Configuration Manual

MSc Research Project
M.Sc. Cybersecurity

Himanshu Sharma
Student ID: 22220135

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Himanshu Sharma		
Student ID:	22220135		
Programme:	M.Sc. Cybersecurity	**Year:**	2023-24
Module:	MSc Research Practicum		
Lecturer:	Prof. Vikas Sahni		
Submission Due Date:	16/09/2024		
Project Title:	Enhancing Biometric Security Systems Against Deepfake Threats		
Word Count:	751	**Page Count:**	8

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: Himanshu Sharma

Date: 15/09/2024

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):	

Configuration Manual

Himanshu Sharma
22220135

1. Introduction

This is a step by step tutotial to create, train and deploy a deepfake detection model using Google Colab with TPU and Flask for API implementation. This manual is designed for developers and data scientists, who want to train and integrate a deepfake detection model.

2. System Requirements

- **Hardware**:
  - Google Colab with TPU v2 support for model training.
  - Local machine for API deployment and testing.
- **Software**:
  - Python 3.10.12
  - Libraries: TensorFlow 2.17.0, NumPy 1.23.5, OpenCV 4.5.5.62, Flask 2.1.1, and others as specified in the code.
  - Ngrok for exposing the Flask API to the internet.

3. Mount Google Drive

- **Purpose**: Installing Google Drive is crucial to be able to call datasets stored in the cloud and save the processed data, models, and output directly to the Drive.

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

4. Data Preprocessing

- **Objective**: Get frames from real and fake videos, resize them and store in smaller sizes while processing to avoid memory issues.
- **Steps**:
  1. **Extract Frames**: Video frames are extracted and resized to 224x224 pixels.
  2. **Save in Batches**: Frames are saved in smaller batches (e.g., 10 videos per batch) for both real and fake videos.

FaceForensics++ dataset. Available at: https://www.kaggle.com/datasets/hungle3401/faceforensics/data
GitHub Repository – https://github.com/himanshu5991/DeepfakeDetectionEBS.git

```python
import os
import numpy as np
import cv2
from tqdm import tqdm

output_path = '/content/drive/MyDrive/FaceForensics_preprocessed'
os.makedirs(output_path, exist_ok=True)

real_videos_path = '/content/drive/MyDrive/FaceForensics++/real'
fake_videos_path = '/content/drive/MyDrive/FaceForensics++/fake'

def extract_frames(video_path, label, frame_rate=1):
    cap = cv2.VideoCapture(video_path)
    count = 0
    success = True
    frames = []
    while success:
        success, frame = cap.read()
        if count % frame_rate == 0 and success:
            frame = cv2.resize(frame, (224, 224))
            frames.append((frame, label))
        count += 1
    cap.release()
    return frames

def preprocess_data_in_batches(videos_path, label, prefix, batch_size=10, frame_rate=1):
    video_files = os.listdir(videos_path)
    for i in range(0, len(video_files), batch_size):
        all_frames = []
        batch_files = video_files[i:i + batch_size]
        for video_name in tqdm(batch_files, desc=f'Processing batch {i // batch_size + 1}')
            video_path = os.path.join(videos_path, video_name)
            frames = extract_frames(video_path, label, frame_rate)
            all_frames.extend(frames)

        X = np.array([frame[0] for frame in all_frames])
        y = np.array([frame[1] for frame in all_frames])

        np.save(os.path.join(output_path, f'{prefix}_X_batch_{i // batch_size + 1}.npy'), X
        np.save(os.path.join(output_path, f'{prefix}_y_batch_{i // batch_size + 1}.npy'), y

# Preprocess real and fake videos in batches
preprocess_data_in_batches(real_videos_path, 0, 'real')
preprocess_data_in_batches(fake_videos_path, 1, 'fake')
```

5. Combine and Split Data

- **Objective**: Combine the smaller batches into larger datasets and then split the combined data into training, validation, and test sets.
- **Steps**:
    1. **Combine Batches**: Combine batches of real and fake frames.
    2. **Split Data**: Split the combined data into training (80%), validation (10%), and test (10%) datasets.
    3. **Save Split Data**: Save the split datasets for efficient loading during model training.

