Project Titled

“Detection of Deepfakes using Temporal Features and Frequency Analysis”

Submitted in Partial Fulfillment of
The requirement of the degree for the
Bachelor’s Degree (Information Technology)

Submitted by

Pratim Ugale - 171080033
Jainil Shah - 171080077
Suvidh Shah - 171080027
Prathmesh Bendal - 171080076

Under the guidance of
Dr. M. M. Chandane

DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION TECHNOLOGY
VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE
(An Autonomous Institute Affiliated to Mumbai University)
(Central Technological Institute, Maharashtra State)
Matunga, MUMBAI - 400019
A.Y. 2020-2021

DECLARATION OF CANDIDATES

We state that work embodied in this Project entitled “DETECTION OF DEEPFAKES USING TEMPORAL FEATURES AND FREQUENCY ANALYSIS" forms our own contribution of work under the guidance Dr. M.M Chandane at the Department of Computer Engineering and Information Technology, Veermata Jijabai Technological Institute. The report reflects the work done during the period of candidature but may include related preliminary material provided that it has not contributed to an award of previous degree. No part of this work has been used by us for the requirement of another degree except where explicitly stated in the body of the text and the attached statement.

Signature of Candidates:

Pratim Ugale________________________
(171080033)

Jainil Shah ________________________
(171080077)

Suvidh Shah ________________________
(171080027)

Prathmesh Bendal ________________________
(171080076)

Date: ________________________

ACKNOWLEDGEMENT

We would like to take this opportunity to express our sincere gratitude towards our mentor and project guide, **Dr. M. M. Chandane** for his constant support and encouragement.

We would like to thank him for devoting his time and effort to this project. His words of encouragement and wisdom have kept us on track throughout this project and he has helped us achieve our results.

We would also like to take this opportunity to thank our parents for bearing with us and taking care of us while we set about this journey. They have been the pillars of support that helped us work on this project with dedication and spirit.

Finally, we would like to express our gratitude towards our alma mater, Veermata Jijabai Technological Institute, for the extremely conducive environment it has provided and the infrastructure it has made available to students like us to pursue our projects also during the Covid Pandemic.

REPORT ORIENTATION

The report will comprise five chapters with different content and scenarios providing the complete details about the project. The report is completed in such a way that it first provides the background knowledge about the project and then gives the thorough details about it. It also has methodology and the conclusion. The different chapters of the report are as follows.

Chapter 1: Background and Motivation

This chapter will provide introduction to the project and motivation for performing it. This chapter gives a review of the current deepfake generation techniques, followed by the problem statement, objective and goals. It will also contain the flaws in deepfakes that our technique will look to exploit.

Chapter 2: Literature Review

This chapter will provide literature studied for the project. The focus would be on the technology going to be used.

Chapter 3: System Analysis and Design

This chapter is divided into System analysis and System Design. In this chapter we provide system architecture and analyze it. We also focus on detailed information of the data sets and also have a Gantt chart for the project timeline.

Chapter 4: Implementation Details

This chapter describes the implementation and results obtained. All the models trained are compared with each other to find out which one works better.

Chapter 5: Conclusion

This chapter gives the conclusion and the issues, improvements and future scope.

ABSTRACT

In recent years, deep learning based software tools have emerged that make it easy for anyone with a basic computer system to create believable face swaps in videos and such manipulated pieces of media, called “Deepfakes” result in the spread of misinformation, hoaxes, and abusive content. These realistic fake videos may create political distress, can be used to blackmail someone or fake terrorism events. ***Convolutional Autoencoders*** and ***Generative Adversarial Networks (GAN)*** models have made tampering images and videos, which used to be reserved to highly-trained professionals, a broadly accessible operation within reach of almost any individual with a computer.

New deepfake videos have become so realistic that it is almost impossible to differentiate a real and fake video by the naked eye. This project hence uses a CNN+LSTM architecture to exploit the temporal inconsistencies in between the frames. This is because of the nature of the way that deepfakes are created-frame by frame. There is a high probability that a current frame might have features that are inconsistent with previous frames. The CNN is used as a feature extractor by taking a pretrained model with the last layer removed and the LSTM is used to detect the sequence inconsistencies. Different pre-trained CNNs are tried to compare which one generalizes better and performs better classification.

We further propose a model in which the CNN feature vectors are concatenated with Azimuthally averaged frequency domain vectors to check if the accuracy can be improved. This is because some GAN generators may leave a frequency fingerprint on the image which is not visible in the spatial domain. We develop a model for this classification but find out that the DFDC and Celeb-df datasets don’t result in an increased accuracy with this method as reasoned out in the report.

The detection of DeepFake videos is a binary classification problem where classifiers are used to classify between authentic videos and tampered ones.

Table of Contents

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

1.1 Background:

Introduction to DeepFakes

The term “DeepFake" refers to all those multimedia contents (specially video) scientifically altered or created by using Machine Learning Generative models. Various examples of Deepfakes involving celebrities are easily discoverable on the web. They represent a big problem especially when mass media are involved.

Creating a DeepFake

1. **GAN**: A generative adversarial network (GAN)[1] is a class of machine learning framework designed in 2014 in which two neural networks contest with each other in a form of a zero-sum game, where one agent's gain is another agent's loss. The idea of a GAN is based on the "indirect" training through the discriminator, which itself is also being updated dynamically. This means that the generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner. The generative network generates candidates while the discriminative network evaluates them. The contest operates in terms of data distributions. Typically, the generative network learns to map from a latent space to a data distribution of interest, while the discriminative network distinguishes candidates produced by the generator from the true data distribution. The generative network's training objective is to increase the error rate of the discriminative network. GANs[1] can be used to generate unique, realistic profile photos of people who do not exist, in order to automate creation of fake social media profiles. The Generator is a Deconvolutional Neural Network [2](Tensorflow uses Conv2DTranspose() to increase the dimension/go in the opposite direction of Conv2D()). This upsampling process leaves some convolutional traces (or frequency fingerprints) which some detection methods look forward to identifying.

   The discriminator is also trained when the generator generates images. So as training progresses the generator gets better at generating fake images and the discriminator gets better at detecting fake images as it was also already trained on the training dataset. Hence there is an “adversarial” relationship between the generator and discriminator which is the key component of GANs.

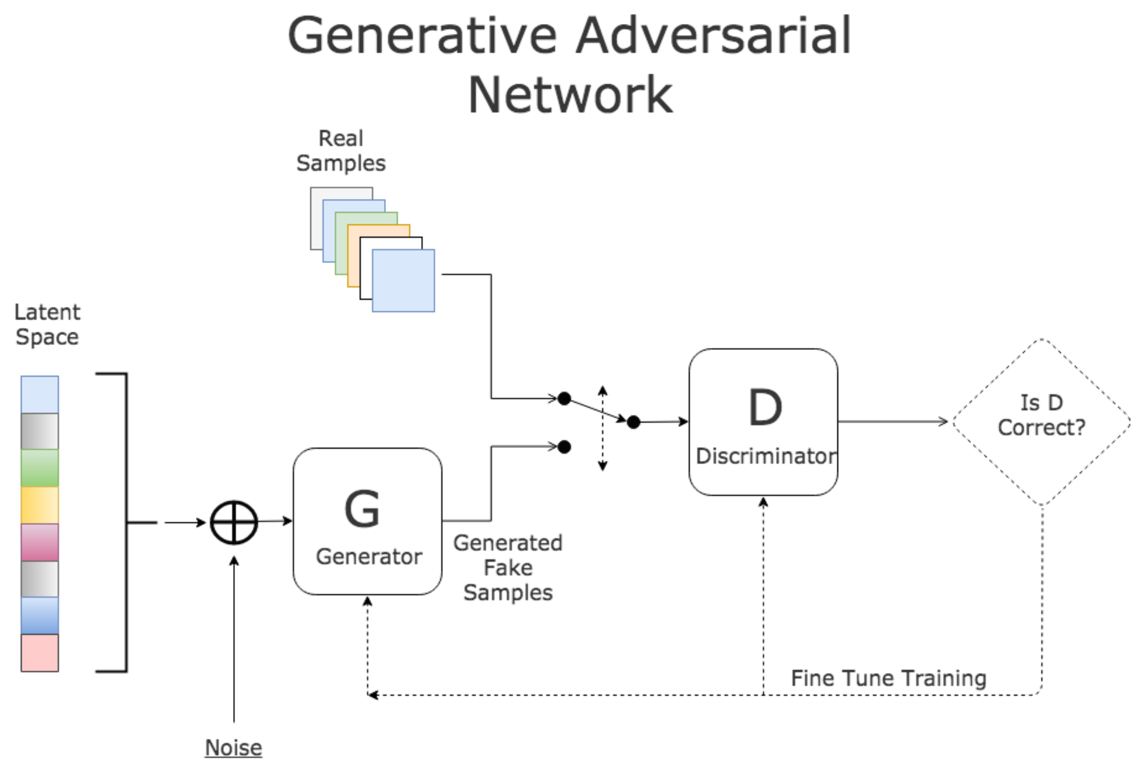


Figure 1.1

2. **Autoencoders:** An Autoencoder[3] is a deep neural network that learns how to take an input, compress it down into a small representation or encoding, and then to regenerate the original input from this encoding. In this standard autoencoder setup, the network is trying to learn how to create an encoding (the bits in the middle), from which it can regenerate the original image. With enough data, it easily learns how to do this.
   Putting a bottleneck in the middle forces the network to recreate these images instead of just returning what it sees.

   Deepfake goes further by having one encoder to compress a face into an encoding, and two decoders, one to turn it back into person A, and the other to person B (this produces face swap).

   Step 1 shown below builds a common encoder to encode the latent factors of pictures for two different persons. In steps 2 and 3, it builds two separate decoders to reconstruct the first and second photo respectively. To reconstruct the image correctly, the encoder must capture all the variants in a person's photos, i.e. the latent factors that apprehend information like the pose, the expression, illumination, etc

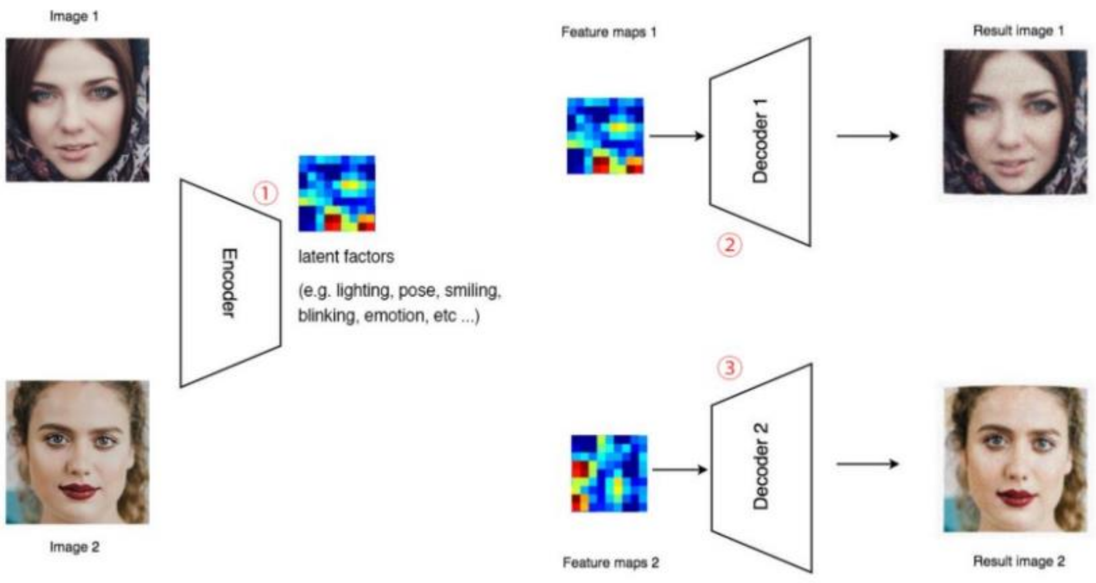


Figure 1.2

For creating a Deepfake, Person A’s faces in a video are replaced with Person B. The encoder will capture the latent factors of A’s face in the video and render it with B’s decoder. Therefore, the rendered B face will have the same pose, lighting, and emotional expression as the original video.

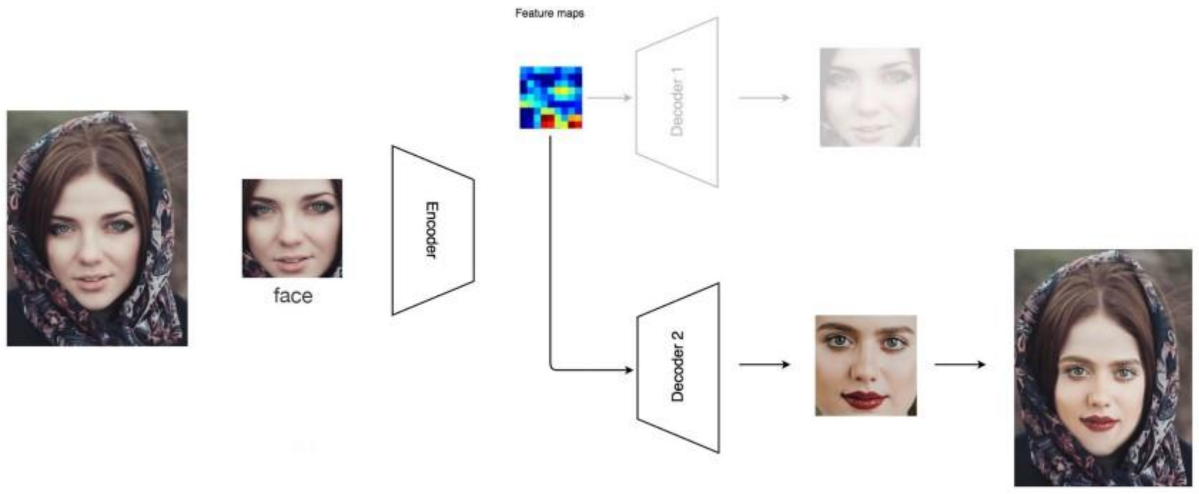


Figure 1.3

By adding adversarial loss and perceptual loss implemented in VGGFace[4] to the encoder-decoder architecture, an improved version of deepfakes based on the generative adversarial network (GAN), i.e. faceswap-GAN[5], was proposed.

1.2 Motivation

Deepfakes[6] have become popular due to the quality of tampered videos and also the easy-to-use ability of their applications to a wide range of users with various computer skills from professional to novice.
However there are not many easy-to-use applications that help the layman person to identify whether a video is pristine or manipulated.

For example, a voice deepfake was used to scam a CEO out of $243,000 according to Forbes.
These false content generation and modification technologies may affect the opinion of the public. Also it poses a challenge to the safeguarding of human rights given that deepfakes may be used maliciously as a source of misinformation, manipulation, persuasion and harassment to women. Identifying manipulated media is a demanding and rapidly evolving challenge that would require collaborations across the entire tech industry and beyond.

1.3 Problem Statement:

Deepfake detection techniques have been proposed since Deepfakes were started to be made.
These techniques have been proved to be effective at exploiting specific characteristics of manipulated videos. Some of the detection techniques like inconsistent eye blinking have been overcome by adding more largely available training data and manually or automatically removing bad frames.

A key concern of all of the methods is that they can be easily learnt by the GAN[1]. By incorporating them into the GAN's[1] discriminator, the discriminator will learn those characteristics of Deepfakes and the generator will fine-tune itself to learn a countermeasure for any differentiable characteristic/forensic.
One of the central unsolved challenges of detecting deepfakes is that it is hard to generalize from known examples to unfamiliar instances.

In June 2020, the results of Facebook's million dollar DeepFake Detection Challenge[7] which was conducted on Kaggle were announced. The top-performing model on the public data set achieved 82.56% accuracy. But when evaluated against a black box data set, the highestperforming model achieved an accuracy of 65.18%. This motivates further research on the generalisation ability of the fake detectors against unseen conditions.

Moreover, none of the top-performing solutions used digital forensics techniques or characteristics derived from the image creation process, according to Facebook. This suggests that either techniques that operate at a pixel level aren't useful for this task or they aren't currently in widespread use among those who entered the DFDC[7].

1.4 Goals:

In our work we will be using frequency analysis for developing a model to detect Deepfake images. We will extend this method and apply it to videos frame by frame after detecting and cropping faces to check if the method gives good results on mainstream Deepfake video datasets like Celeb-df[8] and Google DFD[9]. Temporal features will be used to detect manipulated videos. A front end web interface will be provided which will allow a user to do analysis on an image or a video using our implementation of the techniques.

The goals of our project can be summarized as:

1. To use RNNs to detect temporal inconsistencies between the frames of a video once the features have been calculated using a CNN. This is the main method that we intend to use for detecting deepfake videos.
2. To use Azimuthally averaged frequency spectrum features along with the above CNN features on videos and check its accuracy on mainstream video datasets like Celeb-df[8] and DFD[9] to find out if the accuracy improves.
3. Comparing the results of [12]LSTM+RNN architectures with different tuning parameters to find out which model generalizes better so that the web-interface would serve that model.
4. To provide an easy to use front-end application which will input videos from a user, detect faces, extract features and run inference on the trained model to find out whether the video is pristine or manipulated.

CHAPTER 2
LITERATURE SURVEY

2.1 Paper 1: Deepfake Video Detection Using Recurrent Neural Networks[6]:

- David Guera, Edward J. Delp

The main reason for using this technique is that different camera angles, differences in lightning conditions or the use of different codecs makes it difficult for autoencoders to produce realistic faces under all conditions. This usually leads to swapped faces that are visually inconsistent with the rest of the scene. This is a frame level scene inconsistency that is exploited with this approach. Since while generating deepfakes only faces are swapped, the encoder is not aware of other scene information due to which boundary effects are created between the new face and the rest of the frame.

