Jyothi Hills, Panjal Road,
Vettikattiri PO, Cheruthuruthy , Thrissur,
Kerala 679531

Jyothi Engineering College
NAAC Accredited College with NBA Accredited Programmes*
Approved by AICTE & affiliated to APJ Abdul Kalam Technological University
A CENTRE OF EXCELLENCE IN SCIENCE & TECHNOLOGY BY THE CATHOLIC ARCHDIOCESE OF TRICHUR
JYOTHI HILLS, VETTIKATTIRI P.O, CHERUTHURUTHY, THRISSUR. PIN-679531 PH : +91- 4884-259000, 274423 FAX : 04884-274777
NBA accredited B.Tech Programmes in Computer Science & Engineering, Electronics & Communication Engineering, Electrical & Electronics Engineering and Mechanical Engineering valid for the academic years 2016-2022. NBA accredited B.Tech Programme in Civil Engineering valid for the academic years 2019-2022.

DeepFake Detection Using Machine Learning

PROJECT REPORT

Karthik PC **(JEC17CS061)**

M P Adithya Vijayan **(JEC17CS069)**

Sanjana S **(JEC17CS088)**

Thushara P **(JEC17CS103)**

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY (B.Tech)

in

COMPUTER SCIENCE & ENGINEERING

of

A P J ABDUL KALAM TECHNOLOGICAL UNIVERSITY

Under the guidance of

Ms. Aswathy Wilson

CREATING TECHNOLOGY LEADERS OF TOMORROW

JUNE 2021

Department of Computer Science & Engineering

Jyothi Hills, Panjal Road,
Vettikattiri PO, Cheruthuruthy , Thrissur,
Kerala 679531

Jyothi Engineering College
NAAC Accredited College with NBA Accredited Programmes*
Approved by AICTE & affiliated to APJ Abdul Kalam Technological University
A CENTRE OF EXCELLENCE IN SCIENCE & TECHNOLOGY BY THE CATHOLIC ARCHDIOCESE OF TRICHUR
JYOTHI HILLS, VETTIKATTIRI P.O, CHERUTHURUTHY, THRISSUR. PIN-679531 PH : +91- 4884-259000, 274423 FAX : 04884-274777
NBA accredited B.Tech Programmes in Computer Science & Engineering, Electronics & Communication Engineering, Electrical & Electronics Engineering and Mechanical Engineering valid for the academic years 2016-2022. NBA accredited B.Tech Programme in Civil Engineering valid for the academic years 2019-2022.

DeepFake Detection Using Machine Learning

PROJECT REPORT

Karthik PC **(JEC17CS061)**

M P Adithya Vijayan **(JEC17CS069)**

Sanjana S **(JEC17CS088)**

Thushara P **(JEC17CS103)**

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY (B.Tech)

in

COMPUTER SCIENCE & ENGINEERING

of

A P J ABDUL KALAM TECHNOLOGICAL UNIVERSITY

Under the guidance of

Ms. Aswathy Wilson

CREATING TECHNOLOGY LEADERS OF TOMORROW

JUNE 2021

Department of Computer Science & Engineering

Department of Computer Science and Engineering

JYOTHI ENGINEERING COLLEGE, CHERUTHURUTHY

THRISSUR 679 531

JANUARY 2021

BONAFIDE CERTIFICATE

This is to certify that the main project report entitled **Deep Fake Detection Using Machine Learning** submitted by **Karthik PC(JEC17CS061), M P Adithya Vijayan (JEC17CS069), Sanjana S(JEC17CS088) and Thushara P (JEC17CS103)** in partial fulfillment of the requirements for the award of **Bachelor of Technology** degree in **Computer Science and Engineering** of **A P J Abdul Kalam Technological University** is the bonafide work carried out by them under our supervision and guidance.

Ms. Aswathy Wilson	**Dr. Swapna B Sasi**	**Dr. Saju P John**
Project Guide	Project Coordinator	Head of The Dept
Assistant Professor	Associate Professor	Associate Professor
Dept. of CSE	Dept. of CSE	Dept. of CSE

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

COLLEGE VISION

Creating eminent and ethical leaders through quality professional education with emphasis on holistic excellence.

COLLEGE MISSION

- To emerge as an institution par excellence of global standards by imparting quality engineering and other professional programmes with state-of- the-art facilities.
- To equip the students with appropriate skills for a meaningful career in the global scenario.
- To inculcate ethical values among students and ignite their passion for holistic excellence through social initiatives.
- To participate in the development of society through technology incubation, entrepreneurship and industry interaction.

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

DEPARTMENT VISION

Creating eminent and ethical leaders in the domain of computational sciences through quality professional education with a focus on holistic learning and excellence.

