Fall 12-17-2020

Quantifying DeepFake Detection Accuracy for a Variety of Natural Settings

Pratikkumar Prajapati
San Jose State University

Quantifying DeepFake Detection Accuracy for a Variety of Natural Settings.

A Project Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Pratikkumar Prajapati

Dec 2020

The Designated Project Committee Approves the Master's Project Titled

Quantifying DeepFake Detection Accuracy for a Variety of Natural Settings.

by

Pratikkumar Prajapati

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY (SJSU)

Dec 2020

Dr. Chris Pollett, Department of Computer Science, SJSU

Dr. Mark Stamp, Department of Computer Science, SJSU

Dr. Teng Moh, Department of Computer Science, SJSU

ABSTRACT

Deep fakes are videos generated from a starting video of a person where that person's face has been swapped for someone else's. In this report, we describe our work to develop general, deep learning-based models to classify Deep Fake content. Our first experiments involved simple Convolution Neural Network (CNN)-based models where we varied how individual frames from the source video were passed to the CNN. These simple models tended to give low accuracy scores for discriminating fake versus non-fake videos of less than 60%. We then developed three more sophisticated models: one based on choosing test frames, one based on video Optical Flow, and one that uses Generative Adversarial Networks (GANs) to determine structural differences in images. This last technique we call MRI-GAN and is new to the literature. We tested our models using the Deep Fake Detection Challenge dataset and found our plain frames-based model achieves 90% test accuracy, our MRI model achieves 79% test accuracy, and Optical Flow-based model achieves 69% test accuracy.

Index terms: Deep learning, GAN, MRI-GAN, DeepFakes, Convolution Neural Networks (CNNs), Machine learning, Artificial Intelligence.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my special appreciation and thanks to my advisor Dr. Chris Pollett for his continuous support of my project, for his enthusiasm, motivation, and deep knowledge. He has been very supportive since day one. He has guided me to make this complex project a great learning experience as well as enabled me to contribute to the research of the Artificial Intelligence field.

Besides my advisor, I would like to thank the rest of my project committee: Dr. Mark Stamp and Dr. Teng Moh, for their encouragement, guidance, and valuable feedback.

Finally, I would like to thank my wonderful family for their countless hours of support and encouragement along the way.

TABLE OF CONTENTS

ABSTRACT 4

ACKNOWLEDGEMENTS 5

Table of Contents 6

List of Figures 8

List of Tables 9

1. Introduction 10

2. Background 12

    2.1. Artificial Neural Networks 12

    2.2. Common Face Synthesis Methods. 16

    2.3. Prior work 18

3. Hardware, Frameworks, and Tools 19

4. Design 21

    4.1. Overall design and pipeline of the data flow. 21

    4.2. Variants of different features. 23

    4.2.1. Plain images as features 24

    4.2.2. Optical Flow as features 24

    4.2.3. Learning to generate MRIs using proposed MRI-GAN 26

    4.2.3.1. Structural Similarity Index Measurement (SSIM) 26

    4.2.3.2. MRI-GAN 29

4.3. Model architecture 32

5. Implementation 34

5.1. Source code 34

5.2. Dataset 34

5.3. Data preprocessing 35

5.4. Model implementation 41

5.5. Aggregation algorithm 41

5.6. Replication steps 42

5.6.1. Data preprocessing 43

5.6.2. Features extraction 44

5.6.2.1. Optical Flow features extraction 44

5.6.2.2. MRI features extraction 44

5.6.3. Model execution 45

6. Experiments and Results 46

7. Limitations and Future work 52

8. Conclusion 54

List of References 56

LIST OF FIGURES

Figure 2.1 Comparison of various CNN-based models 14

Figure 2.2 Samples of common face synthesis methods 17

Figure 4.1 Overall flow of the design using plain frames 23

Figure 4.2 Optical Flow for original (left) and DeepFake (right) 24

Figure 4.3 Samples of Optical Flow for consecutive frame 25

Figure 4.4 Architecture of the Optical Flow-based model 26

Figure 4.5 Example MRI images of the fake samples 30

Figure 4.6 Samples generated by MRI-GAN 31

Figure 4.7 Architecture of the MRI-based model 32

Figure 4.8 Model architecture 33

Figure 5.1 Data preprocessing methods 36

Figure 5.2 Examples of augmentations and distractions 40

Figure 5.3 The Aggregation algorithm 42

Figure 6.1 Plot of train and validation accuracies for all models 46

Figure 6.2 Plot of train and validation losses for all models 47

Figure 6.3 Plot of accuracies with different aggregation parameters for all models 49

Figure 6.4 Plot of ROC curves for the test of all models 49

Figure 6.5 Confusion matrixes, left: for plain frames, middle: for Optical Flow, right: for MRI 50

LIST OF TABLES

Table 3.1 Hardware configuration of the workstation 19

Table 5.1 Distribution of various data augmentation and distractions methods applied. 38

Table 6.1 Thresholds matrix for aggregating frame-level results. 48

Table 6.2 Various matrices for all models. 51

1. INTRODUCTION

DeepFakes are synthesized images or videos that look like real content. Many DeepFakes can be found on www.youtube.com such as [1]. Fake videos of many political leaders, cinema actors, and other celebrities are generated to spread propaganda or to diminish their social reputation. Imagine a fake video of a political leader announcing something horrific. Such things can lead to catastrophic outcomes in society. Synthesized fake content can be a threat to society when seeing does not translate to believing [2]. In this research, we explore ways to detect such fake content. Our goal is to quantify the accuracy of DeepFake detection for a variety of natural settings using deep learning-based techniques.

Fake media can be of many kinds. One of such kind is swapping the identity of one person with another. This specific identity swap method is called DeepFake. We briefly explain other kinds of fake contents generation methods later in this report. DeepFakes are generally created with specific kinds of neural networks called Generative and adversarial Networks (GANs). While a kind of neural network architectures such as Autoencoders (AEs) are also used, GAN-based models can generate realistic looking DeepFakes [3, 4]. GANs, AEs, and variants of such neural networks are deep learning subareas of machine learning methods in the Artificial Intelligence domain. Machine learning methods such as deep learning need a very high amount of data and computation to find patterns and make meaningful insights into the data. Deep learning-based models can be trained in a supervised manner to learn some specific kind of universal approximations to translate data into some insightful information.

We propose a generic model that does not rely on a specific DeepFake generation method and tries to generalize the DeepFake detection holistically. We have used DeepFake Detection Challenge (DFDC) dataset [7] in this research to train our proposed models. DFDC dataset is

chosen for this research as it contains DeepFake samples generated with many state-of-the-art methods, contains subjects with rich ethnic diversity, and also the video samples are recorded in a variety of natural settings.

In this report, Chapter 2 builds the background to explain an overview of neural networks, common methods used to generate synthetic contents and explore prior work done in the DeepFake detection problem space. Then in Chapter 3, we mention the hardware, frameworks, and tools used to develop the code and utilized to perform various experiments. Further, we explain the design of our models and various regularization techniques used to improve the efficiency of our model in Chapter 4. We have proposed a novel approach to detect DeepFake by introducing MRI-GAN. We explain this architecture in detail with the mathematical background, in Chapter 4. We have extended the idea of [10] in the MRI-GAN and proposed new way to learn features of the fake content. We also used Optical Flow method discussed in [22] to detect the DeepFakes. After that, in Chapter 5 we explain the implementation of the design with links to access the source code of this research, various methods of data preprocessing we applied to train the model to generalize the predictions, and steps to replicate our work. Then we explain the various experiments we performed and quantify our results in Chapter 6. Later, we specify the limitations of our approaches and future work related to that in Chapter 7. Finally, we end the discussion with our conclusion in Chapter 8.

2. BACKGROUND

In this Chapter, we briefly explain the technical background needed to understand the research performed in this report. This explanation serves as an overview only for the technical concepts. Details of these concepts can be explored by visiting references mentioned in the Chapter. First, we begin with an overview of various kinds of neural networks as they are used extensively to implement the algorithms we proposed. Then we explain common kinds of face synthesis methods used to understand the overall spectrum of fake contents and the scope of our work. At last, we review prior work done by different researchers in the DeepFake detection problem space.

2.1. Artificial Neural Networks

Perceptrons are building blocks of Artificial Neural Networks (ANN). A perceptron has one or more inputs with associated weights and an output. The perception multiplies the weights with the corresponding inputs and adds them together. This operation is also represented as a dot-product of weight and input vectors. The output of the dot-product is given to an activation function before generating the output result. The activation function can bound the inputs and help the perceptron learn a non-linear relationship between the inputs. Sigmoid, ReLu, Tanh, leaky ReLu are some of the many activation functions. A bias term is also given as an additional input to the perceptron. The bias is always ON and has its own weight, it helps to shift the input function and enable the model to be more flexible about the learned function. ANN consists of one or more perceptrons. The output of one perceptron can be given as input to the next perceptron to form series or layers of the perceptron. One or more perceptrons can be connected back-to-back, and in parallel with each input vector and some combinations of it to learn

complex non-linear function. This can be roughly considered as a directed acyclic graph. This architecture is collectively referred to as Multi-Layer Perception (MLP) [11].

While MLP provides good results on many of the classification tasks, it has its own limitations. One of the core limitations of MLP is that it needs a huge number of perceptrons to achieve some reasonable amount of accuracy and in many of the complex problems, such as computer vision, it fails to get any acceptable accuracy. The main reason the MLP does not work well with images is that it does not learn geometric information of the subjects in images. Convolution Neural Network (CNN) was proposed as an attempt to solve these limitations. CNNs uses convolution operation with kernels of specified dimension as filters to learn geometric information of the subjects. Kernels are used as a sliding window with specified strides and learn to detect many features of the inputs. Usually, max-pooling or average-pooling layers are also applied after the convolution and activation functions. CNNs are applied on 1D, 2D, and 3D data. There are many variations of the CNNs proposed by researchers. Some of the variations downsample the inputs, upsamples the inputs, and some large architectures with hundreds of layers are also proposed. CNNs have proven to achieve better accuracy in computer vision problems with a lower number of model parameters compared to MLP [11]. ImageNet dataset is commonly used for benchmarking various CNN architectures, as this dataset is very rich in diversity of images and 1000 classes of images are provided with it. ResNet-50, Xception Net, ResNext-101 are a few of the very large and complex architectures claimed to achieve state-of-the-art accuracy with the ImageNet dataset. These architectures have millions of parameters needed for training.

Design of the CNNs brings a new set of challenges, which include tuning the depth and width of the layers with respect to the resolution of the images. The increasing complexity of the model results in vanishing gradients problems and it is not guaranteed that complex architectures will

perform better [11, 12]. To overcome the vanishing gradient problem, the U-Net architecture was proposed which has skip-connections to transfer gradient from one layer to some other layer directly and skipping some intermediate layers [13]. Tan M. et al. [12] proposed a set of architectures called Efficient-Nets which is claimed to achieve accuracy comparable to complex architectures with relatively a smaller number of parameters. Tan M. et al. have proposed eight different models called B0 to B7. Figure 2.1 shows the comparison of the accuracy of these models with respect to model parameters of recent state-of-the-art architectures [12].