Because auto encoder is used frame by frame it is completely unaware of previous frames.
This causes multiple anomalies in the face region eg: blinking of eyes.

This paper proposes a temporal-aware pipeline as opposed to single convolutions on frames. It consists of a convolutional neural network (ImageNet CNN) which is pretrained with the last layer removed, to extract frame-level features which are then used to train a recurrent neural network (RNN) that learns to classify if a video has been subject to manipulation or not based on the inconsistencies between the frames. The architecture is explained below:

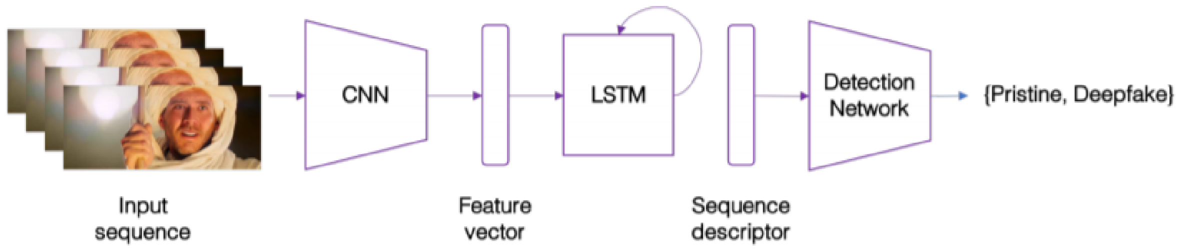


Figure 2.1: CNN-LSTM architecture

CNN: The model architecture uses the InceptionV3[11] CNN with the fully-connected layer at the top of the network removed to directly output a deep representation of each frame using the ImageNet pre-trained model. 2048-dimensional feature vectors after the last pooling layers are then used as the sequential LSTM[12] input.

RNN: Followed by the CNN, this is takes the CNN input and inserts it into a 2048-wide LSTM[12] unit with 0.5 chance of dropout.

Fully Connected Layer: The LSTM[12] is followed by a 512 fully-connected layer with 0.5 chance of dropout. This layer is needed to make the final predictions on the video.

Softmax Layer: To compute the probabilities of the frame sequence being either pristine or deepfake.

LSTM 's and GRU's were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information.

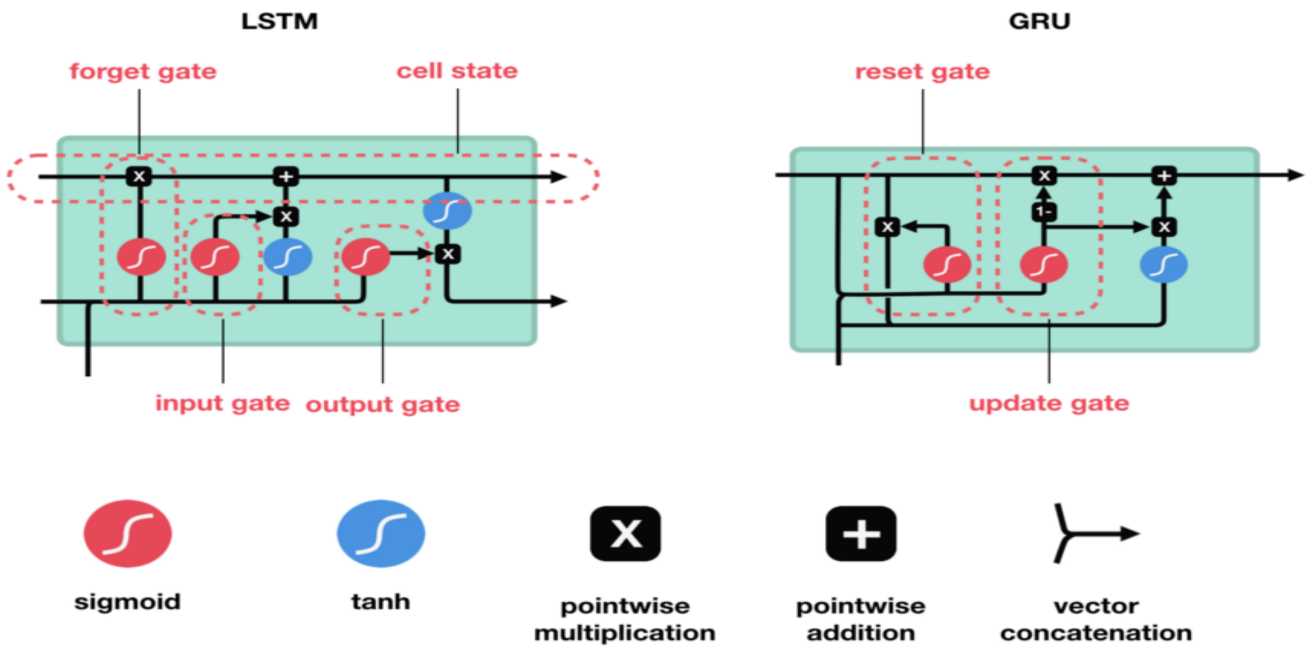


Figure 2.2: LSTM and GRUs

These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions. This is why they are used to process a sequence of video frames in our case.

Dataset Used: HOHA (Proprietary Dataset)[13]

Accuracy on Dataset: 97%

Conclusion: The model/pipeline finds out the inconsistencies in a series of frames and is practically found to overfit datasets due to which frame augmentations are required.
This can be further improved by using a newer preprocessing technique using MTCNN[14] which will do face recognition for one frame and use the same bounding box coordinates for a number of frames after it. This will save computational time and improve accuracy. This technique was used in a winning Kaggle solution.

2.2 Paper 2: Unmasking DeepFakes with Simple Features[10]:

- Ricard Durall, Margret Keuper, Franz-Josef Pfreundt, Janis Keuper.

There are many deep learning methods that detect deepfakes that show good performance. But a key concern is that all these methods can be easily learnt by the GAN[1]. In particular, by incorporating them into the GAN's discriminator, the generator can be fine-tuned to learn a countermeasure for any differentiable forensic.

This paper[10] says that GAN generated images fail at reproducing spectral properties during upsampling. By only using the 1-D power spectrum features and a simple (SVM and Logistic Regression) classifier, they differentiate between real and generated images. Generative deep neural networks rely on convolution based on sampling methods to produce images. The paper investigates techniques to try to find out what effects these upsampling methods have on the spectral properties.

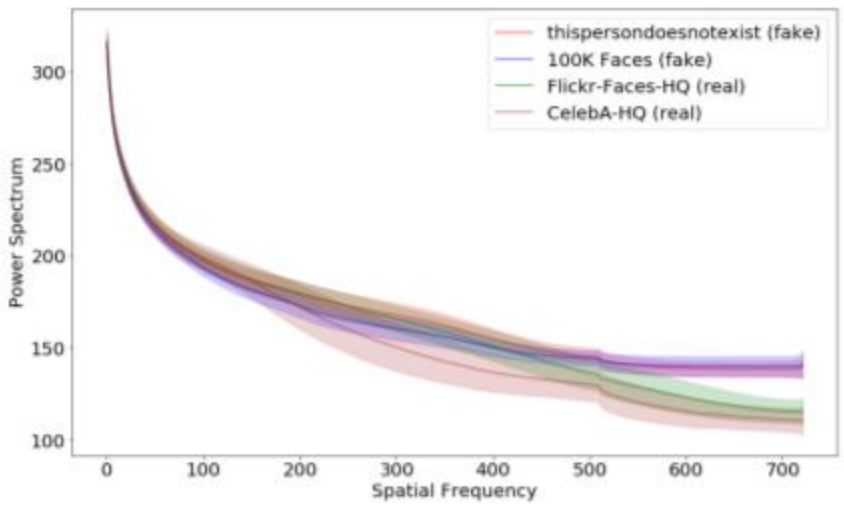


Figure 2.3: Power Spectrum differences between Real and Fake Images

Fourier Transform: A Fourier transform[15] is generally used to decompose a wave function into its constituent sine wave frequencies. This is done by winding the wave around a circle, then gradually increasing the winding frequency and noticing the centre of mass of the wave. When the winding frequency equals the wave frequency, the centre of mass shifts far towards the right. Thus on plotting the x-coordinate of the center of mass, a spike is observed when the frequencies match.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

Fourier Transform[15] decomposes an image into its sine and cosine components. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

For a square image of size N×N, the two-dimensional DFT is given by:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-\iota 2\pi \left(\frac{ki}{N} + \frac{lj}{N}\right)}$$

Where *f(a,b)* is the image in the spatial domain and the exponential term is the basis function corresponding to each point *F(k,l)* in the Fourier space.

The number of frequencies corresponds to the number of pixels in the spatial domain image.
The Fourier Transform[15] produces a complex number valued output image which can be displayed with two images, either with the *real* and *imaginary* part or with *magnitude* and *phase*. In most implementations the Fourier image is shifted in such a way that the DC-value (*i.e.* the image mean) *F(0,0)* is displayed in the center of the image. The further away from the center an image point is, the higher is its corresponding frequency. A line on the logarithmic transformation corresponds to the direction on the spatial domain in which the frequency is observed.

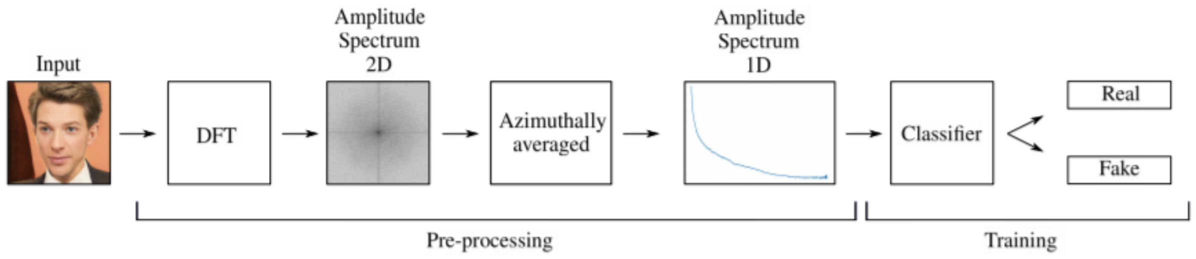


Figure 2.4: Model pipeline

Technique: Convert preprocessed image of face to grayscale, take Discrete Fourier Transform of the Image and represent the 2D image in 1D by computing Azimuthal Average.
Azimuthal averaging is applied to compute a robust 1D representation of the FFT power spectrum. (Radial Average). It is a method for feature extraction in which we do compression, gathering and averaging similar frequency components into a vector of features. Thus, the amount of features is reduced without losing relevant information. It is a more robust representation of the input. Then use supervised techniques (SVM[16] / Logistic Regression[17]) or unsupervised techniques (K means clustering[18]) to classify the data.

Conclusion: Is better to use mainly for Images which have been tampered with and not Videos. Since the classification is based only on frame image data, the temporal information is lost.
For image classification, it is basically a high-frequency component analysis.
Drawback for Images: Low-resolution content is harder to identify since the available frequency spectrum is much smaller.

2.3 Paper 3: Deepfakes Detection with Automatic Face Weighting[19]:

- Daniel Mas Montserrat, Hanxiang Hao, S. K. Yarlagadda, Sriram Baireddy, Ruiting Shao Janos Horvath, Emily Bartusiak, Justin Yang, David Guera, Fengqing Zhu, Edward J. Delp

Reasons for using this technique:
When performing deepfake detection on videos it is done by frame by frame basis and at the end results of each frame are combined or average is taken, this approach of taking direct average has the drawback that frames which do not have face region or blurry frame have same contribution to final result as compared to other frames - this produces inconsistent results. This paper describes an approach to solve the above problem an additional automatic face weighing layer is added which assigns weights to each frame according to its quality which boost the performance
Their pipeline has the following steps:

- Face detection and cropping using MTCNN[14] across multiple frames, because all visual manipulations are located within face regions.
- Feature extraction with EfficientNet[20] on face regions (Newer EfficientNet has performed better than ImageNet[21] using lesser parameters).
- Logit prediction and weighted aggregation using the automated face weighting (AFW) which is similar to attention mechanism[22].
- Final prediction with a Gated Recurrent Unit.

MTCNN[14] is a face detection tool which detects faces and facial landmarks on images.
It is one of the most popular and accurate face detection tools available today, most face extraction (specially for DeepFakes) pipelines use MTCNN to extract the faces from a video.

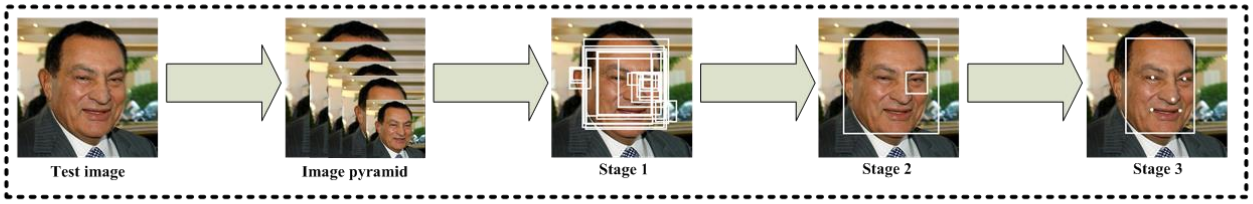


Figure 2.5: MTCNN Working

It is a Multi-Task cascaded three stage architecture that can simultaneously produce face bounding boxes and facial landmarks.

An image pyramid is generated by resizing the input image to different scales.

At each stage, a different subnetwork is used to refine the estimate from the previous stage in a coarse-to-fine manner.

Methodology:

- They crop and resize the regions with detected faces to 224*224 pixels (from HD)
- They use the EfficientNet-b5[20] network as it provides a good trade-off between network parameters and classification accuracy. This network has been designed using neural architecture search (NAS) algorithms, resulting in a network that is both compact and

accurate. The network has outperformed previous state-of-the-art approaches in datasets such as ImageNet[21] while having fewer parameters. (* Neural Architecture Search (NAS) is the process of automating architecture engineering i.e. finding the design of our machine learning model, where we need to provide a NAS system with a dataset and a task (classification, regression, etc), and it will give us the architecture. This architecture will perform best among all other architecture for that given task when trained by the dataset provided. This is how complicated CNN[32] architectures are created.)

- They use the ArcFace loss (additive angular margin loss)[23] which reduces the intra class difference and enlarges the inter-class difference between the learnt classification features. This allows better discrimination between real and fake faces.
- The extracted features are used to predict a logit and a weight (confidence score) at each face region within every input frame.
- The probability $\boldsymbol{p_w}$ of a video being false is estimated by performing a weighted average of logits and then applying a softmax layer.

  This allows them to obtain a prediction on a video level by using features from multiple face regions and multiple frames.
- Finally they use a Gated Recurrent Unit (GRU). They have input as extracted features from EfficientNet[20], and the logits, weights and the prediction of the automatic face weighting layer. This is to capture temporal information from short and long sequences.
- The GRU is composed of 3 stacked bi-directional layers and a uni-directional layer with a hidden layer with dimension 512. The output of the last layer of the GRU is mapped through a linear layer and a Sigmoid function to estimate a final probability $\boldsymbol{p_{RNN}}$ of the video being manipulated.

2.4 Table for comparison of different papers studied

The above papers are the 3 main papers that we refer to while implementing the models.
The following table contains the other papers that we have studied the literature for. The table contains:
1. Description and methodology used in the paper
2. The dataset they have used their method on
3. The accuracy they obtained on their dataset in various experiments
4. Conclusion and drawbacks of using the method in other circumstances.

The above mentioned papers are also summarized in this table to compare with other techniques in a tabular manner.

Sr. No.	**Premise and Methodology**	**Dataset**	**Accuracy**	**Conclusion and Drawbacks**
1.	**In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking[24]** Many Deepfake videos have faces collected from still images. This creates an imbalance of training classes. And there is a lack of side images and images that are blinking. The trained decoder will have a hard time decoding images that should be blinking. The paper uses object detection to extract the eye area and uses the combination of CNN, LSTM[12], and fully-connected networks to determine whether the eye in a particular frame is blinking (1 if it blinks). Then they use long term properties of LSTM to see if the face in the video has a normal blinking pattern.	CEW Dataset images of eyes closed and open to train the CNN[32]. 40 videos from EBV dataset for overall LRCN Model.	LRCN shows the best performance with AUC 0.99 compared to CNN 0.98 and EAR 0.79.	This type of detection approach can be defeated easily by identifying what is missing and augmenting the data manually or automatically by feeding more data.