DEPARTMENT MISSION

- To create technically competent and ethically conscious graduates in the field of Computer Science & Engineering by encouraging holistic learning and excellence.
- To prepare students for careers in Industry, Academia and the Government.
- To instill Entrepreneurial Orientation and research motivation among the students of the department.
- To emerge as a leader in education in the region by encouraging teaching, learning, industry and societal connect

PROGRAMME OUTCOMES (POs)

1. **Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem Analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct Investigations of Complex Problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern Tool Usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The Engineer and Society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and Sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project Management and Finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-Long Learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

1. The graduates shall have sound knowledge of Mathematics, Science, Engineering and Management to be able to offer practical software and hardware solutions for the problems of industry and society at large.
2. The graduates shall be able to establish themselves as practising professionals, researchers or Entrepreneurs in computer science or allied areas and shall also be able to pursue higher education in reputed institutes.
3. The graduates shall be able to communicate effectively and work in multidisciplinary teams with team spirit demonstrating value driven and ethical leadership.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

1. An ability to apply knowledge of data structures and algorithms appropriate to computational problems.
2. An ability to apply knowledge of operating systems, programming languages, data management, or networking principles to computational assignments.
3. An ability to apply design, development, maintenance or evaluation of software engineering principles in the construction of computer and software systems of varying complexity and quality.
4. An ability to understand concepts involved in modeling and design of computer science applications in a way that demonstrates comprehension of the fundamentals and trade-offs involved in design choices.

COURSE OUTCOMES (COs)

C410.1 The students will be able to analyse a current topic of professional interest and present it before an audience.

C410.2 Students will be able to identify an engineering problem, analyse it and propose a work plan to solve it.

C410.3 Students will have gained thorough knowledge in design, implementations and execution of Computer science related projects.

C410.4 Students will have attained the practical knowledge of what they learned in theory subjects.

C410.5 Students will become familiar with usage of modern tools.

C410.6 Students will have ability to plan and work in a team.

ACKNOWLEDGEMENT

We take this opportunity to express our heartfelt gratitude to all respected personalities who had guided, inspired and helped us in the successful completion of this interim project. First and foremost, we express my thanks to **The Lord Almighty** for guiding us in this endeavour and making it a success.

We take immense pleasure in thanking the **Management** of Jyothi Engineering College and **Dr. Sunny Joseph Kalayathankal**, principal, Jyothi Engineering College for having permitted me to carry out this seminar. Our sincere thanks to **Dr. Saju P John**, Head of the Department of Computer Science and Engineering for permitting us to make use of the facilities available in the department to carry out the interim project successfully.

We express our sincere gratitude to **Mr. Shaiju Paul & Dr. Swapna B Sasi**, project coordinators for their invaluable supervision and timely suggestions. We are very happy to express our deepest gratitude to our mentor **Ms. Aswathy Wilson**, Assistant Professor,Department of Computer Science and Engineering, Jyothi Engineering College for her able guidance and continuous encouragement.

Last but not least, we extend our gratefulness to all teaching and non-teaching staff who directly or indirectly involved in the successful completion of this interim project work and to all friends who have patiently extended all sorts of help for accomplishing this undertaking.

ABSTRACT

Deep fake videos are AI-generated videos that look real but are actually fake.Deep fake videos are generally created by face-swapping techniques. It started out as fun but like any technology, it is being misused. In the beginning, these videos could be identified by human eyes. But due to the development of machine learning, it became easier to create deep fake videos. It has almost become indistinguishable from real videos.Deep fake videos are usually created by using GANs (Generative Adversarial Network) and other deep learning technologies. The danger of this is that technology can be used to make people believe something is real when it is not. Smartphone desktop applications like Face-App and Fake App are built on this process.These videos can affect a person's integrity.

So identifying and categorizing these videos has become a necessity. We are proposing a new method for detecting Deep Fake Videos. Hopefully, we will be able to make the internet a safer place.

Keywords - Detection, Identification, Convolutional Recurrent Neural Networks, Recurrent Neural Networks, Convolutional neural networks.

CONTENTS

List of Figures

List of Abbreviations

CNN	: *Convolutional Neural Network*
PIL	: *Plot Image Learning*
KNN	: *K Nearest Neighbour*
KWS	: *Keyword Spotting*
SDLC	: *Software Development Life Cycle*
PCL	: *Passive Coherent Location*
DCT	: *Discrete Cosine Transformation*
UAV	: *Unmanned Aerial Vehicle*
FFT	: *Fast Fourier Transformation*
RCS	: *Radar Cross Section*
LSS	: *Low altitude Slowspeed SmallRCS*
RD	: *Range Doppler*
CFAR	: *Constant False Alarm Rate*
DoA	: *Direction of Arrival*
MUSIC	: *Multiple Signal Classification*
EKF	: *Extended Kalman Filter*
UHF	: *Ultra High Frequency*

CHAPTER 1
INTRODUCTION

1.1 Overview

Deep Fakes are AI generated fake videos that look real but are actually fake.The word deepfake combines the terms “deep learning” and “fake,” and is a form of artificial intelligence. Deepfakes are falsified videos made by means of deep learning. Deep learning is “a subset of AI,” and refers to arrangements of algorithms that can learn and make intelligent decisions on their own. But the danger of that is the technology can be used to make people believe something is real when it is not.A deep-learning system can produce a persuasive counterfeit by studying photographs and videos of a target person from multiple angles, and then mimicking its behavior and speech patterns. Once a preliminary fake has been produced, a method known as GANs, or generative adversarial networks, makes it more believable. The GANs process seeks to detect flaws in the forgery, leading to improvements addressing the flaws.

