1027
27477    0.3818
27478    0.9136
27479    0.3291
27480    0.8074
Name: scores, Length: 27480, dtype: float64
```

```
np.min(data['scores']), np.max(data['scores'])
```

```
(-0.9726, 0.9819)
```

Figure 9: Getting sentiments for the text data

```
s=[]
for compound in data['scores']:
        if compound > 0.5:
            s.append('positive')    # putting 2 if compound score is greater than 0 for positive sentiment
        elif compound > -0.2:
            s.append('neutral')   # putting 1 if compound score is less than 0 for neutarl sentiment
        else:
            s.append('negative')    # putting 0 if compound score is less than 0 for negative sentiment
```

```
data['scores'] = s
```

```
data['scores'].value_counts()
```

```
scores
neutral     13975
positive     7847
negative     5658
Name: count, dtype: int64
```

```
data['sentiment'].value_counts()
```

```
sentiment
neutral     11117
positive     8582
negative     7781
Name: count, dtype: int64
```

Figure 10: Getting sentiments based on score

4.1 Text Preprocessing

```
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

```
def decontracted(phrase):
    # This function decontract words like it's to it is.

    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    phrase = re.sub(r"\n", " ", phrase)
    return phrase
```

Figure 11: Text cleaning

```
stop = stopwords.words('english') #set of stopwords
print(stop)
```

Figure 12: Stopword Removal using NLTK

```
data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
data['text']
```

Figure 13: Tokenization using Lambda Function

```
#Basic variables
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_neutral_words=[] # store words from neutral reviews here.
all_negative_words=[] # store words from -ve reviews here.

s=''
```

Figure 14: Initializing the lists to store results

```
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer\
sno
```

```
<nltk.stem.snowball.SnowballStemmer at 0x17c1a3d70>
```

```
for sent in tqdm(data['text']):
    filtered_sentence=[]
    sent = decontracted(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (data['sentiment'].values)[i] == 'positive':
                            all_positive_words.append(s) #list of all words used to describe positive reviews
                    if (data['sentiment'].values)[i] == 'neutral':
                            all_neutral_words.append(s) #list of all words used to describe negative reviews reviews
                    if (data['sentiment'].values)[i] == 'negative':
                            all_negative_words.append(s) #list of all words used to describe negative reviews reviews
                else:
                    continue
            else:
                continue

    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    final_string.append(str1)
    i+=1

data["clean_text"] = final_string
data['clean_text']=data['clean_text'].str.decode("utf-8")
```

```
100%|████████████████████████████████| 27480/27480 [00:02<00:00, 12957.31it/s]
```

Figure 15: Getting the tokenized data

4.2 Label Encoding the Text Labels

```
le = LabelEncoder()
```

```
data['sentiment'] = le.fit_transform(data['sentiment'])
data['sentiment']
```

Figure 16: Label Encoding the Sentiment

```
# Spliting data in test and train data set
x= data.drop(['sentiment'], axis=1)
y = data["sentiment"].values
```

Figure 17: Separating the dependent and independent variables

4.3 Splitting the dataset

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.15)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((23358, 14), (4122, 14), (23358,), (4122,))
```

Figure 18: Splitting the dataset using the sklearn's model selection module

4.4 Extracting the features

```
vec=TfidfVectorizer(max_features=10000)
X_train=vec.fit_transform(X_train['clean_text']).toarray()
X_test=vec.transform(X_test['clean_text']).toarray()
```

Figure 19: Extracting TFIDF Features

4.5 Modelling

```
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
pred_dt = dt.predict(X_test)
```

```
score_dt = dt.score(X_test, y_test)
score_dt
```

```
0.6540514313440078
```

```
dt = DecisionTreeClassifier(max_depth = 100, random_state=123)
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=100, random_state=123)
```

```
pred_dt = dt.predict(X_test)
```

```
score_dt = dt.score(X_test, y_test)
score_dt
```

```
0.6809801067442989
```

Figure 20: Decision Tree Implementations

```
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
pred_rfc = rfc.predict(X_test)
```

```
score_rfc = rfc.score(X_test, y_test)
score_rfc
```

```
0.7113051916545367
```

```
rfc = RandomForestClassifier(max_depth = 100, criterion='entropy', random_state=123)
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=100, random_state=123)
```

```
pred_rfc = rfc.predict(X_test)
```

```
score_rfc = rfc.score(X_test, y_test)
score_rfc
```

```
0.6962639495390587
```

Figure 21: Random Forest Implementations

```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train, y_test
```

Figure 22: Conversion of the Labels to Categorical for Neural Network Models

```
model = Sequential()
model.add(Dense(32, activation='tanh', input_shape=(X_train.shape[1],)))
model.add(Dense(3, activation='softmax'))
model.compile(loss="categorical_crossentropy", metrics=["accuracy"], optimizer='adam')
model.summary()
```

```
history = model.fit(X_train, y_train, validation_data= (X_test, y_test), batch_size=32, epochs=5)
    loss, accuracy = model.evaluate(X_test, y_test)
    print("Loss = ", loss)
    accRNNBert= accuracy*100
    print("Accuracy = ", accRNNBert)
```