2.	**Deepfake Video Detection Using Recurrent Neural Networks**[6] : Proposes a temporal-aware pipeline. It consists of a convolutional neural network (CNN[32]) (pretrained with the last layer removed) to extract frame-level features which are then used to train a recurrent neural network (RNN) that learns to classify if a video has been subject to manipulation or not based on the inconsistencies between the frames. A novel solution. **CNN**: InceptionV3 with the fully-connected layer at the top of the network removed to directly output a deep representation of each frame using the ImageNet pre-trained model. 2048-dimensional feature vectors after the last pooling layers are then used as the sequential LSTM[12] input. **RNN**: 2048-wide LSTM[12] unit with 0.5 chance of dropout. **Fully Connected Layer:** LSTM is followed by a 512 fully-connected layer with 0.5 chance of dropout. **Softmax Layer:** To compute the probabilities of the frame sequence being either pristine or deepfake.	HOHA (Proprietary Dataset)	97%	The model/pipeline does **not** try to detect manipulation/blurring done in a single frame, rather it finds out the inconsistencies in a series of frames.
3.	**Unmasking DeepFakes with simple Features** [10]: While many deep learning methods that detect deepfakes show promising performance, a key concern is that all these methods can be easily learnt by the GAN. In particular, by incorporating them into the GAN's discriminator, the generator can be fine-tuned to learn a countermeasure for any differentiable forensic. **Technique**: Convert image to grayscale, take discrete **Fourier Transform** of the Image and represent the 2D image in 1D by computing **Azimuthal Average**. Azimuthal averaging is applied to compute a robust 1D representation of the FFT power spectrum. (Radial Average). It can be seen as a compression, gathering and averaging similar frequency components into a vector of features. Thus, the amount of features is reduced without losing relevant information. It is a more robust representation of the input. Then use supervised techniques (SVM / Logistic Regression) or unsupervised techniques (K means clustering) to classify the data.	Faces-HQ, Celeb-A, DFDC	Many experiments explained in paper ***SVM:*** 85% (Image), 90% (Video) ***Logistic Regression:*** 78% (Image), 81% (Video)	Is better to use mainly for Images which have been tampered with and **not Videos**. Since the classification is based only on frame image data, the temporal information is lost. For image classification, it is basically a high-frequency component analysis. **Drawback for Images**: Low-resolution content is harder to identify since the available frequency spectrum is much smaller.

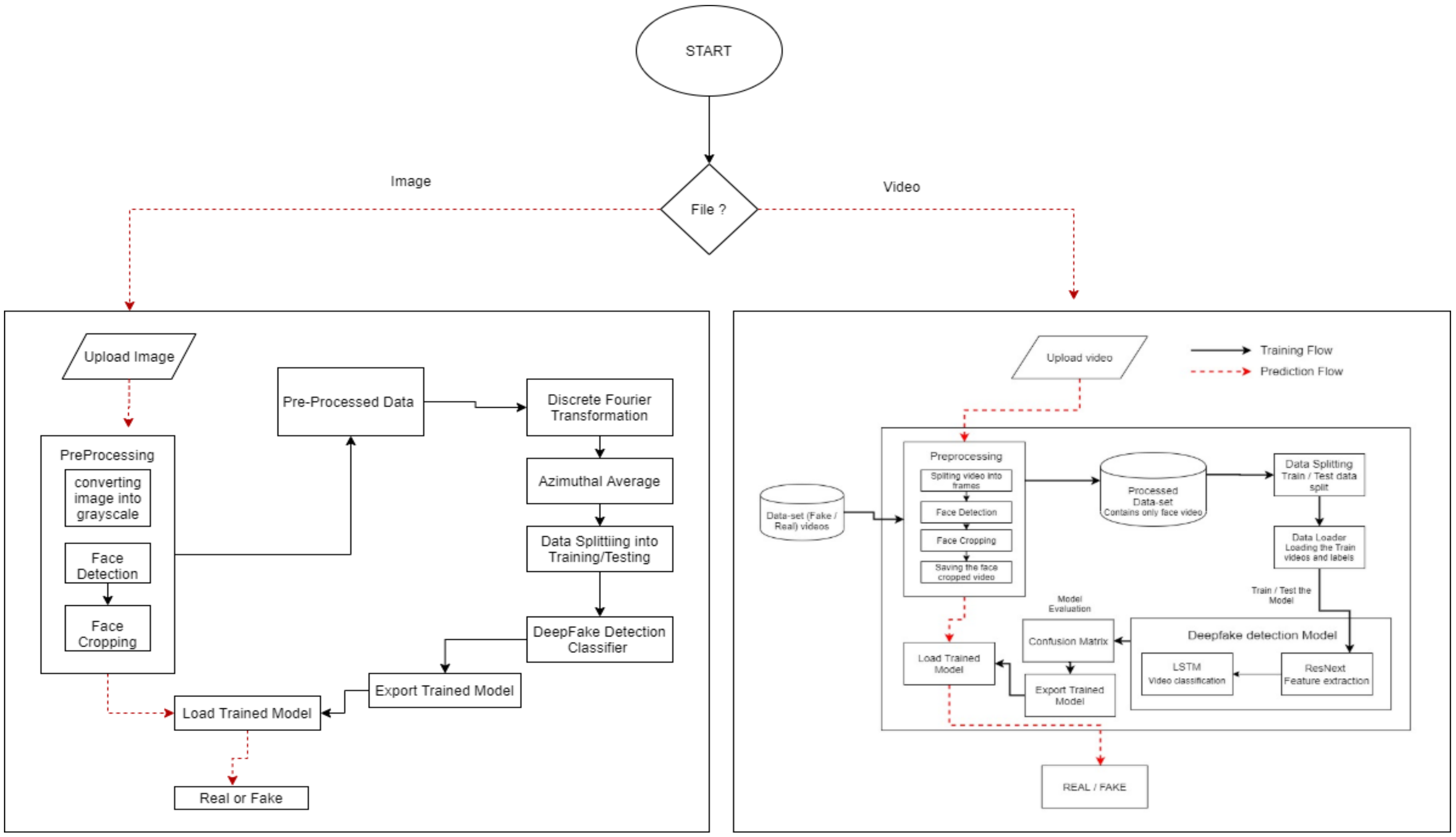
4.	**Deepfakes Detection with Automatic Face Weighting**[19] A novel model architecture that combines a Convolutional Neural Network (CNN[32]) with a Recurrent Neural Network (RNN), detecting facial manipulations in videos. The network automatically selects the most reliable frames to detect these manipulations with a weighting mechanism combined with a Gated Recurrent Unit (GRU) that provides a final probability of a video being real or being fake. **Technique**: The method has three distinct steps: (1) Face detection across multiple frames using MTCNN (2) Feature extraction with a CNN, and (3) Prediction estimation with a layer they refer to as Automatic Face Weighting (AFW) along with a Gated Recurrent Unit (GRU).	Deepfake Detection Challenge dataset **(DFDC)[7]**	0.321 log likelihood error 92.61%	Focuses on face manipulation detection and dismisses any analysis of audio content which could provide a significant improvement of detection accuracy in future work. 117 of 2275 teams (top 6%) of the public leader-board in DFDC.
5.	**Exposing DeepFake Videos By Detecting Face Warping Artifacts[25]** This proposed method is based on the observations that the current DeepFake algorithms can only generate images of limited resolutions, which are then needed to be further transformed to match the faces to be replaced in the source video. Such transforms leave certain distinctive artifacts in the resulting DeepFake Videos, which can be effectively captured by a dedicated deep neural network model.	DeepFake video dataset UADFV, DeepfakeTIMIT, VidTIMIT dataset	The VGG16, ResNet50, ResNet101 and ResNet152 models achieve AUC performance 83.3%, 97.4%, 95.4%, 93.8%, respectively	Poor detection method of deepfake with respect to multiple video compression. Currently uses predesigned network structure like resnet [30]or VGG which is not efficient.
6.	**Video Manipulation detection through ensemble of CNN[26]** The proposed method takes inspiration from the family of EfficientNet models and improves upon a recently	DFDC[7] Face Forensics	B4 FF-93.82% DFDC-87.66% B4ST	Future work will be devoted to the embedding of temporal information. As a matter of fact,

	proposed solution, investigating an ensemble of models trained using two main concepts: (i) an attention mechanism which generates a human comprehensible inference of the model, increasing the learning capability of the network at the same time; (ii) a triplet siamese training strategy which extracts deep features from data to achieve better classification performances.		FF-93.37% DFDC-86.58% B4Att FF-93.60% DFDC-86.42% B4AttST FF-92.93% DFDC-83.6%	intelligent voting schemes when more frames are analyzed at once might lead to an increased accuracy
7.	**Deepfake detection with clustering embedding regularization[27]** In this paper the technique proposed by Li (Using generated simulated samples) and (clustering based embedding) proposed by Zhou have been combined along with some regularization factor to improve performance on adversarial examples According to LI During the postprocessing step certain traces are left which are called artifacts. During the experiments they trained model on real samples and simulates samples rather than real and deepfake samples.Their model only focused on artifacts generated during processing and ignored label information of dataset Zhou proposed a model to capture both artifact tampering and local noise residual evidence. After training the model it was observed that the patches from same images are closer in learning space , while the distance from two different images was large but the dnn was highly vulnerable to adversarial example like when in positive path was extracted from face of an image and negative patch was extracted from non-face region of same image . the local smoothness can be improved by adding a regularization term	Celeb-DF DFDC[7]	Without regularization-98.03% With regularization-98.17	Useful for adversarial examples. The model achieved good results on UADFV, Celeb-DF[8] and DeepFakeDetection datasets including low-quality videos and high quality videos. With the continuous improvement of video tampering technology and the improvement of video quality, it does bring certain challenges to deepfake detection method proposed by the paper.

CHAPTER 3
SYSTEM ANALYSIS AND DESIGN

3.1 System Description

The Web Interface will allow a user to upload a video file to determine its originality.
The important parts are Data Preprocessing on the input video (face extraction, resizing, normalizing) and then importing the trained model and running inference. Then we have our CNN to extract features from the frame and apply RNN to classify whether the video is real or fake.
Hence the system response depends on the file uploaded by the user. It only allows .mp4 as an input file to the system to prevent misuse. The user can play their uploaded video on the browser itself and then click a Predict Button which will start the inference process on the video. This is output to the user in json format and displayed in a table along with the confidence of the prediction.


START
File ?
Image
Video
Upload Image
PreProcessing
converting image into grayscale
Face Detection
Face Cropping
Pre-Processed Data
Discrete Fourier Transformation
Azimuthal Average
Data Splittiing into Training/Testing
DeepFake Detection Classifier
Export Trained Model
Load Trained Model
Real or Fake
Upload video
Training Flow
Prediction Flow
Data-set (Fake / Real) videos
Preprocessing
Spliting video into frames
Face Detection
Face Cropping
Saving the face cropped video
Processed Data-set Contains only face video
Data Splitting Train / Test data split
Data Loader Loading the Train videos and labels
Train / Test the Model
Model Evaluation
Confusion Matrix
Deepfake detection Model
LSTM Video classification
ResNext Feature extraction
Load Trained Model
Export Trained Model
REAL / FAKE
Figure 3.1: System Architecture

3.2 Project Timeline - Gantt Chart

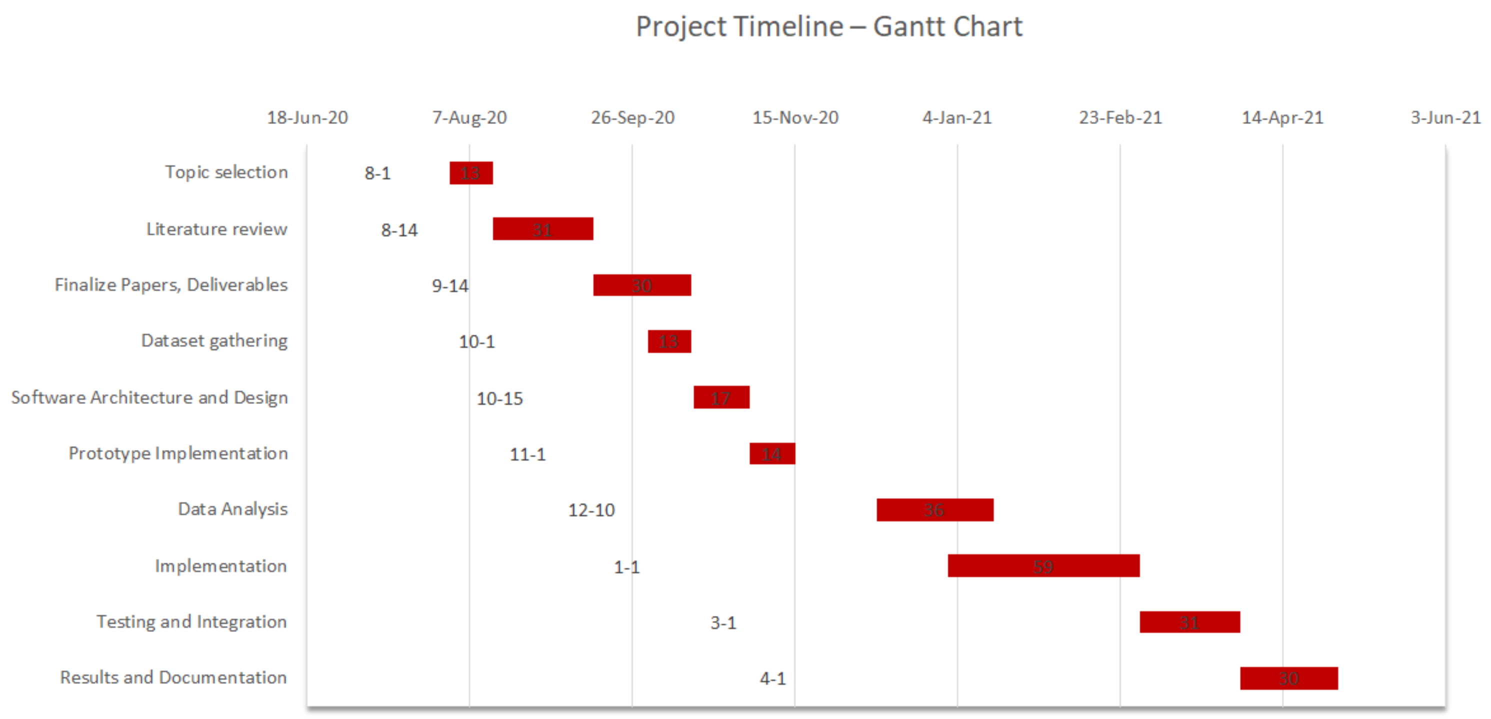


Figure 3.2: Project Timeline

3.3 Study of available Datasets

Name	**Description**
Celeb-df[8]	Celeb-DF dataset includes 590 original videos collected from YouTube with subjects of different ages, ethnic groups and genders, and 5639 corresponding DeepFake videos.
Face Forensics[28]	FaceForensics++ is a forensics dataset consisting of 1000 original video sequences that have been manipulated with four automated face manipulation methods: Deepfakes, Face2Face, FaceSwap and NeuralTextures. The data has been collected from 977 youtube videos and all videos contain a trackable mostly frontal face which enables automated tampering methods to generate realistic forgeries.
Flickr-Faces-HQ Dataset (FFHQ)[29]	HD Image dataset of human faces, Generative Adversarial Networks (GAN); 70,000 high-quality PNG images at 1024×1024 resolution The dataset consists of 52,000 high-quality PNG images at 512×512 resolution and contains considerable variation in terms of age, ethnicity and image background. It also has good coverage of accessories such as eyeglasses, sunglasses, hats, etc
DFDC [7]	Largest and latest Dataset. 124k videos Featuring eight facial modification algorithms. The DFDC[7] dataset is by far the largest currently and publicly available face swap video dataset, with over 100,000 total clips sourced from 3,426 paid actors, produced with several Deepfake, GAN-based, and non-learned methods
DFD	The Google/Jigsaw DeepFake detection dataset - has 3, 068 DeepFake videos generated based on 363 original videos of 28 consented individuals of various genders, ages and ethnic groups. The details of the synthesis algorithm are not disclosed.

Datasets chosen:

We are going to use a mixture of Celeb-DF[8] Dataset, the Deepfake Detection Challenge (DFDC) dataset and FaceForensics dataset for our project. This is because training the model on only one dataset starts to overfit that dataset deepfake technique and won't not generalize well for other videos. Also these are the best datasets for videos as they give us a great diversity of different deepfakegeneration techniques. The videos contain people of a variety of cultures and ages, and the backgrounds vary from bright indoors to dark outdoors.

CHAPTER 4
IMPLEMENTATION AND COMPARISON

The objective of this section is to implement the models for deepfake detection and do a comparative analysis of the models to determine where improvements can be made for better generalization.

Objectives:

1) Selection of suitable dataset
2) Dataset exploration
3) Preprocessing
4) Algorithms
5) Implementation of algorithm

4.1 Selection of suitable dataset:

To make our model as generalized as possible, we use a mixed dataset which consists of equal amounts of videos from different datasets, of Celeb-df[cite], DFDC[cite], and FaceForensics++[cite]. These are the best datasets for videos as they give us a great diversity of different techniques using which deepfakes are made. The videos contain people of a variety of cultures and ages, and the backgrounds vary from bright indoors to dark outdoors. We balanced the classes to contain an equal number of pristine and deepfake videos. The dataset is split into 70% train and 30% test set.

4.2 Dataset Exploration:

Data exploration refers to the initial step in data analysis which involves using data visualization and statistical techniques to describe dataset characterizations, such as size, quantity, and accuracy, in order to better understand the nature of the data. Exploratory analysis, noise removal, missing value treatment, identifying outliers and correct data inconsistencies, and more, all are a part of the process called data preparation and exploration. Exploring various datasets and feature engineering in those datasets is a key step. EDA is used to understand, summarize and analyze the contents of a dataset, usually to investigate a specific question or to prepare for more advanced modeling.

In the Data Exploration we perform an Exploratory Data Analysis (EDA) on the training and testing data of the sample version of the DFDC[7] and Celeb-df dataset.

The file types are checked, then we examine the metadata files by importing the dataframes. After this, we explore video files, by looking first at a sample of fake videos, then the real videos. After that, we explore some of the deepfake videos that have the same origin. We visualize one frame extracted from the video, for both real and fake videos. We also played a few videos for visual inspection and found that while some of the videos are very obviously fake, others have a very high quality and are almost impossible to be recognized by the naked eye.

