Configuration Manual

MSc Research Project
Data Analytics

Aniket Suryakant Ghadge
Student ID: x23106786

School of Computing
National College of Ireland

Supervisor: Dr. David Hamill

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aniket Suryakant Ghadge
Student ID:	x23106786
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. David Hamill
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	1204
Page Count:	8

Signature:	Aniket Suryakant Ghadge
Date:	12th August 2024

Configuration Manual

Aniket Suryakant Ghadge
x23106786

1 Introduction

This configuration manual provides step-by-step instructions to replicate the project (Deepfake Detection in AV1 Compressed Videos with EfficientNet and Stacked Bi-LSTM Model). With the help of this manual, detailed instructions on setting up the environment, installation of required software, data collection and storage, pre-processing of data, model training and evaluation, loading the trained model, and testing new videos are covered. Follow each step carefully to ensure successful replication of the project.

2 Setting Google Colab Pro

This project requires high GPU power for processing videos and training the Efficient and the 3-layered bidirectional LSTM models. For this purpose, a Google Colab Pro[1] account is recommended, as it offers a tensor processing unit (TPU) with a 15GB GPU. The following steps can be followed for setting the Google Colab environment:

- Login to the Google Colab account via Google account.
- Although this platform provides 3 hours of free TPU power for pre-processing the videos, model training would be difficult to complete. It is suggested that paid versions of Google Colab Pro be used, as it offers 100 units of computational power that are sufficient for building a model simulation version.
- Once the Colab Pro version is enabled, the user should enable the TPU power by changing the runtime option for executing the code files provided.

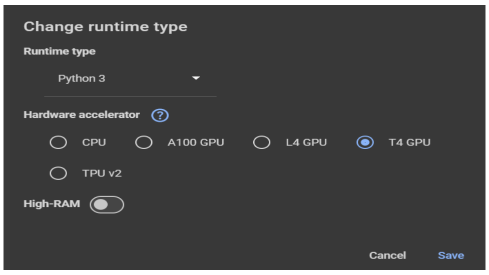

Change runtime type
Runtime type: Python 3
Hardware accelerator: CPU, A100 GPU, L4 GPU, [✓ T4 GPU], TPU v2
High-RAM
Cancel Save

Figure 1: Change runtime to TPU

The next section will help in understanding the data collection, data loading and required data pre-processing steps.

[1]https://colab.google/

3 Data Layer

This project has used the Python script for downloading the entire raw dataset after getting permission to access the dataset from the FaceForensics team. To gain access to the dataset, users can fill out the request form that is available on the FaceForensics[2] github site. After downloading the raw videos, the balancing of the dataset is performed using random sampling techniques using random-sample.py, and a total of 280 real and 280 fake videos were selected for this project. The compression of videos into the AV1 codec is executed with the help of the handbrake application. Below are the example steps to be followed:

3.1 Compression of videos using handbrake

- **Download and installing handbrake:** By visiting the official site of handbrake[3] application user can download latest version of the application.

- **Load the videos folder:** Open the application and load the dataset folder, as seen in Figure 2 below.

