National College of Ireland

Unmasking Deception: Deepfake detection using Shallow CNN

MSc Research Project
Data Analytics

Ashwyn Singh
Student ID: 22128310

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ashwyn Singh
Student ID:	22128310
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Christian Horn
Submission Due Date:	14/12/2023
Project Title:	Unmasking Deception: Deepfake detection using Shallow CNN
Word Count:	1050
Page Count:	5

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:	31st January 2024

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):	

Unmasking Deception: Deepfake detection using Shallow CNN

Ashwyn Singh
22128310

1 Introduction

The configuration manual gives detailed information about doing the study and running it on the original equipment. Following that, it provides details on the hardware configuration of the original device, emphasizing the importance of an identical or superior hardware configuration to enable successful replication of the research. It also describes the program prerequisites required for running the study code prior to execution. The last part elaborates on the critical project configurations necessary inside the source code to allow the research code to execute.

2 Hardware Details

The code of the project was developed and executed on the local machine, specifications of which are documented in the Table 1.

Table 1: Machine Configuration

Feature	**Config**
Operating System	Windows 11
System Memory	16GB
Processor	AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz
Graphic Card	NVIDIA Geforce RTX 3050 Laptop GPU (4GB Memory)
Storage	1 TB

The machine with the above specifications was able to do the pre-processing tasks and run the shallow CNN model on CPU. The only issue was with the system memory. Both the VRAM of GPU and system ram of 16GB were not enough to store the data of frames. Becuase of this, only the first 7 subfolders of the dataset were used in this project.

3 Software Requirements

This project was implemented using python language. Python is a flexible, high-level programming language noted for its usability and simplicity. Jupyter notebook was used as IDE for both pre-processing and implementing the model. Jupyter Notebook provides

a dynamic web-based ecosystem which enables for program execution, graphical representation, and documentation, which makes it ideal for analyzing data, research, and teaching. Python's version 3.9.18 was used to developed this project. Table 2 gives the information about the libraries, their versions being used in the project and the line of code to install them.

Table 2: Machine Configuration

Library	**Version**	**Install command**
pandas	2.1.1	pip install pandas
numpy	1.26	pip install numpy
cv2	4.8.0	pip install opencv-python
plotly	5.18.0	pip install plotly
keras	2.14.0	pip install keras
tensorflow	2.14.0	pip install tensorflow
sklearn	1.3.2	pip install scikit-learn
seaborn	0.11.2	pip install seaborn
matplotlib	3.8.0	pip install matplotlib
mtcnn	0.1.0	pip install mtcnn

4 Project Configuration

This section discusses about the dataset, how it is utilized and the steps to reproduce the results with the artefact.

4.1 Dataset Details

The DFDC dataset is available on the Kaggle [1] after accepting the rules and regulations of the competition. There is single file which has all the data as well as the dataset is divided into 50 sub-folders as can be seen in Figure 1 . For this project, the first 7 sub-folders were downloaded.