The details for the above steps are as follows:

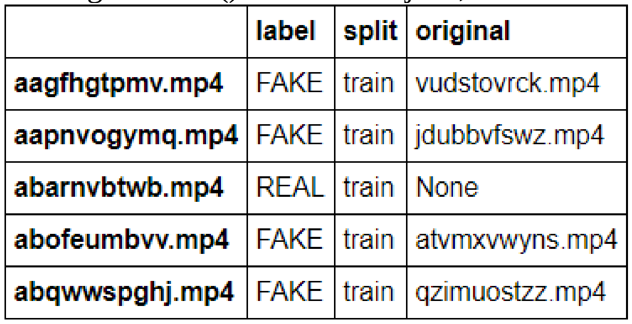
1. Checking the train data files extensions.

   Output: Extensions: ['mp4', 'json']
   All files present in the training folder are videos except for the metadata file which is present in the json format.

2. Exploring the metadata.json file:

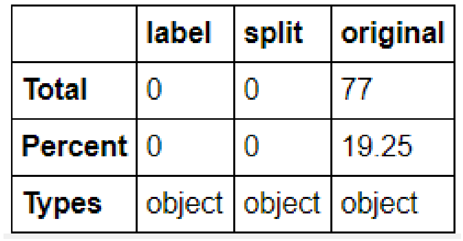
   Output: Files with extension `json`: 1

   Printing the head() of metadata.json, we find the format of the metadata file:



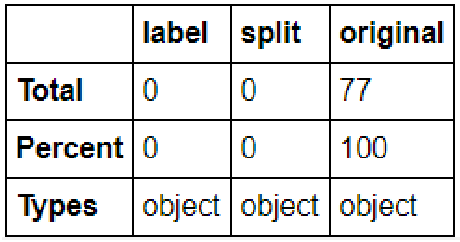
	label	split	original
aagfhgtpmv.mp4	FAKE	train	vudstovrck.mp4
aapnvogymq.mp4	FAKE	train	jdubbvfswz.mp4
abarnvbtwb.mp4	REAL	train	None
abofeumbvv.mp4	FAKE	train	atvmxvwyns.mp4
abqwwspghj.mp4	FAKE	train	qzimuostzz.mp4

3. Checking the metadata for missing values as we can see a None already in the ‘original’ column.



	label	split	original
Total	0	0	77
Percent	0	0	19.25
Types	object	object	object

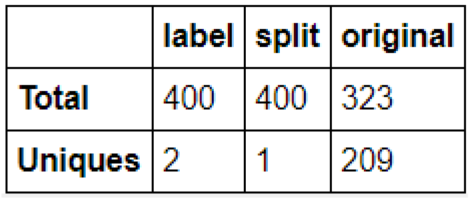
   There are 19.25% of the samples (or 77) missing data points. However this can be because REAL data will have missing ‘original’ value. To confirm this, we need to find out the missing values among the real videos in the dataset.
   Output:



	label	split	original
Total	0	0	77
Percent	0	0	100
Types	object	object	object

Hence, we conclude that all the missing values were only due to the fact that those videos were REAL. Thus there are no missing values or data quality issues in the metadata.json of the dataset.

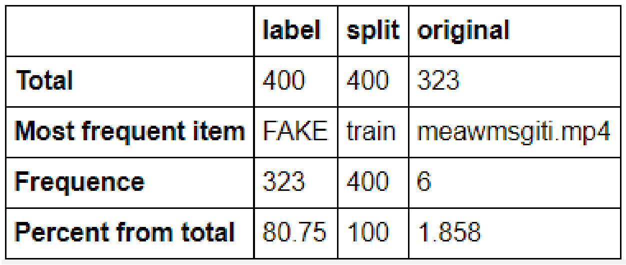
4. Finding out unique values in each column



	label	split	original
Total	400	400	323
Uniques	2	1	209

Each row (object) in the json file is the name of a video. The column ‘label’ indicates whether the video is REAL or FAKE. The ‘split’ indicates whether the video is a part of the TRAIN dataset or the TEST dataset. The ‘original’ column indicates the name of the original video, if the video is FAKE. It contains a total of 323 values as we know that 77 of them are None. Also it contains 209 unique videos meaning that the same videos have been repeated again to create the FAKE videos.

5. Finding the most frequent item in each column to find out whether there is an imbalance in REAL and FAKE videos.



	label	split	original
Total	400	400	323
Most frequent item	FAKE	train	meawmsgiti.mp4
Frequence	323	400	6
Percent from total	80.75	100	1.858

We find out that the most frequent ‘label’ is FAKE (80.75%), and that meawmsgiti.mp4 is the most frequent ‘original’ i.e. it has been repeated 6 times to make a FAKE video.

6. Visualizing class imbalance:

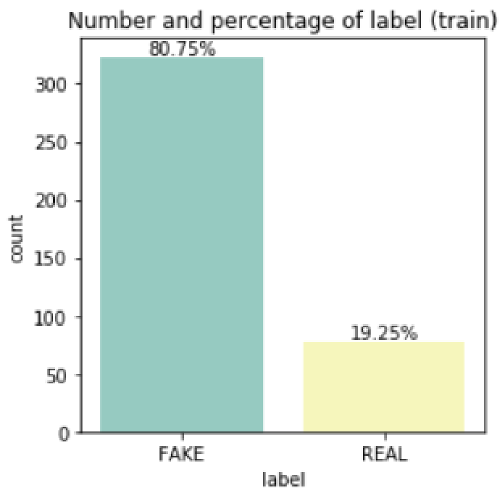


Figure 4.1

We see that 80.75% of the sample dataset contains FAKE videos and only 19.25% are REAL. Although this is just a sample dataset, the same distribution is found in the full dataset. Hence before training, we make sure that equal numbers of REAL and FAKE sequences are passed to the classifier. One other way to balance classes is to augment the real videos using techniques like rotation, shifting, flipping etc. But since the full dataset is very large, we can balance the classes even without augmentation.

7. Exploring the Videos:

There are no missing videos that are present in the metadata but not available in the dataset.

Visualizing a few Fake Video Frames:

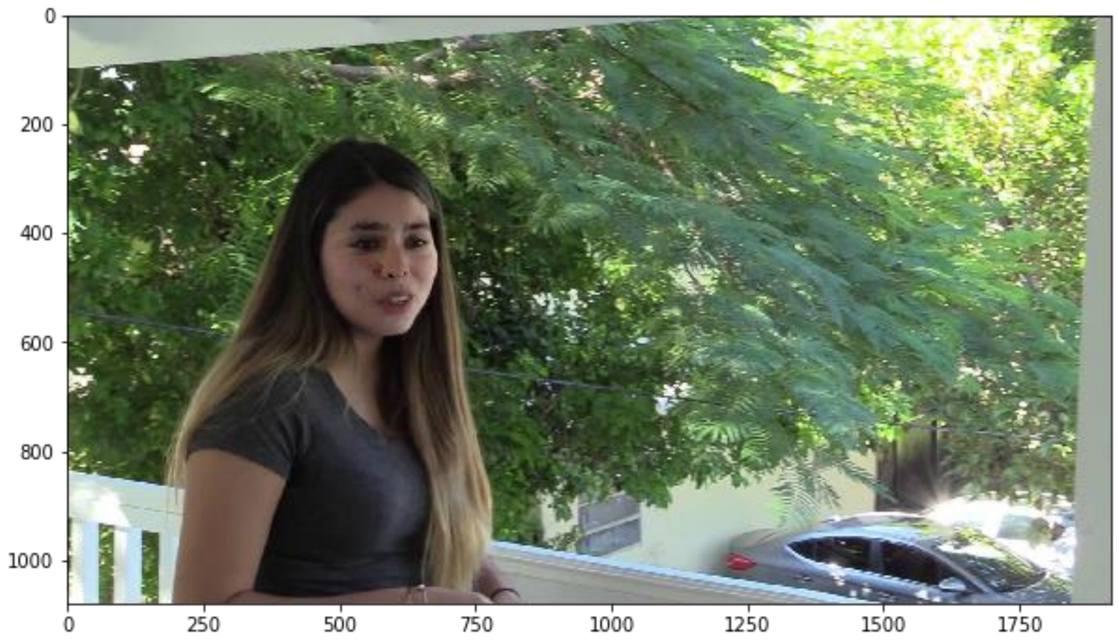


Figure 4.2

Figure 4.3

Figure 4.4

Figure 4.5

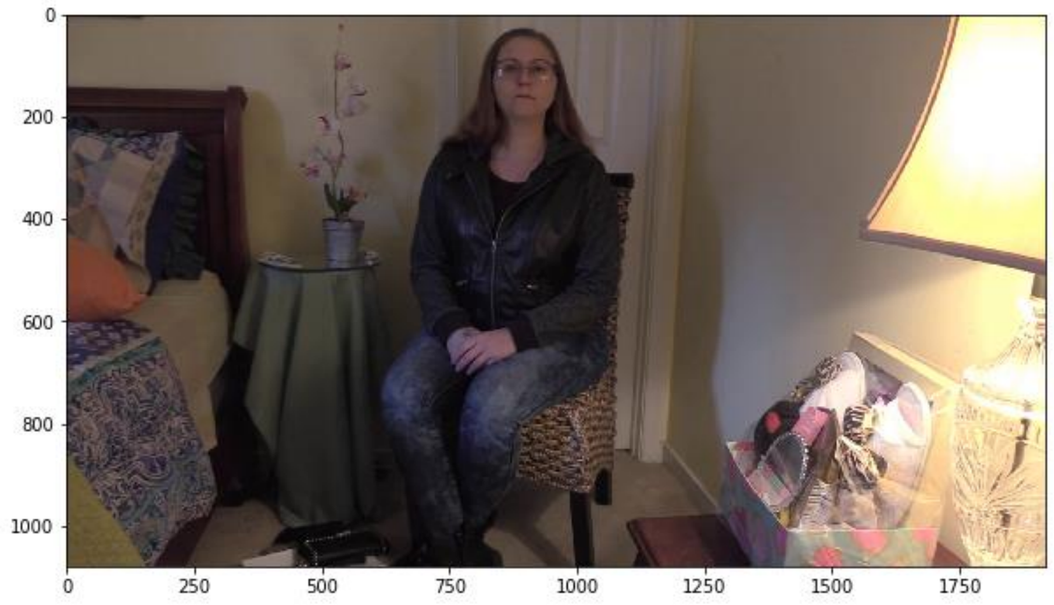


Figure 4.6

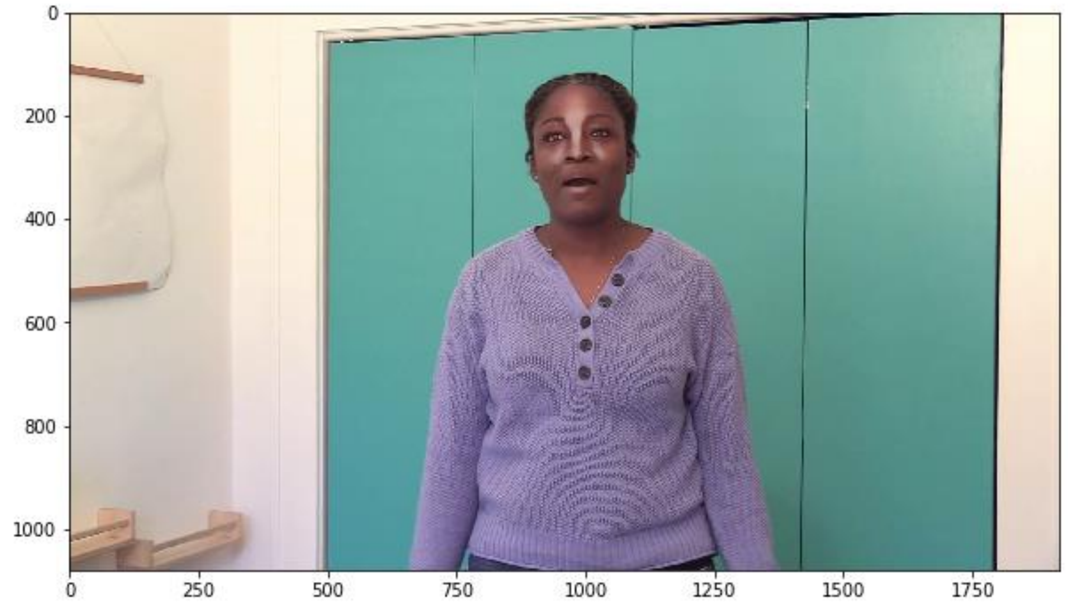


Figure 4.7

Hence, on examining even more videos we find out that although the human eye can identify some of the deepfakes, most of the fake videos are difficult to be identified by the naked eye.

Visualizing a few Real Video Frames:

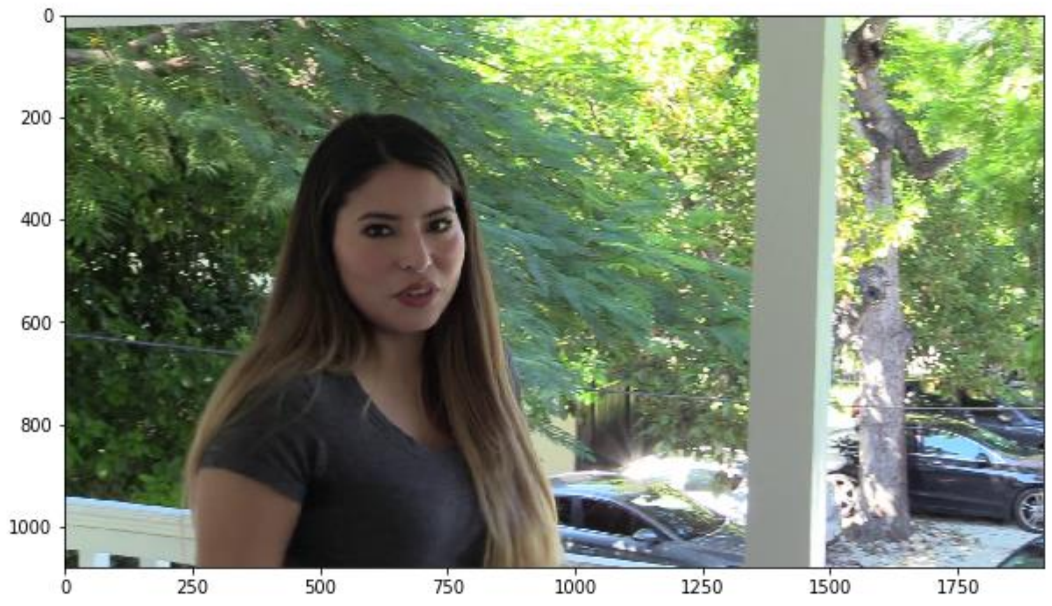


Figure 4.8

Figure 4.9

These are the real people corresponding to the fake ones above.

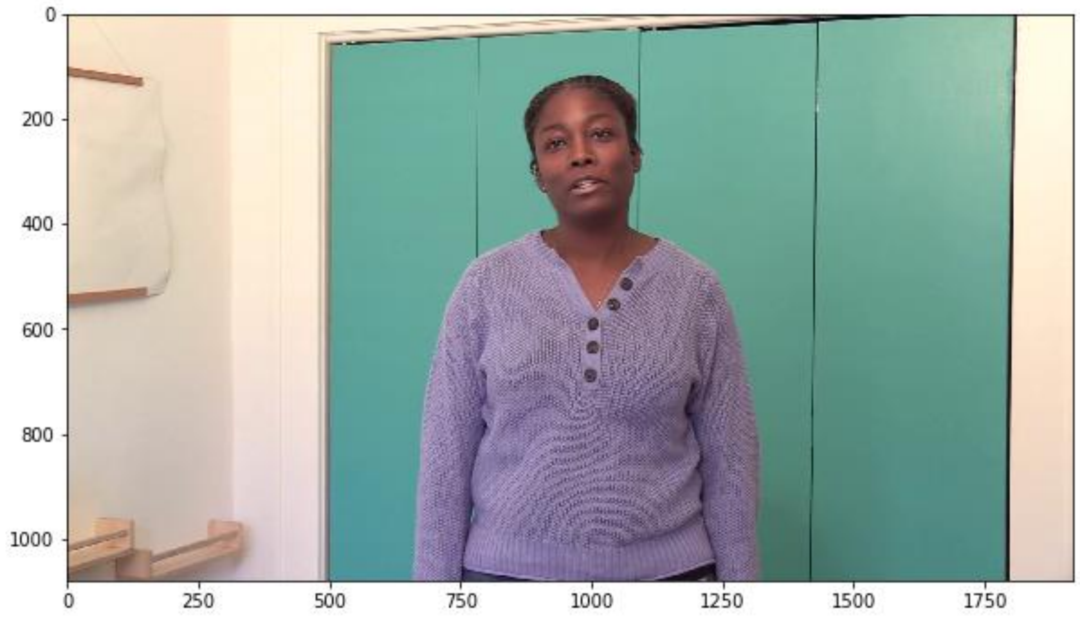


Figure 4.10

Celeb-df[8] Data Exploration and Visualizing a few Fake and Real Video Frames:

Celeb-DF[8] includes 590 original videos collected from YouTube with subjects of different ages, ethic groups and genders, and 5639 corresponding DeepFake videos.