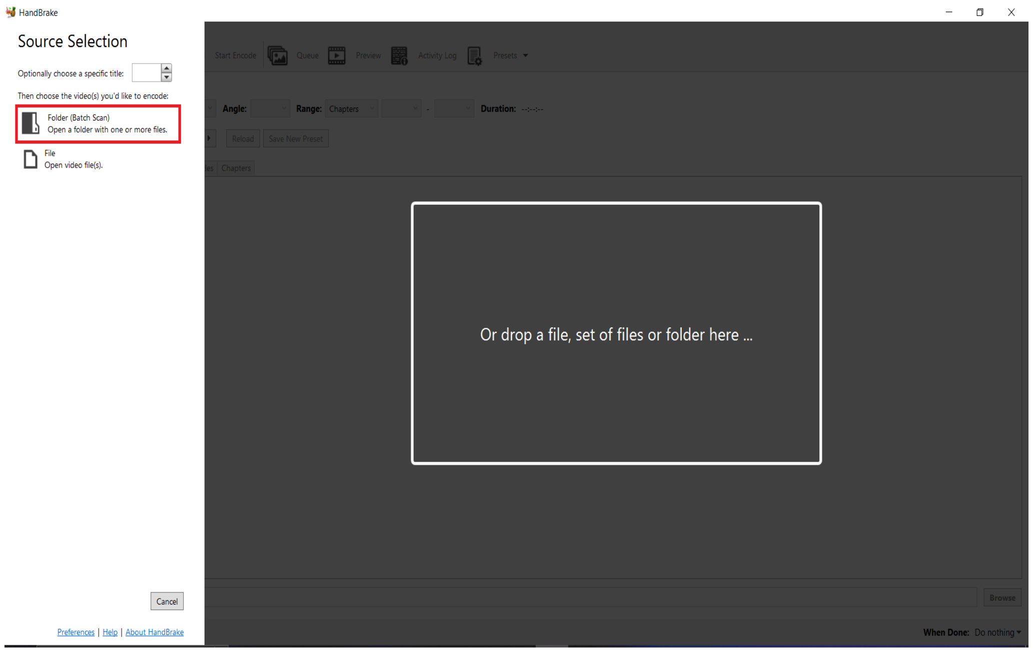
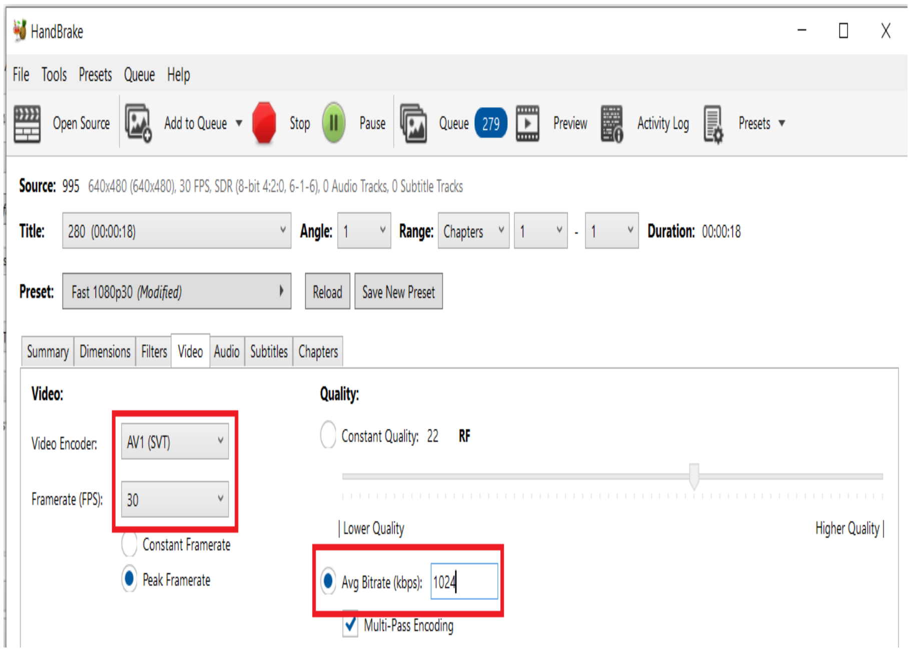

HandBrake
Source Selection
Optionally choose a specific title:
Then choose the video(s) you'd like to encode:
Folder (Batch Scan)
Open a folder with one or more files.
File
Open video file(s).
Cancel
Preferences | Help | About HandBrake
Or drop a file, set of files or folder here ...

Figure 2: Loading batch of videos on handbrake

- The below setting for high-bitrate AV1 video compressions has been used, similarly for low can be achieved by setting bitrates to 250 kbps. Once all the real and fakes are compressed and stored into two different folders. The further pre-processing techniques can be applied (Refer Figure 3).

- **Connect Google Drive:** Mount the Google Drive so that you can access the stored files as an alternative to cloud storage (Refer Figure 4).

[2]https://github.com/ondyari/FaceForensics
[3]https://handbrake.fr/downloads.php



Source: 995 640x480 (640x480), 30 FPS, SDR (8-bit 4:2:0, 6-1-6), 0 Audio Tracks, 0 Subtitle Tracks

Title: 280 (00:00:18) Angle: 1 Range: Chapters 1 - 1 Duration: 00:00:18

Preset: Fast 1080p30 *(Modified)*

Video:

Video Encoder: AV1 (SVT)

Framerate (FPS): 30

Constant Framerate [✓ Peak Framerate]

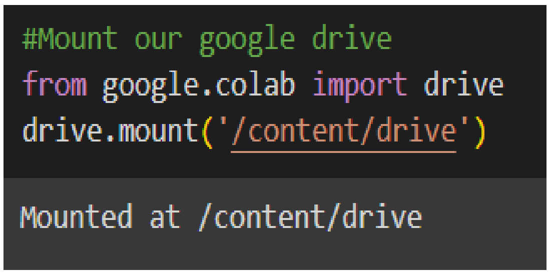
Quality:

Constant Quality: 22 RF

[✓ Avg Bitrate (kbps): 1024]