It started out as fun but like any technology, it is being misused. In the beginning, these videos could be identified by human eyes. But due to the development of machine learning, it became easier to create deep fake videos. It has almost become indistinguishable from real videos. Smartphone desktop applications like FaceApp and Fake App are built on this process. These videos can affect a person’s integrity. It became necessary to identify these fake videos from real videos. While AI can be used to make deepfakes, it can also be used to detect them. We are proposing a new method to detect deepfake videos.

1.1.1 Artificial Neural Networks

An artificial neural network (ANN) is the piece of a computing system designed to simulate the way the human brain analyzes and processes information. It is the foundation of artificial intelligence (AI) and solves problems that would prove impossible or difficult by human or statistical standards. ANNs have self-learning capabilities that enable them to produce better results as more data becomes available. [1]

An ANN has hundreds or thousands of artificial neurons called processing units, which

are interconnected by nodes. These processing units are made up of input and output units. The input units receive various forms and structures of information based on an internal weighting system, and the neural network attempts to learn about the information presented to produce one output report

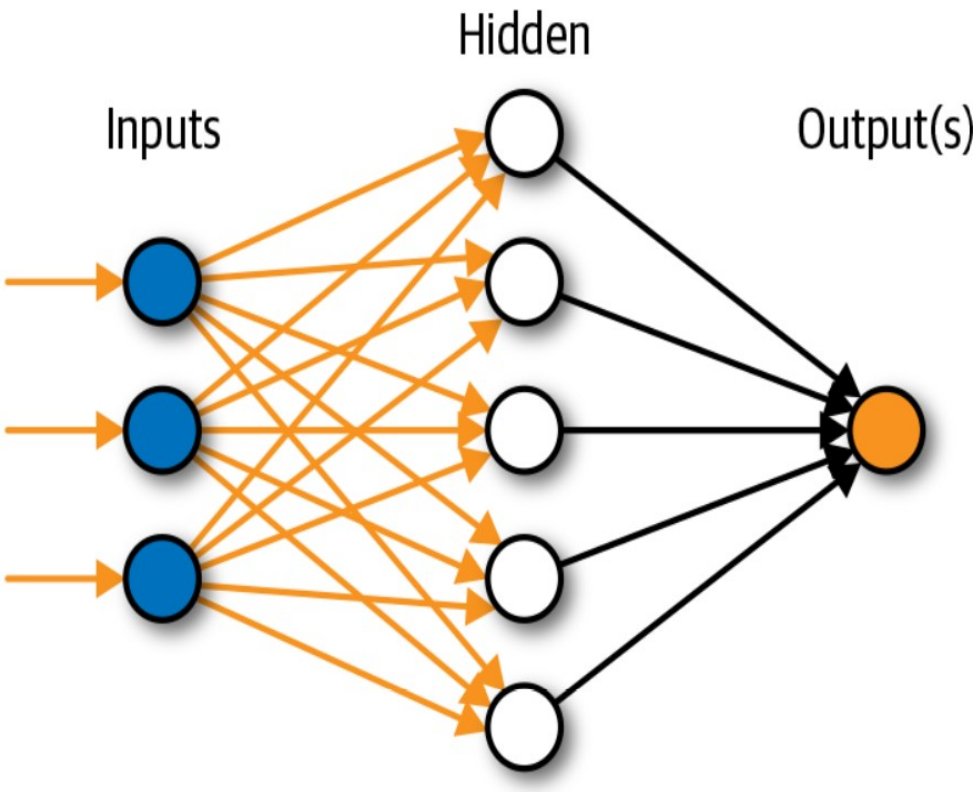


Figure 1.1: Artificial Neural Networks

1.1.2 Recurrent Neural Networks

Recurrent Neural Network(RNN) are a type of Neural Network where the output from previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.[8]

RNN have a “memory” which remembers all information about what has been calculated.