The real data is stored in the form of videos having names ‘id0_0000.mp4’. The id refers to the identity of a particular person (celebrity). So there are multiple real videos of the same person. (id11_0000.mp4, id11_0001.mp4, id11_0002.mp4, etc)

The fake data is a face swap of any two people from the real data and is stored in the form of id0_id11_0000.mp4 meaning that this would be the first face swap video of person 0 and person11. The same real videos are permuted with each other resulting in more fake videos than real videos in the dataset. A few examples are shown below:

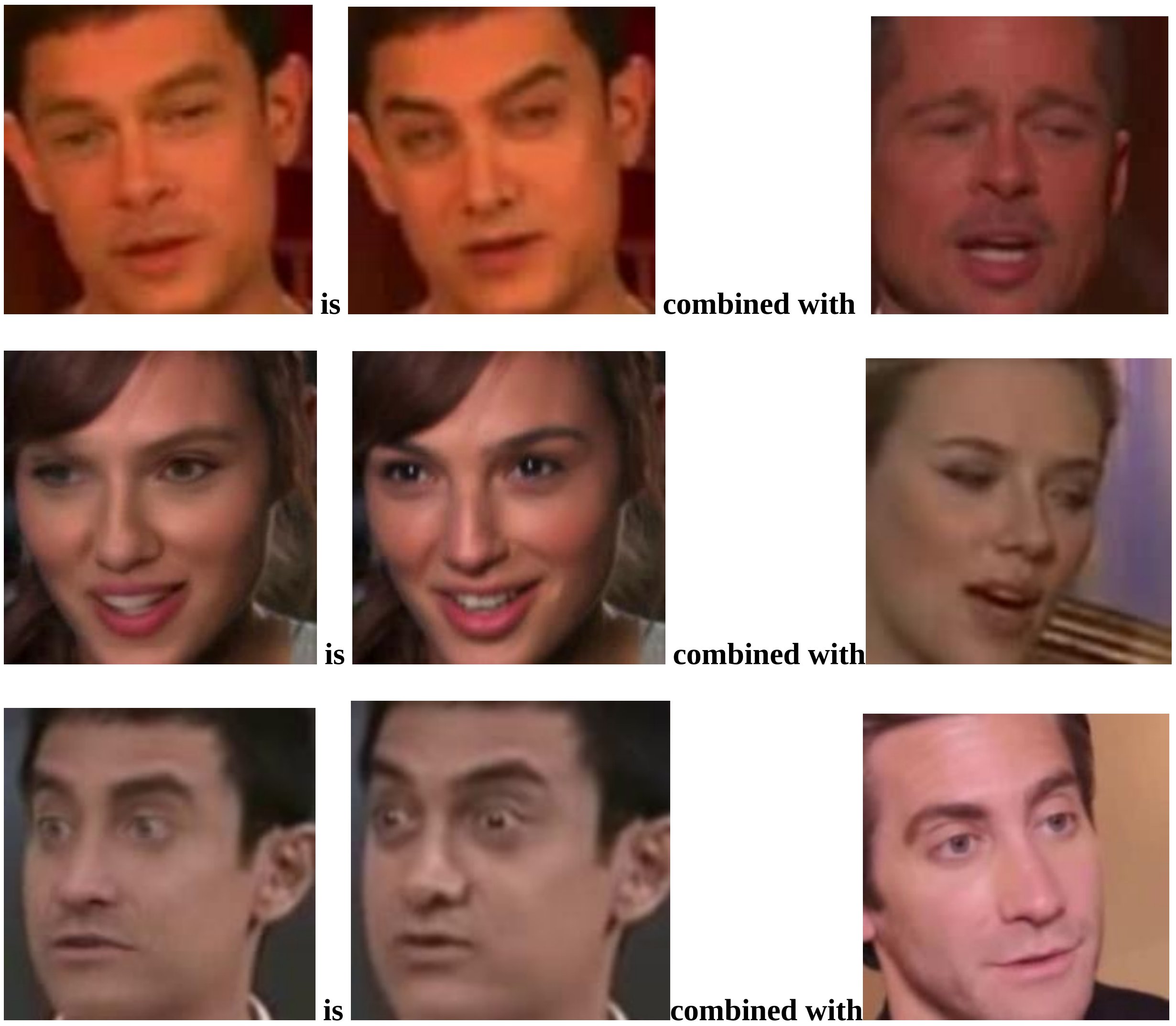


Figure 4.11

An example of different Fake videos made from the same real video is shown below for DFDC:

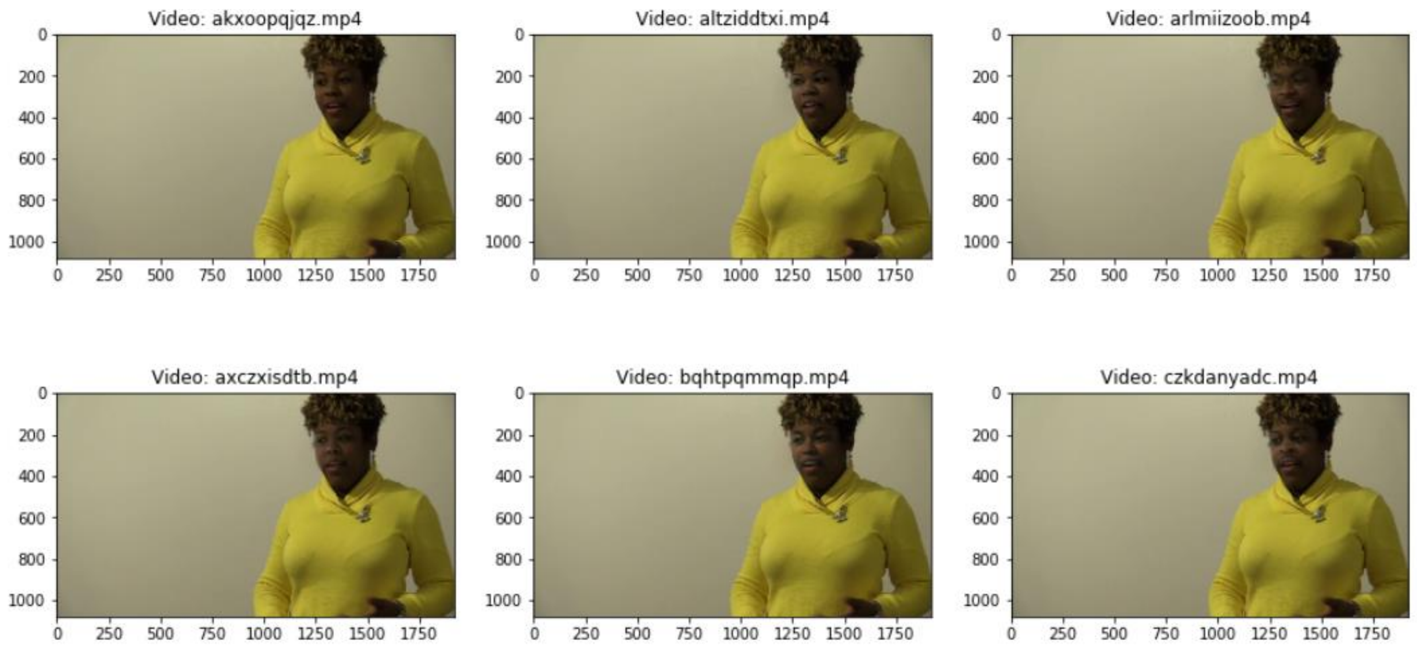


Figure 4.12

4.3 Preprocessing:

Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is an important step to prepare the data to create any machine learning model. Data preprocessing is a data mining technique that involves transforming raw data into a format understandable by computer. There are various techniques and tools to perform data preprocessing. Data pre-processing has a significant impact on the performance of supervised learning models because unreliable samples probably lead to wrong outputs.

For preprocessing, we split the dataset into training, validation and test datasets while keeping the classes balanced. This ensures that each final set has exactly 50% videos of each class which allows us to report our results in terms of accuracy without having to take into account biases due to the appearance frequency of each class.

In terms of data preprocessing of the video sequences, we:

1. Split the video into frames.
2. Extract facial region of the frame by using a face detector and then cropping the detected face.
3. Remove the frames that don't have faces in it as they add noise to the final prediction.
4. Resize the cropped frame to 112*112 to pass to CNN.
5. ResNeXt[31] model expects the input images normalized in the same way as it was pre-trained on, i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be maximum of size 224. The images have to be loaded in to a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225] because the dataset that ResNeXt[31] was trained on has this mean and std deviation.
6. As processing the 10 second video at 30 frames per second i.e total 300 frames will require a lot of computational power, we decide to use only the first 100 frames for training the model.
7. We take a batch size of 4 video sequences at a time. Now the shape of the train_data is [BatchNo, SeqNo, ChannelNo, Height, Width] where SeqNo corresponds to the frame in the video.

4.4 Modelling the classifier:

1. **CNN - for Feature Extraction**

   A CNN is used as a feature extractor to pass to the LSTM. We don't train a new CNN, but instead adopt the ResNeXt[31] architecture which is a Residual Network CNN developed by Facebook. We use the PyTorch implementation of Re sNeXt, and download the weights using resnext50_32x4d(pretrained=True). This downloads the model and trained weights for all the layers. However, since we want to use the CNN as a feature extractor and not a classifier by itself, we remove the last fully connected layer of dimension 2048. Hence, now passing an input image to the network results in the output of a 2048 dimension feature map. Thus we use transfer learning for feature extraction.

   Explaining and Comparing ResNet[30] and ResNeXt to keep as a feature extractor:

   - **ResNet** [30]

   The authors of this paper discovered that a multi-layer deep neural network like a deep CNN can produce unexpected results. In this case, the training accuracy dropped as the layers increased, technically known as vanishing gradients. While training, the vanishing gradient effect on network output with regard to parameters in the initial layer becomes extremely small. The gradient becomes further smaller as it reaches the minima. As a result, weights in initial layers update very slowly or remain unchanged, resulting in an increase in error.
   A residual network, or ResNet[30] for short, is an artificial neural network that helps to build deeper neural networks by utilizing skip connections or shortcuts to jump over some layers. skipping helps build deeper network layers without falling into the problem of vanishing gradients.

   - ResNeXt [31]

   The model name, ResNeXt[31], contains Next. It means the next dimension, on top of the ResNet[30]. This next dimension is called the "cardinality" dimension. The network expands along this new dimension and convolutions are performed for each path.

   The dimension of cardinality controls the number of more complex transformations. The dimension is increased directly from 4 to 256, and then added together, and also added with the skip connection path.

   Compared with Inception-ResNet[30] that needs to increase the dimension from 4 to 128 then to 256, ResNeXt[31] requires minimal extra effort designing each path.

   Unlike ResNet[30], in ResNeXt[31], the neurons at one path are not connected to the neurons at other paths.

Thus ResNet-50 is a special case of ResNeXt-50 with C=1, d=64.

Experimentally, it is found that ResNeXt-50 (32×4d) obtains 22.2% top-1 error for ImageNet-1K (1K means 1K classes) dataset while ResNet-50 only obtains 23.9% top-1 error.

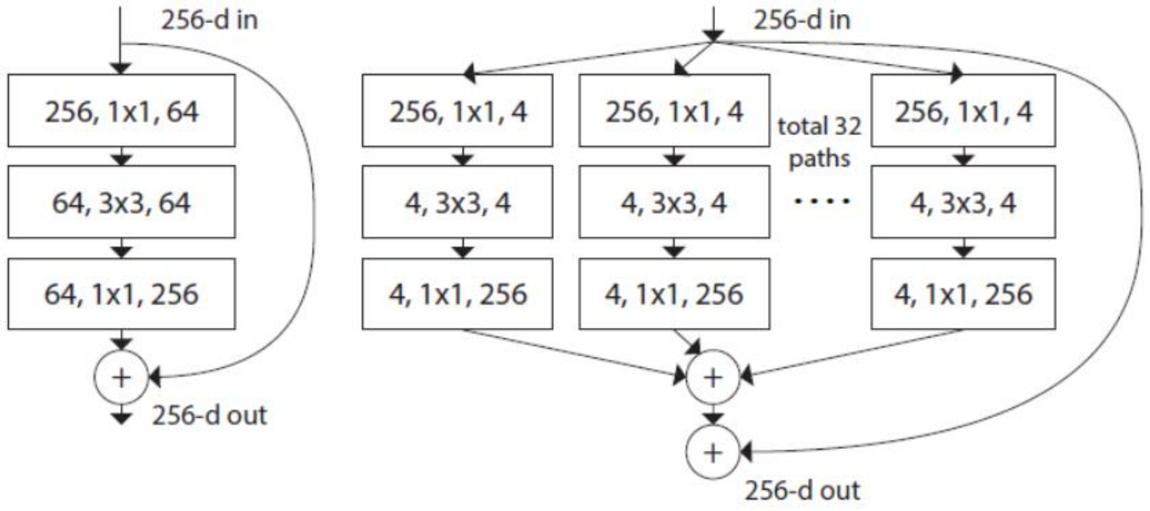


Figure 4.15

Hence we employ the ResNeXt model for feature extraction to get better features.

2. **Adding the Frequency Spectrum Features**

To the 2048 dimensions gathered from the CNN, we concatenate the frequency spectrum features. To do this we perform the following steps:

1. Convert the (3*112*112) dimension image to grayscale by taking mean along the 0th dimension axis.
2. Take the 2 dimensional discrete Fourier transform of the image using torch.fft.fft2().
3. This Fourier image is shifted such that the DC-value (*i.e.* the image mean) *F(0,0)* is displayed in the center of the image. The further away from the center an image point is, the higher is its corresponding frequency.
4. DC-value is by far the largest component of the image. However, the intensity values in the Fourier image are too large to be displayed or processed easily, therefore all other values appear as black. We apply a logarithmic transformation to the image to obtain visible components of all frequencies.
5. Then we take the Azimuthal Average or the Radial Average of the Fourier image

because just like the CNN outputs 1D tensor of size 2048, we need a 1D feature vector rather than a 2D Fourier transform. Hence azimuthal averaging is used to compute a 1D representation of the FFT power spectrum.

Each frequency component is the radial average from the 2D spectrum. Hence, it is a compression, gathering and averaging similar frequency components into a vector of features. In this way, we can reduce the amount of features to 1D without losing relevant information.

6. This 1D tensor of size 77 is concatenated to the CNN features to make a tensor of size 2048+77=2125 to be fed to the LSTM[12]

3. **LSTM for Sequence Processing**

Now we have a sequence of CNN and frequency feature vectors of input frames as input and towards the end we would need a 2-node neural network with the probabilities of the sequence being part of a deepfake video or an untampered video. The key challenge that we need to address is the design of a model to recursively process a sequence in a meaningful manner. For this problem, we resort to the use of a 2048-wide LSTM unit with some chance of dropout.

During training, if frequency spectrum features are being used, the LSTM[12] model takes a sequence of 2048-dimensional ResNeXt[31] feature vectors concatenated with the 77 frequency features. The LSTM is followed by a 512 fully-connected layer.

4. **Softmax Layer for final classification**

Finally, we use a softmax layer to compute the probabilities of the frame sequence being either pristine or deepfake. The LSTM module is an intermediate unit in our pipeline, which is trained entirely end-to-end. The CNN (ResNeXt) is not trained as the weights are loaded and its purpose is only for feature extraction.

4.5 Implementation with different algorithm variations:

This section describes how we implement the CNN-LSTM model architecture and how we try different variations of it to compare them to find and analyze which model runs best.

We use the PyTorch library because it is easier and allows us to write Object Oriented Code to use classes to define models and the forward propagation steps, etc.
Tensorflow works on a static graph concept that means the user first has to define the computation graph of the model and then run the ML model, whereas PyTorch uses a dynamic graph that allows defining/manipulating the graph on the go.
PyTorch offers an advantage with its dynamic nature of creating the graphs. The graphs are built, interpreting the line of code corresponding to that particular aspect of the graph. However, in the case of TensorFlow, as the construction is static and the graph is required to go through compilation and then executed on the execution engine.
To implement the model in Pytorch, we use the following libraries:

- torch - We use the Pytorch library to create, train, and test the model.
- torchvision - The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision. We use this library to load the pretrained ResNeXT[31] model that it provides for PyTorch.

4.5.1 CNN + LSTM:

In this model, we first use a plain CNN with last layer removed, followed by a LSTM, with a linear layer to output 2 class neurons.

Code for model:

```
class Model(nn.Module):
        def __init__(self, num_classes,latent_dim= 2048, lstm_layers=1,
                    hidden_dim = 512, bidirectional = False):
           # Initialize nn.Module class
           super(Model, self).__init__()

           # load Pretrained resnext model
           model = models.resnext50_32x4d(pretrained = True)

           # Remove the last two layers of the resnext architecture
             and put the rest of the modules in resnext in the Sequential module.
           self.model = nn.Sequential(*list(model.children())[:-1])

           # Define lstm layer/module
           self.lstm = nn.LSTM(latent_dim,hidden_dim, lstm_layers,  bidirectional)
           self.relu = nn.LeakyReLU()
           self.dp = nn.Dropout(0.4)
           self.linear1 = nn.Linear(512,num_classes)
           self.avgpool = nn.AdaptiveAvgPool2d(1)

        def forward(self, x):
           batch_size,seq_length, c, h, w = x.shape
           # Stack all the sequences of all the batches together
           x = x.view(batch_size * seq_length, c, h, w)
           x = self.model(x)
           # Seperate the batches as sequence of frames for lstm
           x = x.view(batch_size,seq_length,2048)
           x_lstm,_ = self.lstm(x,None)
           return self.dp(self.linear1(torch.mean(x_lstm,dim = 1)))
```

OUTPUT :

The following graph shows the training and validation accuracy.
For every epoch the training and test accuracy is recorded and plotted using a graph. As we can see in Figure Below the X-axis consists of Epochs ranging from 1 to 20, both inclusive. Y-Axis consists of the Accuracy calculated by our model. At first, the difference between training and validation accuracy is quite significant but as the process continues , i.e as epochs increase we see both the accuracy gradually are almost same and range from 85%-90% during the end of epoch cycle .