✓ Multi-Pass Encoding

Figure 3: High-bitrate videos setting on handbrake



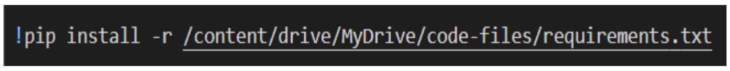
```
#Mount our google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Figure 4: Mount google Drive

Once successfully mounted, open the Google Drive account and upload the provided “code-files” folder. This folder contains three different (.ipynb) files—the raw, low-bitrate, and high-bitrate videos along with the trained models—code files, pre-processed videos, datasets (videos), and requirements.txt file.

- **Install tools and libraries:** Run the below code for installing all the prerequisites libraries.

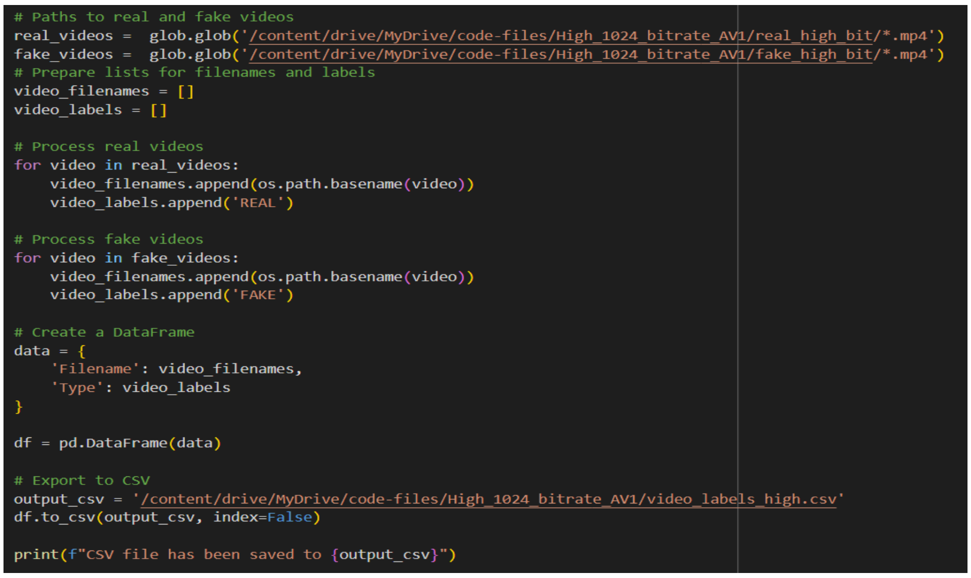


```
!pip install -r /content/drive/MyDrive/code-files/requirements.txt
```

Figure 5: Intalling required libraries and tools

- **Import libraries:** After installing all the prerequisite libraries and tools, import all the libraries by running the “Import Libraries” cell.

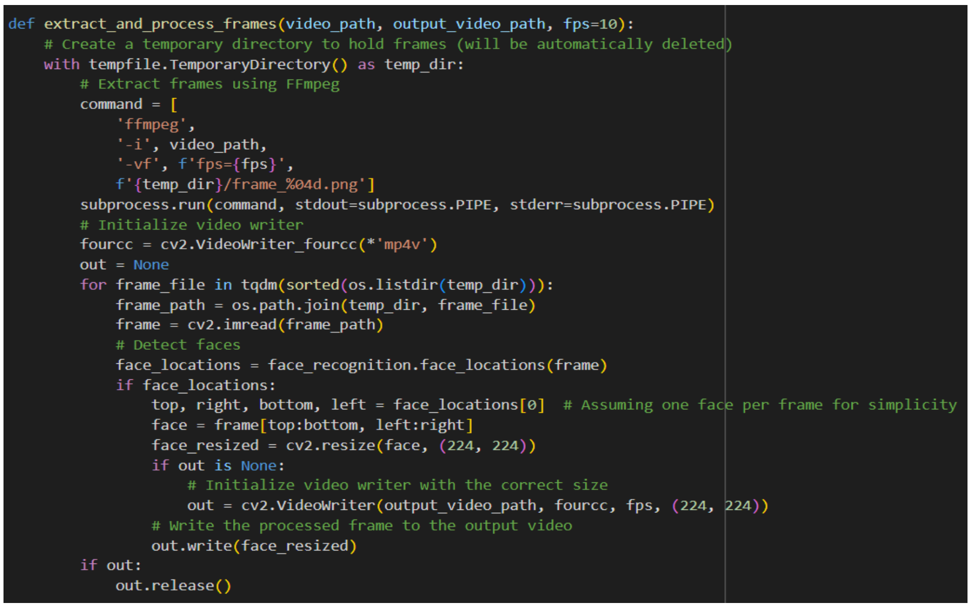
- **Data loading and labelling:** In this section, we load the data from two different directories, perform labelling on the videos, and export the labelling into a csv file for further use in the model training section. Figure 6 represents the code snippet used:



```python
# Paths to real and fake videos
real_videos =  glob.glob('/content/drive/MyDrive/code-files/High_1024_bitrate_AV1/real_high_bit/*.mp4')
fake_videos =  glob.glob('/content/drive/MyDrive/code-files/High_1024_bitrate_AV1/fake_high_bit/*.mp4')
# Prepare lists for filenames and labels
video_filenames = []
video_labels = []

# Process real videos
for video in real_videos:
    video_filenames.append(os.path.basename(video))
    video_labels.append('REAL')

# Process fake videos
for video in fake_videos:
    video_filenames.append(os.path.basename(video))
    video_labels.append('FAKE')

# Create a DataFrame
data = {
    'Filename': video_filenames,
    'Type': video_labels
}

df = pd.DataFrame(data)

# Export to CSV
output_csv = '/content/drive/MyDrive/code-files/High_1024_bitrate_AV1/video_labels_high.csv'
df.to_csv(output_csv, index=False)

print(f"CSV file has been saved to {output_csv}")
```

Figure 6: Data loading and labelling

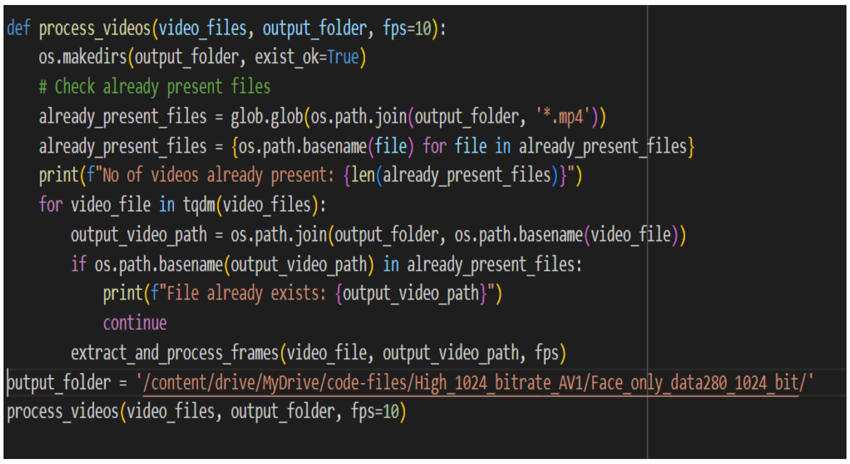
- **Extracting frames and face cropping:** Figures 7 and 8 show the function that stores the cropped face-only videos into the defined output_folder path. Also checks if the files already exist.



```python
def extract_and_process_frames(video_path, output_video_path, fps=10):
    # Create a temporary directory to hold frames (will be automatically deleted)
    with tempfile.TemporaryDirectory() as temp_dir:
        # Extract frames using FFmpeg
        command = [
            'ffmpeg',
            '-i', video_path,
            '-vf', f'fps={fps}',
            f'{temp_dir}/frame_%04d.png']
        subprocess.run(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        # Initialize video writer
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = None
        for frame_file in tqdm(sorted(os.listdir(temp_dir))):
            frame_path = os.path.join(temp_dir, frame_file)
            frame = cv2.imread(frame_path)
            # Detect faces
            face_locations = face_recognition.face_locations(frame)
            if face_locations:
                top, right, bottom, left = face_locations[0]  # Assuming one face per frame for simplicity
                face = frame[top:bottom, left:right]
                face_resized = cv2.resize(face, (224, 224))
                if out is None:
                    # Initialize video writer with the correct size
                    out = cv2.VideoWriter(output_video_path, fourcc, fps, (224, 224))
                # Write the processed frame to the output video
                out.write(face_resized)
        if out:
            out.release()
```

Figure 7: Extracting frames and face cropping

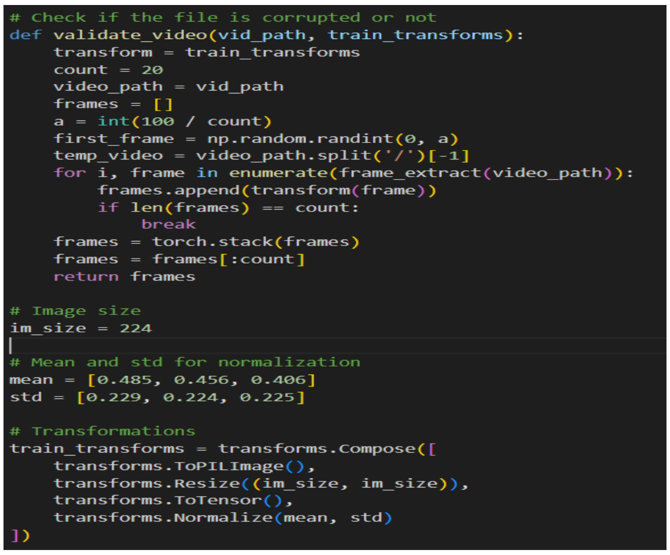
- **Checking corrupted videos and resizing:** It is important to filter out any corrupted videos in advance so that it doesn't affect model performance. This