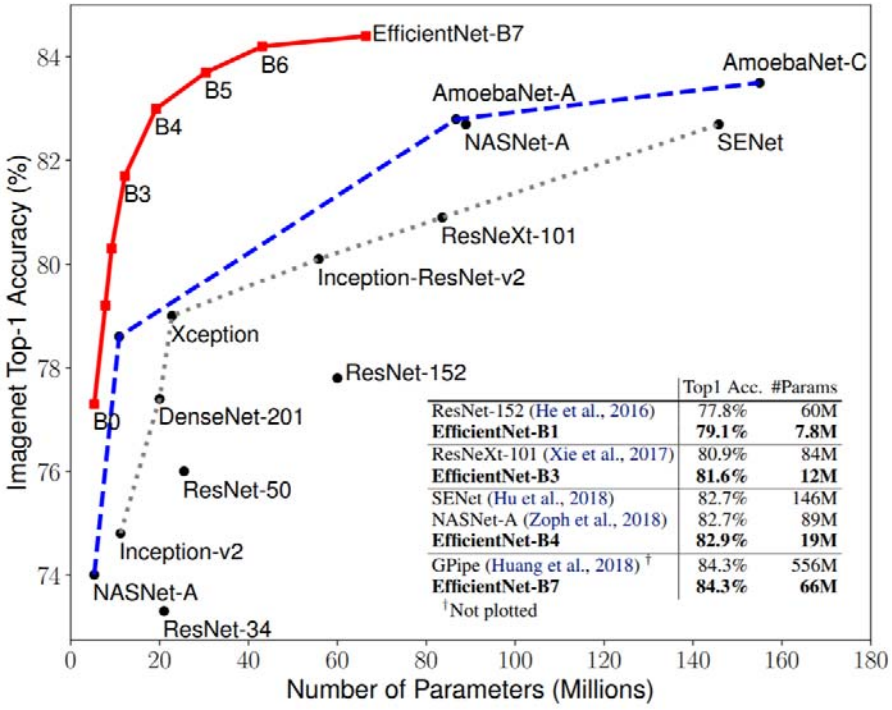

Figure 2.1 Comparison of various CNN-based models

We have used Efficient-Net B0 in our research to extract the feature-maps of the images. Model weights are given with the model architecture and we use this pre-trained model to utilize the

transfer-learning aspect of deep learning. We explain the architecture of our model in detail in the Chapter 4.

Input is forwarded to various layers before getting the final output in the MLP and CNN-based model. They do not utilize a feed-back loop kind of mechanism; hence these kinds of models are collectively referred to as feed-forward networks. To handle a sequence of input data or time-series data, a new set of models are proposed which uses temporal information of the inputs and passes output as one of the inputs to the model. Such models are called Recurrent Neural Networks (RNNs). These kinds of models can be roughly considered as cyclic graphs. These models employ memory cells to remember previous inputs in addition to current inputs to predict the output. Long Short-Term Memory (LSTM) and Gated-Recurrent Unit (GRU) are some of the examples of such models [11]. We have used LSTM in one of our models to learn temporal discrepancy between a sequence of frames to detect DeepFake videos. Details of these models are only briefly discussed in our report, as our attempt to use LSTM did not produce acceptable results.

Yet other kinds of architectures are invented to learn to generate images. Autoencoders (AEs) and Generative adversarial Networks (GAN) some of the popular model architectures. Powerful architectures of AEs, GANs, and variants of these models are capable to generate a new set of media contents that can be fake in nature but looks realistic to humans. This side-effect of technology is one of the sources of the rise of DeepFakes and associated threats. Design of AEs has encoder and decoder sub-modules. Images are encoded into a latent space by downsampling the features using the encoder module. Latent space is generally in a smaller dimension than input and it can represent encoded feature maps of the input images. Decoders are trained to sample from this latent space and upsampled to generate the images of the targeted dimension.

GAN-based models have two main modules called generator and discriminator. Utilizing game theory, both of these components are trained in a zero-sum game. The role of the generator is to sample vectors from a latent space and learn to generate an image that looks realistic to the discriminator. The role of the discriminator is to classify the images generated by the generator is from the training set (which gets considered as a real image from the discriminator's point of view) or not. While both modules of the networks are trained it's expected that the overall model will reach equilibrium at some time where the generator is able to generate realistic-looking fake images and able to fool the discriminator such that the discriminator fails to classify if the image generated is real or fake. Training GAN models are very difficult [11, 14]. There are many variants of the GANs proposed by researched to solve a specific set of problems. One such architecture is called Pix2Pix GAN [15] which is designed to transform the domain of one image sets into another one. Pairs of images are given as inputs to the model and Pix2Pix GANs are trained to transform one component of the pair to the other. We have utilized this architecture and trained a novel model, called MRI-GAN, which generates the MRI of any human face and it is used to detect if the face is synthesized or real. We have explained the details of the MRI-GAN in our report.

In this subsection, we have briefly explained some of the common and very powerful architectures of neural networks. In the next subsection, we explain some of the common methods used in synthesizing fake faces and prior work done in detecting the DeepFakes.

2.2. Common Face Synthesis Methods.

Entire face synthesis, identity swap, face attributes manipulation, and facial expression manipulation are four common types of face synthesis methods. Entire none-existent faces are generated using AEs and GANs. The generated faces are completely fictitious yet resembles a

human face. Identity swap methods use source and target images. The face of the source image is used to replace the face of the target image while maintaining the background environment of the target image. Fake contents generated using this method is also known as the DeepFakes. The primary scope of our project is to detect if a video is DeepFake or not. With facial attribute manipulation methods, the identity of the face is not changed but some attributes of the faces like hair or skin tone, age, etc. are changed. Facial expression manipulation techniques take source and target images as inputs and generate a fake image with a facial expression of source image imposed on the target images [3]. See Figure 2.2 for the samples of each manipulation method [3].

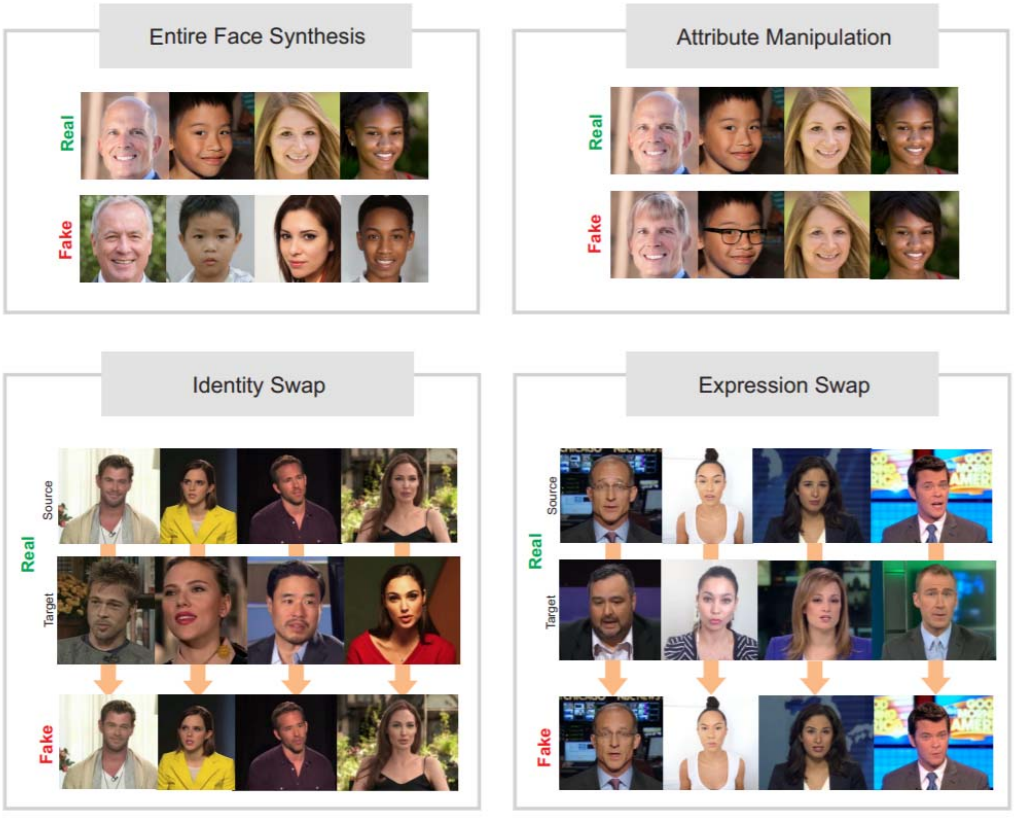

Figure 2.2 Samples of common face synthesis methods

2.3. Prior work

DeepFakes are generally created with off-the-shelf models including but not limited to StyleGAN [16], CycleGAN [17], FaceSwapGAN [18] and sometimes combined with some post-processing of the images and videos such as color balancing to make the fake video look realistic. There have been quite a few researches done on detecting DeepFakes. Many of them used different matrixes to report the outcome such are accuracy, Area Under the Curve (AUC), Equal Error Rate (EER), etc. and they used different datasets so it's hard to compare all such researches [3]. Facebook has launched the DeepFake detection challenge (DFDC) [19] and also provided a dataset for the competition [7]. We have used this same dataset in our research. The winner of the competition [19, 20] achieved 82% of the test accuracy on the complete DFDC test dataset. We have inherited some ideas such as frame-based detections from the winning solutions and greatly extended the idea with new features and a different classification model. Rossler et al. Tang et al. [5] used mesoscopic and deep learning features using CNN on Face Forensics (FF)++ FaceSwap Low Quality (LQ) dataset [21] and achieved around 94% accuracy. Sabir et al. [6] used temporal information of the video frames such as discrepancies between consecutive frames with CNN and RNN based models and achieved 96.3% AUC on the FF++ FaceSwap, LQ dataset. Amerini et al. [22] used Optical Flow fields on fake media generated with Face Manipulation techniques and achieved 81.6% accuracy. We extended this idea to generate the Optical Flow between consecutive frames of each identified person in the fake video and used it as a feature for our model to classify the DeepFake media. Li et al. [10] used x-ray of the images as features to detect DeepFakes. They used the DFDC dataset and reported an AUC of 80.92. We have extended this idea and proposed a novel method to generate structural dissimilarity of images and we call it Magnetic Resonance Imaging (MRI) of the images.

3. HARDWARE, FRAMEWORKS, AND TOOLS

Deep learning-based projects are very compute-intensive for data processing and training. In this Chapter, we explain the configuration of the hardware, framework, and tools we utilized to achieve the overall development and various experiments.