It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

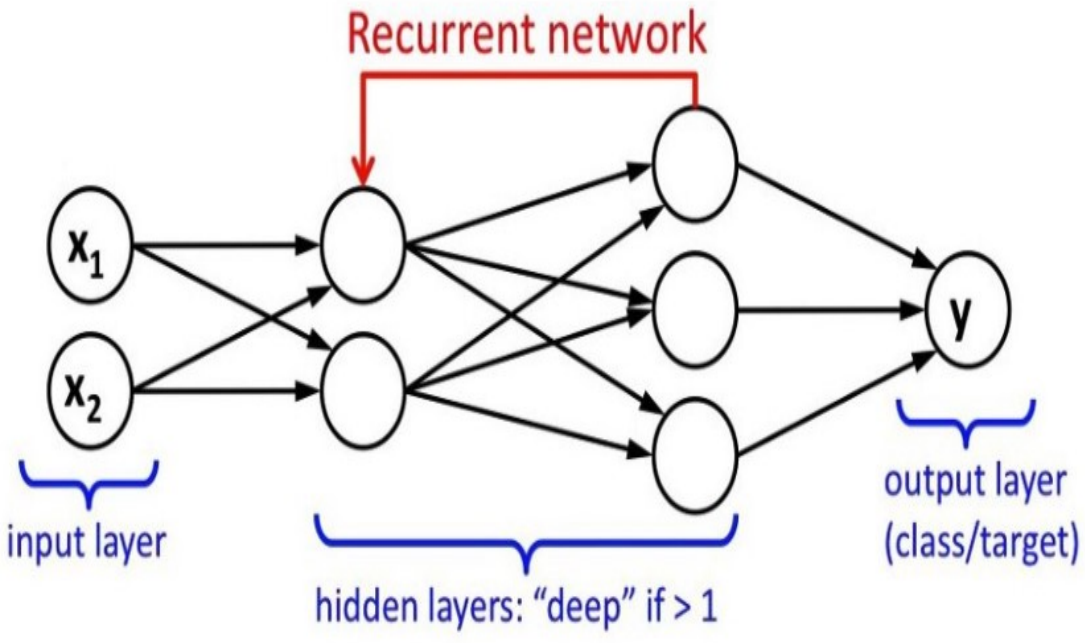


Figure 1.2: Recurrent Neural Networks

1.1.3 Convolutional Neural Networks

A convolutional neural network (CNN) is a specific type of artificial neural network that uses perceptrons, a machine learning unit algorithm, for supervised learning, to analyze data. CNNs apply to image processing, natural language processing and other kinds of cognitive tasks.[3]

CNNs are a fundamental example of deep learning, where a more sophisticated model pushes the evolution of artificial intelligence by offering systems that simulate different types of biological human brain activity.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of

weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

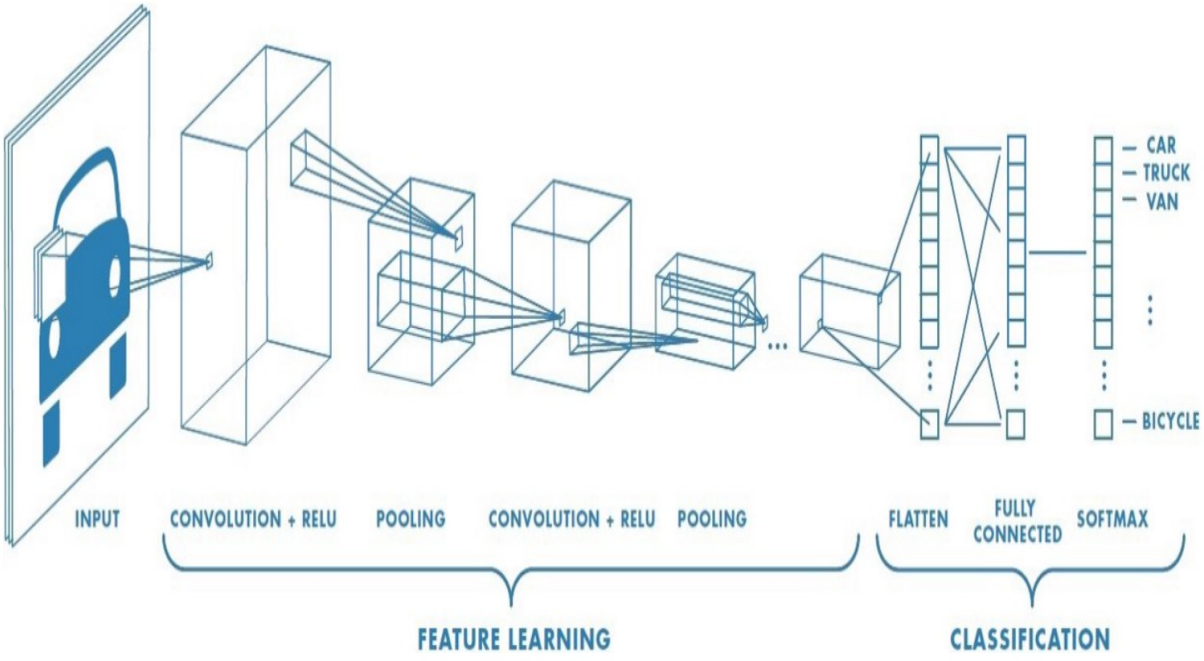


Figure 1.3: Convolutional Neural Networks

1.1.4 DeepFakes

Deepfakes are synthetic media in which a person in an existing image or video is replaced with someone else's likeness. While the act of faking content is not new, deepfakes leverage powerful techniques from machine learning and artificial intelligence to manipulate or generate visual and audio content with a high potential to deceive. [7]

Deepfakes extend the idea of video (or movie) compositing, which has been done for decades. Significant video skills, time, and equipment go into video compositing; video deepfakes require much less skill, time and equipment, although they are often unconvincing to careful observers

The main machine learning methods used to create deepfakes are based on deep learning and involve training generative neural network architectures, such as autoencoders or generative adversarial networks (GANs).