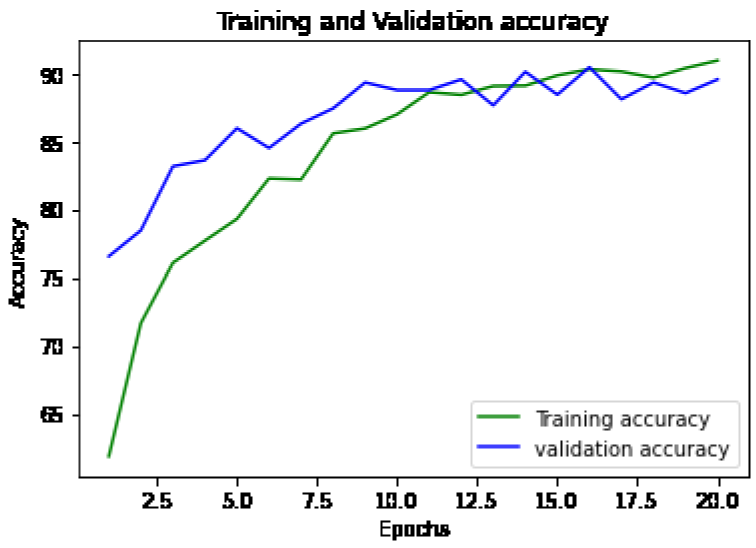


Figure 4.16

The below figure shows the training and testing Loss for our particular model. As we saw above, the loss has been recorded for every epoch ranging from 1 to 20 epochs, both inclusive. X-axis, as usual, consists of a number of Epochs and Y-axis consists of the Loss encountered at each epoch for training and validation. As seen in the below figure , the Loss for training process is High at the start of the process but eventually it decreases and tries to minimize the loss as much as possible . But for validation loss , it did start a little higher but after a certain point , there is no sharp increase nor decrease , only after 15 epoch we can see that the loss is increasing . Since the validation loss is greater than the training loss , it might be a case of overfitting but since the validation accuracy was similar to training accuracy , there is not much evidence to prove the overfitting.

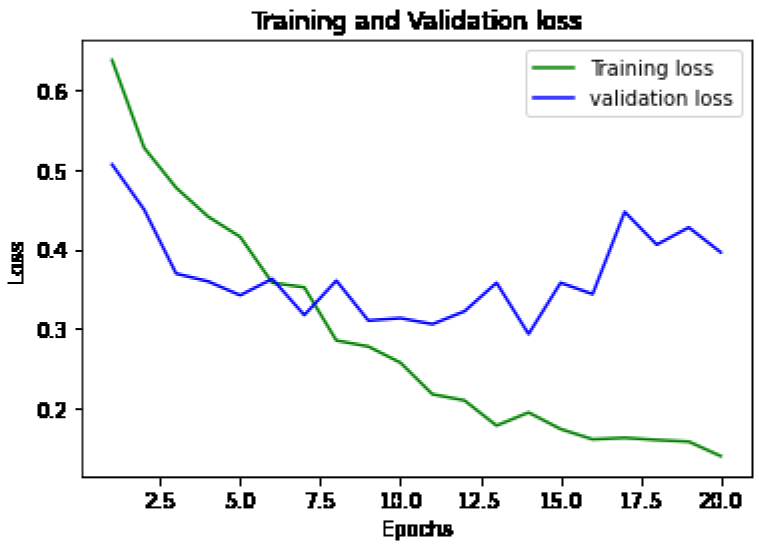


Figure 4.17

The below figure is the Confusion Matrix . The Y-Label shows the Actual Label from the data and the X-Label shows the predicted Label .

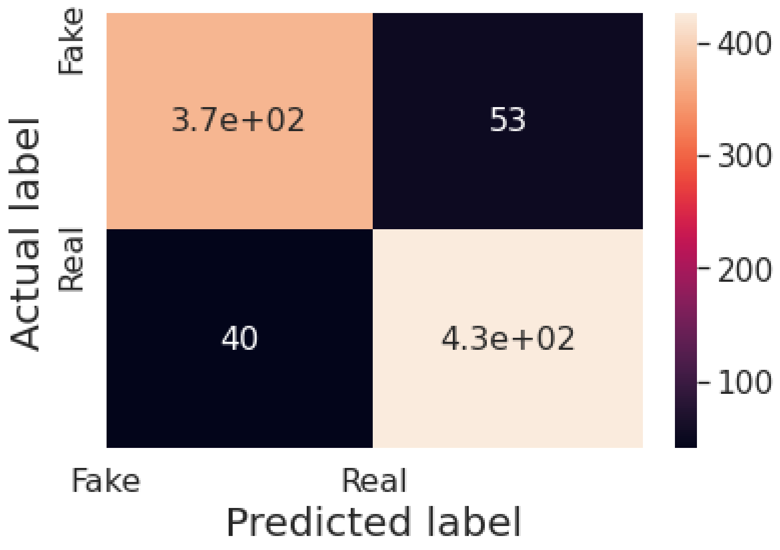


Figure 4.18

From the confusion matrix we can derive the following :

1) True Positive (TP) : 430
2) True Negative (TN) : 370

3) False Positive (FP) : 53
4) False Negative (FN) : 40

These were whole numbers , now let's look at the rates that we can derive for better understanding of the model :

1) Accuracy : 89.56%
2) Misclassification Rate (Error Rate) : 10.44%
3) True Positive Rate (Also known as Sensitivity or Recall) : 93.47%
4) False Positive Rate : 10.33%
5) True Negative Rate(Also known as Specificity) : 89.67%
6) Precision : 91.489%
7) Prevalence : 52.63%

So we find out that the Test Accuracy for the model is **: 89.56%**

4.5.2 Frequency + CNN + LSTM:

As explained in section 4.4, this model uses the CNN concatenated with azimuthally averaged frequency features resulting in a 2125 dim output. All the layers after feature extraction are the same as 4.5.1. (LSTM, linear layer).

The code for the model is given below. The Azimuthal Averaging function needs to be re-implemented using tensors instead of numpy because the model is run on CUDA GPUs and numpy is a CPU library.
We write the calculateFreqSpectrum(self, x) function in the Model class which does the required preprocessing (converting the image to grayscale) and FFT, azimuthal averaging to yield the features.

The code for the same is given below:

```
import torch
from torch import nn
from torchvision import models
import numpy as np

class Model(nn.Module):
   def __init__(self, num_classes,latent_dim= 2125, lstm_layers=1 , hidden_dim = 512,
bidirectional = False):
      super(Model, self).__init__()
      model = models.resnext50_32x4d(pretrained = True) #Residual Network CNN
      self.model = nn.Sequential(*list(model.children())[:-a])
      self.lstm = nn.LSTM(latent_dim,hidden_dim, lstm_layers,  bidirectional)
      self.relu = nn.LeakyReLU()
      self.dp = nn.Dropout(0.4)
      self.linear1 = nn.Linear(512,num_classes)
      # number of frequency features to be considered
      self.N = 300

   def azimuthalAverage(self, image, center=None):
      """
      Calculate the azimuthally averaged radial profile.
      image - The 2D image
      center - The [x,y] pixel coordinates used as the center. The default is
            None, which then uses the center of the image (including
            fracitonal pixels).

      """
      # Calculate the indices from the image
      y, x = np.indices(image.shape)
      #print(x)
      #print(y)
      if not center:
         center = np.array([(x.max()-x.min())/2.0, (y.max()-y.min())/2.0])

      x = torch.from_numpy(x)
      y = torch.from_numpy(y)
      center = torch.from_numpy(center)
      r = torch.hypot(x - center[0], y - center[1])
```

```
    #print(r)
    # Get sorted radii
    ind = torch.argsort(r.flatten())
    #print(ind)
    temp_r = r.flatten()
    temp_image = image.flatten()
    r_sorted = temp_r[ind]
    i_sorted = temp_image[ind]

    # Get the integer part of the radii (bin size = 1)
    r_int = r_sorted.int()

    # Find all pixels that fall within each radial bin.
    deltar = r_int[1:] - r_int[:-1]  # Assumes all radii represented
    rind = torch.where(deltar)[0]       # location of changed radius
    nr = rind[1:] - rind[:-1]        # number of radius bin
    nr = nr.to(torch.device('cuda:0'))

    # Cumulative sum to figure out sums for each radius bin
    csim = torch.cumsum(i_sorted, dim=0, dtype=torch.float)
    tbin = csim[rind[1:]] - csim[rind[:-1]]
    tbin = tbin.to(torch.device('cuda:0'))
    radial_prof = tbin / nr
    return radial_prof

def calculateFreqSpectrum(self, x):
    # expecting x in shape [batch_size*seq_length, c, h, w]
    epsilon = 1e-8

    # Change this to number of frames.
    number_iter = len(x)

    #batch_frequency_spectrum = torch.empty(77)
    #batch_frequency_spectrum = batch_frequency_spectrum.to(torch.device('cuda:0'))
    batch_frequency_spectrum = []
    for i in range(number_iter):
        # Read the image
        img_chw = x[i]

        # For frequency, image must be in hwc
        #img = np.transpose(img_chw, (2, 0, 1))

        # convert the image into grayscape
        img = torch.mean(img_chw, axis=0)

        # No need to crop the center as the image already contains face extracted/
        h = img.shape[0]
        w = img.shape[1]

        # Taking Fourier Transform of the image
        f = torch.fft.fft2(img)

        # Shift the quadrants so that low spatial frequencies are in the center
        fshift = torch.fft.fftshift(f)
        #fshift += epsilon

        # Calculating 2D Power Spectrum
        magnitude_spectrum = 20*torch.log(torch.abs(fshift))

        # Azimuthally averaged 1D Power spectrum
        psd1D = self.azimuthalAverage(magnitude_spectrum)

        # Calculate the azimuthally averaged 1D power spectrum
        points = torch.linspace(0,self.N,len(psd1D)) # coordinates of a
        xi = torch.linspace(0,self.N,self.N) # coordinates for interpolation

        # This is the frequency spectrum for the current image.
        #interpolated = griddata(points,psd1D,xi,method='cubic')
        #interpolated /= interpolated[0]
        psd1D = torch.divide(psd1D, psd1D[0])

        # append to the list which contains the spectra for all sequences in the batch
        #print(psd1D.shape)
        #batch_frequency_spectrum.append(psd1D)
        #batch_frequency_spectrum = torch.stack((batch_frequency_spectrum, psd1D))
        batch_frequency_spectrum.append(psd1D)

    #print(batch_frequency_spectrum)
    #print("concatenated")
    #print(torch.stack(batch_frequency_spectrum, dim=0))
```

```
        return torch.stack(batch_frequency_spectrum, dim=0)


    def forward(self, x):
        batch_size,seq_length, c, h, w = x.shape
        x = x.view(batch_size * seq_length , c, h, w)
        frequencyFeatures = self.calculateFreqSpectrum(x)
        frequencyFeatures = frequencyFeatures.view(batch_size, seq_length, 77)
        #print(frequencyFeatures.shape)
        x = self.model(x)
        x = x.view(batch_size,seq_length,2048)

        x = torch.cat((x, frequencyFeatures), axis=2)
        #print(x.shape)
        x_lstm,_ = self.lstm(x,None)
        return self.linear1(torch.mean(x_lstm,dim = 1))
```

OUTPUT :

As we have seen with the above graphs , it's the same graph but for a different model. As expected, in case of training, the accuracy at the initial stages is low but as we progress we get better accuracy. But for validation**,** there is no sharp increase in accuracy and after 7-8 epochs there is a huge difference between training and validation accuracy, which might be a case of overfitting.

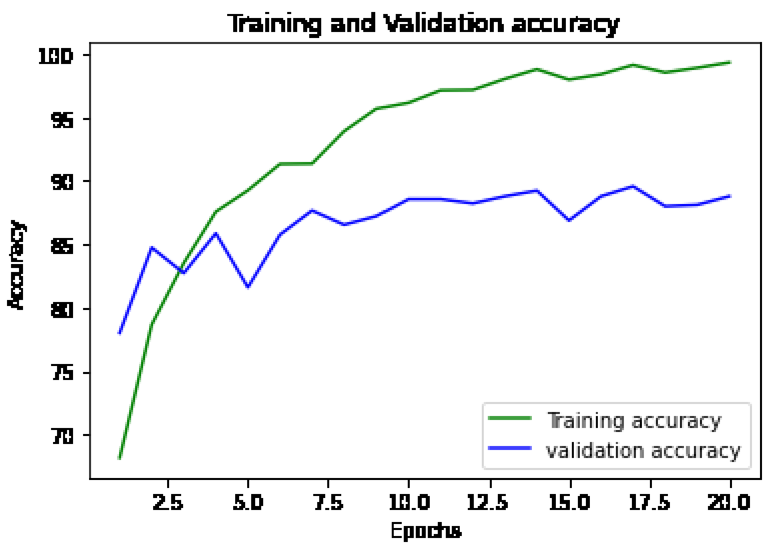


Figure 4.19

The below graph plots the training and validation accuracy for different epochs. The training loss is gradually decreasing as expected but the validation loss is in the range of 0.3-0.5 and can be seen that it increases at the later stage.From the two graphs given we can say that the model is overfitting to our data .

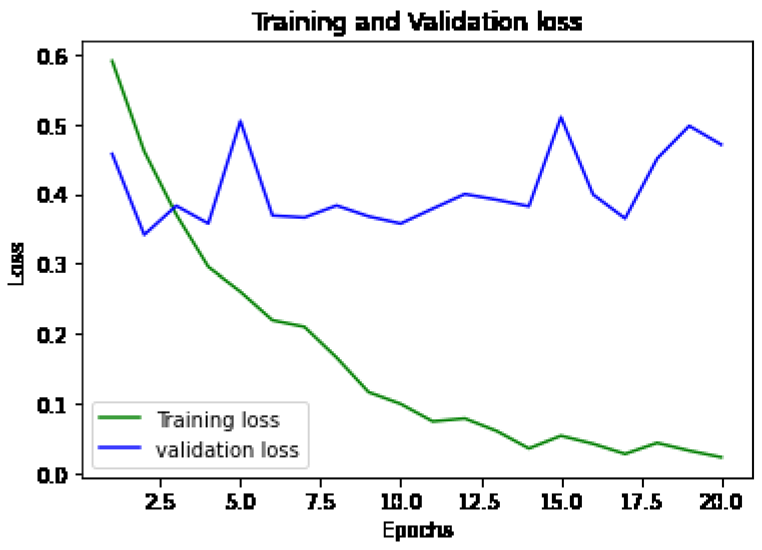


Figure 4.20

Below is the confusion matrix for the model :

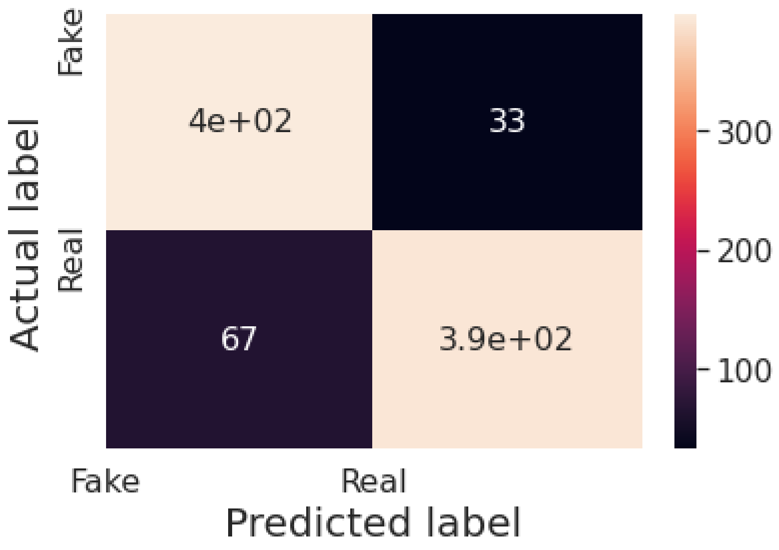


Figure 4.21

The confusion Matrix gives us the following values :

1. Accuracy : 88.77%
2. Misclassification Rate (Error Rate) : 11.23%
3. True Positive Rate (Also known as Sensitivity or Recall) : 85.33
4. False Positive Rate : 7.62%
5. True Negative Rate(Also known as Specificity) : 92.38
6. Precision : 92.2%
7. Prevalence : 51.3%

Accuracy: 88.77%

4.5.3 Frequency Analysis on Video Frames:

To analyze 4.5.2 better, we investigate the Azimuthal averaging method on images (frames alone) from our dataset to see if it is showing the characteristic properties as given in the paper. We get the following plot after finding out the 1D power spectra of the Real and Fake frames.

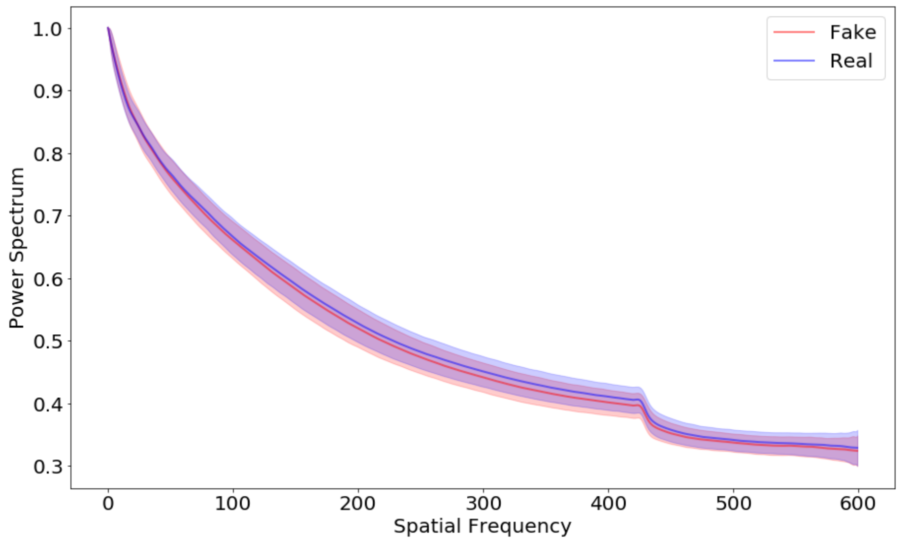


Figure 4.22

The authors have shown the following plot (4.23) in their paper for their dataset.