FaceForensics++ dataset. Available at: https://www.kaggle.com/datasets/hungle3401/faceforensics/data
GitHub Repository – https://github.com/himanshu5991/DeepfakeDetectionEBS.git

```python
import os
import numpy as np

output_path = '/content/drive/MyDrive/FaceForensics_preprocessed'

def combine_batches(output_path, prefix, num_batches):
    all_X = []
    all_y = []

    for i in range(1, num_batches + 1):
        X_path = os.path.join(output_path, f'{prefix}_X_batch_{i}.npy')
        y_path = os.path.join(output_path, f'{prefix}_y_batch_{i}.npy')
        if os.path.exists(X_path) and os.path.exists(y_path):
            print(f"Loading {X_path} and {y_path}")
            X_batch = np.load(X_path)
            y_batch = np.load(y_path)
            all_X.append(X_batch)
            all_y.append(y_batch)
        else:
            print(f"Files {X_path} or {y_path} do not exist. Skipping these batches.")

    X = np.concatenate(all_X, axis=0)
    y = np.concatenate(all_y, axis=0)

    return X, y

# Combining real and fake batches separately to handle memory
real_X, real_y = combine_batches(output_path, 'real', 10)
fake_X, fake_y = combine_batches(output_path, 'fake', 10)

np.save(os.path.join(output_path, 'combined_real_X.npy'), real_X)
np.save(os.path.join(output_path, 'combined_real_y.npy'), real_y)
np.save(os.path.join(output_path, 'combined_fake_X.npy'), fake_X)
np.save(os.path.join(output_path, 'combined_fake_y.npy'), fake_y)
```

```python
def load_and_split_data(output_path):
    real_X = np.load(os.path.join(output_path, 'combined_real_X.npy'))
    real_y = np.load(os.path.join(output_path, 'combined_real_y.npy'))
    fake_X = np.load(os.path.join(output_path, 'combined_fake_X.npy'))
    fake_y = np.load(os.path.join(output_path, 'combined_fake_y.npy'))

    X = np.concatenate((real_X, fake_X), axis=0)
    y = np.concatenate((real_y, fake_y), axis=0)

    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)
    X = X[indices]
    y = y[indices]

    split_1 = int(0.8 * len(X))
    split_2 = int(0.9 * len(X))
    X_train, X_val, X_test = np.split(X, [split_1, split_2])
    y_train, y_val, y_test = np.split(y, [split_1, split_2])

    # Save the splits in smaller chunks
    for i in range(0, len(X_train), batch_size):
        np.save(os.path.join(output_path, f'combined_X_train_batch_{i//batch_size+1}.npy'), X_train[i:i+batch_size])
        np.save(os.path.join(output_path, f'combined_y_train_batch_{i//batch_size+1}.npy'), y_train[i:i+batch_size])

    for i in range(0, len(X_val), batch_size):
        np.save(os.path.join(output_path, f'combined_X_val_batch_{i//batch_size+1}.npy'), X_val[i:i+batch_size])
        np.save(os.path.join(output_path, f'combined_y_val_batch_{i//batch_size+1}.npy'), y_val[i:i+batch_size])

    for i in range(0, len(X_test), batch_size):
        np.save(os.path.join(output_path, f'combined_X_test_batch_{i//batch_size+1}.npy'), X_test[i:i+batch_size])
        np.save(os.path.join(output_path, f'combined_y_test_batch_{i//batch_size+1}.npy'), y_test[i:i+batch_size])

batch_size = 32
load_and_split_data(output_path)
```

6. Load and Prepare Data for TensorFlow

- **Objective**: Load the split datasets from .npy files and use them to create TensorFlow Records datasets that are suitable for model training.

- **Steps**:

  1. **Load Data**: Load training, validation, and test datasets.