- Autoencoders

  Essentially, autoencoders for deepfake faces in images run a two-step process. Step one is to use a neural network to extract a face from a source image and encode that into a set of features and possibly a mask, typically using several 2D convolution layers, a couple of dense layers, and a softmax layer. Step two is to use another neural network to decode the features, upscale the generated face, rotate and scale the face as needed, and apply the upscaled face to another image. Training an autoencoder for deepfake face generation requires a lot of images of the source and target faces from multiple points of view and in varied lighting conditions.

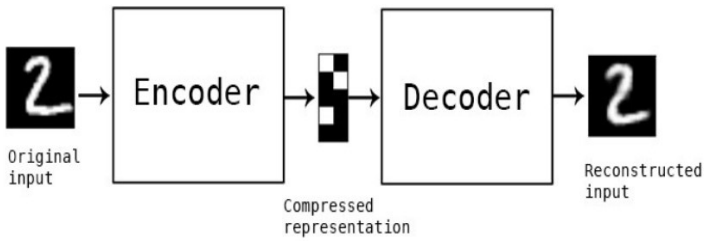


Figure 1.4: Autoencoders example

- GANs

Generative adversarial networks can refine the results of autoencoders, for example, by pitting two neural networks against each other. The generative network tries to create examples that have the same statistics as the original, while the discriminative network tries to detect deviations from the original data distribution.

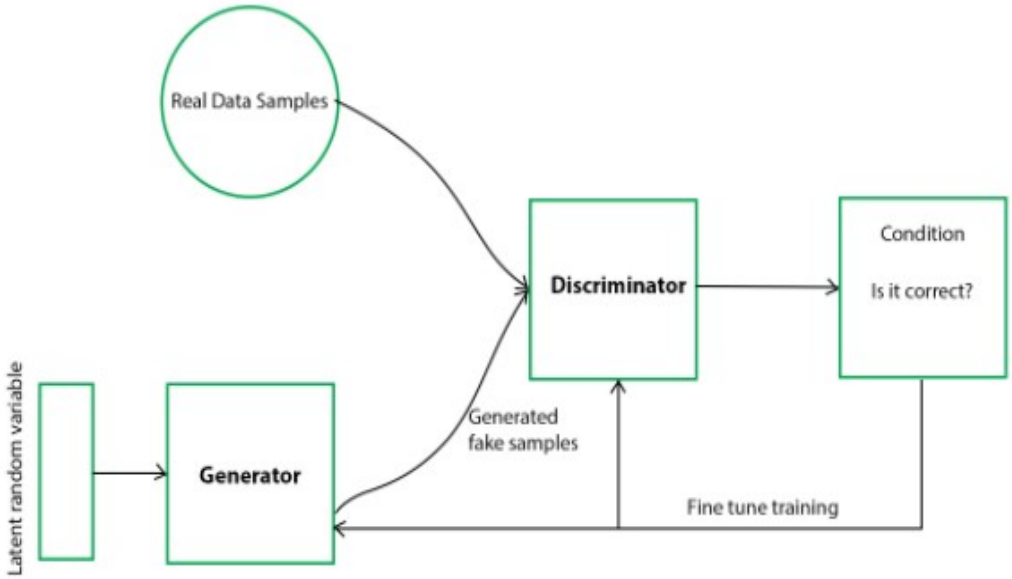


Figure 1.5: Generative Adversarial Networks

1.2 Objectives

The main objective of this project is to introduce a new method used to identify deepfake videos and classify them as real and fake videos based on their features using deep learning.This methods identify fake videos from real videos and there by prevent the usage of these videos in creating political distress, blackmailing, fake terrorism events, etc.