The implementation of the overall project is based on Python. We used Pandas, Numpy, Matplotlib, Sklearn, facenet_pytorch, and many common python libraries throughout the project. The PyTorch framework is used to implement ANN models. For video file manipulation we used FFMPEG. The development is done on the Ubuntu 18.04 operating system. The hardware configuration of the workstation is mentioned in Table 3.1.

Table 3.1 Hardware configuration of the workstation

CPU	Brand and model	Intel i9-9940X
	Clock Frequency	3.30GHz
	Num of threads	28
	Cache	19.25 MB Intel Smart Cache
CPU Liquid Cooling	Brand and model	Corsair Hydro Series H115i PRO
	Fan Speed	1200 RPM
	Fan size	140mm
	radiator size	280mm
DRAM	Brand and model	Corsair CMK32GX4M2A2666C16

	Speed	2666MHz
	Capacity	16x 8GB = 128GB
Motherboard	Brand and model	Asus WS x299 sage
GPU	Brand and model	Nvidia Titan RTX
	Total Video Memory	24 GB GDDR6
	Tensor Cores	576
	CUDA Cores	4608
	Base Clock (MHz)	1350 MHz
	Single-Precision Performance	16.3 TFLOPS
	Tensor Performance	130 TFLOPS
Primary Storage	Brand and model	Sabrent 2TB Rocket NVMe
	Read Speed	3400 MB/s
	Write Speed	2750 MB/s
Secondary Storage	Brand and model	Seagate IronWolf 12TB HDD
	Read Speed	150 MB/s
	Write Speed	100 MB/s

4. DESIGN

In this Chapter, we discuss the design of the models we implemented to detect the DeepFake videos and various methods performed for feature engineering. This is critical to understand the core ideas we have applied to classify the DeepFakes. We have tried various model architectures using RNNs and CNNs. We pass a set of ordered frames to RNN and the output of the RNN is passed to the classifier for final prediction of the video. We tried some combinations of parameters such as ten consecutive frames, twenty constitutive frames, one and two layers of LSTMs, dropout layer with 50% probability, and label smoothing. These all resulted in accuracy of less than 57%. We also tried several experiments with CNN-based models, such as pass only the first frame to the model, pass three consecutive frames placed next to each other as a single feature vector, using custom CNN-based model, and using features extraction utilizing Efficient-Net B0. These all also resulted in less than 60% test accuracy. One particular architecture based on CNN worked well for the DFDC dataset, we explain this model in detail. We also experimented with extracting different features from frames and used the same model to classify DeepFake detection. Since other model architectures did not produce any reasonable accuracy, we omit them in our discussion. Now we begin by explaining the overall design of our best performing CNN-based model and pipeline of the data flow. Then we discuss various techniques used to extract the features from the videos to help the model learn better and finally we explain the architecture of the CNN-based model. Only the frames of the video are used and the audio stream of the videos is dropped in our experiments.

4.1. Overall design and pipeline of the data flow.

Figure 4.1 shows the overall core design. We begin with a video sample in Step 1. Then we extract frames from the video in Step 2. We used pre-trained MTCNN [23] in Step 3 to locate the

faces of the humans in each frame. There could be more than one person in each frame. Detected faces are cropped from the frames and saved on disk. We repeat this process for each video of the dataset. At the end of Step 3, we have a set of frames for all videos. These frames contain only the faces of humans. The faces could be either fake or real. Then in Step 4, we use our model to predict if each of these frames is of a fake or real person. In the model, each frame is passed to the pre-trained Efficient-Net B0 model to extract the features. These features are passed to a custom classifier to predict if the given frame is fake or real. In Step 5 we need to aggregate the results of each frame and at last in Step 6 we need to predict if the given video is fake or real. There could be multiple ways to aggregate the results of individual frames to predict the overall video classification. E.g., if 50% of the frames are detected as fake with a probability of more than 60% then we predict the entire video as fake. In the Implementation Chapter, we discuss our algorithm to translate frame-level classification to overall video-level classification. Also due to limitations of computing resources we only extracted every tenth frame of each video, rather than all the frames. In the report, we use the term 'consecutive frames' to refer to every tenth frame. There could be more than one face in the video and so our classification approach is to count how many fake faces are detected in the entire video. If there is more than one face in a frame each face is detected independently and passed to the classifier to detect if the face is fake or real. A final decision of the entire video is made using aggregation of each face-based classification.

We choose the Efficient-Net B0 model to extract the features of the images as it achieves the accuracy of the current state-of-the-art models and uses much less computational resources [12]. The output of the Efficient-Net B0 is passed to the average pooling layer and then to the

classifier. The classifier is an MLP-based model with the output of a single logit to perform binary classification. We explain the architecture of the model later in this Chapter.

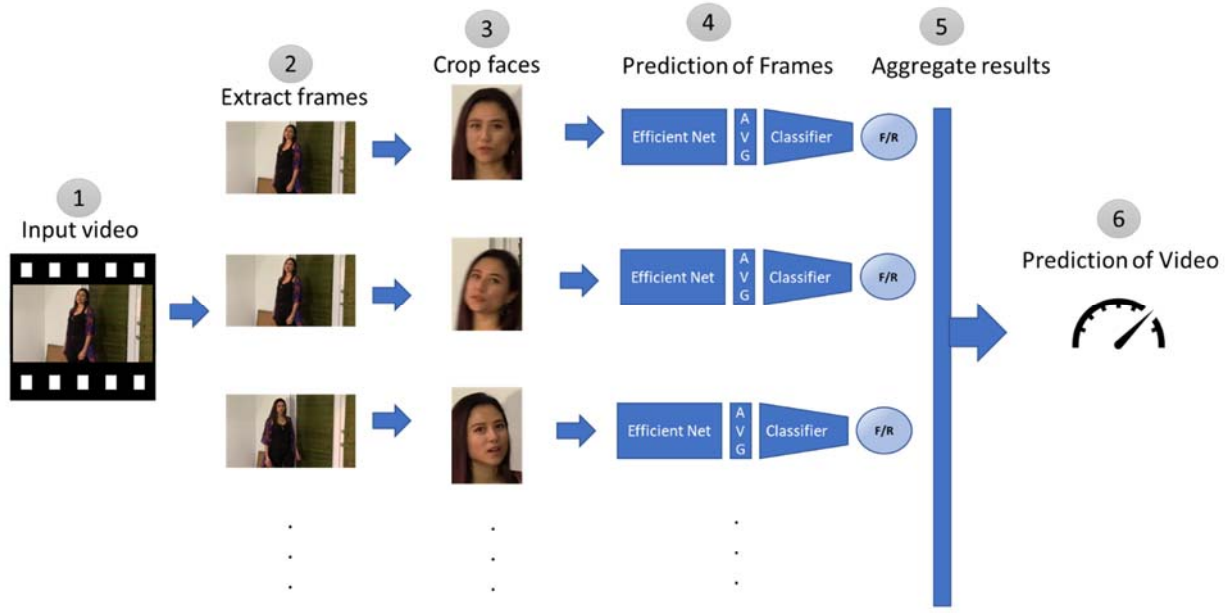

Figure 4.1 Overall flow of the design using plain frames

4.2. Variants of different features.

The pipeline discussed in Figure 4.1 shows the flow of 'plain frames'. We have applied data augmentation statically and dynamically, which is discussed in the Data preprocessing subsection of the Implementation Chapter. We also extracted features such as Optical Flow and Magnetic resonance imaging (MRI) of the images after Step 3 and before passing them to our model in Step 4. We borrowed the term MRI from medical science. Now we discuss all three variants of features we have used to design our architecture.

4.2.1. Plain images as features

This is the simplest of all the features design. We do not extract any feature explicitly from the frames after Step 3. We only apply dynamic augmentation after Step 3 and then pass the frames to our model. Figure 4.1 shows this design flow.

4.2.2. Optical Flow as features

Amerini et al. [22] experimented with extracting the Optical Flow of the images to detect Face manipulations. We have adopted the same idea but applied it to detect DeepFakes. According to [22], Optical Flow is a vector field which is computed on two consecutive frame *f(t)* and *f(t+1)* to extract apparent motion between the observer and the scene itself. They claim that the Optical Flow can exploit discrepancies in motion across frames synthetically created. Figure 4.2 shows an example of the Optical Flow of fake and real frames [22].

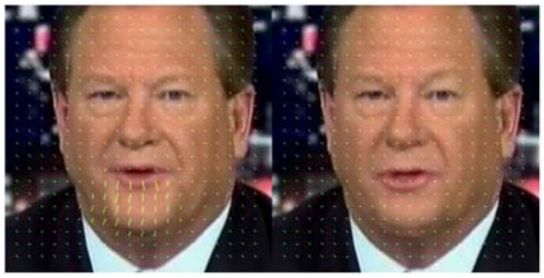

Figure 4.2 Optical Flow for original (left) and Fake (right) video

We created new features using Optical Flow with similar methods. We take two consecutive frames and find the Optical Flow from frame *f(t)* to *f(t+1)* using model [24]. This Optical Flow is a set of vectors. We use the Flowiz library [25] to covert these vectors to images. The Flowiz uses the direction and dimension of the vectors to generate various hues of the image. Figure 4.3 shows an example of the Optical Flow, the top row has two consecutive frames on the left and

the corresponding Optical Flow is on the right. Same for the bottom row with the other two consecutive frames.

Figure 4.3 Samples of Optical Flow for consecutive frame

With the Optical Flow variant, we use the same pipeline as described in Figure 4.1, but we pass images of Optical Flow at Step 3 instead of 'plain frames'. Figure 4.4 shows the end-to-end flow for the Optical Flow design. In addition to architecture defined in Figure 4.1, we have Step 3a and Step 3b depicted in Figure 4.4. At Step 3a we use two consecutive frames and generate an Optical Flow image. At Step 3b we pass these images for dynamic augmentation and then to our model for classification.