```python
def process_videos(video_files, output_folder, fps=10):
    os.makedirs(output_folder, exist_ok=True)
    # Check already present files
    already_present_files = glob.glob(os.path.join(output_folder, '*.mp4'))
    already_present_files = {os.path.basename(file) for file in already_present_files}
    print(f"No of videos already present: {len(already_present_files)}")
    for video_file in tqdm(video_files):
        output_video_path = os.path.join(output_folder, os.path.basename(video_file))
        if os.path.basename(output_video_path) in already_present_files:
            print(f"File already exists: {output_video_path}")
            continue
        extract_and_process_frames(video_file, output_video_path, fps)
output_folder = '/content/drive/MyDrive/code-files/High_1024_bitrate_AV1/Face_only_data280_1024_bit/'
process_videos(video_files, output_folder, fps=10)
```

Figure 8: Store extracted frames

function in Figure 9 performs such investigations of corrupted videos, and once the videos are validated, they're resized and normalized to 224*224 size for EfficientNet-B0.

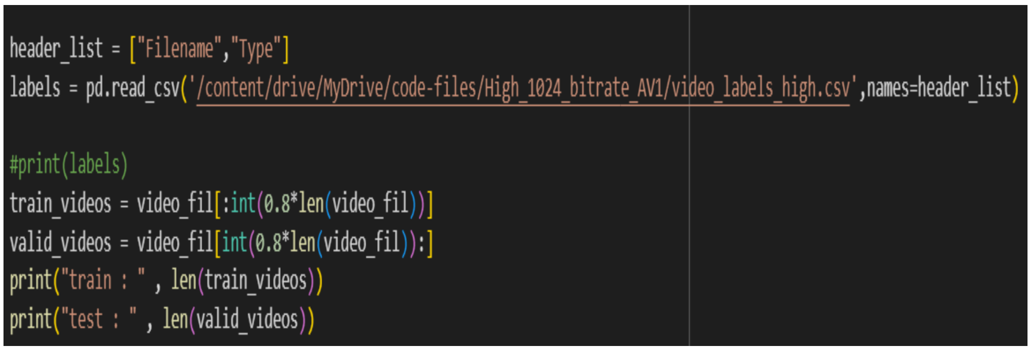


```python
# Check if the file is corrupted or not
def validate_video(vid_path, train_transforms):
    transform = train_transforms
    count = 20
    video_path = vid_path
    frames = []
    a = int(100 / count)
    first_frame = np.random.randint(0, a)
    temp_video = video_path.split('/')[-1]
    for i, frame in enumerate(frame_extract(video_path)):
        frames.append(transform(frame))
        if len(frames) == count:
            break
    frames = torch.stack(frames)
    frames = frames[:count]
    return frames