1.3 Organization of the project

The report is organised as follow:

- **Chapter 1:Introduction**
- **Chapter 2:Literature Survey**
- **Chapter 3: Problem Statement**
- **Chapter 4:Project Management**
- **Chapter 5:Proposed System** D
- **Chapter 5:System Requirements & Specification**
- **Chapter 6:Conclusion**
- **References**

CHAPTER 2

LITERATURE SURVEY

2.1 Deepfake Video Detection Using Recurrent Neural Networks

In recent months a machine learning based free software tool has made it easy to create believable face swaps in videos that leaves few traces of manipulation, in what are known as "deepfake" videos. Scenarios where these realistic fake videos are used to create political distress, black-mail someone or fake terrorism events are easily envisioned.This paper proposes a temporal-aware pipeline to automatically detect deepfake videos. Our system uses a convolutional neural network (CNN) to extract frame-level features.These features are then used to train a recurrent neural net-work (RNN) that learns to classify if a video has been subject to manipulation or not. [4]

This paper propose a two-stage analysis composed of a CNN to extract features at the frame level followed by a temporally-aware RNN network to capture temporal inconsistencies between frames introduced by the face-swapping process.A collection of 600 videos have been used to evaluate the proposed method, d, with half of the videos being deepfakes collected from multiple video hosting website.

2.1.1 Creating Deepfake Videos

It is well known that deep learning techniques have been successfully used to enhance the performance of image compression. Especially, the autoencoder has been applied for dimensionality reduction, compact representations of images, and generative models learning . Thus, autoen-coders are able to extract more compressed representations of images with a minimized loss function and are expected to achieve better compression performance than existing image compression standards. The compressed representations or latent vectors that current convolutional autoencoders learn are the first cornerstone behind the faceswap-ping capabilities of . The second insight is the use of two sets of encoder-decoders with shared weights for the encoder networks.[4]

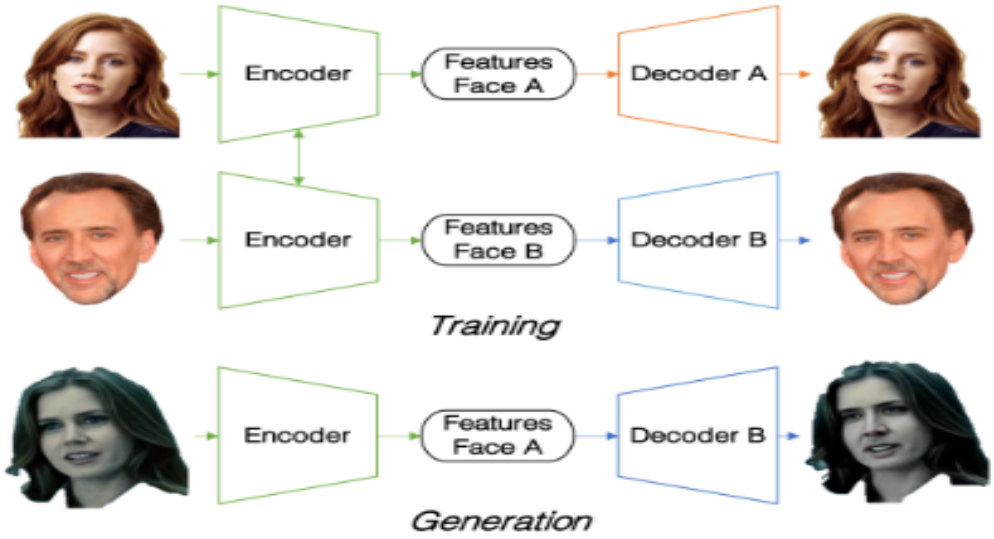


Figure 2.1: Generation of deepfakes using auto encoders

2.1.2 Training

Two sets of training images are required. The first set only has samples of the original face that will be replaced, which can be extracted from the target video that will be manipulated. This first set of images can be further extended with images from other sources for more realistic results. The second set of images contains the desired face that will be swapped in the target video. To ease the training processof the autoencoders, the easiest face swap would have both the original face and target face under similar viewing and illumination conditions. However, this is usually not the case. Multiple camera views, differences in lightning conditions or simply the use of different video codecs makes it difficult for autencoders to produce realistic faces under all conditions. This usually leads to swapped faces that are visually inconsistent with the rest of the scene. This frame-level scene inconsistency will be the first feature that we will exploit with our approach.

If two autoencoders are trained separately on different sets of faces, their latent spaces and representations will be different. This means that each decoder is only able to decode a single kind of latent representations which it has learnt during the training phase. This can be overcome by forcing the two set of autoencoders to share the weights for the encoder networks, yet using two different decoders.

2.1.3 Video Generation

When the training process is complete, we can pass a latent representation of a face generated from the original subject present in the video to the decoder network trained on faces of the subject we want to insert in the video.The decoder will try to reconstruct a face from the new subject, from the information relative to the original subject face present in the video. This process is repeated for every frame in the video where we want to do a face swapping operation. This is usually a second source of scene inconsistency between the swapped face and the re-set of the scene. Because the encoder is not aware of the skin or other scene information it is very common to have boundary effects due to a seamed fusion between the new face and the rest of the frame.