```
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 521us/step - accuracy: 0.6657 - loss: 0.9044
Loss =  0.9054167866706848
Accuracy =  65.2838408946991
```

```
model = Sequential()
model.add(Dense(256, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(320, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(loss="categorical_crossentropy", metrics=["accuracy"], optimizer='adam')
model.summary()
    loss, accuracy = model.evaluate(X_test, y_test)
    print("Loss = ", loss)
    accRNNBert= accuracy*100
    print("Accuracy = ", accRNNBert)
```

```
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6776 - loss: 1.6838
Loss =  1.726218581199646
Accuracy =  66.08442664146423
```

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(448, activation='relu'))
model.add(Dense(320, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(loss="categorical_crossentropy", metrics=["accuracy"], optimizer='adam')
model.summary()
```

Figure 23: RNN Implementations

```
model = Sequential()
model.add(Input(shape=(X_train.shape[1],1)))
model.add(LSTM(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()
model = Sequential()
model.add(Input(shape=(X_train.shape[1],1)))
model.add(LSTM(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

Figure 24: BiLSTM Implementations

5 Emotion Based Sentiment Analysis

Installing opencv library using Python pip.

```
!pip install opencv-python
```

```
uirement already satisfied: opencv-python in /opt/anaconda3/lib/python3.12/site-packages (4.10.0.84)
uirement already satisfied: numpy>=1.21.2 in /opt/anaconda3/lib/python3.12/site-packages (from opencv-python) (1.26.4)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')

import os
import tensorflow as tf
import keras
import cv2

from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.utils import plot_model
from tensorflow.keras import layers , models, optimizers

from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import *
from tensorflow.keras.applications import ResNet50V2
```

Figure 25: Installing and Importing Necessary Library

5.1 Data Collection and Exploration

```
train_dir = 'images/train/'
test_dir = 'images/validation/'

def Classes_Count( path, name):
    Classes_Dict = {}

    for Class in os.listdir(path):

        Full_Path = path + Class
        Classes_Dict[Class] = len(os.listdir(Full_Path))

    df = pd.DataFrame(Classes_Dict, index=[name])

    return df

Train_Count = Classes_Count(train_dir, 'Train').transpose().sort_values(by="Train", ascending=False)
Test_Count = Classes_Count(test_dir, 'Test').transpose().sort_values(by="Test", ascending=False)
```

Figure 26: Reading the image data files

```
Train_Count.plot(kind='barh')
```

```
<Axes: >
```

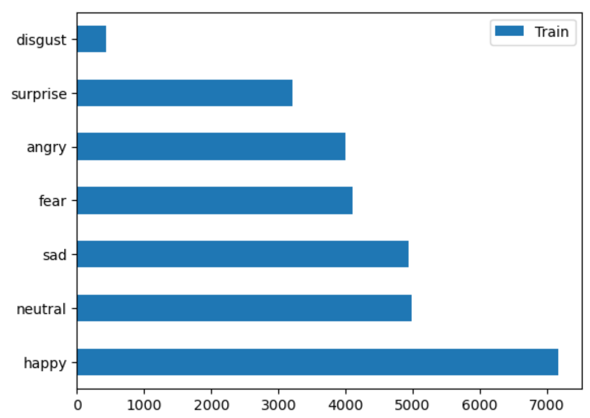

Figure 27: Getting the emotion sample count

```python
plt.style.use('default')
plt.figure(figsize = (25, 8))
image_count = 1
BASE_URL = 'images/train/'

for directory in os.listdir(BASE_URL):
    if directory[0] != '.':
        for i, file in enumerate(os.listdir(BASE_URL + directory)):
            if i == 1:
                break
            else:
                fig = plt.subplot(1, 7, image_count)
                image_count += 1
                image = cv2.imread(BASE_URL + directory + '/' + file)
                plt.imshow(image)
                plt.title(directory, fontsize = 20)
```

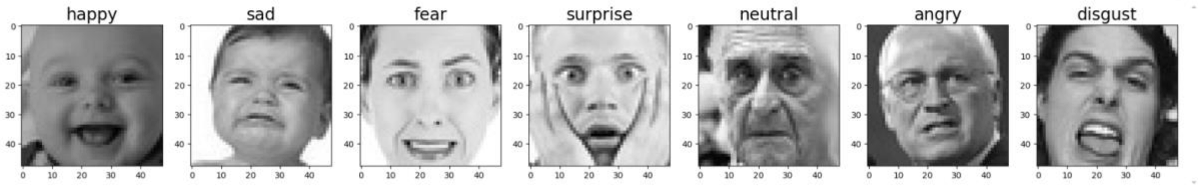

Figure 28: Exploration of the dataset

```python
img_shape = 48
batch_size = 64
train_data_path = 'images/train/'
test_data_path = 'images/validation/'
```

Figure 29: Setting image and training parameters

5.2 Image Preprocessing

The images in the dataset are augmentated using the ImageDataGenerator object of the Tensorflow library.