# Image size
im_size = 224

# Mean and std for normalization
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]

# Transformations
train_transforms = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((im_size, im_size)),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])
```

Figure 9: Checking corrupted videos and resizing

4 Model building layer:

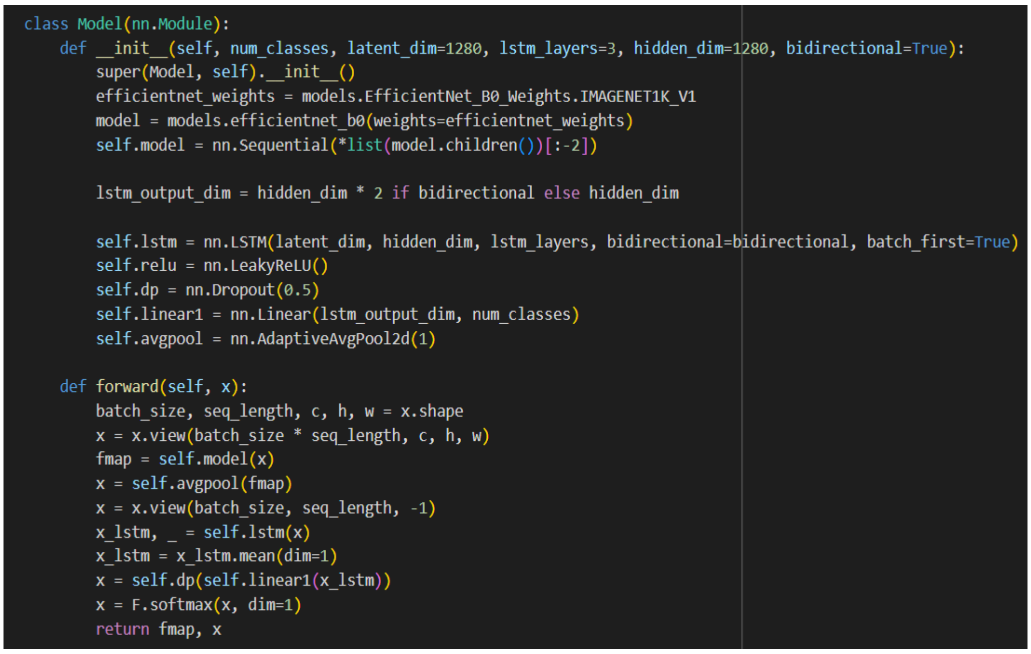
- **Training and testing data:** In this section, the pre-processed only face videos are used as the input for the model, where, firstly, the videos are split into train and test. Here, the data is split into an 80:20 ratio, i.e., 80 percent for training the models and 20 percent for testing the models. Figure 10 shows the code for splitting the data using a csv file that was exported after data labelling.
- **Creating model function:** After splitting the data into training and testing, the model is defined in this step. Firstly, using the torchvision library, the pretrained



```python
header_list = ["Filename","Type"]
labels = pd.read_csv('/content/drive/MyDrive/code-files/High_1024_bitrate_AV1/video_labels_high.csv',names=header_list)

#print(labels)
train_videos = video_fil[:int(0.8*len(video_fil))]
valid_videos = video_fil[int(0.8*len(video_fil)):]
print("train : " , len(train_videos))
print("test : " , len(valid_videos))
```

Figure 10: Split into train and test data

model EfficientNet-B0 is loaded at the initial layer, and secondly, using sequential(), a bidirectional LSTM model is connected to the EfficientNet-B0 model's output layer with a dropout layer of 0.5 percent. Lastly, with the help of the SoftMax activation function, the probability of the prediction of the model is calculated.

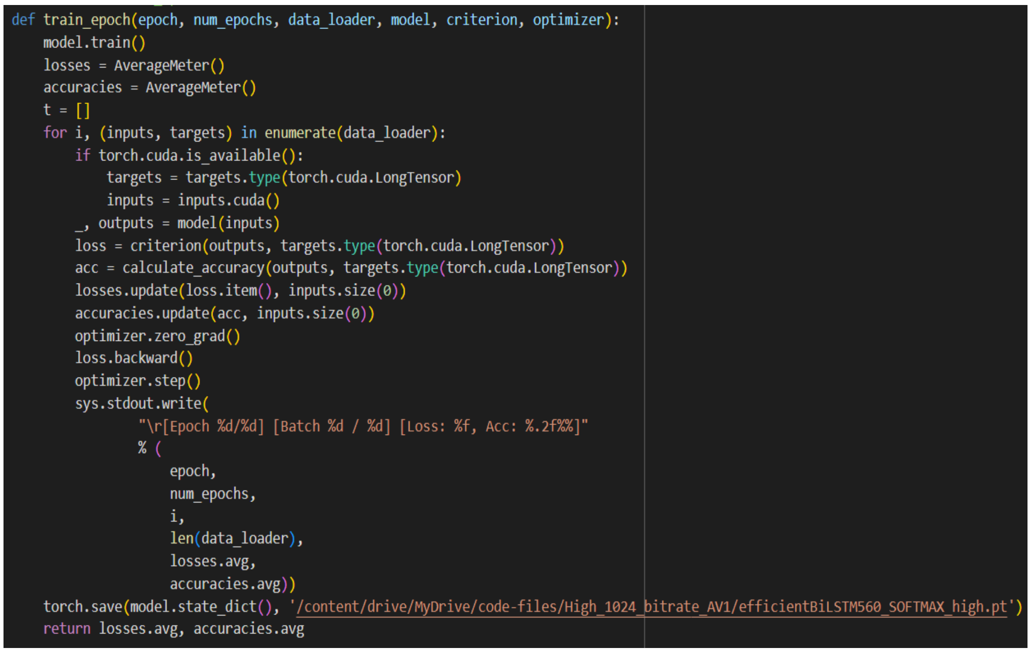