2.1.4 Recurrent Network for Deepfake Detection

The proposed system is composed by a convolutional LSTM structure for processing frame sequences. There are two essential components in a convolutional LSTM:

- CNN for frame feature extraction.
- LSTM for temporal sequence analysis

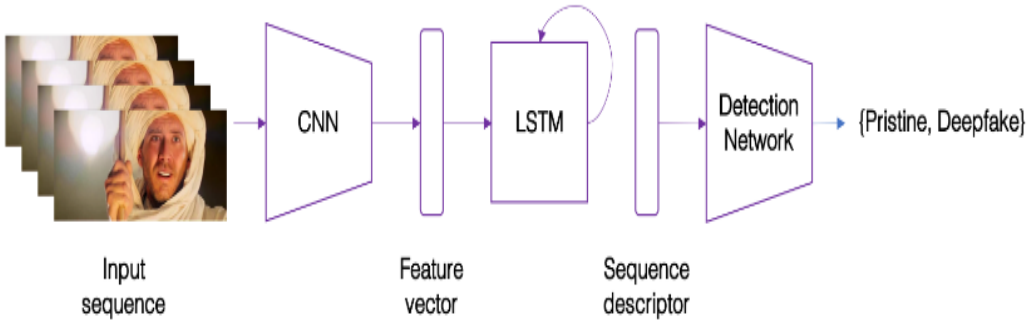


Figure 2.2: Overview of our detection system.

2.1.5 Convolutional LSTM

A convolutional LSTM is employed to produce a temporal sequence descriptor for image manipulation of the shot frame. Aiming at end-to-end learning, an integration of fully-connected layers is used to map the high-dimensional LSTM descriptor to a final detection probability. Specifically, our shallow network consists of two fully-connected layers and one dropout layer to minimize training over-fitting. The convolutional LSTM can be divided into a CNN and a LSTM, which we will describe separately in the following paragraphs.

CNN for Feature Extraction.This method uses InceptionV3 with fully connected layers.The 2048 dimensional feature vectors after the last pooling are used as the sequential LSTM input.

LSTM for Sequence Processing.Sequence of CNN feature vectors of input frames are input and a 2-node neural network with the probabilities of the sequence being part of deep fake video or real video.LSTM model takes a sequence of 2048-dimensional ImageNet feature vectors. LSTM is followed by 512 fully-connected layers with 0.5 chance of dropout. Finally softmax layer is used to compute the probabilities of the frame sequence being either pristine or deep fake.

2.1.6 Architecture

Dataset is selected with 50% of deep fake videos and 50% of real videos. Random split is used to generate 3 disjoint set which is used for training, validation and testing. Balanced splitting on real videos and fake videos are done. This made sure that final set has exactly 50% of each class. Resized every frame to 299 X 299. Length of input sequence is controlled using subsequence sampling. This allows us to identify how many frames are necessary per video to have an accurate detection. Optimizer is set for end to end training of the complete model.

2.2 Effective Fast DeepFake Detection Method Based on Haar Wavelet Transform

DeepFake using Generative Adversarial Networks (GANs) tampered videos reveals a new challenge in today's life. With the inception of GANs, generating high-quality fake videos becomes much easier and in a very realistic manner. Therefore, the development of efficient tools that can automatically detect these fake videos is of paramount importance.he proposed DeepFake detection method takes the advantage of the fact that current DeepFake generation algorithms cannot generate face images with varied resolutions, it is only able to generate new faces with a limited size and resolution, a further distortion and blur is needed to match and fit the fake face with the background and surrounding context in the source video. This transformation causes exclusive blur inconsistency between the generated face and its background in the outcome DeepFake videos, in turn, these artifacts can be effectively spotted by examining the edge pixels in the wavelet domain of the faces in each frame compared to the rest of the frame. A blur inconsistency detection scheme relied on the type of edge and the analysis of its sharpness using Haar wavelet transform is used.Haar wavelet is a sequence of rescaled "square-shaped" functions which together form a wavelet family.[9]

2.2.1 Detection

Linear blurred images can be described as G = H * F + N (1). Where G, F N represent noisy image and Blur function is represented by H matrix. The blur extent is measured by testing the edge features in an image. The edge feature is one of these features that can be used. If the blur is present, both the edge sharpness and its type will be changed and that will indicate whether the face image has been manipulated or not.[9]

Different types of edges are present in an image. Generally, there are three classes of edge type: Dirac-Structure, Step-Structure, and Roof-Structure.The Step-Structure type is divided according to the change of intensity whether it is gradual or not into: "A Step-Structure" and "G Step-Structure". Every image has all types of edges more or less, most of the G Step-Structure and Roof-Structure are sharp enough. In case it is blurred, the edges lose their sharpness. The sharpness parameter is measured by the sharpness parameter alpha.This method detects whether a face image is blurred or not based on Dirac-structure and A Step-Structure. A blur extent is identified by taking sharpness of Roof-Structure and G Step-Structure into account. The sharpness of the edge is indicated by the parameter alpha , if alpha is larger, means

the edge is sharper.