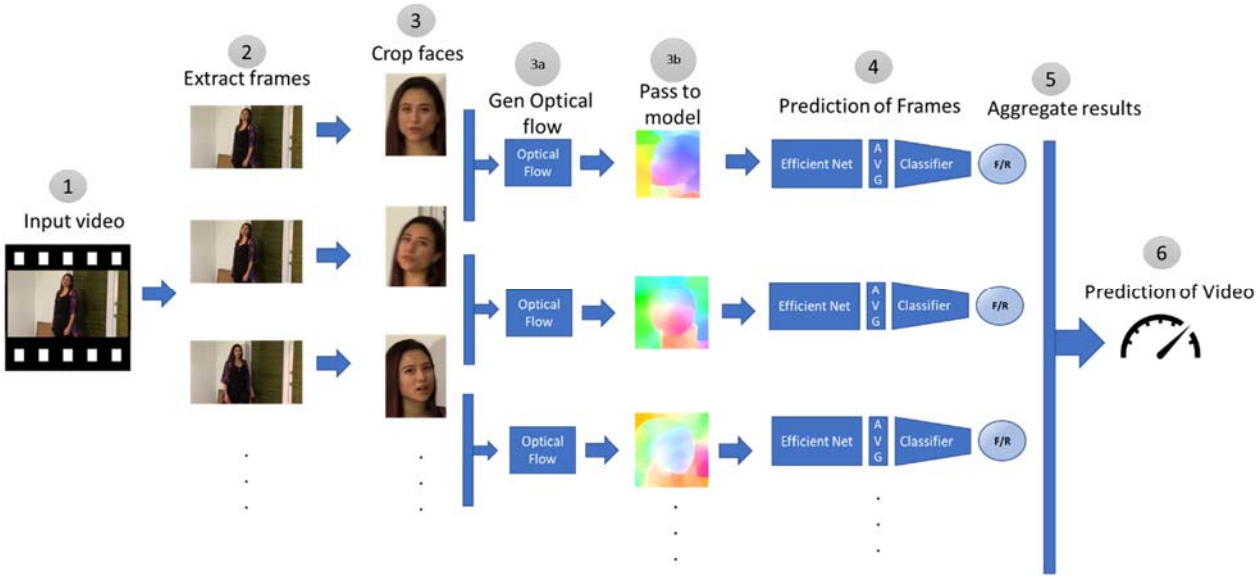

Figure 4.4 Architecture of the Optical Flow-based model

4.2.3. Learning to generate MRIs using proposed MRI-GAN

This is the most complex feature we have utilized and needs extensive explanation. MRI term is used as an enhancement to the x-ray technique discussed in [10]. Facial masks are learned for fake images and these masks are used to detect the fake images. We extended this idea to learn structure similarity index. Before we propose our novel architecture of MRI-GAN, we explain how two images can be compared effectively. Zhou et al. [8] and Wang et al. [9] shows that structural similarity considers texture of the images as well, unlike Mean Squared Error (MSE). We utilized this idea in our MRI-GAN.

4.2.3.1. Structural Similarity Index Measurement (SSIM)

The core idea behind generating MRI is finding the structural difference between a set of images. SSIM is proposed by [8] and we have provided a brief discussion here from the same paper. The normalized Structural Similarity Index between two images is a value in the range of 0 to 1. Where 0 indicates both images are very different and 1 indicates both images are very similar.

While the Structural Similarity Index is calculated with the proposed set of mathematical equations, the actual SSIM image can also be derived. Zhou et al. [8] have proposed to use similarity measurement using three components: luminance, contrast, and structure. Let's define these terms first and then we need to compare these terms between the set of images to get the SSIM image.

a. Luminance

Luminance μ_x is defined as the mean intensity over all pixel values.

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \quad \text{… Eq. (4.1)}$$

Luminance comparison function *l(x, y)* as given in Eq. (4.2) ,

$$l(X,Y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \quad \text{… Eq. (4.2)}$$

where X and Y are the two images being compared and $C1$ denotes a constant to ensure the stability of the equation when the denominator becomes zero. The $C1$ is defined in Eq. (4.3), where $K1 = 0.001$ and L = the data range of the input image (distance between minimum and maximum possible values)

$$C_1 = (K_1 L)^2 \quad \text{… Eq. (4.3)}$$

b. Contrast

The standard deviation of all pixel values is used as an estimation of the contrast σ_x. This function simply removes the mean intensity from the image.

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)^2 \right)^{\frac{1}{2}} \quad \text{… Eq. (4.4)}$$

Same way the contrast comparison function $c(x, y)$ is defined as,

$$c(x, y) = \frac{2\sigma_x \sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \quad \text{… Eq. (4.5)}$$

$$\text{where, } C_2 = (K_2 L)^2 \quad \text{… Eq. (4.6)}$$

Also, $K2 = 0.003$, and L is the same as defined for $C1$.

c. Structure

Now, both images are normalized by their standard deviation to get both of them in unit standard deviation for comparison purposes.

$$S(X) = \frac{X - \mu_x}{\sigma_x}, \quad S(Y) = \frac{Y - \mu_y}{\sigma_y} \quad \text{… Eq. (4.7)}$$

The structural similarity or correlations between $S(X)$ and $S(Y)$, is equivalent to the correlation coefficient between and X and Y. So, the Structure comparison function $s(X, Y)$ is defined by

$$s(X, Y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3} \quad \text{… Eq. (4.8)}$$

$$where, \sigma_{xy} = \frac{1}{N-1}\left(\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)\right) \quad \ldots \text{Eq. (4.9)}$$

To simplify the equations Eq. (4.8) and Eq. (4.9) the authors proposed to use, $C_3 = \frac{C_2}{2}$

And finally, the SSIM is defined by Eq. (4.10).

$$SSIM(X,Y) = [l(X,Y)]^{\alpha}.[c(X,Y)]^{\beta}.[s(X,Y)]^{\gamma} \quad \ldots \text{Eq. (4.10)}$$

To simplify the Eq.(4.10), authors have suggested to use $\alpha = \beta = \gamma = 1$, and so,

$$SSIM(X,Y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\sigma_x^2 + \sigma_y^2 + C_2)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad \ldots \text{Eq. (4.11)}$$

The structural similarity index can be calculated by simply taking the mean of Eq. (4.11) over all the pixel values. The authors of the paper also suggest that Eq. (4.11) may not work well in practice as this calculates the SSIM over the entire image globally and better results may be derived when using the small patches of the images in a sliding window manner and calculating the statistics over the local regions.

4.2.3.2. MRI-GAN

In the previous Chapter, we discussed how to get the SSIM of two images. If two images are similar SSIM will generate a tensor of all zeros otherwise the structural difference between the images. For our DeepFake detection purpose, we need dissimilarity between the fake and real images and we call it MRI. So, we simply use inversion of SSIM as dissimilarity tensor if SSIM is non zero. We define the MRI of two images X and Y in Eq. (4.12)

$$MRI\ (X,Y) = 1 - SSIM(X,Y), \qquad where\ 0 \leq SSIM(X,Y) \leq 1 \qquad \ldots \text{Eq. (4.12)}$$

The DFDC training set has metadata to identify real video used to generate a given fake video. We use this fake and real video pair to learn about the structural dissimilarity of frames. So, if there is dissimilarity in the structure of a pair of images it can be captured as some kind of artifact and there are no dissimilarities then the difference between them would be all zeros. All zeros can be considered as a black image. We use the structural dissimilarity of two images as MRI of the pair. Figure 4.5 shows an example of MRIs of pairs of fake images. In the top row, the left image is of a real person, the middle image is the synthesized image, and the rightmost image is the MRI of these fake and real images. Same for the bottom row.

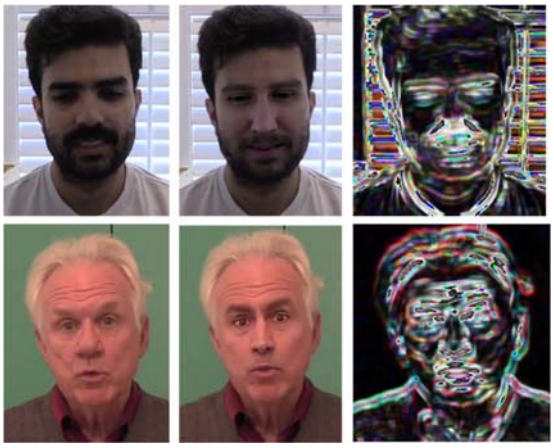

Figure 4.5 Example MRI images of the fake samples

Now our challenge is to learn a way to generate an MRI of any unseen image. A Pix2Pix GAN can be handy in this scenario. A Pix2Pix GAN is used to convert one image to another image. For training, it expects a pair of images, say Image-A and Image-B. The Pix2Pix GAN learns to convert Image-A into Image-B using supervised learning. In our case we trained the Pix2Pix

GAN with pair of an image and its MRI, we call this model MRI-GAN. This is a novel architecture we have come up with. If the image is of a real person the MRI is specified as a black image and if the image is of a fake person then its MRI would have corresponding artifacts. Again, we have pairs of real and fake videos from the training set and we use it extensively here. We trained our MRI-GAN using this technique from scratch and generated MRIs of all samples in the dataset. Figure 4.6 shows samples generated by the MRI-GAN. The top row is the image of persons which can be either fake or real. The middle row is the MRI of the corresponding persons from the top row. This row serves as the ground truth for the training. If the person is fake its MRI will have some kind of artifacts and if the person is real then the MRI will be a black image. The bottom row is the MRIs generate by our trained MRI-GAN. As you can see, the MRI-GAN seems to have learned to generate MRI in most of the cases. For the fourth person, the MRI-GAN generated some artifacts even though the person is real. This may be a limitation of the MRI-GAN we trained.

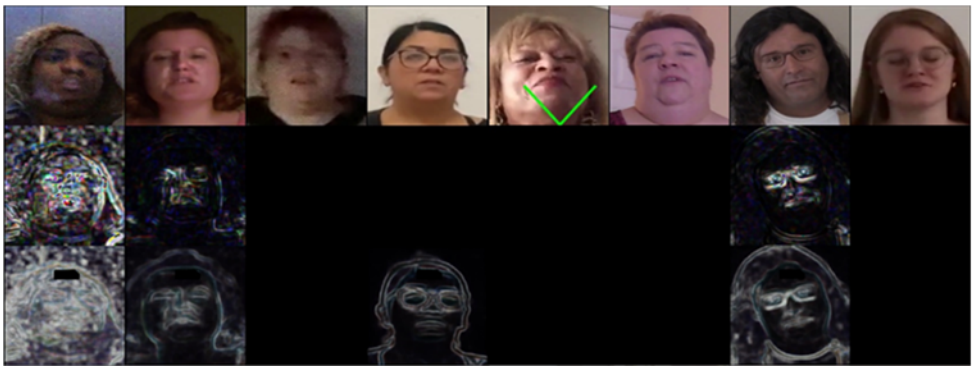

Figure 4.6 Samples generated by MRI-GAN

Using this technique, we have generated MRIs of all samples in the dataset. Just like ‘plain frames’ and Optical Flow methods, we have also passed MRIs as inputs to the model to classify if the video is fake or real. Figure 4.7 shows the end-to-end flow for the model utilizing MRIs. In addition to architecture defined in Figure 4.1, we have Step 3a and Step 3b explain in Figure 4.7.

At Step 3a we pass the frame to MRI-GAN and generate an MRI of the face. At Step 3b we pass these MRIs for dynamic augmentation and then to our model for classification.