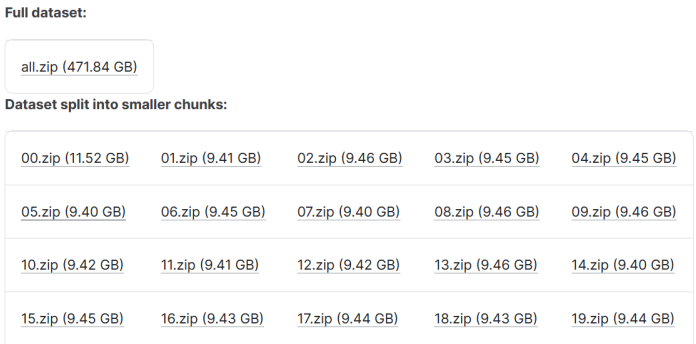

Figure 1: Dataset available on Kaggle

[1] https://www.kaggle.com/competitions/deepfake-detection-challenge/data

4.2 Sequence in which the code is to be run

The first python code which needs to be executed is "Frame from Videos.ipynb". This code file is responsible for the pre-processing of the videos and extracting images of subject's faces from them. Class imbalance problem has been handled in this code itself as an equal number of real and fake videos are selected for the process of extracting frames. Figure 2 shows the jupyter cell in which MTCCN algorithm is applied to each selected frame and then the image of the face is resized and stored in the folder for which the details are given in the next section.

```
In [99]: plt.rcParams['figure.figsize'] = [10, 5]
desired_face_size = (224, 224)
t=0
## It will generate 12 faces from each set of videos
for fn in meta_train_df.index:
    t=t+1
    print("number of images processed:" + str(t))
    #print("in main for loop")
    label = meta_train_df.loc[fn]['label']
    orig = meta_train_df.loc[fn]['label']
    video_file = f'{DATA_FOLDER}{TRAIN_SAMPLE_FOLDER}{fn}'
    print(f"{fn.split('.')[0]} n {label}")
    count=0
    cap = cv2.VideoCapture(video_file)
    #cap.set(cv2.CAP_PROP_FRAME_COUNT, frame_seq-1)
    while cap.isOpened():
        #print("in while loop")
        success, image = cap.read()
        if success :
            #print("in success if")
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            face_locations =  detector.detect_faces(image)
            print(len(face_locations))
            if len(face_locations) > 0:
                #print("in face_locations if")
            # Print first face
                for person in face_locations:
                    print(fn)
                    #print("in  second for loop")
                    if person['confidence']>0.98:
                        #print("in faces if condition")
                        i=0
                        bounding_box = person['box']
                        keypoints = person['keypoints']
                        print(bounding_box)
                        print(image.shape)
                        cropped_face=cropnpad(bounding_box,image,0)
                        resized_face = cv2.resize(cropped_face, desired_face_size)
                        plt.imsave(f"{OUTPUT_PATH}{fn.split('.')[0]}-{label}-{str(i)}-{str(count)}.png",res
                        #plt.imsave(f"{OUTPUT_PATH}{fn.split('.')[0]}-{label}-{str(i)}-{str(count)}.png",cr
                        #plt.imshow(cropped_face)
                        #plt.show()
                        i+=1
                    count += 25 # i.e. at 25 fps, each video creates 12 images . I want to limit the output
                    cap.set(1, count)
            else:
                #print("exit")
                cap.release()
                break
        else:
                #print("exit")
                cap.release()
                break
```

Figure 2: Extracting frames from videos and applying MTCNN

In the second code file "Meso-4 model implementation.ipynb", the model is implemented using keras, trained and evaluated. Figure 3 shows the code to create the model structure, figure 4 shows the training part and saving the model and figure 5 shows the calculation of logloss, accuracy, precision, recall along with confusion matrix.

```
In [24]:
from keras.layers import Input, Conv2D, BatchNormalization, MaxPooling2D, Flatten, Dropout, Dense, LeakyReLU
from keras.models import Model
from keras.optimizers import Adam

# Define the function to create the custom model
def create_custom_model(image_dimensions, learning_rate=0.001):
    x = Input(shape=(image_dimensions['height'],
                     image_dimensions['width'],
                     image_dimensions['channels']))

    x1 = Conv2D(8, (3, 3), padding='same', activation='relu')(x)
    x1 = BatchNormalization()(x1)
    x1 = MaxPooling2D(pool_size=(2, 2), padding='same')(x1)

    x2 = Conv2D(8, (5, 5), padding='same', activation='relu')(x1)
    x2 = BatchNormalization()(x2)
    x2 = MaxPooling2D(pool_size=(2, 2), padding='same')(x2)

    x3 = Conv2D(16, (5, 5), padding='same', activation='relu')(x2)
    x3 = BatchNormalization()(x3)
    x3 = MaxPooling2D(pool_size=(2, 2), padding='same')(x3)

    x4 = Conv2D(16, (5, 5), padding='same', activation='relu')(x3)
    x4 = BatchNormalization()(x4)
    x4 = MaxPooling2D(pool_size=(4, 4), padding='same')(x4)

    y = Flatten()(x4)
    y = Dropout(0.5)(y)
    y = Dense(16)(y)
    y = LeakyReLU(alpha=0.1)(y)
    y = Dropout(0.5)(y)
    y = Dense(1, activation='sigmoid')(y)

    model = Model(inputs=x, outputs=y)

    optimizer = Adam(lr=learning_rate)
    model.compile(optimizer=optimizer,
                  loss='mean_squared_error',
                  metrics=['accuracy'])

    return model

# Define image_dimensions
image_dimensions = {
    'height': 224,
    'width': 224,
    'channels': 3
}

# Create the custom model
custom_model = create_custom_model(image_dimensions)

# Print model summary
custom_model.summary()
```

