Configuration Manual

MSc Research Project
Cyber Security

Raj Yatin Koli
Student ID: x21154678

School of Computing
National College of Ireland

Supervisor: Michael Pantridge

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name	Raj Yatin Koli
Student ID	X21154678
Programme	Cyber Security
Year:	2023
Module:	Msc Research Project
Supervisor:	Michael Pantridge
Submission Due Date:	18th September 2023
Project Title:	Deepfake Detection System by Integrating Deep Learning and Blockchain Technology
Word Count:	946
Page Count:	13

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	Raj Yatin Koli
Date 18th September 2023	

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

Raj Yatin Koli
Student ID: x21154678

1 Introduction

This Configuration Manual lists together all prerequisites needed to duplicate the studies and its effects on a specific setting. A glimpse of the source for Data Importing & Video frame analysis and after that Block chain after that all the created algorithms, and Evaluations is also supplied, together with the necessary hardware components as well as Software applications. The report is organized as follows, with details relating environment configuration provided in Section 2.

Information about data gathering is detailed in Section 3. Data exploration is done in Section 4. Video Frame Analysis is included in Section 5. In section 6, the Blockchain is described. Details well about models that were created and tested are provided in Section 7. How the results are calculated and shown is described in Section 8.

2 System Requirements

The specific needs for hardware as well as software to put the research into use are detailed in this section.

2.1 Hardware Requirements

The necessary hardware specs are shown in Figure 1 below. MacOs M1 Chip, macOS 10.15.x (Catalilna) operating system, 8GB RAM, 256GB Storage, 24'' Display.

Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 (Version 4.8.0)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

2.3 Code Execution

The code can be run in jupyter notebook. The jupyter notebook comes with Anaconda 3, run the jupyter notebook from startup. This will open jupyter notebook in web browser. The web browser will show the folder structure of the system, move to the folder where the code file is located. Open the code file from the folder and to run the code, go to Kernel menu and Run all cells.

3 Data Collection

The dataset is taken from Kaggle public repository from the link https://www.kaggle.com/competitions/deepfake-detection-challenge/data. Facebook, Microsoft, the Partnership on AI's Media Integrity Steering Committee, and academics have come together to build the Deepfake Detection Challenge (DFDC).

4 Data Exploration

Figure 2 includes a list of every Python library necessary to complete the project.

```
import numpy as np
import pandas as pd
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import os
import datetime
import hashlib
import json
from uuid import uuid4
import socket

from sklearn.model_selection import train_test_split
import keras
from keras.layers import Conv1D, Conv2D, Conv3D, ConvLSTM2D, Dense, Flatten, Dropout, BatchNormalization, GRU
from keras.layers import Input
from keras.models import Sequential, Model
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

Using TensorFlow backend.
```

Figure 2: Necessary Python libraries

The Figure 3 represents the block of code to import the training videos and check for the data.

```
train_dir = 'train_sample_videos/'
train_video_files = [train_dir + x for x in os.listdir(train_dir)]
test_dir = 'test_videos/'
test_video_files = [test_dir + x for x in os.listdir(test_dir)]
```

```
df_train = pd.read_json('train_sample_videos/metadata.json').transpose(
df_train.head()
```

	label	split	original
aagfhgtpmv.mp4	FAKE	train	vudstovrck.mp4
aapnvogymq.mp4	FAKE	train	jdubbvfswz.mp4
abarnvbtwb.mp4	REAL	train	None
abofeumbvv.mp4	FAKE	train	atvmxvwyns.mp4
abqwwspghj.mp4	FAKE	train	qzimuostzz.mp4

Figure 3: Importing training videos and Checking Data Information

As seen in Figure 4, the information about the training data.

```
df_train.shape # We have 400 vide
```

```
(400, 3)
```

```
df_train.original.nunique()  # fr
```

```
209
```

```
df_train.label.value_counts()
```

```
FAKE    323
REAL     77
Name: label, dtype: int64
```

Figure 4: Training data

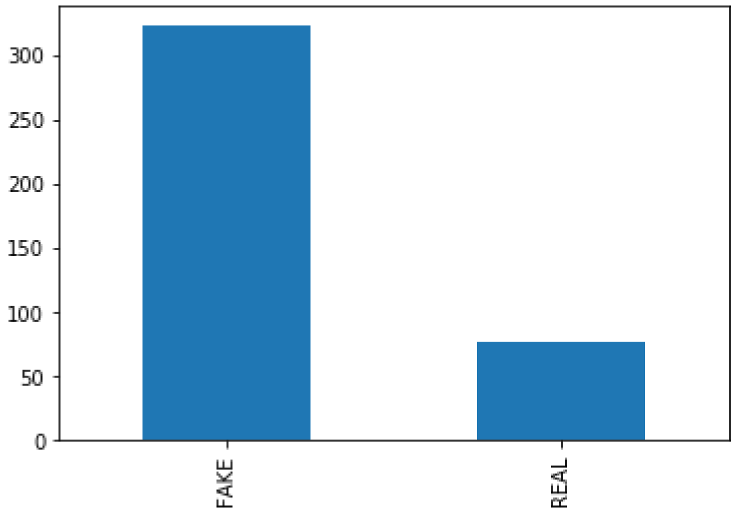
In figure 5, the code to value counts for real and fake video in the dataset.

```
df_train.label.value_counts()
```

```
FAKE    323
REAL     77
Name: label, dtype: int64
```

```
df_train.label.value_counts().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ad7f2c90c8:
```




```
df_train.label.value_counts(normalize=True
```

```
FAKE    0.8075
REAL    0.1925
Name: label, dtype: float64
```