```python
train_preprocessor = ImageDataGenerator(
        rescale = 1 / 255.,
        # Data Augmentation
        rotation_range=10,
        zoom_range=0.2,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest',
    )


test_preprocessor = ImageDataGenerator(
    rescale = 1 / 255.,
)

train_data = train_preprocessor.flow_from_directory(
    train_data_path,
    class_mode="categorical",
    target_size=(img_shape,img_shape),
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    subset='training',
)


test_data = test_preprocessor.flow_from_directory(
    test_data_path,
    class_mode="categorical",
    target_size=(img_shape,img_shape),
    color_mode="rgb",
    shuffle=False,
    batch_size=batch_size,
)
```

Figure 30: Data Augmentation using ImageDataGenerator

6 Modelling

```python
def Create_CNN_Model():

    model = Sequential()
    model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Flatten())
    model.add(Dense(7,activation='softmax'))
    return model
```

```python
CNN_Model = Create_CNN_Model()

CNN_Model.summary()

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

Figure 31: Implementation of the Baseline CNN Model

```python
def plot_curves(history):

    loss = history.history["loss"]
    val_loss = history.history["val_loss"]

    accuracy = history.history["accuracy"]
    val_accuracy = history.history["val_accuracy"]

    epochs = range(len(history.history["loss"]))

    plt.figure(figsize=(15,5))

    #plot loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs, loss, label = "training_loss")
    plt.plot(epochs, val_loss, label = "val_loss")
    plt.title("Loss")
    plt.xlabel("epochs")
    plt.legend()

    #plot accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, accuracy, label = "training_accuracy")
    plt.plot(epochs, val_accuracy, label = "val_accuracy")
    plt.title("Accuracy")
    plt.xlabel("epochs")
    plt.legend()

  #plt.tight_layout()
```

Figure 32: Defining a function to plot the model history

```python
CNN_history = CNN_Model.fit( train_data , validation_data= test_data , epochs=50, batch_size= batch_size)
```

Figure 33: Training the CNN model

```python
CNN_Score = CNN_Model.evaluate(test_data)

print("Test Loss: {:.5f}".format(CNN_Score[0]))
print("Test Accuracy: {:.2f}%".format(CNN_Score[1] * 100))
```

```
111/111 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.4139 - loss: 1.4832
Test Loss: 1.46712
Test Accuracy: 45.02%
```

Figure 34: Evaluation the model

```python
def Create_CNN_Model():
    model = Sequential()
    model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))
    model.add(Dense(7,activation='softmax'))
    return model
```

```python
CNN_Model = Create_CNN_Model()

CNN_Model.summary()

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

```python
def Create_CNN_Model():

    model = Sequential()

    model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(64, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

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

```python
CNN_Model = Create_CNN_Model()

CNN_Model.summary()

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

Figure 35: Implementation of Deeper CNN models

```python
# ResNet50
ResNet50V2 = tf.keras.applications.ResNet50V2(input_shape=(48, 48, 3),
                                               include_top= False,
                                               weights='imagenet'
                                               )

ResNet50V2.summary()
```

Figure 36: Downloading the ResNet50V2 model

```python
def Create_ResNet50V2_Model():

    model = Sequential([
                    ResNet50V2,
                    Flatten(),
                    Dense(7,activation='softmax')
                   ])
    return model
```

```python
ResNet50V2_Model = Create_ResNet50V2_Model()

ResNet50V2_Model.summary()

ResNet50V2_Model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 37: Adding output layers for the study

```python
def Create_ResNet50V2_Model():

    model = Sequential([
                    ResNet50V2,
                    Flatten(),
                    Dense(64, activation='relu'),
                    Dropout(.5),
                    Dense(7,activation='softmax')
                   ])
    return model
```

```python
ResNet50V2_Model = Create_ResNet50V2_Model()

ResNet50V2_Model.summary()

ResNet50V2_Model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
def Create_ResNet50V2_Model():

    model = Sequential([
                    ResNet50V2,
                    Dropout(.25),
                    BatchNormalization(),
                    Flatten(),
                    Dense(64, activation='relu'),
                    BatchNormalization(),
                    Dropout(.05),
                    Dense(7,activation='softmax')
                   ])
    return model
```

```python
ResNet50V2_Model = Create_ResNet50V2_Model()

ResNet50V2_Model.summary()

ResNet50V2_Model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 38: Adding more layers for a deeper network

7 References

- **OpenCV**: OpenCV, 2024. *OpenCV Documentation*. [online] Available at: https://docs.opencv.org/ [Accessed 11 December 2024].

- **scikit-learn**: scikit-learn, 2024. *scikit-learn: Machine Learning in Python*. [online] Available at: https://scikit-learn.org/stable/ [Accessed 11 December 2024].

- **TensorFlow**: TensorFlow, 2024. *TensorFlow Documentation*. [online] Available at: https://www.tensorflow.org/docs [Accessed 11 December 2024].