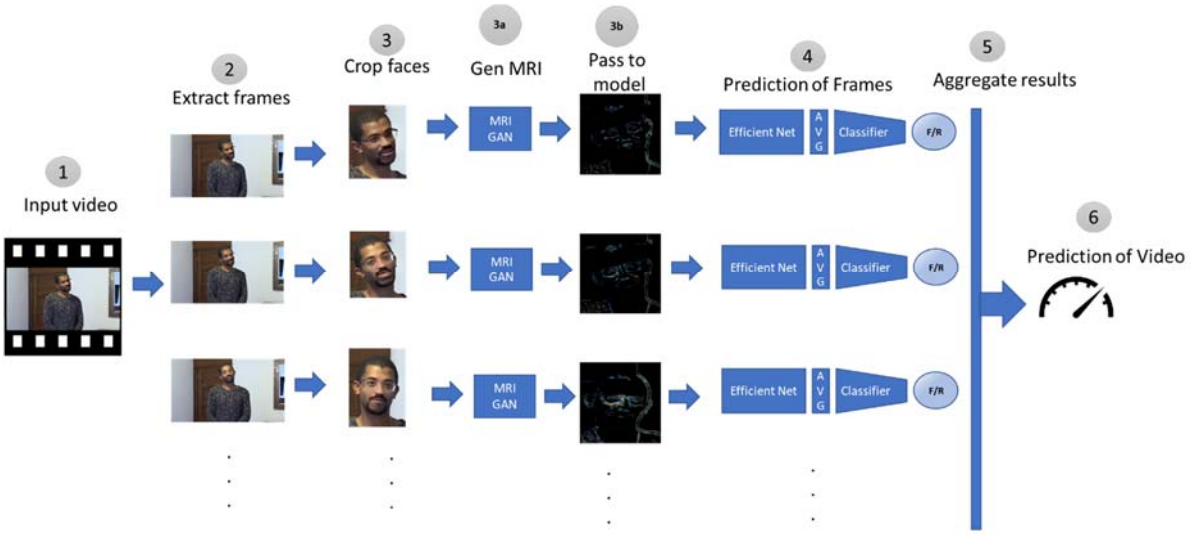

Figure 4.7 Architecture of the MRI-based model

4.3. Model architecture

Figure 4.8 shows the model architecture. In Step 1, we provide an input image to the model. This image can be a plain frame, Optical Flow image, or MRI of the face. The input image is expected to be *224 x 224* to match the native size of the Efficient-Net B0 model. The pre-trained Efficient-Net B0 is deployed at Step 2, in our model to extract the convolution features of the input image. At Step 3 we apply adaptive average pooling. The output of Step 3 is flattened and send to the classifier in Step 4. The classifier is two layers of fully connected MLP with dropout and Relu activation applied in between. Finally Step 6 is the final output of the model which is a single neuron to indicate if the input image is fake or real. This model architecture is the same for all kinds of input features we have experimented with. In the implementation Chapter, we explain other optimizations we have performed.

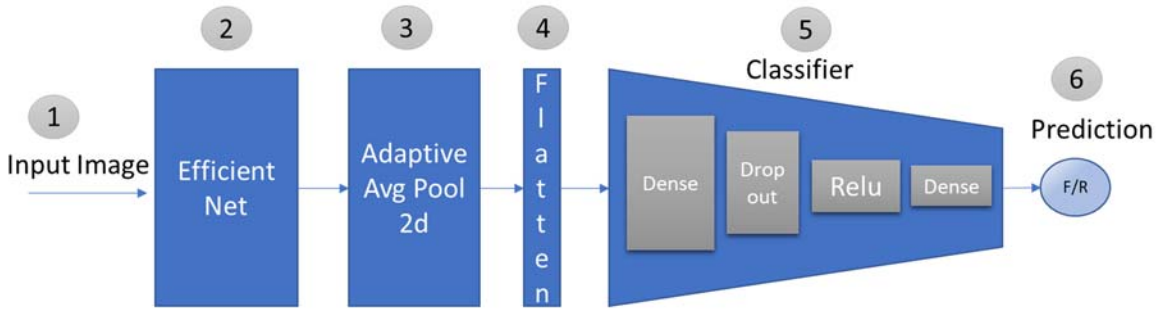

Figure 4.8 Model architecture

DeepFake detection problem involves detecting given media is fake or real, so it is a binary classification problem. Since our goal is to detect DeepFakes we declare it as succuss and denote the label to one for all fake samples. For real images, we denote the label as zero. We use Binary cross-entropy (with logits loss) to find the loss between the predictions and the actual labels. The loss function is defined in Eq. (4.13).

$$Loss\ l(f,r) = -\frac{1}{N}\sum_{i=1}^{N} r_i \,.\, log\big(p(f_i)\big) + (1 - r_i).\, log\big(1 - p(f_i)\big) \quad \text{… Eq. (4.13)}$$

Adaptive average pooling is the average pooling layer but we don't have to explicitly specify the kernel size, stride, and padding hyper-parameters. We only need to specify the target image dimension; the needed hyper-parameters are set by the layer to get the targeted image dimensions. We have set the target image size to be the same as the input image size.

5. IMPLEMENTATION

In this Chapter, we walk through the details of the implementation. We begin by providing links to access the source, the dataset used, and data preprocessing. Then we explain the implementation of the model using PyTorch and how to replicate our work with the source code provided.

5.1. Source code

The source code of the entire implementation can be found at http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring20/pratik/index.php?Bio.php#top as well as at https://github.com/pratikpv/deep_fake_detection

5.2. Dataset

We have used DeepFake Detection Challenge (DFDC) [7] dataset. Approximately 3426 paid actors and actresses were hired by Facebook and explicit consent was made available to modify their identity. The videos are recorded in natural settings and with 1080p resolution. Some DeepFake methods applied are less-realistic to represent low-effort results. The number of videos per-method is not equal. Training, validation, and test set are provided with the dataset. Some of the videos of the test set are kept private and not made available publicly.

The training data set contains 119,154 video clips each being ten seconds long. There are 486 unique subjects in the training set. Total 100,000 clips (approx. 84%) are DeepFakes and no augmentations are applied to the training set. The validation set contains 4,000 video clips each being ten seconds long. Total 2,000 clips (50%) are DeepFakes in the validation set. There are 214 unique subjects recorded in this set. These subjects are not part of the training set. The test set is comprised of two parts, private and public. The private test set is not available and

Facebook used it to evaluate the DFDC challenge at Kaggle [19]. The public set is made available and contains 5,000 clips each being ten seconds long. Total 2,500 of them (50%) are DeepFakes. There are 260 unique subjects recorded in the test set which does not appear in the train and validation set. We have used the entire training, validation and public test set in our experiments. The validation and test set are heavily augmented with various kinds of methods. Different types of noise, rotation, color transformations, different frame-rates, different video resolutions are some of the augmentation methods applied. The validation and test set also contains distractions in many videos. Some of the distraction methods include the appearance of random text, random geometric shapes, and random logo to appear either at random frames of scrolls from any of the directions. Faces may also have dog and flower crown filters in random videos. This makes the whole dataset very heavily augmented and challenges arise to generalize the model accuracy. To overcome this, we have to apply a similar kind of augmentation and distractions to the training set so that model may become more resilient and learns to generalize better.

5.3. Data preprocessing

To make our model generalize better we have applied augmentation and distractions to the training dataset heavily with various methods. We used static and dynamic mode of augmentation and the only static way to apply distractions. With the static method, we applied both kinds of data preprocessing methods before Step 1 of the data pipeline explained in Figure 4.1. We choose to apply the static way of augmentation and distractions to resemble real-world scenarios such as low-quality, low-resolution, distraction applied in various settings, etc. The MTCNN face detection model will need to detect faces in this heavily manipulated data set. In practice, many times MTCNN failed to recognize faces in some of the frames or fails completely

to detect any faces in the entire video. This becomes one of the limitations of our implementation as MTCNN fails to detect faces for some of the videos. We discuss more on this case in the Limitation Chapter of this report. The static data preprocessing methods are applied to a random set of videos from the training set. Note that the dynamic mode of augmentation is applied on the final frames which have only the faces detected by the MTCNN. The static mode of augmentation is applied to the entire frame before passing the frame to MTCNN to detect the faces. Figure 5.1 gives a pictorial view of all data preprocessing we have applied. With dynamic mode, we apply simple preprocessing or complex based on the features we chose to pass to the model. Later in the results Chapter, we clarify which dynamic mode of preprocessing we applied on the given feature set.

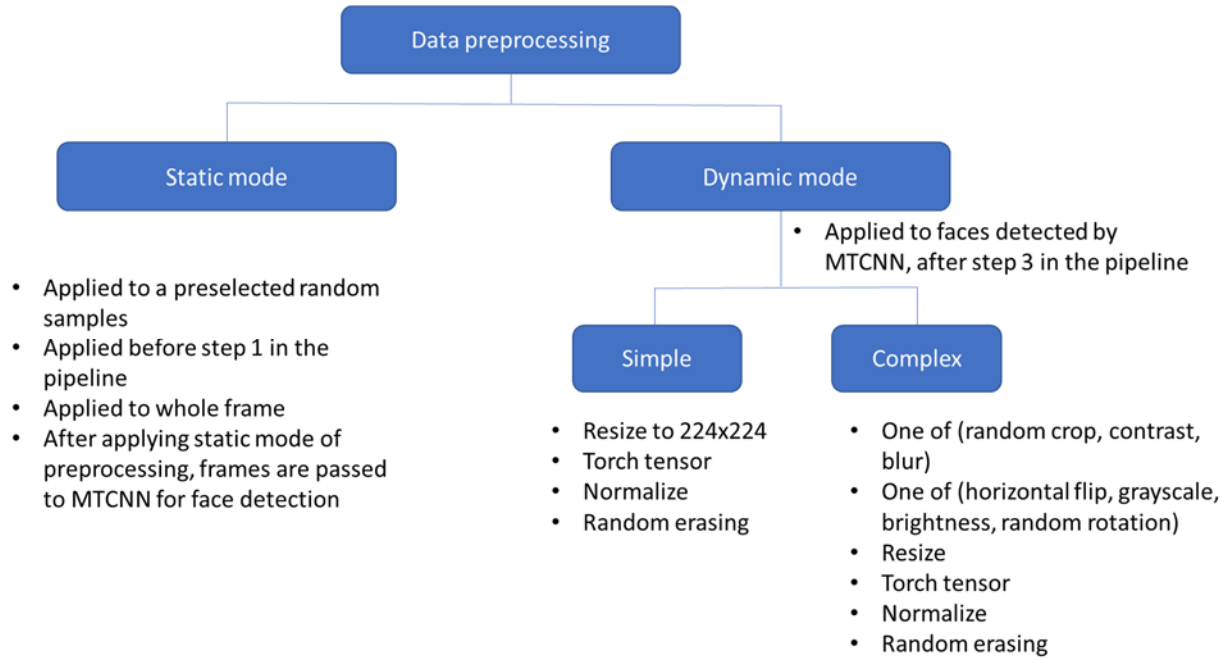

Figure 5.1 Data preprocessing methods

Now we discuss methods implemented for static data preprocessing methods.

a. Augmentation methods

Various types of noise are added to each frame of the videos. Types of noise implemented are Gaussian, Speckle, Salt-and-Pepper, Pepper, Salt, Poisson, and Localvar. We also applied other augmentation methods such as blur, rotation, horizontal flip, rescale, brightness, and contrast.

b. Distraction methods

We implemented static, rolling, and spontaneous methods of data distractions. Each method includes text and geometric shapes as an object to generate distractions. When a text is used as a distraction, a random alpha-numeric text of length eight is generated and applied to the video. The font size of the text is chosen randomly from the scaling of one to six units. The font thickness is chosen randomly from the scaling of one to three units. The font colors are chosen randomly supported colors. Red, blue, green, white, and black is supported colors. In the case of static text, a random text is generated and it is overlaid on the video at a fixed location. The location is chosen randomly and fixed for the entire video. In the rolling text scenario, a random text moves from a fixed direction for the duration of the video. The starting location of the text is chosen randomly at the start and the rolling direction could be any of the following: right-to-left, left-to-right, up-to-down, down-to-up. The spontaneous text method overlays a random text at random locations at random times in the video. All the various options implemented for text-based distraction applies to shapes-based-distractions as well. Circle and rectangle shapes are chosen for the shapes-based-distractions method.