  2. **Create TensorFlow Datasets**: Load the data and form TensorFlow data sets which will involve shuffling and Batching so as to enhance the training process.

```python
import os
import numpy as np
import tensorflow as tf

# Function to load data from .npy files
def load_data(output_path, prefix_X, prefix_y, batch_size):
    X = []
    y = []
    batch_num = 1
    while True:
        X_path = os.path.join(output_path, f'{prefix_X}_batch_{batch_num}.npy')
        y_path = os.path.join(output_path, f'{prefix_y}_batch_{batch_num}.npy')
        print(f"Checking for files: {X_path} and {y_path}")
        if os.path.exists(X_path) and os.path.exists(y_path):
            X.append(np.load(X_path))
            y.append(np.load(y_path))
            print(f"Loaded batch {batch_num} from {X_path} and {y_path}")
            batch_num += 1
        else:
            break
    if not X or not y:
        print(f"No data found for prefixes {prefix_X} and {prefix_y}. Please ensure the files exist and are correctly named.")
        return np.array([]), np.array([])
    X = np.concatenate(X, axis=0)
    y = np.concatenate(y, axis=0)
    return X, y

# Load datasets
output_path = '/content/drive/MyDrive/FaceForensics_preprocessed'
print("Loading training dataset...")
X_train, y_train = load_data(output_path, 'combined_X_train', 'combined_y_train', 32)
print("Loading validation dataset...")
X_val, y_val = load_data(output_path, 'combined_X_val', 'combined_y_val', 32)
print("Loading test dataset...")
X_test, y_test = load_data(output_path, 'combined_X_test', 'combined_y_test', 32)

# Print dataset shapes and data types to verify successful loading
if X_train.size > 0 and y_train.size > 0:
    print(f'train X shape: {X_train.shape}')
    print(f'train y shape: {y_train.shape}')
    print(f'train X dtype: {X_train.dtype}')
    print(f'train y dtype: {y_train.dtype}')
if X_val.size > 0 and y_val.size > 0:
    print(f'val X shape: {X_val.shape}')
    print(f'val y shape: {y_val.shape}')
    print(f'val X dtype: {X_val.dtype}')
    print(f'val y dtype: {y_val.dtype}')
if X_test.size > 0 and y_test.size > 0:
    print(f'test X shape: {X_test.shape}')
    print(f'test y shape: {y_test.shape}')
    print(f'test X dtype: {X_test.dtype}')
    print(f'test y dtype: {y_test.dtype}')

# Create TensorFlow datasets
def create_tf_dataset(X, y, batch_size):
    if X.size == 0 or y.size == 0:
        print("Empty dataset received. Skipping creation of TensorFlow dataset.")
        return None
    dataset = tf.data.Dataset.from_tensor_slices((X, y))
    dataset = dataset.shuffle(buffer_size=len(y))
    dataset = dataset.batch(batch_size)
    dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
    return dataset

batch_size = 32
train_dataset = create_tf_dataset(X_train, y_train, batch_size)
val_dataset = create_tf_dataset(X_val, y_val, batch_size)
test_dataset = create_tf_dataset(X_test, y_test, batch_size)

# Check the first batch if the dataset is not empty
if train_dataset is not None:
    for images, labels in train_dataset.take(1):
        print("Image batch shape:", images.shape)
        print("Label batch shape:", labels.shape)
        print("Image batch dtype:", images.dtype)
        print("Label batch dtype:", labels.dtype)
```


FaceForensics++ dataset. Available at: https://www.kaggle.com/datasets/hungle3401/faceforensics/data
GitHub Repository – https://github.com/himanshu5991/DeepfakeDetectionEBS.git

7. Model Training

- **Objective**: It is necessary to set the deep learning model based on MobileNetV2, train it on the prepared datasets, and make evaluation.

- **Steps**:

  1. **Preprocess Dataset**: Normalize image data to [0, 1] and ensure labels are properly cast.

  2. **Define Model**: Use MobileNetV2 as the base, add custom layers, and compile the model.

  3. **Train Model**: Train the model with early stopping and model checkpointing.

  4. **Evaluate and Save Model**: Evaluate the model on the test set and save the final trained model.

```python
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# Function to preprocess the dataset
def preprocess(image, label):
    image = tf.cast(image, tf.float32) / 255.0  # Normalize the images to [0, 1]
    label = tf.cast(label, tf.int32)
    return image, label

# Apply preprocessing to the datasets
train_dataset = train_dataset.map(preprocess)
val_dataset = val_dataset.map(preprocess)
test_dataset = test_dataset.map(preprocess)

# Load the MobileNetV2 model without the top layer
base_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the convolutional base
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Define callbacks
checkpoint_path = "/content/drive/MyDrive/model_checkpoint.h5"
checkpoint_cb = ModelCheckpoint(checkpoint_path, save_best_only=True)
early_stopping_cb = EarlyStopping(patience=5, restore_best_weights=True)

# Train the model
history = model.fit(train_dataset, epochs=20, validation_data=val_dataset, callbacks=[checkpoint_cb, early_stopping_cb])

# Evaluate the model
test_loss, test_acc = model.evaluate(test_dataset)
print(f'Test accuracy: {test_acc}')

# Save the final model
model.save("/content/drive/MyDrive/final_model.h5")

# Plot training history
import matplotlib.pyplot as plt

def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs_range = range(len(acc))

    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()

plot_history(history)
```


FaceForensics++ dataset. Available at: https://www.kaggle.com/datasets/hungle3401/faceforensics/data
GitHub Repository – https://github.com/himanshu5991/DeepfakeDetectionEBS.git

8. Deploying the Model with Flask

- **Objective -** Convert the deployed deepfake detection model into the API using Flask where user can upload the video files and in return, user will get the binary (real or fake) and the confidence level.

