National College of Ireland

Configuration Manual

MSc Research Project
MSc. Data Analytics

Harshpreet Singh
Student ID: 18197396

School of Computing
National College of Ireland

Supervisor: Rashmi Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Harshpreet Singh……

Student ID: 18197396……

Programme: MSc. Data Analytics…… **Year:** 2020……

Module: MSc. Research Project……

Lecturer: Rashmi Gupta……

Submission Due Date: 17th August 2020……

Project Title: Deepfake Detection Configuration Manual……

Word Count: 437…… **Page Count:** 9……

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: 28th August 2020……

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

Harshpreet Singh
Student ID: 18197396

1 Pre-Requisites

Your first section. Change the header and label to something appropriate.

2 Access Colab

Mounting Google drive with Colab

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Result:

```
Enter your authorization code:
··········
Mounted at /content/gdrive
```

3 Download Dataset

```
# Mounting google drive to colab notebook to the path where
kaggle.json is present in the Google Drive
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/gdrive/My Drive/Colab
Notebooks/Thesis/input/Dataset"

# Changing the working directory to where dataset need to be
downloaded
%cd /content/gdrive/My Drive/Colab Notebooks/Thesis/input/Dataset

!kaggle competitions download -c deepfake-detection-challenge

!kaggle competitions download -c deepfake-detection-challenge
Unzip the dataset and remove the zip file
```

4 Import Libraries

```
import os
import glob
import json
import torch
import torch.nn as nn
import torch.optim as optim
import cv2
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
%matplotlib inline

import dlib
import re
import tensorflow as tf
import seaborn as sn
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
img_to_array, load_img
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from tensorflow.keras.models import load_model
from pylab import *
from PIL import Image, ImageChops, ImageEnhance
from tqdm.notebook import tqdm
from facenet_pytorch import MTCNN, InceptionResnetV1, extract_face
from sklearn.model_selection import train_test_split

from tensorflow.keras.applications import InceptionResNetV2
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import InputLayer
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras import optimizers
from     tensorflow.keras.callbacks     import     ReduceLROnPlateau,
EarlyStopping
```

5 Define Functions

Function to count train and test dataset features

```
def plot_count(feature, title, df, size=1):
    '''
    Plot count of classes / feature
    param: feature - the feature to analyze
    param: title - title to add to the graph
    param: df - dataframe from which we plot feature's classes
distribution
    param: size - default 1.
    '''
    f, ax = plt.subplots(1,1, figsize=(4*size,4))
    total = float(len(df))
    g = sns.countplot(df[feature], order =
df[feature].value_counts().index[:20], palette='Set3')
    g.set_title("Number and percentage of {}".format(title))
    if(size > 2):
        plt.xticks(rotation=90, size=8)
    for p in ax.patches:
        height = p.get_height()
        ax.text(p.get_x()+p.get_width()/2.,
                height + 3,
                '{:1.2f}%'.format(100*height/total),
                ha="center")
    plt.show()
```

Function to display frames from the video

```
      def display_image_from_video(video_path):
    '''
    input: video_path - path for video
    process:
    1. perform a video capture from the video
    2. read the image
    3. display the image
    '''
    capture_image = cv.VideoCapture(video_path)
    ret, frame = capture_image.read()
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111)
    frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    ax.imshow(frame)
```

Function to extract and compare different images from videos

```
      def display_image_from_video_list(video_path_list,
video_folder=TRAIN_SAMPLE_FOLDER):
    '''
    input: video_path_list - path for video
    process:
    0. for each video in the video path list
        1. perform a video capture from the video
        2. read the image
        3. display the image
    '''
```

```
    plt.figure()
    fig, ax = plt.subplots(2,3,figsize=(16,8))
    # we only show images extracted from the first 6 videos
    for i, video_file in enumerate(video_path_list[0:6]):
        video_path = os.path.join(DATA_FOLDER,
video_folder,video_file)
        capture_image = cv.VideoCapture(video_path)
        ret, frame = capture_image.read()
        frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        ax[i//3, i%3].imshow(frame)
        ax[i//3, i%3].set_title(f"Video: {video_file}")
        ax[i//3, i%3].axis('on')
```

Function to detect face features from the image

```
      def detect_objects(image, scale_factor, min_neighbors,
min_size):
    '''
    Objects detection function
    Identify frontal face, eyes, smile and profile face and display
the detected objects over the image
    param: image - the image extracted from the video
    param: scale_factor - scale factor parameter for `detect`
function of ObjectDetector object
    param: min_neighbors - min neighbors parameter for `detect`
function of ObjectDetector object
    param: min_size - minimum size parameter for f`detect` function
of ObjectDetector object
    '''

    image_gray=cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    eyes=ed.detect(image_gray,
                   scale_factor=scale_factor,
                   min_neighbors=min_neighbors,
                   min_size=(int(min_size[0]/2), int(min_size[1]/2)))

    for x, y, w, h in eyes:
        #detected eyes shown in color image
        cv.circle(image,(int(x+w/2),int(y+h/2)),(int((w + h)/4)),(0,
0,255),3)

    profiles=pd.detect(image_gray,
                   scale_factor=scale_factor,
                   min_neighbors=min_neighbors,
                   min_size=min_size)

    for x, y, w, h in profiles:
        #detected profiles shown in color image
        cv.rectangle(image,(x,y),(x+w, y+h),(255, 0,0),3)

    faces=fd.detect(image_gray,
                   scale_factor=scale_factor,
                   min_neighbors=min_neighbors,
                   min_size=min_size)

    for x, y, w, h in faces:
        #detected faces shown in color image
        cv.rectangle(image,(x,y),(x+w, y+h),(0, 255,0),3)
```

```
    # image
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    ax.imshow(image)
```