We have applied various combinations of random augmentation and random distractions. So a video might have one of more random augmentation applied as well as one or more random distractions also applied to it. We applied this static method of data preprocessing to 30% of the training set. Table 5.1 shows the distribution of various data augmentation and distractions methods applied to 30% of the training dataset. The method column mentions if we only one of the data preprocessing methods of a combination of the methods on the same sample. The corresponding row mentions how many samples we used to apply the specified set of data preprocessing methods.

Table 5.1 Distribution of various data augmentation and distractions methods applied.

Method	**Percentage of videos**
Random distractions only	35%
Random distraction, followed by random noise on the same sample.	35%
Random distractions, followed by random noise, and finally random augmentation	15%
Random noise	5%
Random augmentation followed by random noise	5%
Random augmentation	5%

Also, note that the static mode of data preprocessing is applied at the very beginning of the implementation phase on randomly chosen samples in the training set. After this data preprocessing step, we follow the pipeline process mentioned in Figure 4.1. We also apply dynamic data preprocessing after Step 3, before passing the frames to our model. We use the transform library of the PyTorch to apply data augmentation in dynamic mode. With this mode every time a batch of data is passed to the model the specified random set of augmentation is applied. Figure 5.2 shows samples of all types of augmentation and distractions we have implemented and applied to the training set.

Figure 5.2 Examples of augmentations and distractions

5.4. Model implementation

The model implementation is pretty straight forward. We use the timm [26] library to get a pre-trained Efficient-Net B0 model. We use the torch.nn library of the PyTorch for AdaptiveAvgPool2d, Dropout, Relu, and Linear layers.

We have defined `'DeepFakeDetectModel_6'` class for the CNN-based discussed in this report. This class is inherited from torch.nn.Module. We initialize the Efficient-Net B0 and all the layers discussed in the model architecture in the init method of this class. In the `forward()` method of the class, we transform the input data and pass it to various layers of the model. Finally, the logits are returned by the `forward()` method. We have used Binary cross-entropy (with logits loss) as our loss criterion. Adam optimizer is used for gradient descent steps. We have use label smoothing [27] and dropout for regularization.

5.5. Aggregation algorithm

We classify each frame independently if it's fake or real. The model may perform poorly to classify certain frames and so our challenge is to transform the frame-level classification to the entire video-level classification.

We have used a simple algorithm to aggregate the frame-level classification results to quantify the video-level classification. We can check the probability of each frame being fake and we can also calculate how many of such frames are detected by the model. We can use these two statistics along with the total number of frames of the video to classify the entire video. The algorithm to transform frame-level classification to video-level classification is given in Figure 5.3

```
get_video_classification(video):

        set fake_frames_set = {}

        for each frame in the video:
                if probability_of_fake(frame) > FAKE_FRAME_THRESHOLD:
                        fake_frames_set.add(frame)

        if size(fake_frames_set) >= FAKE_FRACTION * total_num_of_frames:
                return "Fake"
        else
                return "Real"
```

Figure 5.3 The Aggregation algorithm

`probability_of_fake()` simply returns, as the name suggests, probability if the frame being fake. `FAKE_FRAME_THRESHOLD` is the threshold we need to set to create a set of frames that are detected as fake with at least this threshold value. `FAKE_FRACTION` can be between 0 and 1 to represent how many such high probable frames are needed to classify the entire video as fake. We have experimented with few values of these parameters and the results are discussed later in the report.

5.6. Replication steps

In this Chapter, we explain how to use our code to replicate the overall project. There are three main steps to use our code. The first step is data preprocessing to perform static augmentation, detecting faces using MTCNN, and saving crops of the detected faces. The second step is to extract Optical Flow features and MRIs. We also explain how to train MRI-GAN and use it to extract the MRI of faces. The third step includes model training and testing. All steps are very time and computation resources consuming. It can easily take several days for data preprocessing and few days for training. We have used a moderately high-end workstation and implemented

most of the data preprocessing code using python's multiprocessing library to run the preprocessing in parallel. The training, validation, and testing is performed using GPU. Overall system configuration is described in the Chapter of "Frameworks, Tools, And Hardware".

The entire implementation is controlled using the `config.yml` file. It contains various configuration options, paths of the dataset, and paths of the intermediate files generated. We explain an overview of the steps needed to replicate our work; further details of the implementation can be found in the code.

5.6.1. Data preprocessing

First, we applied static augmentation and distractions to the training set. Following commands can be used to apply static augmentation and distractions.

```
$ python data_preprocess.py --gen_aug_plan
```

```
$ python data_preprocess.py --apply_aug_to_all
```

Then we use MTCNN to detect faces in the training, validation, and test set and save the location of faces in the JSON files for each sample.

```
$ python data_preprocess.py --extract_faces
```

In the next step, we extract the faces from each sample and save the crops of the faces on the disk. These crops are plain faces of the subjects detected in the samples.

```
$ python data_preprocess.py --crop_faces
```

Then we need to generate a CSV file with all frames and a separate image and its label. The label can be either fake (1) or real (0). This label is replicated several frames times to assign a

dedicated label to each frame. This is needed as we pass each frame independently to the model for classification.

```
$ python data_preprocess.py --gen_frame_label
```

5.6.2. Features extraction

In this subsection, we explain how to extract Optical Flow and MRI features.

5.6.2.1. Optical Flow features extraction

Use the below command to generate Optical Flow data.

```
$ python features_extract.py --gen_optic_data
```

5.6.2.2. MRI features extraction

Extracting MRI features is a very involved process. Overall we need to train the MRI-GAN model, and use the generator part of the GAN to extract MRIs for the whole dataset.

a. MRI-GAN training

MRI-GAN is trained in a supervised manner. This is a Pix2Pix GAN and it expects pairs of images. In our case, a pair can be the face of the person and its MRI. In this pair, if the person is real, we use a black image as its MRI. If the face of the person is fake, we use structural differences between this fake image and its corresponding real image. The DFDC training dataset contains a mapping between all fake and real videos, which is used to generate the pairs. Use the below command to generate an MRI of fake samples.

```
$ python features_extract.py --gen_xray
```

Use the below command to generate pairs of the samples. This command will generate a CSV file with paths of the images in the pair and its label.

```
$ python features_extract.py --gen_xray_pairs
```

Now we train the MRI-GAN with the below command.

```
$ python train_pix2pix.py --train_from_scratch
```

b. MRIs extraction

At this point, we are ready to generate MRIs of all samples in our dataset using the trained MRI-GAN. The below command generated the MRIs of all the samples. These MRIs can be passed to the model for classification.

```
$ python features_extract.py --extract_mri
```

5.6.3. Model execution

Model execution is controlled by `config.yml`. Some of the important configuration options are model_name, label_smoothing, label_smoothing, train_transform, batch_format, epochs, learning_rate, batch_size, and dataset. The code contains detailed descriptions of all of these parameters. We can set these parameters and execute the training with the below command.

```
$ python detect_deepfake.py --train_from_scratch
```

We can resume the training from the saved checkpoint with the below command.

```
$ python detect_deepfake.py --train_resume <path to chkpt file>
```

If we want to test our saved model in the checkpoint file, the below command can be used.

```
$ python detect_deepfake.py --test_saved_model <path to chkpt file>
```

6. EXPERIMENTS AND RESULTS

We discuss various experiments performed and quantify the output of our experiments in this Chapter. Although we have implemented various CNN-based models and an RNN-based model, we discuss the experiments done with one of the CNN-based models i.e., `'DeepFakeDetectModel_6'`, which gives better accuracy. We have discussed this model widely in this report so far and the results discussed in this Chapter are with the same model only. As we have used three ways (plain frames, Optical Flow, and MRI) to pass images or theirs features to the model we provide accuracy, loss, confusion matrix, Receiver Operating Curve (ROC), etc. matrices of all of them to compare the results. Figure 6.1 shows the training and validation (Val) accuracy of the model with all three features we utilized.

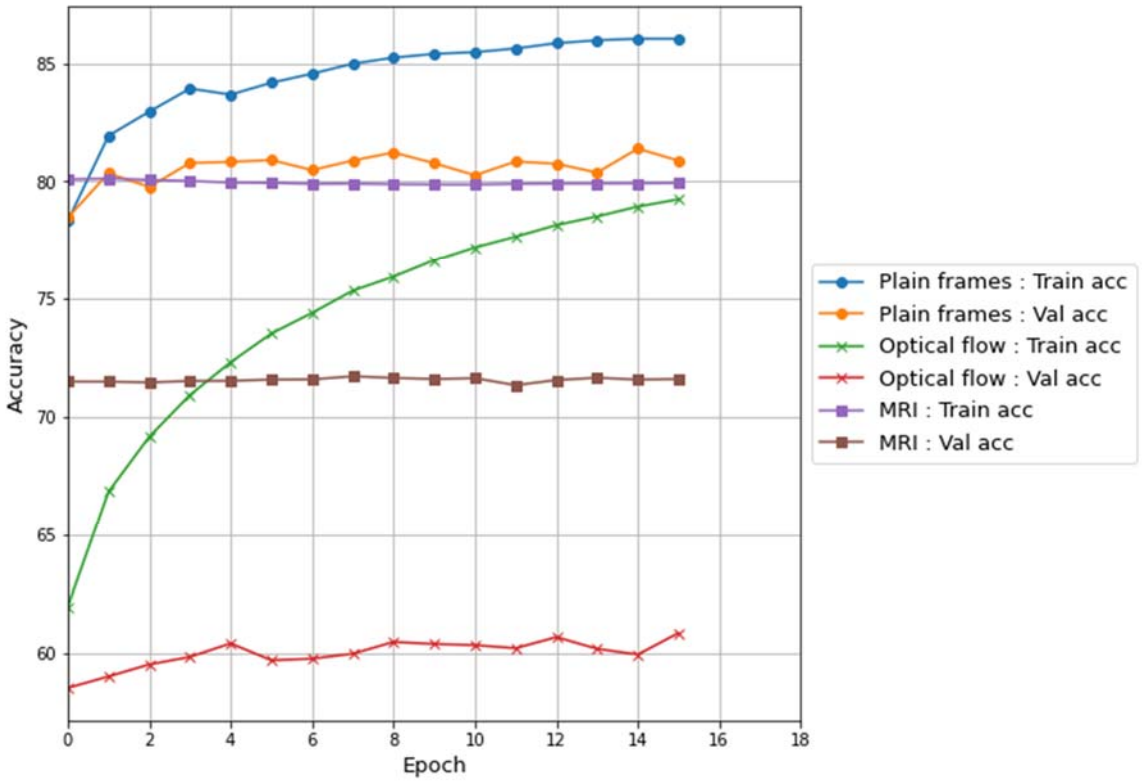

Figure 6.1 Plot of train and validation accuracies for all models

Figure 6.2 shows the mean binary cross-entropy loss for all the methods during the execution of each epoch.