Figure 5: Class count

The Figure 6, illustrate the code to check the value counts of original columns giving the count of fake videos available for that video.

```
df_train['original'].value_counts()
```

```
atvmxvwyns.mp4    6
meawmsgiti.mp4    6
qeumxirsme.mp4    5
kgbkktcjxf.mp4    5
fysyrqfguw.mp4    4
                 ..
cizlkenljw.mp4    1
xagsvjctmp.mp4    1
uuxqylnzls.mp4    1
brwrlczjvi.mp4    1
bdnaqemxmr.mp4    1
Name: original, Length: 209, dtype: int64
```

```
df_train[df_train['original']=='meawmsgiti.mp4']
```

	label	split	original
akxoopqjqz.mp4	FAKE	train	meawmsgiti.mp4
altziddtxi.mp4	FAKE	train	meawmsgiti.mp4
arlmiizoob.mp4	FAKE	train	meawmsgiti.mp4
axczxisdtb.mp4	FAKE	train	meawmsgiti.mp4
bqhtpqmmqp.mp4	FAKE	train	meawmsgiti.mp4
czkdanyadc.mp4	FAKE	train	meawmsgiti.mp4

Figure 6: Multi Convolver

5 Video Frame Analysis

The Figure 7, illustrate the read the video frame and show each frame image.

```
def display_img(video):
    cap = cv2.VideoCapture(video)  # take 1 picture
    ret, frame = cap.read()
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    ax.imshow(frame)
```

```
display_img(video1)
```

Figure 7: Read video

Figures 8 show the code used read training video and find its data entry in the dataset.

```
first_Video = train_video_files[8]
first_Video
```

```
'train_sample_videos/acxwigylke.mp4'
```

```
name_video = first_Video.split('/', 5)[1]
```

```
df_train[df_train.index == name_video]
```

	label	split	original
acxwigylke.mp4	FAKE	train	ffcwhpnpuw.mp4

Figure 8: read video

The Figure 9, illustrate the code to generate frames into images and display each frame by getting their current frame position.

```
count = 0
cap = cv2.VideoCapture(first_Video)
ret,frame = cap.read()

while count < 3:
    cap.set(cv2.CAP_PROP_POS_MSEC,(count*1000))
    ret,frame = cap.read()
    if count == 0:
        image0 = frame
    elif count == 1:
        image1 = frame
    elif count == 2:
        image2 = frame

    #cv2.imwrite( filepath+ "\frame%d.jpg" % count
    count = count + 1
```

```
def display(img):

    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    ax.imshow(img)
```

```
display(image0)  # frame 1
```

Figure 9: read video

6 Blockchain

The Figure 10, illustrate the code to read features and target from the data and to set global variables for blockchain.

```
x =df_train['original']
y =df_train['label']
```

```
img_size = 224
block_size = 64
blocks = 5
input_shape = (img_size,img_size,3)
max_chain_length = 20
num_features = 2048
```

Figure 10: Global Variables

Figures 11 show the code to create functions for cropping to centre and loading the block..

```
def crop_center_square(frame):
    y,x = frame.shape[0:2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
    return frame[start_y :start_y + min_dim, start_x : start_x + min_dim]

def load_block(path, max_frames=0, resize=(img_size, img_size)):
    cap = cv2.VideoCapture(path)
    frames = []
    try:
        while 1:
            ret, frame = cap.read()
            if not ret:
                break
            frame = crop_center_square(frame)
            frame = cv2.resize(frame, resize)
            frame = frame[:, :, [2, 1, 0]]
            frames.append(frame)

            if len(frames) == max_frames:
                break
    finally:
        cap.release()
    return np.array(frames)
```

Figure 11: Load block

Figures 12 show the code to analyze blocks and generating features of the block using InceptioNet.

```
def blockAnalyse():
    feature_extractor = keras.applications.InceptionV3(weights = "imagenet", include_top=False, pooling="avg", input_shape = inp
ut_shape)
    inputs = Input(input_shape)
    outputs = feature_extractor(inputs)
    return Model(inputs, outputs, name="feature_extractor")

feature_extractor = blockAnalyse()
```