Function play videos during initial exploration of dataset

```
def play_video(video_file, subset=TRAIN_SAMPLE_FOLDER):
    '''
    Display video
    param: video_file - the name of the video file to display
    param: subset - the folder where the video file is located (can
be TRAIN_SAMPLE_FOLDER or TEST_Folder)
    '''
    video_url = open(os.path.join(DATA_FOLDER,
subset,video_file),'rb').read()
    data_url = "data:video/mp4;base64," +
b64encode(video_url).decode()
    return HTML("""<video width=500 controls><source src="%s"
type="video/mp4"></video>""" % data_url)
```

Function to put box on the face found in the image

```
      def extract_image_objects(video_file,
video_set_folder=TRAIN_SAMPLE_FOLDER):
    '''
    Extract one image from the video and then perform
face/eyes/smile/profile detection on the image
    param: video_file - the video from which to extract the image
from which we extract the face
    '''
    video_path = os.path.join(DATA_FOLDER,
video_set_folder,video_file)
    capture_image = cv.VideoCapture(video_path)
    ret, frame = capture_image.read()
    frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    detect_objects(image=frame,
            scale_factor=1.3,
            min_neighbors=5,
            min_size=(50, 50))
```

6 Implementation

To generate a folder with captured frames of videos

```
train_frame_folder = 'input/Dataset/deepfake-detection-
dataset/train_sample_videos/'
with open(os.path.join(train_frame_folder, 'metadata.json'), 'r') as
file:
    data = json.load(file)
list_of_train_data = [f for f in os.listdir(train_frame_folder) if
f.endswith('.mp4')]
detector = dlib.get_frontal_face_detector()
for vid in tqdm(list_of_train_data):
    count = 0
    cap = cv2.VideoCapture(os.path.join(train_frame_folder, vid))
    frameRate = cap.get(5)
    while cap.isOpened():
        frameId = cap.get(1)
        ret, frame = cap.read()
        if ret != True:
            break
        if frameId % ((int(frameRate)+1)*1) == 0:
            face_rects, scores, idx = detector.run(frame, 0)
            for i, d in enumerate(face_rects):
                x1 = d.left()
                y1 = d.top()
                x2 = d.right()
                y2 = d.bottom()
                crop_img = frame[y1:y2, x1:x2]
                if data[vid]['label'] == 'REAL':

cv2.imwrite('dataset/real/'+vid.split('.')[0]+'_'+str(count)+'.png',
cv2.resize(crop_img, (128, 128)))
                elif data[vid]['label'] == 'FAKE':
cv2.imwrite('dataset/fake/'+vid.split('.')[0]+'_'+str(count)+'.png',
cv2.resize(crop_img, (128, 128)))
                count+=1
```

To flatten the image and split test and training data

```
input_shape = (128, 128, 3)
data_dir = 'dataset'

real_data = [f for f in os.listdir(data_dir+'/real') if
f.endswith('.png')]
fake_data = [f for f in os.listdir(data_dir+'/fake') if
f.endswith('.png')]
X = []
Y = []

for img in real_data:
    X.append(img_to_array(load_img(data_dir+'/real/'+img)).flatten()
/ 255.0)
    Y.append(1)
for img in fake_data:
    X.append(img_to_array(load_img(data_dir+'/fake/'+img)).flatten()
/ 255.0)
    Y.append(0)
```

```
Y_val_org = Y

#Normalization
X = np.array(X)
Y = to_categorical(Y, 2)

#Reshape
X = X.reshape(-1, 128, 128, 3)

#Train-Test split
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size =
0.2, random_state=5)
```

To run the model for training

```
early_stopping = EarlyStopping(monitor='val_loss',
                               min_delta=0,
                               patience=2,
                               verbose=0, mode='auto')
EPOCHS = 20
BATCH_SIZE = 100
history = model.fit(X_train, Y_train, batch_size = BATCH_SIZE, epochs
= EPOCHS, validation_data = (X_val, Y_val), verbose = 1)
```

7 Evaluation

To generate graph of the result comparing accuracy and loss value in each epoch

```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 4))
t = f.suptitle('Pre-trained InceptionResNetV2 Transfer Learn with
Fine-Tuning & Image Augmentation Performance ', fontsize=12)
f.subplots_adjust(top=0.85, wspace=0.3)

epoch_list = list(range(1,EPOCHS+1))
ax1.plot(epoch_list, history.history['accuracy'], label='Train
Accuracy')
ax1.plot(epoch_list, history.history['val_accuracy'],
label='Validation Accuracy')
ax1.set_xticks(np.arange(0, EPOCHS+1, 1))
ax1.set_ylabel('Accuracy Value')
ax1.set_xlabel('Epoch #')
ax1.set_title('Accuracy')
l1 = ax1.legend(loc="best")

ax2.plot(epoch_list, history.history['loss'], label='Train Loss')
ax2.plot(epoch_list, history.history['val_loss'], label='Validation
Loss')
ax2.set_xticks(np.arange(0, EPOCHS+1, 1))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch #')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")
```

To generate confusion matrix for evaluation

```
def print_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    print('True positive = ', cm[0][0])
    print('False positive = ', cm[0][1])
    print('False negative = ', cm[1][0])
    print('True negative = ', cm[1][1])
    print('\n')
    df_cm = pd.DataFrame(cm, range(2), range(2))
    sn.set(font_scale=1.4) # for label size
    sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
    plt.ylabel('Actual label', size = 20)
    plt.xlabel('Predicted label', size = 20)
    plt.xticks(np.arange(2), ['Fake', 'Real'], size = 16)
    plt.yticks(np.arange(2), ['Fake', 'Real'], size = 16)
    plt.ylim([2, 0])
    plt.show()

print_confusion_matrix(Y_val_org, model.predict_classes(X))
```