- **Steps -**

  1. **Set Up Flask Application**: Load the trained model, define the API routes, and handle video file uploads.
  2. **Running the Flask App**: Host the Flask application locally and optionally expose it to the internet using Ngrok.

8.1 Code for Flask API

1. **Flask App Setup**:
   - Create a Flask app that loads the trained model and sets up an endpoint to receive video files, process them, and return predictions.

```python
import os
import cv2
import numpy as np
import tensorflow as tf
from flask import Flask, request, jsonify
from werkzeug.utils import secure_filename
from custom_layers import CustomDepthwiseConv2D

# Initialize the Flask application
app = Flask(__name__)

# Define the path to your saved model
model_path = 'final_model.h5'

# Load the model using the custom layer
model = tf.keras.models.load_model(model_path, custom_objects={'DepthwiseConv2D': CustomDepthwiseConv2D})

# Define the allowed extensions for video files
ALLOWED_EXTENSIONS = {'mp4', 'avi', 'mov', 'mkv'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def process_video(video_path):
    cap = cv2.VideoCapture(video_path)
    frame_number = 0
    total_prediction_value = 0.0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        # Preprocess the frame as required
        frame_resized = cv2.resize(frame, (224, 224))
        frame_array = np.expand_dims(frame_resized, axis=0)
        # Make prediction
        prediction = model.predict(frame_array)
        prediction_value = float(prediction[0][0])
        total_prediction_value += prediction_value
        frame_number += 1
    cap.release()

    average_prediction_value = total_prediction_value / frame_number if frame_number > 0 else 0
    threshold = 0.5  # Example threshold value
    video_classification = "fake" if average_prediction_value > threshold else "real"

    return video_classification, average_prediction_value

@app.route('/predict', methods=['POST'])
def predict():
    # Check if the post request has the video file part
    if 'file' not in request.files:
        return jsonify({'error': 'No file part in the request'}), 400
    file = request.files['file']
    # If the user does not select a file, the browser submits an empty file without a filename
    if file.filename == '':
        return jsonify({'error': 'No selected file'}), 400
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file_path = os.path.join('/tmp', filename)
        file.save(file_path)
        video_classification, average_prediction_value = process_video(file_path)
        os.remove(file_path)  # Clean up the saved file after processing
        return jsonify({
            'classification': video_classification,
            'score': average_prediction_value
        }), 200
    else:
        return jsonify({'error': 'File type not allowed'}), 400

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=int(os.environ.get('PORT', 8000)))
```

8.2 Running the Flask App:

- Run the Flask application locally using the following command:

```
(mynewenv) (base) himanshusharma@Himanshus-Laptop DDEBS % python app.py
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.comp
ile_metrics` will be empty until you train or evaluate the model.
WARNING:absl:Error in loading the saved optimizer state. As a result, your model is starting with
a freshly initialized optimizer.
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
WARNING:werkzeug: * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
INFO:werkzeug: * Running on http://10.13.228.120:8000/ (Press CTRL+C to quit)
INFO:werkzeug: * Restarting with stat
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.comp
ile_metrics` will be empty until you train or evaluate the model.
WARNING:absl:Error in loading the saved optimizer state. As a result, your model is starting with
a freshly initialized optimizer.
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 116-169-714
```

8.3 Expose the Flask API Using Ngrok:

- Ngrok is a tool that creates a secure tunnel to your local server, making it accessible over the internet.
- **Install Ngrok**: Download and install Ngrok from its official website.
- **Run Ngrok**: Expose the Flask API using Ngrok
- After running this command, Ngrok will provide a public URL that can be used to access your Flask API.