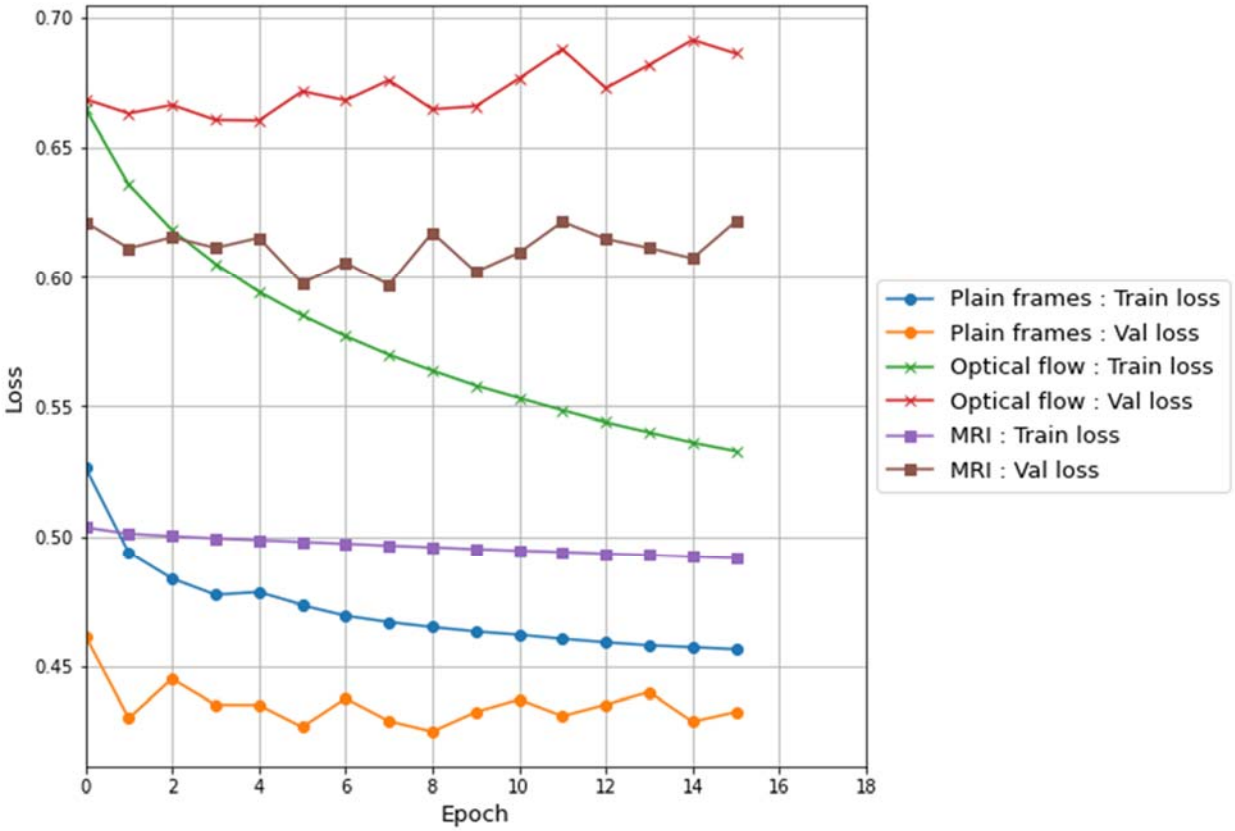

Figure 6.2 Plot of train and validation losses for all models

These accuracy and loss values are for the frames we have passed to the model. These values do not represent the aggregated accuracies for the video-level classification. We have tried two sets of values for `FAKE_FRAME_THRESHOLD` and `FAKE_FRACTION` parameters for the aggregation algorithm discussed in the Implementation Chapter.

Table *6.1* shows the matrix we tried and the name given to the specific configuration. We can infer that if we use frame-level classification only then, `FAKE_FRAME_THRESHOLD` and

FAKE_FRACTION would be 0.5 and one respectively, but we are relaxing these thresholds with the aggregation algorithm. We have classified the frame as fake if the probability of being fake is at least 50% in our core model.

Table 6.1 Thresholds matrix for aggregating frame-level results.

Configuration name	**FAKE_FRAME_THRESHOLD**	**FAKE_FRACTION**
Video Agg1	0.6	0.3
Video Agg2	0.5	0.5
Frame level	0.5	1

Figure 6.3 shows the results of using these thresholds on the overall accuracy of the model. We can see that by relaxing the thresholds with the algorithm, the accuracy of the model improves. Amongst both sets of values we tried for the aggregation, Video Agg2 set gives better accuracy. So, in nutshell, we classify a given video being fake or real if we find at least 50% of all the frames are detected fake and each of these frames are classified as fake with at least 50% probability.

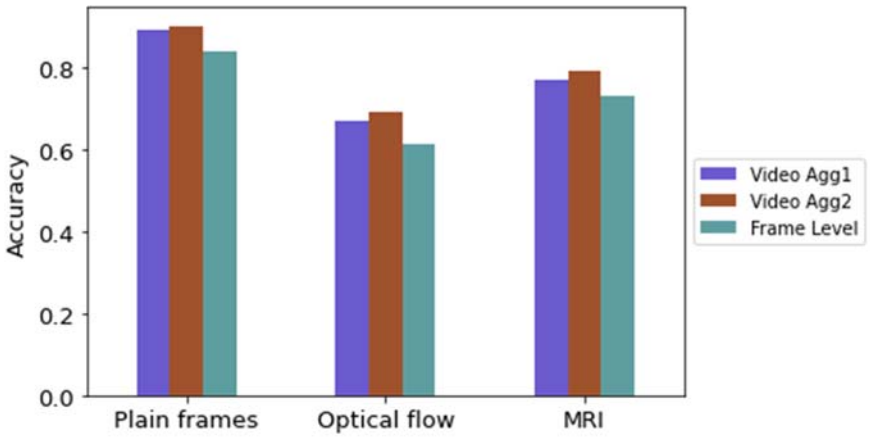

Figure 6.3 Plot of accuracies with different aggregation parameters for all models

Figure 6.4 shows the ROC curves and Figure 6.5 shows confusion matrixes for all the methods with utilizing Video Agg2 parameters for transforming the frame-level classification to the video-level.

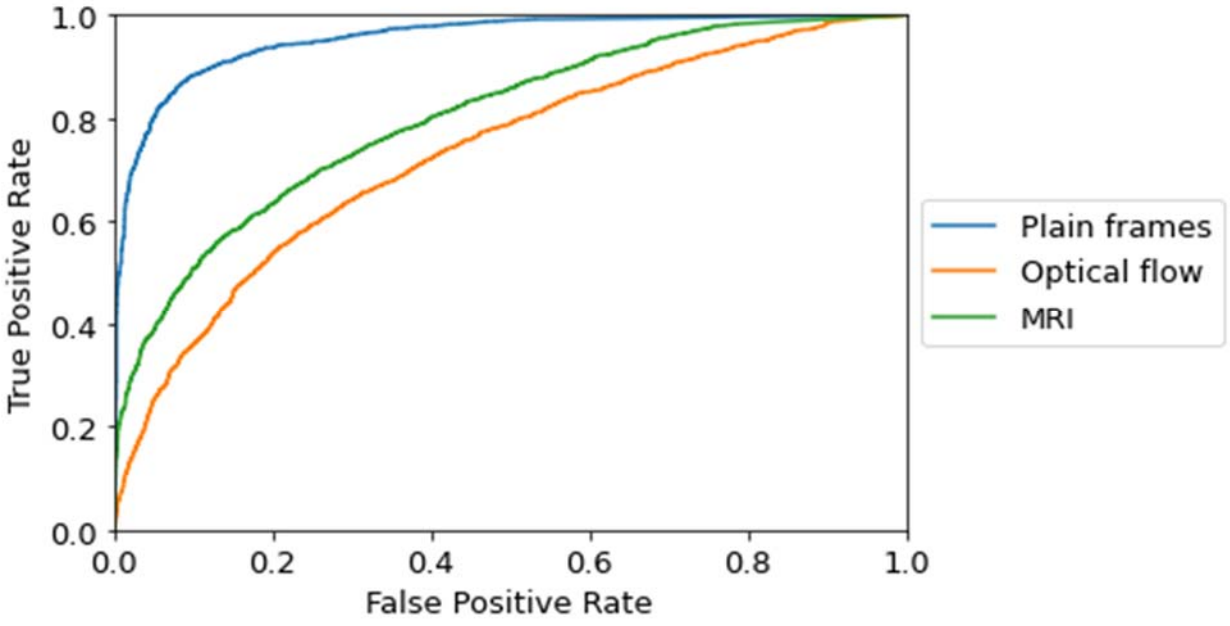

Figure 6.4 Plot of ROC curves for the test of all models

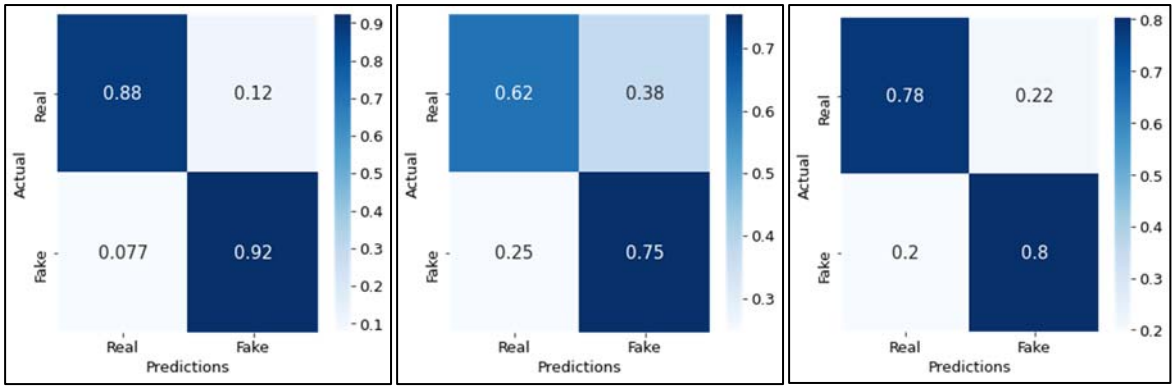

Figure 6.5 Confusion matrixes, left: for plain frames, middle: for Optical Flow, right: for MRI