Figure 12: Block Analysis

The Figure 13, illustrate the code to create the function for generating the blockchain for the video.

```python
def prepareBlockchain(df, root_dir):
    num_samples = len(df)
    video_paths = list(df.index)
    labels = df["label"].values
    labels = np.array(labels=='FAKE').astype(np.int)

    frame_masks = np.zeros(shape=(num_samples, max_chain_length), dtype="bool")
    frame_features = np.zeros(
        shape=(num_samples, max_chain_length, num_features), dtype="float32"
    )

    for idx, path in enumerate(video_paths):
        frames = load_block(os.path.join(root_dir, path))
        frames = frames[None, ...]

        temp_frame_mask = np.zeros(shape=(1, max_chain_length,), dtype="bool")
        temp_frame_features = np.zeros(shape=(1, max_chain_length, num_features), dtype="float32"

        for i, batch in enumerate(frames):
            video_length = batch.shape[0]
            length = min(max_chain_length, video_length)
            for j in range(length):
                temp_frame_features[i, j, :] =feature_extractor.predict(batch[None, j, :])
            temp_frame_mask[i, :length] =1 # 1 = not masked, 0 = masked

        frame_features[idx,] =temp_frame_features.squeeze()
        frame_masks[idx,] =temp_frame_mask.squeeze()

    return (frame_features, frame_masks), labels
```

Figure 13: Function to generate blockchain

The Figure 14, illustrate the code to build training and testing data and implementation to generate the training and testing set into blockchain.

```python
train , test = train_test_split(df_train, test_size=0.1,random_state=42, stratify=df_train['label'])
x_train, y_train = prepareBlockchain(train, "train")
x_val, y_val = prepareBlockchain(test, "test")
```

Figure 14: Blockchain

7 Deep Learning Models

7.1 RNN

```
blockchain_input = Input((max_chain_length, num_features))
mask_input = Input((max_chain_length,),dtype="bool")
```

```
x = GRU(16, return_sequences=True)(blockchain_input, mask = mask_input)
x = GRU(8)(x)
x = Dropout(0.4)(x)
x = Dense(8, activation="tanh")(x)
output = Dense(1, activation="tanh")(x)
```

```
rnn = Model([blockchain_input, mask_input], output)
rnn.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
rnn.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 20, 2048)          0
_________________________________________________________________
gru_1 (GRU)                  (None, 20, 16)            99120
_________________________________________________________________
gru_2 (GRU)                  (None, 8)                 600
_________________________________________________________________
dropout_1 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 99,801
Trainable params: 99,801
Non-trainable params: 0
_________________________________________________________________
```

```
es = EarlyStopping(monitor='accuracy', verbose=1, patience=3)
history = rnn.fit([x_train[0], x_train[1]],y_train,validation_data=([x_val[0], x_val[1]], y_val),epochs=blocks, callbacks=[es]
```

```
Train on 360 samples, validate on 40 samples
Epoch 1/5
360/360 [==============================] - 1s 4ms/step - loss: 12.4685 - accuracy: 0.1917 - val_loss: 12.3400 - val_accuracy:
0.2000
Epoch 2/5
360/360 [==============================] - 0s 789us/step - loss: 12.4685 - accuracy: 0.1917 - val_loss: 12.3400 - val_accuracy
0.2000
```

Figure 15: Implementation of RNN

7.2 Convultional RNN

```
x = Conv1D(18, kernel_size=3, activation='relu', padding='same')(blockchain_input)
x = Conv1D(64, kernel_size=3, activation='relu', padding='same')(x)
x = GRU(32)(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(16, activation="relu")(x)
output = Dense(1, activation="sigmoid")(x)
```

```
cnn = Model([blockchain_input, mask_input], output)
cnn.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
cnn.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 20, 2048)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 20, 18)            110610
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 20, 64)            3520
_________________________________________________________________
gru_3 (GRU)                  (None, 32)                9312
_________________________________________________________________
batch_normalization_95 (Batc (None, 32)                128
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                528
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 124,115
Trainable params: 124,051
Non-trainable params: 64
_________________________________________________________________
```

```
es = EarlyStopping(monitor='accuracy', verbose=1, patience=3)
```

```
history = cnn.fit([x_train[0], x_train[1]],y_train,validation_data=([x_val[0], x_val[1]], y_val),epochs=blocks, callbacks=[es]
```

```
Train on 360 samples, validate on 40 samples
Epoch 1/5
360/360 [==============================] - 1s 3ms/step - loss: 0.6904 - accuracy: 0.7639 - val_loss: 0.6868 - val_accuracy: 0.
000
Epoch 2/5
360/360 [==============================] - 0s 989us/step - loss: 0.6835 - accuracy: 0.8083 - val_loss: 0.6798 - val_accuracy:
0.8000
Epoch 3/5
```

Figure 16: Implementation of C-RNN

7 Model result

This section explains the performance of the models.

7.1 Model Scores

```
loss, accuracy = rnn.evaluate([x_val[0], x_val[1]], y_val
accuracy*100
```

```
40/40 [==============================] - 0s 601us/step

20.000000298023224
```

Figure 18: Model Performance RNN

```
loss, accuracy = cnn.evaluate([x_val[0], x_val[1]], y_val
round(accuracy*100)
```

```
40/40 [==============================] - 0s 509us/step

80
```

Figure 19: Model Performance CNN

References

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

OpenCV: OpenCV modules

https://www.opensourceforu.com/2019/08/using-python-tools-and-libraries-for-blockchain-programming/

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

Convolutional Neural Network (CNN) | TensorFlow Core