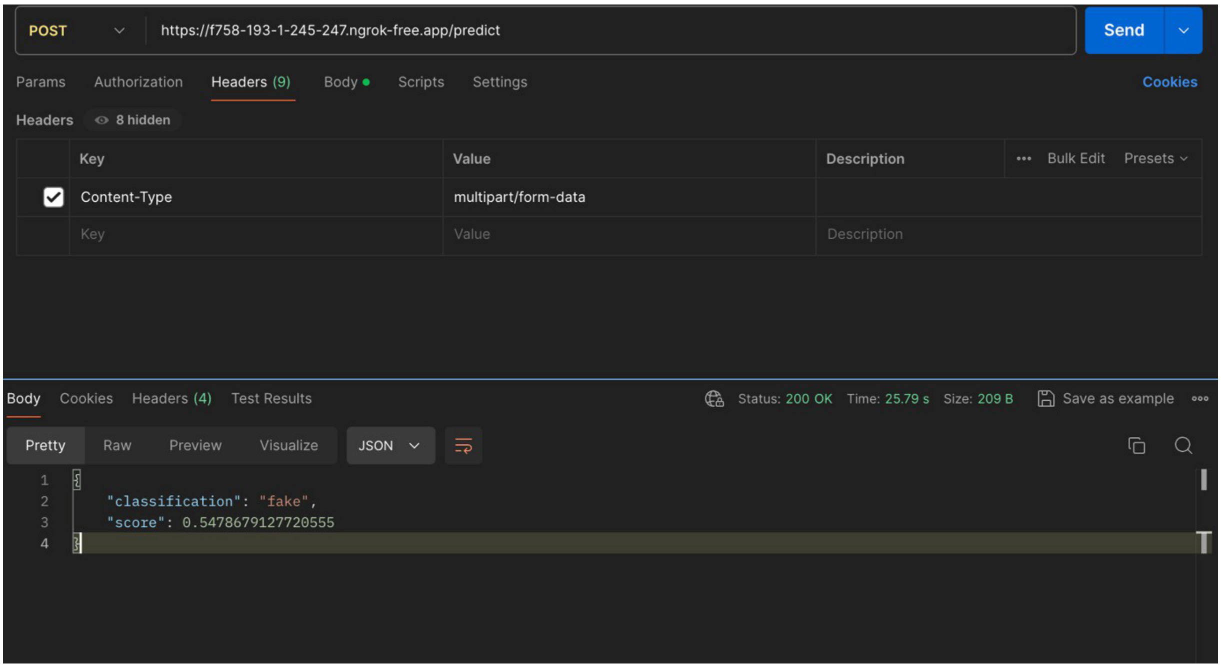
```
ngrok

Sign up to try new private endpoints https://ngrok.com/new-features-update?ref=private

Session Status                online
Account                       hs17596@gmail.com (Plan: Free)
Version                       3.14.0
Region                        Europe (eu)
Latency                       51ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://f758-193-1-245-247.ngrok-free.app -> http://localhost:8000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              1       0       0.00    0.00    25.55   25.55

HTTP Requests
-------------

14:11:15.341 IST POST /predict                200 OK
```

8.4 Testing the API:

- Use `curl` or Postman to send a POST request to the `/predict` endpoint with a video file.

```
curl --location 'https://your-ngrok-url.ngrok.io/predict' \
--header 'Content-Type: multipart/form-data' \
--form 'file=@"/path/to/your/video.mp4"'
```


FaceForensics++ dataset. Available at: https://www.kaggle.com/datasets/hungle3401/faceforensics/data
GitHub Repository – https://github.com/himanshu5991/DeepfakeDetectionEBS.git



9. Conclusion

It has expanded the specified workflow from the development of a deepfake detection model to the training of a deepfake detection model and its deployment through a Flask API. The application uses TensorFlow for model training and evaluation, and for end-users, it is possible to upload videos and obtain corresponding predictions using the Flask API layer. Here, Ngrok is employed to expose the API for external access to facilitate testing and demonstration.

References

Chollet, F. (2015). *Keras: The Python Deep Learning library*. Retrieved from https://keras.io

Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media. Retrieved from https://flask.palletsprojects.com/

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Retrieved from https://arxiv.org/abs/1801.04381

TensorFlow Developers. (n.d.). *TensorFlow*. Retrieved from https://www.tensorflow.org

Google Research. (n.d.). *Google Colaboratory*. Retrieved from https://colab.research.google.com

Hung, L. (n.d.) *FaceForensics dataset*. Available at: https://www.kaggle.com/datasets/hungle3401/faceforensics/data (Accessed: 11 August 2024).

FaceForensics++ dataset. Available at: https://www.kaggle.com/datasets/hungle3401/faceforensics/data
GitHub Repository – https://github.com/himanshu5991/DeepfakeDetectionEBS.git