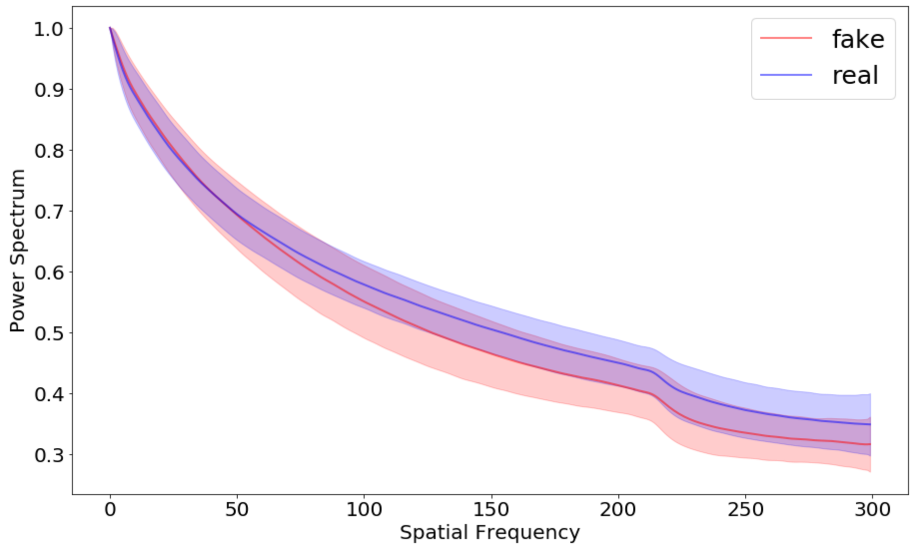


Figure 4.23.

The figure 4.22 shows us the power spectrum graph for the deepfake videos and the Figure 4.23 shows us the power spectrum for Deepfake images . As we can see there is a clear separation in case of images i.e Figure 4.23, of power spectrum for Real and Fake images

but in case of Figure 4.22 , there is no clear boundary to be seen between Real and Fake Images.
Thus we can expect that this method won't affect the accuracy of models on deepfake videos dataset. So for proper classification we expect that there is a proper separation between Real and Fake images.

This can also be attributed to the fact that the dataset they use (Celeb-A, Faces-HQ) have more easily identifiable deepfakes with blur seen by the naked eye as well, due to which the frequency spectra are different. However, datasets that we use (DFDC, Celeb-df) have more realistic deepfakes that don’t show any difference in frequency spectra on their frames.

4.5.4 Wide_Resnet (for CNN) + LSTM:

For the following model , the code is similar to that one we saw in 4.5.1 , the only change is that the Resnext model is replaced by Wide_Resnet for CNN.
As we can see the general trend for validation loss is increasing and for training loss as expected it is decreasing after every epoch.

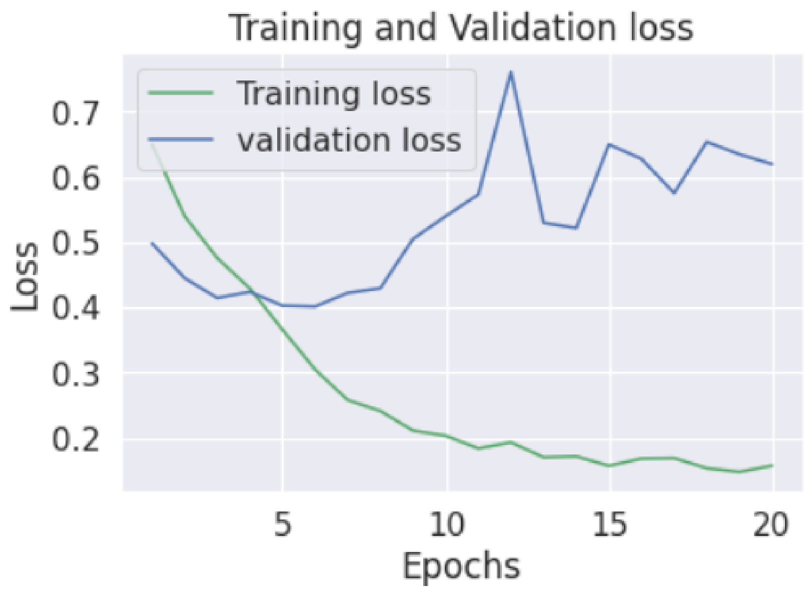


Figure 4.24

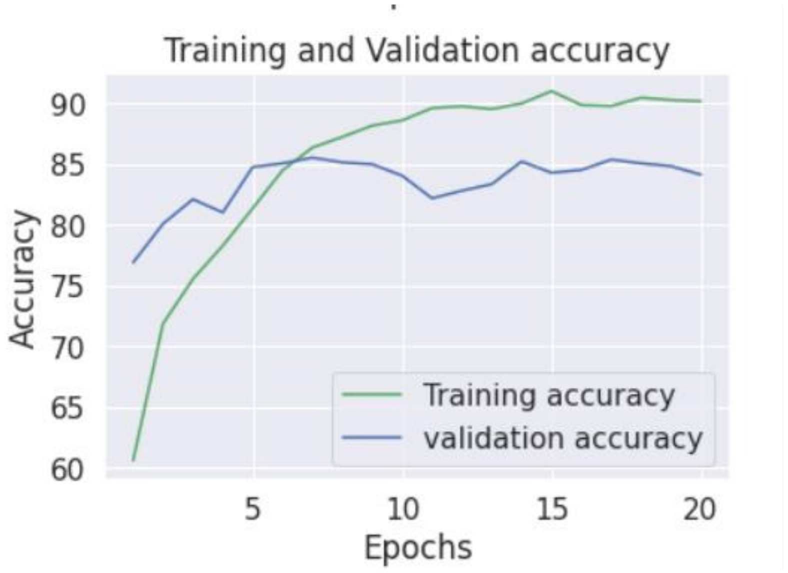
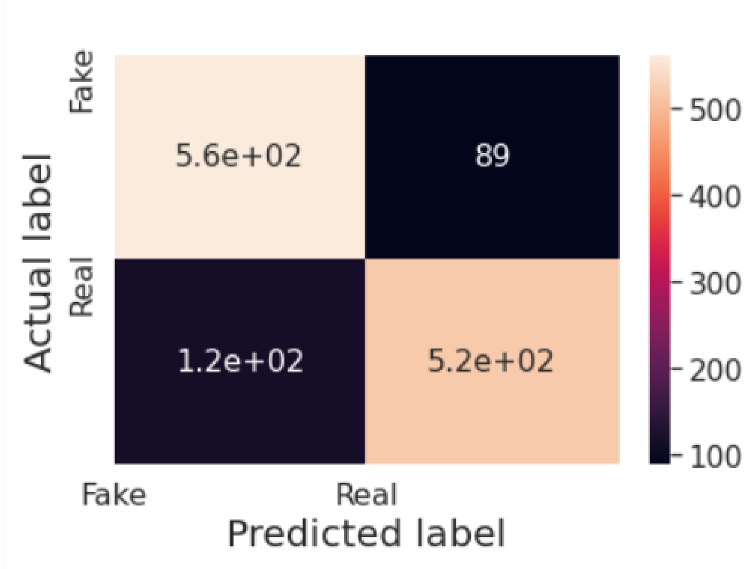


Figure 4.25

In Figure 4.25 we see that the validation accuracy is 84.13%



```
Calculated Accuracy 84.13685847589424
```

Figure 4.26

From the Confusion Matrix we calculate the following metrics :

1. Accuracy : 84.13%
2. Misclassification Rate (Error Rate) : 15.87%

3. True Positive Rate (Also known as Sensitivity or Recall) :81.25%
4. False Positive Rate : 13.7
5. True Negative Rate(Also known as Specificity) : 86.3
6. Precision : 85.38%
7. Prevalence : 49.65%

4.5.5 CNN + Averaging without LSTM:

The training loss is decreasing after every epoch but the validation loss is increasing which may suggest the model has started overfitting.

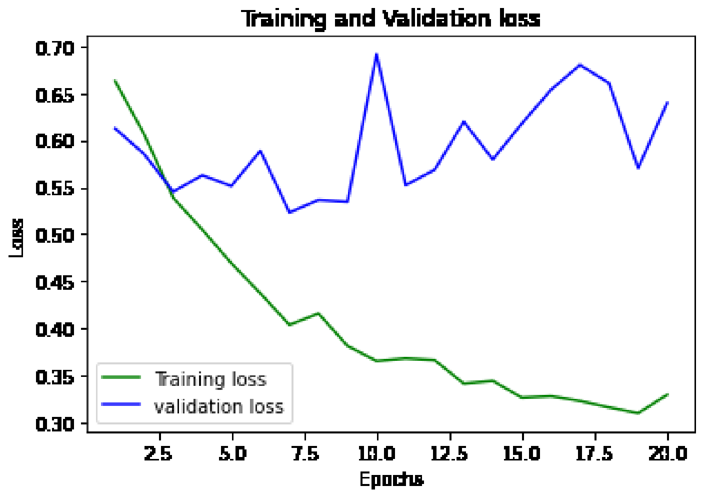


Figure 4.27

In Figure 4.28 , we see that the training accuracy as well as the validation accuracy is increasing after every epoch

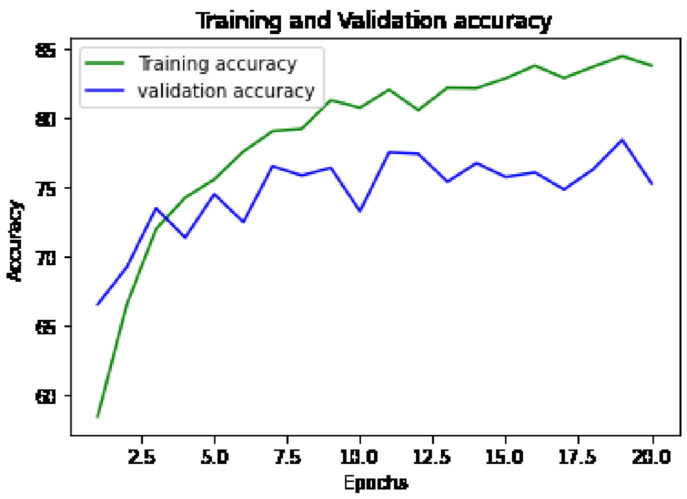


Figure 4.28

In Figure 4.29 we see the confusion matrix given by the model.

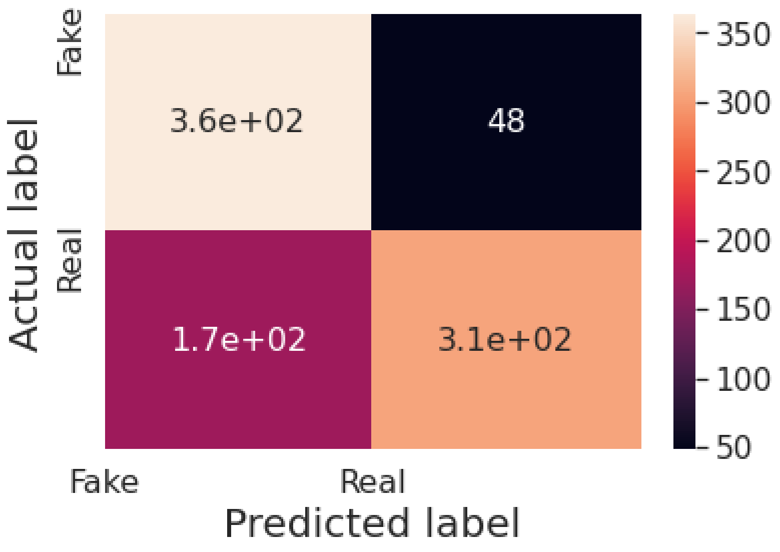


Figure 4.29

From the Confusion Matrix we can come up with following key metrics :

1. Accuracy : 75.45%
2. Misclassification Rate (Error Rate) : 24.55%
3. True Positive Rate (Also known as Sensitivity or Recall) :64.58%
4. False Positive Rate : 11.76%
5. True Negative Rate(Also known as Specificity) : 88.23%
6. Precision : 86.6%
7. Prevalence : 54%

4.5.7 ResNeXt-101-32x8d + LSTM:

In the below figure we see that the training loss is decreasing and validation is slightly increasing and there is quite a difference in training and validation loss.

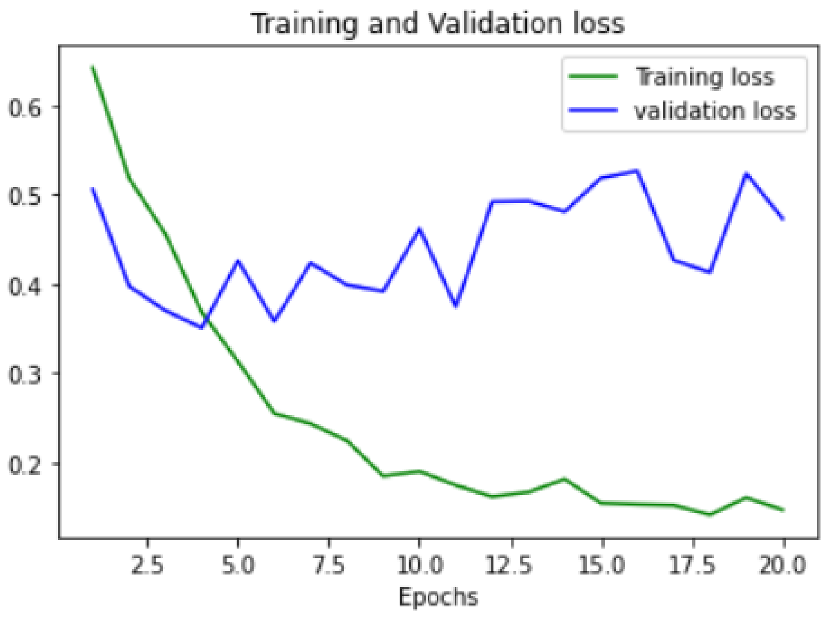


Figure 4.30

In Figure 4.31 , we seethta the training accuracy and validation accuracy follows each other closely after each epoch and the accuracy is increasing as the epoch increase.

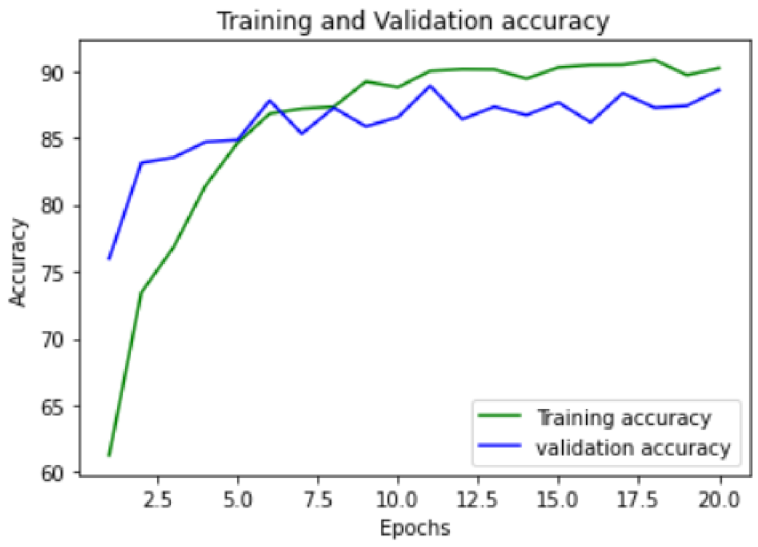
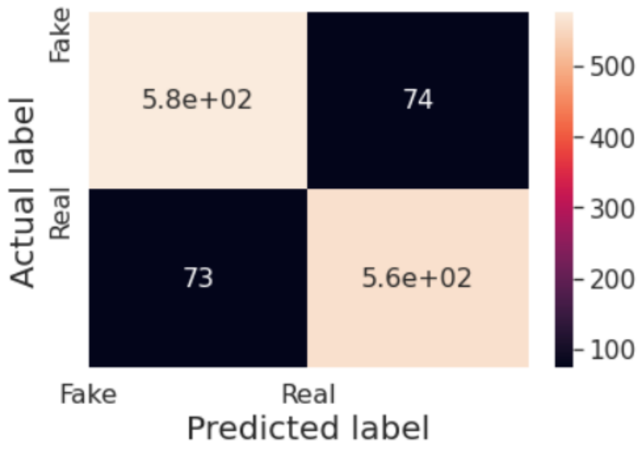


Figure 4.31

Let's look at the confusion Matrix , shown in Figure 4.32



```
Calculated Accuracy 88.56920684292379
```

Figure 4.32

From the Figure 4.32 we can find the following Key Performance Ratio :

1. Accuracy : 88.57
2. Misclassification Rate (Error Rate) : 11.53%
3. True Positive Rate (Also known as Sensitivity or Recall) :88.46%
4. False Positive Rate : 11.31%
5. True Negative Rate(Also known as Specificity) : 88.68
6. Precision : 88.32%
7. Prevalence : 49.18%

CHAPTER 5
RESULT

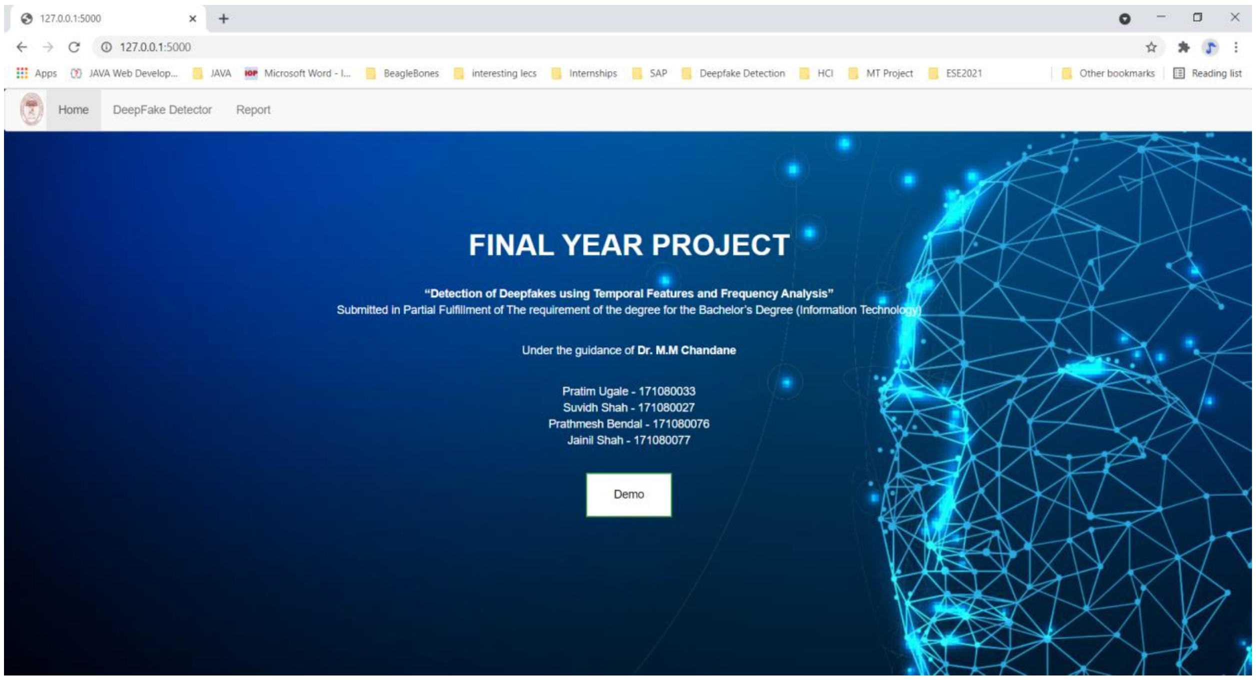
The below table summarises that model architecture used and the corresponding Key Performance Metrics for Binary Classification.