Figure 3: Creating model structure using Keras library

```
In [26]:
history = custom_model.fit(X_train, y_train, epochs=2, batch_size=64,
                           validation_data=(X_val, y_val))

Epoch 1/2
371/371 [==============================] - 298s 793ms/step - loss: 0.7666 - accuracy: 0.5625 - val_loss: 0.6543 - val_accuracy: 0.6029
Epoch 2/2
371/371 [==============================] - 292s 787ms/step - loss: 0.6605 - accuracy: 0.6155 - val_loss: 0.6455 - val_accuracy: 0.6264

In [27]:
from tensorflow.keras.models import load_model
custom_model.save('25_epochs_3.h5')

D:\Anaconda\lib\site-packages\keras\src\engine\training.py:3079: UserWarning:

You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native
Keras format, e.g. `model.save('my_model.keras')`.

In [28]:
from tensorflow.keras.models import load_model
loaded_model = load_model('25_epochs_3.h5')

In [ ]:

In [29]:
score = loaded_model.evaluate(X_test, y_test)

205/205 [==============================] - 17s 81ms/step - loss: 0.6320 - accuracy: 0.6215
```

Figure 4: Training the model

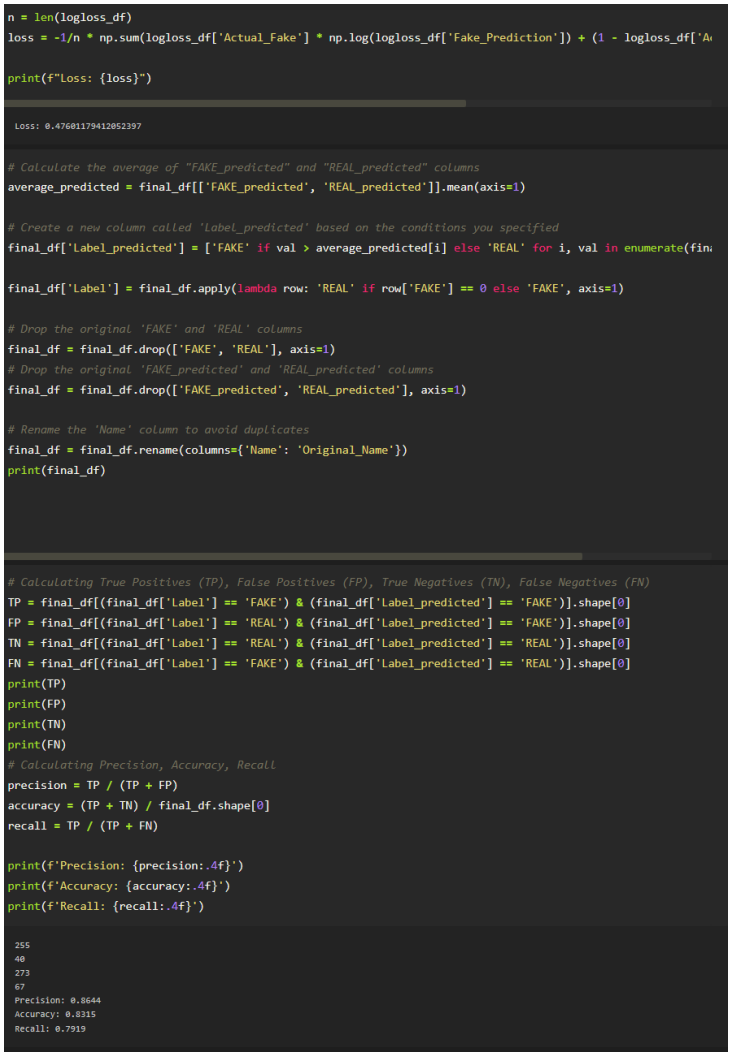

Figure 5: Calculating evaluation metrics

4.3 Folder Configuration

The videos which are downloaded are then extracted to the "train_sample_videos" folder which is the "deepfake-detection-challenge" folder. Then python code "Frame from Videos.ipynb" file is executed. This code does the pre-processing and stores the images in the folder "output" which is located in the "working" folder. This folder acts as the database for the code "Meso-4 model implementation.ipynb" which uses the images stored in the folder to train the model.

5 Artefact details

Figure a shows the contents of the zip file which is shared in the submission. Since each video dataset is 9Gb each and the dataset of facial images created using the fist python code is 1.56gb, that's why they are removed from the submission folder. The image dataset can be created easily by following the steps in the above sections.