We have used label smoothing with a factor of 0.1, Adam optimizer with a learning rate of 0.001, and a batch size of 192 for all the results discussed in the report. We have used a complex level of dynamic augmentation for plain frames and a simple level of dynamic augmentation for Optical Flow and MRI methods. Data used in all three methods go through static augmentation before we extract each frame of the videos. We have used Binary Cross Entropy with logits loss as our loss function for this binary classification problem. We have run the train and validation for sixteen epochs, and then we have run final testing on the test dataset. At each epoch, we save model weights when we find the best accuracy on the validation set. The test results are generated with the model which exhibits the best validation accuracy during the training, we did not use the overfitted model for the final testing. Also, note that the training set contains roughly 83% of fake videos, and both the validation and test set contain 50% of fake videos.

We also tried some combinations of no label smoothing, learning rates of 0.0001 and 0.0002, simple and complex dynamic augmentations. But they did not result in better accuracy.

Table 6.2 shows various matrices for all three models side-by-side to compare them.

Table 6.2 Various matrices for all models.

Score	Plain Frames	Optical Flow	MRI
True Positive Rate (TPR)	0.92	0.72	0.8
False Negative Rate (FNR)	0.08	0.28	0.2
False Positive Rate (FPR)	0.11	0.33	0.21
True Negative Rate (TNR)	0.89	0.67	0.79
Accuracy	0.9	0.69	0.79
Balanced Accuracy	0.9	0.7	0.8
F1-score	0.9	0.67	0.79
Precision	0.88	0.62	0.78
Specificity	0.89	0.67	0.79
Area Under the curve ROC (AUC ROC)	0.95	0.73	0.80

7. LIMITATIONS AND FUTURE WORK

In this Chapter, we explain the limitations of our approaches and how it may be mitigated with future research.

We use the aggregation technique to convert frame-level classification results to video-level classification. We use the probability of being a fake frame and the number of fake frames detected in the video. There could be the optimal set of these parameters which may lead to better overall accuracy but we tried with only a few of them, so our results represent the lower bound of the accuracy. In future work, we may also try to generate a metamodel to learn about the best set of these parameters to find the upper bound of the accuracy. Also, MTCNN falls short to recognize faces in some of the videos, which may be due to heavy augmentation or the nature of the environment in the video. This leads to having no information about such videos. We simply drop such videos from our dataset. In the future, we may choose to try other face detection model. Due to computation limitations, we chose to use the Efficient-Net B0 model to extract convolution features. The B0 model is the simplest and the base-line model of Efficient-Nets. One may try with a complex and bigger model such as Efficient-Net B7 or any such other models to extract convolution features. These limitations apply to all three features we experimented with.

The DFDC dataset is very challenging. There could more than one person in the video. In many of the videos, some person appears only for a certain duration. The persons are walking and changing their positions. This resembles a more natural way of collaboration and appearance in the video. Due to this, the MTCNN might detect some faces in some frames while others in the rest of the frames. Also ordering of face detection can change. E.g., let's consider if there are two persons in the entire video and person A appears for the initial few frames and then person B

appears. In another scenario, person A and person B are walking. In such cases, MTCNN detects faces in an out-of-band manner and the identity of the persons are not matched. Also, we crop the faces detected in each frame as they appear in the sequence of the frame, many times person B gets detected as the first person, instead of the second person in the context of the video. We calculate the Optical Flow of the two consecutive frames. Sometimes these both frames will not be of the same person. E.g. For Optical Flow, we need frame f of person A and frame f+1 of same person A, but we end up with frame f of person A and frame m of person B. This leads to the wrong Optical Flow. To overcome this, we also tried using You Only Look Once (YOLO)-V3 [28] to check if we can maintain the identity of the person in the context of the video. We found, YOLO-V3 based face detection also suffers from the same issue with the DFDC dataset. To enhance this model, we may look for new ways to preserve the identity of the persons detected and generate more accurate Optical Flow.

Also, we have trained MRI-GAN using 80% of the DFDC train dataset. Even though MRI-GAN learns to generate MRIs, there are more opportunities to enhance the MRI-GAN. The test and validation set of the model uses DeepFake techniques which are not present in the training set. So, MRI-GAN may have fallen short to generalize in generating MRIs. To train the MRI-GAN, we may choose other data set as well, so that MRI-GAN learns to generalize better. Researchers are proposing new techniques very frequently to generate DeepFakes. If MRI-GAN is used to detect DeepFake in the future, the MRI-GAN can be retrained in a supervised manner to learn about the newly proposed methods and generate MRIs more realistically.

The accuracy of deep learning-based models depends on the tuning of hyper-parameters. There might be a better set of hyper-parameters than we proposed and more experiments may reveal better accuracy.

8. CONCLUSION

Fake media can be a threat to our society. Face swapping is one of the four common methods used to generate fake content. The method of swapping the faces or identities in the images and videos is called DeepFake. Our scope in this research was to quantify the accuracies of the model to detect DeepFakes in commonly available natural settings.

We used the DFDC dataset to train our models as the dataset contains a very diverse set of DeepFake videos in large quantity. Only images as visual information is used and audio information is not used in our approaches. We have applied static mode of augmentation and distractions to replicate real-world normal scenarios of video clips. We extract the frames from the video, perform features extractions, dynamic augmentation, and send the final tensors to our model for classification. We aggregate the frame-level classification results to transform them into the video-level classification. Our best performing models are based on CNN. We use Efficient Net-B0 to extract encodings of the input images to utilize transfer learning and pass the encodings to our custom classifier. We used many regularization methods such as label smoothing and dropouts to avoid overfitting. One of the core techniques in our research is to extract features from the faces. We have created three main types of feature extractions. They are simply passing the frames to the model, generating an image of Optical Flow between consecutive frames, and generating MRIs of the faces. We used the different levels of dynamic data augmentation to achieve the best possible accuracies of the model. Without complex dynamic augmentation, label smoothing, and dropout the model using plain frames as features achieved 80% test accuracy. After applying the regularization techniques, the model was able to generalize better and achieved 84%. Also, after implementing the aggregation algorithm the model achieved accuracy of 90%. We used the same CNN-based model with all three different

kinds of features we experimented with. We proposed a novel method for extracting MRIs of the faces and demonstrated it with our proposed MRI-GAN architecture.

We have quantified our results with matrices including TPR, FNR, FPR, TNR, Accuracy, Balanced Accuracy, F1-score, Precision, Specificity, and AUC ROC. The model using plain frames performs best with 90% test accuracy, MRI based model achieves 79% accuracy and the Optical Flow-based model achieves 69% test accuracy.

Rather than training the model to detect any specific kind of DeepFake generation method, we attempted to propose a model that generalizes DeepFake detection irrespective of DeepFake generation methods. With future advancement and new research, we expect a new method will be proposed and help our society to combat threats like DeepFakes. With this, we conclude our report and an attempt to contribute to the research of detecting synthesized fake media.

LIST OF REFERENCES

[1] https://www.youtube.com/watch?v=xkqflKC64IM, Accessed: 2020-01-30

[2] https://www.forbes.com/sites/robtoews/2020/05/25/deepfakes-are-going-to-wreak-havoc-on-society-we-are-not-prepared/, Accessed: 2020-08-30

[3] Tolosana, Ruben, et al. "DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection." arXiv preprint arXiv:2001.00179 (2020).

[4] Mirsky, Yisroel, and Wenke Lee. "The Creation and Detection of Deepfakes: A Survey." arXiv preprint arXiv:2004.11138 (2020).

[5] Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and ¨M. Nießner, "FaceForensics++: Learning to Detect Manipulated Facial Images," in Proc. IEEE/CVF International Conference on Computer Vision, 2019.

[6] E. Sabir, J. Cheng, A. Jaiswal, W. AbdAlmageed, I. Masi, and P. Natarajan, "Recurrent Convolutional Strategies for Face Manipulation Detection in Videos," in Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2019.

[7] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The DeepFake Detection Challenge (DFDC) dataset. arXivpreprint arXiv:2006.07397, 2020.

[8] Zhou Wang; Bovik, A.C.; ,"Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures," Signal Processing Magazine, IEEE, vol. 26, no. 1, pp. 98-117, Jan. 2009.

[9] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, Apr. 2004.

[10] L. Li, J. Bao, T. Zhang, H. Yang, D. Chen, F. Wen, and B. Guo, “Face X-ray for More General Face Forgery Detection,” arXiv preprint arXiv:1912.13458v1, 2019.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. The MIT Press

[12] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” Proceedings of the 36th International Conference on Machine Learning (ICML), 2019.

[13] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in International Conference on Medical image computing and computer-assisted intervention. Springer, 2015, pp. 234–241.

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative adversarial networks. CoRR abs/1406.2661

[15] Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on. (2017)

[16] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. arXiv preprint arXiv:1912.04958, 2019

[17] Zhu, J.-Y.; Park, T.; Isola, P.; and Efros, A. A. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2223–2232.

[18] https://github.com/shaoanlu/faceswap-GAN, Accessed 2020-01-15

[19] https://www.kaggle.com/c/deepfake-detection-challenge, Accessed 2020-01-15

[20] https://github.com/selimsef, Accessed 2020-09-15

[21] “FaceForensics++: Learning to detect manipulated facial images,” in ICCV, 2019

[22] Amerini, Irene, Leonardo Galteri, Roberto Caldelli, and Alberto Del Bimbo. "Deepfake video detection through Optical Flow based cnn." In Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 0-0. 2019.

[23] Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Processing Letters, 23(10):1499–1503

[24] https://www.github.com/sniklaus/pytorch-liteflownet.git, Accessed 2020-08-15

[25] https://pypi.org/project/flowiz/, Accessed 2020-08-15

[26] https://github.com/rwightman/pytorch-image-models, Accessed 2020-06-15

[27] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, and G. Hinton. Regularizing neural networks by penalizing confident output distributions. ICLR Workshops, 2017.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in CVPR, pp. 779–788, 2016