```python
class Model(nn.Module):
    def __init__(self, num_classes, latent_dim=1280, lstm_layers=3, hidden_dim=1280, bidirectional=True):
        super(Model, self).__init__()
        efficientnet_weights = models.EfficientNet_B0_Weights.IMAGENET1K_V1
        model = models.efficientnet_b0(weights=efficientnet_weights)
        self.model = nn.Sequential(*list(model.children())[:-2])

        lstm_output_dim = hidden_dim * 2 if bidirectional else hidden_dim

        self.lstm = nn.LSTM(latent_dim, hidden_dim, lstm_layers, bidirectional=bidirectional, batch_first=True)
        self.relu = nn.LeakyReLU()
        self.dp = nn.Dropout(0.5)
        self.linear1 = nn.Linear(lstm_output_dim, num_classes)
        self.avgpool = nn.AdaptiveAvgPool2d(1)

    def forward(self, x):
        batch_size, seq_length, c, h, w = x.shape
        x = x.view(batch_size * seq_length, c, h, w)
        fmap = self.model(x)
        x = self.avgpool(fmap)
        x = x.view(batch_size, seq_length, -1)
        x_lstm, _ = self.lstm(x)
        x_lstm = x_lstm.mean(dim=1)
        x = self.dp(self.linear1(x_lstm))
        x = F.softmax(x, dim=1)
        return fmap, x
```

Figure 11: Modeling

5 Evaluation and reporting layer

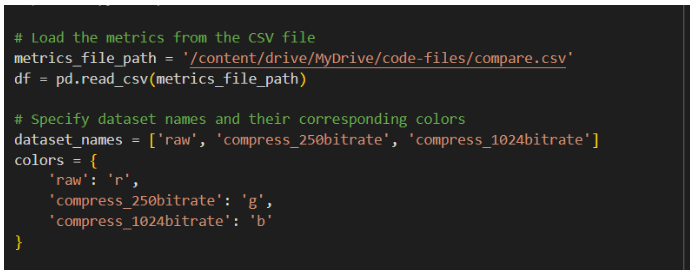
Once the model is trained based on parameters such as the number of epochs, delay weight, batch size, early stopping, and learning rate, it is evaluated and fine-tuned accordingly. Once the fine-tuned model is achieved, it is exported into the defined path as "efficientBiLSTM560_SOFTMAX_high.pt." By using the sklearn.metrics library, model evaluation is carried out, where accuracy, confusion matrix, recall, F1-score, precision, and cross-entropy loss are calculated. Figure 12 represents the code for training and saving the trained model for testing new videos.



```
def train_epoch(epoch, num_epochs, data_loader, model, criterion, optimizer):
    model.train()
    losses = AverageMeter()
    accuracies = AverageMeter()
    t = []
    for i, (inputs, targets) in enumerate(data_loader):
        if torch.cuda.is_available():
            targets = targets.type(torch.cuda.LongTensor)
            inputs = inputs.cuda()
        _, outputs = model(inputs)
        loss = criterion(outputs, targets.type(torch.cuda.LongTensor))
        acc = calculate_accuracy(outputs, targets.type(torch.cuda.LongTensor))
        losses.update(loss.item(), inputs.size(0))
        accuracies.update(acc, inputs.size(0))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        sys.stdout.write(
                "\r[Epoch %d/%d] [Batch %d / %d] [Loss: %f, Acc: %.2f%%]"
                % (
                    epoch,
                    num_epochs,
                    i,
                    len(data_loader),
                    losses.avg,
                    accuracies.avg))
    torch.save(model.state_dict(), '/content/drive/MyDrive/code-files/High_1024_bitrate_AV1/efficientBiLSTM560_SOFTMAX_high.pt')
    return losses.avg, accuracies.avg
```

Figure 12: Saving trained model

- **Storing the model evaluation to csv:** This project research question aims in performing comparison studies of model performances at various bitrates of compressed vidoes, So this step helps in storing the achieved results of "Raw", "Low-bitrate" and "High-bitrate" into a single csv file. Figure is the code snippet used for storing.

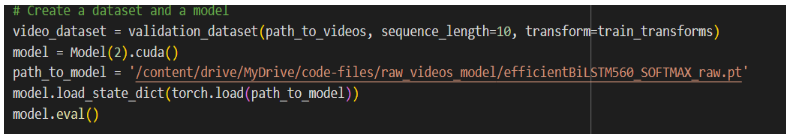


```
# Load the metrics from the CSV file
metrics_file_path = '/content/drive/MyDrive/code-files/compare.csv'
df = pd.read_csv(metrics_file_path)