Model Arc	Testing Accuracy	Training Accuracy	Precision	Recall	F1 score
CNN+LSTM	89.56%	90.96%	91.5%	93.47%	92.47
Freq+CNN+LSTM	88.77%	99%	92.2%	85.33%	88.63
Wide_Resnet +LSTM	84.13%	90.18%	85.38%	81.25%	83.26
CNN +Averaging	75.45%	83.82%	86.6%	64.58%	73.93
Resnext_101+LSTM	88.57%	90.22%	88.32%	88.46%	88.38

Table 5.1

In the table above , we have taken 5 key performance metrics which can give us a clear view of our result. F1 score is the harmonic mean of Precision and Recall and it can be considered a better measure than Accuracy. We can see that the 1st Model architecture gives us the best accuracy as well as F1 score in comparison to other models. Where LSTM is not used we can see a significant drop in its accuracy and F1 score, also its Recall is really low as compared to others.

Hence we serve the 1st CNN + LSTM model on the Web-interface, using Flask as backend. The results of the Web Interface are shown below. It is a ready to deploy model and interface, so it can easily be made publicly available to the world.



FINAL YEAR PROJECT

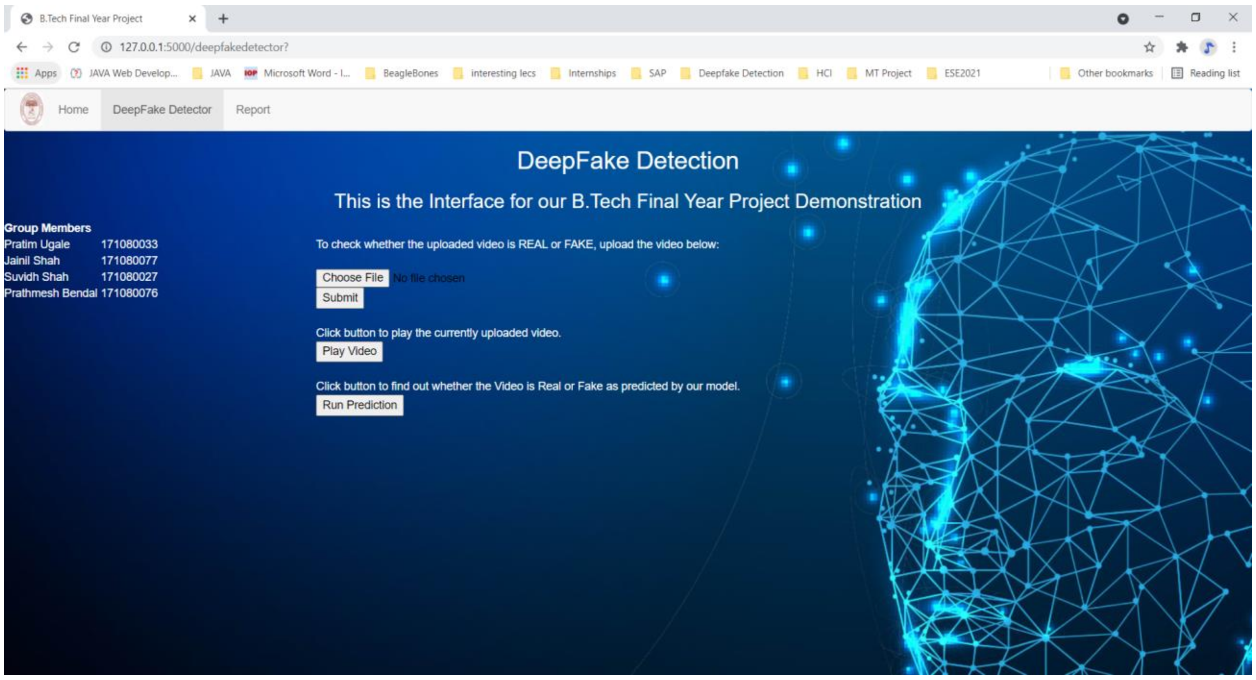
"Detection of Deepfakes using Temporal Features and Frequency Analysis"
Submitted in Partial Fulfillment of The requirement of the degree for the Bachelor's Degree (Information Technology)

Under the guidance of **Dr. M.M Chandane**

Pratim Ugale - 171080033
Suvidh Shah - 171080027
Prathmesh Bendal - 171080076
Jainil Shah - 171080077

Figure 5.1

The user can upload any .mp4 of their choice and play the video on which the prediction is going to be run on the browser.



DeepFake Detection

This is the Interface for our B.Tech Final Year Project Demonstration

Group Members

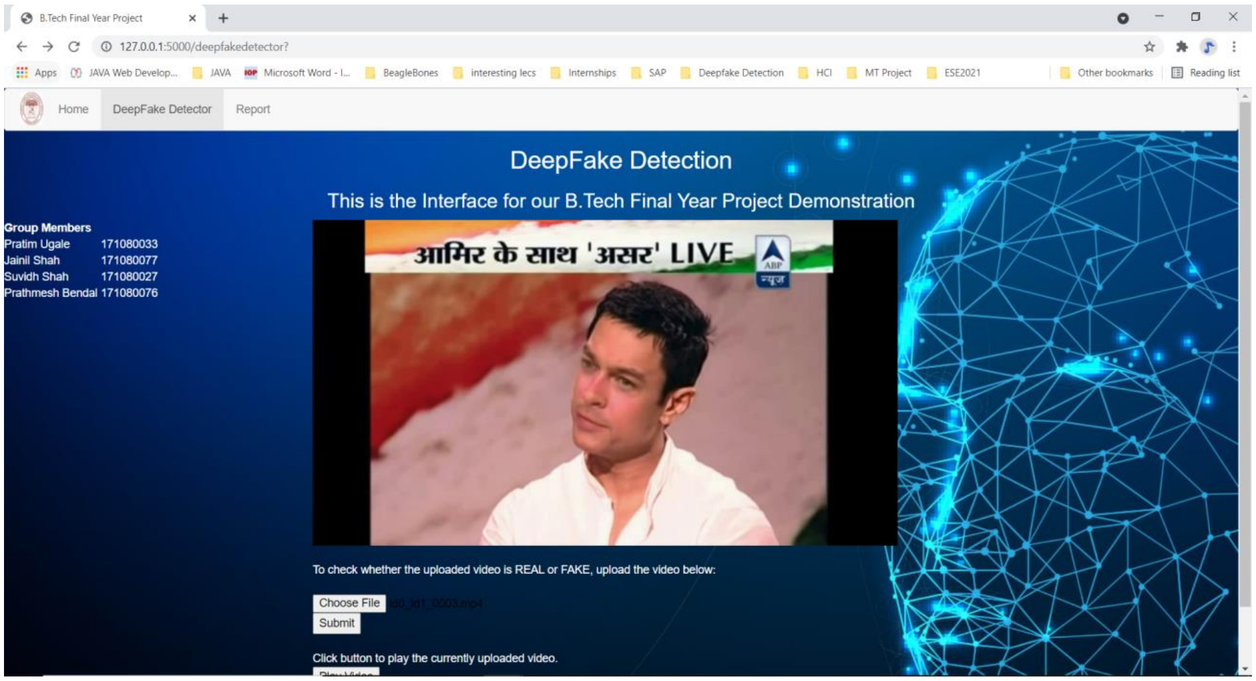
Pratim Ugale	171080033
Jainil Shah	171080077
Suvidh Shah	171080027
Prathmesh Bendal	171080076

To check whether the uploaded video is REAL or FAKE, upload the video below:

Click button to play the currently uploaded video.

Click button to find out whether the Video is Real or Fake as predicted by our model.

Figure 5.2



DeepFake Detection

This is the Interface for our B.Tech Final Year Project Demonstration

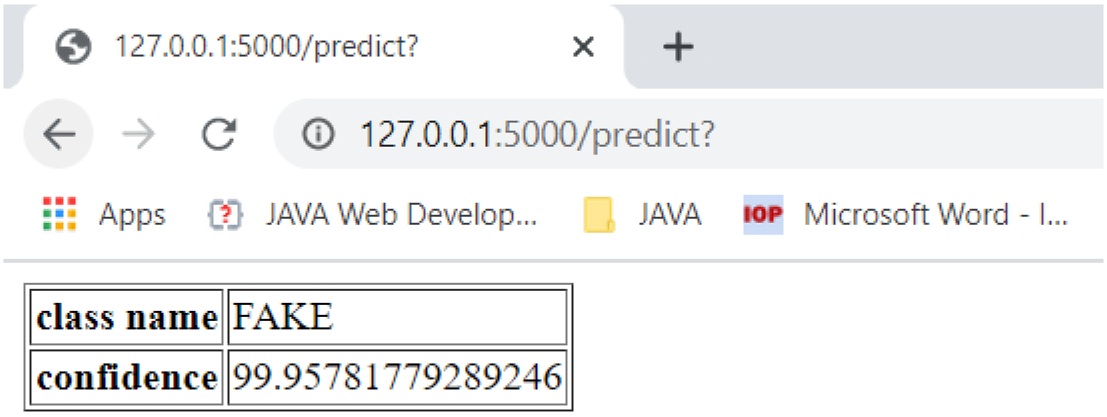
Group Members	
Pratim Ugale	171080033
Jainil Shah	171080077
Suvidh Shah	171080027
Prathmesh Bendal	171080076

To check whether the uploaded video is REAL or FAKE, upload the video below:

Click button to play the currently uploaded video.

Figure 5.3

On clicking the Run Prediction button, the Flask server will load the trained model, start preprocessing the video and then evaluate the on this video input. After the results are obtained, they are returned to the user in JSON format and are here formatted in a table.



class name	FAKE
confidence	99.95781779289246

Figure 5.4

CHAPTER 6
CONCLUSION

We built a [32]CNN + LSTM model using the Resnext CNN as the feature extractor with sequence length of 20 frames which gives us a test accuracy of 88%. The method takes advantage of the way the deepfakes are created frame-by-frame by GANs and Autoencoders. Hence this model can serve as a good first line of defence if one has a doubt if their video has been manipulated.

Adding frequency features to this model does not result in an increase in accuracy or robustness, instead it starts overfitting the training data. Additionally, the frequency spectrum method does not work on images(frames) gathered from the DFDC[7] and Celeb-df[8] dataset as well. So it should be used only if the dataset is following the characteristics as described in the paper. Also the datasets used in the paper use more easily identifiable deepfakes to the eye as compared to the DFDC dataset that we use which is why frequency features are not providing additional information.

We have created a ready to deploy web-interface on which a user can upload a video to find if it is detected as pristine or fake, along with the confidence with which the prediction is made.

FUTURE WORK

Currently we use only the visual part of a DeepFake video to extract features and create a classification model. We can further use a similar RNN-based approach to detect audio tampering, and the results of the two audio and video models could be combined for a complete end-to-end DeepFake video detection model. Instead of using RNN after CNN, new methods use 3DCNN which is capable of detecting temporal inconsistencies by itself so we can test that to check which model generalizes better.

References :

1. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.
2. Noh, H., Hong, S. and Han, B., 2015. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision* (pp. 1520-1528).
3. Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y., 2016. *Deep learning* (Vol. 1, No. 2). Cambridge: MIT press.
4. Mehdipour Ghazi, M. and Kemal Ekenel, H., 2016. A comprehensive analysis of deep learning based representation for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (pp. 34-41).
5. Korshunov, P. and Marcel, S., 2018. Deepfakes: a new threat to face recognition? assessment and detection. *arXiv preprint arXiv:1812.08685*.
6. Güera, D. and Delp, E.J., 2018, November. Deepfake video detection using recurrent neural networks. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)* (pp. 1-6). IEEE.
7. Dolhansky, B., Howes, R., Pflaum, B., Baram, N. and Ferrer, C.C., 2019. The deepfake detection challenge (dfdc) preview dataset. *arXiv preprint arXiv:1910.08854*.
8. Li, Y., Yang, X., Sun, P., Qi, H. and Lyu, S., 2020. Celeb-df: A large-scale challenging dataset for deepfake forensics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 3207-3216).
9. Dolhansky, B., Bitton, J., Pflaum, B., Lu, J., Howes, R., Wang, M. and Ferrer, C.C., 2020. The deepfake detection challenge dataset. arXiv preprint arXiv:2006.07397.
10. Durall, R., Keuper, M., Pfreundt, F.J. and Keuper, J., 2019. Unmasking deepfakes with simple features. *arXiv preprint arXiv:1911.00686*.
11. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z., 2016. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).
12. Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, *9*(8), pp.1735-1780.

13. Laptev, I., Marszalek, M., Schmid, C. and Rozenfeld, B., 2008, June. Learning realistic human actions from movies. In *2008 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1-8). IEEE.
14. Xiang, J. and Zhu, G., 2017, July. Joint face detection and facial expression recognition with MTCNN. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)* (pp. 424-427). IEEE.
15. Weinstein, S. and Ebert, P., 1971. Data transmission by frequency-division multiplexing using the discrete Fourier transform. *IEEE transactions on Communication Technology*, *19*(5), pp.628-634.
16. Cortes, C. and Vapnik, V., 1995. Support-vector networks. *Machine learning*, *20*(3), pp.273-297.
17. Kleinbaum, D.G., Dietz, K., Gail, M., Klein, M. and Klein, M., 2002. *Logistic regression*. New York: Springer-Verlag.
18. Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R. and Wu, A.Y., 2002. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, *24*(7), pp.881-892.
19. Montserrat, D.M., Hao, H., Yarlagadda, S.K., Baireddy, S., Shao, R., Horváth, J., Bartusiak, E., Yang, J., Guera, D., Zhu, F. and Delp, E.J., 2020. Deepfakes detection with automatic face weighting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (pp. 668-669).
20. Tan, M. and Le, Q., 2019, May. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning* (pp. 6105-6114). PMLR.
21. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L., 2009, June. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255). Ieee.
22. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.
23. Deng, J., Guo, J., Xue, N. and Zafeiriou, S., 2019. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4690-4699).

24. Li, Y., Chang, M.C. and Lyu, S., 2018. In ictu oculi: Exposing ai generated fake face videos by detecting eye blinking. *arXiv preprint arXiv:1806.02877*.
25. Li, Y. and Lyu, S., 2018. Exposing deepfake videos by detecting face warping artifacts. *arXiv preprint arXiv:1811.00656*.
26. Bonettini, N., Cannas, E.D., Mandelli, S., Bondi, L., Bestagini, P. and Tubaro, S., 2020. Video face manipulation detection through ensemble of cnns. *arXiv preprint arXiv:2004.07676*.
27. Zhu, K., Wu, B. and Wang, B., 2020, July. Deepfake Detection with Clustering-based Embedding Regularization. In *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)* (pp. 257-264). IEEE.
28. Rossler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J. and Nießner, M., 2019. Faceforensics++: Learning to detect manipulated facial images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 1-11).
29. Karras, T., Laine, S. and Aila, T., 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4401-4410).
30. Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A., 2017, February. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 31, No. 1).
31. Hitawala, S., 2018. Evaluating ResNeXt Model Architecture for Image Classification. *arXiv preprint arXiv:1805.08700*.
32. Lawrence, S., Giles, C.L., Tsoi, A.C. and Back, A.D., 1997. Face recognition: A convolutional neural-network approach. IEEE transactions on neural networks, 8(1), pp.98-113.