# Specify dataset names and their corresponding colors
dataset_names = ['raw', 'compress_250bitrate', 'compress_1024bitrate']
colors = {
    'raw': 'r',
    'compress_250bitrate': 'g',
    'compress_1024bitrate': 'b'
}
```

Figure 13: Load the trained model

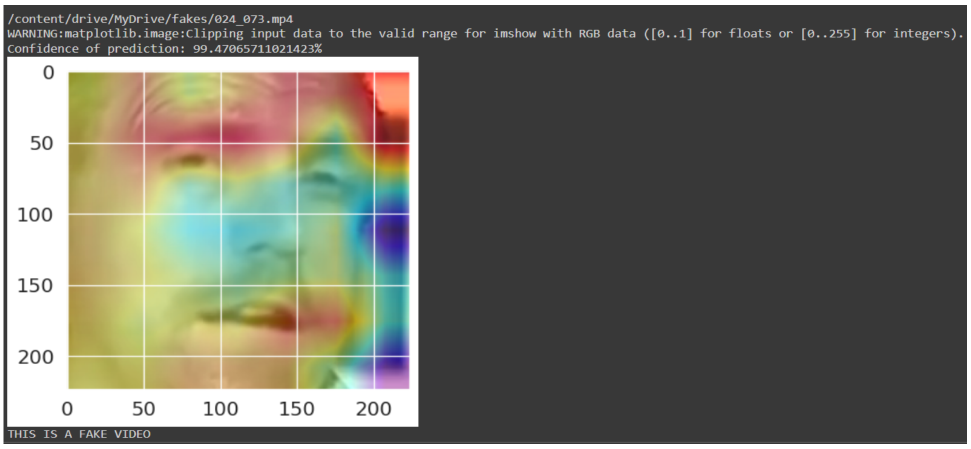
All the above steps can be use for obtaining the model performance of low and high bitrate compressed videos. Apart from this, for raw videos model performance can be achieved by skipping the Section 3.1 of compression using handbrake application.

- **Loading trained model and testing new videos:** The trained model is then loaded with the help of the Torch library to predict whether the new video is real or fake. Figure 14 shows the model loading, and Figure 15 shows the final prediction result.



```
# Create a dataset and a model
video_dataset = validation_dataset(path_to_videos, sequence_length=10, transform=train_transforms)
model = Model(2).cuda()
path_to_model = '/content/drive/MyDrive/code-files/raw_videos_model/efficientBiLSTM560_SOFTMAX_raw.pt'
model.load_state_dict(torch.load(path_to_model))
model.eval()
```

Figure 14: Load the trained model



```
/content/drive/MyDrive/fakes/024_073.mp4
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Confidence of prediction: 99.47065711021423%
```

```
THIS IS A FAKE VIDEO
```

Figure 15: Predicted results with the probability

- **Report generation:** After predicting the new test video, an additional report is generated to show the traditional methods of predicting real or fake videos, which consist of wrapping and unwrapping phases, noise level, entropy value, and blur value.

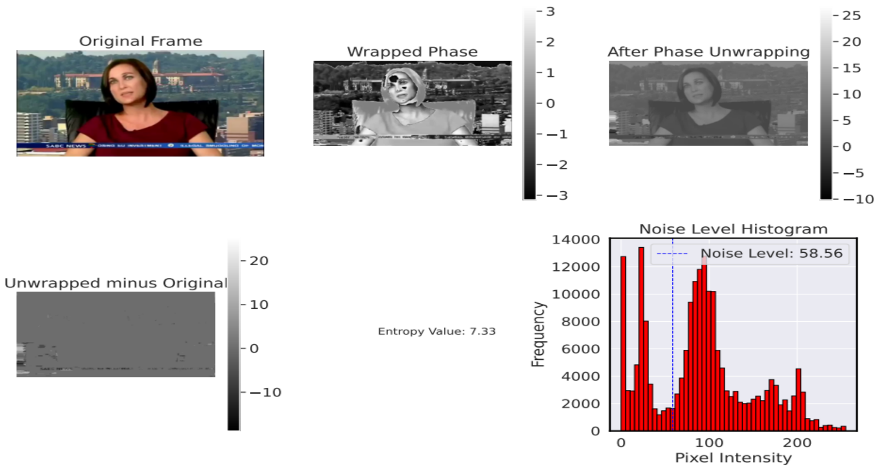


Figure 16: Report generation

- **IMPORTANT NOTE: The Google Drive links are provided along with the code files which includes the pre-processed face-only videos of each model and compressed videos. The pre-processing of videos requires around 5-8 hours (which can differ based on the processing power used).**