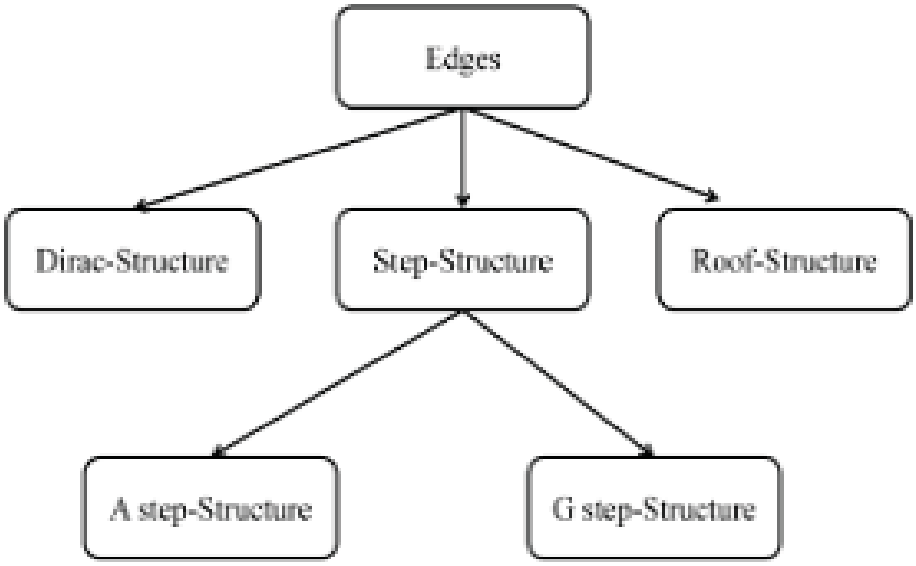


Figure 2.3: Edge Classification

2.2.2 Flow chart of deepfake detection using Haar Wavelet

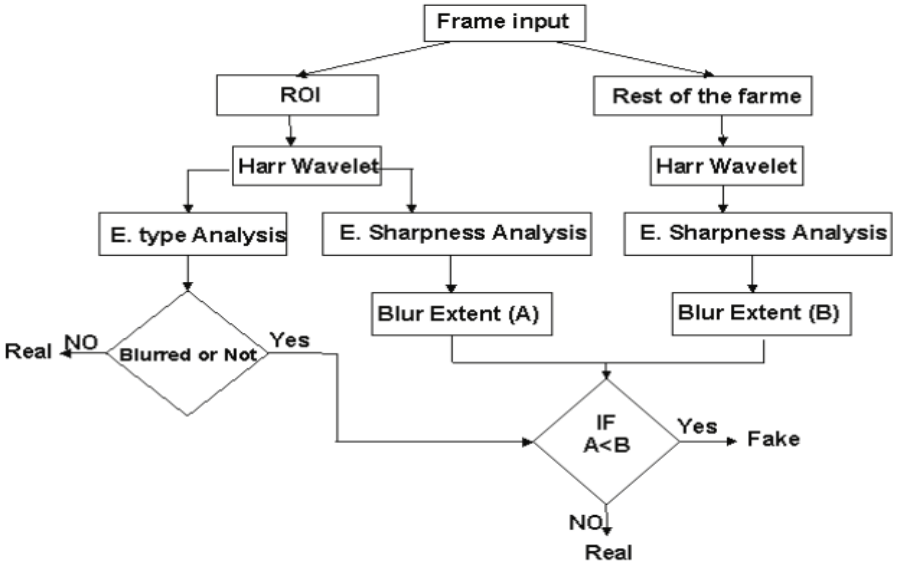


Figure 2.4: DeepFake detection Algorithm using Haar Wavelet

2.2.3 The Effect of Blur on Different Edges

The noise factor N in "(1)" can be neglected since there is a small noise ratio in photos usually acquired from digital cameras.The main blur functions H is a convolution operation that affects the equation and will change the edge property. Take into consideration that there will be no Dirac-Structure or A Step-Structure in the blurred image. On the other hand, both Roof-Structure and G Step-Structure will tend to lose their sharpness (less $\alpha value$).

2.2.4 Sharpness Detection and Edge Type

Discrete Haar functions can be described as functions determined by sampling the Haar functions at 2n points. A matrix form can represent the function in a convenient means. Every row in the matrix H(n), have a discrete Haar sequence Haar(w, t) each alone, where the index (w) represents the number of the Haar function, the discrete point of the function determination interval is identified by index (t). By the following recurrence relation , any dimension of the Haar matrix can be gained;

$$H(n) = \begin{bmatrix} H(n-1) & \otimes [1 \quad 1] \\ 2^{\frac{n-1}{2}} I(n-1) & \otimes [1 \quad 1] \end{bmatrix}, H(0) = 1$$

Where $H(n)$ - the discrete Haar functions of degree 2^n matrix, $I(n)$ - identity matrix of degree 2^n.

Figure 2.5: Recurrence relation for Haar matrix

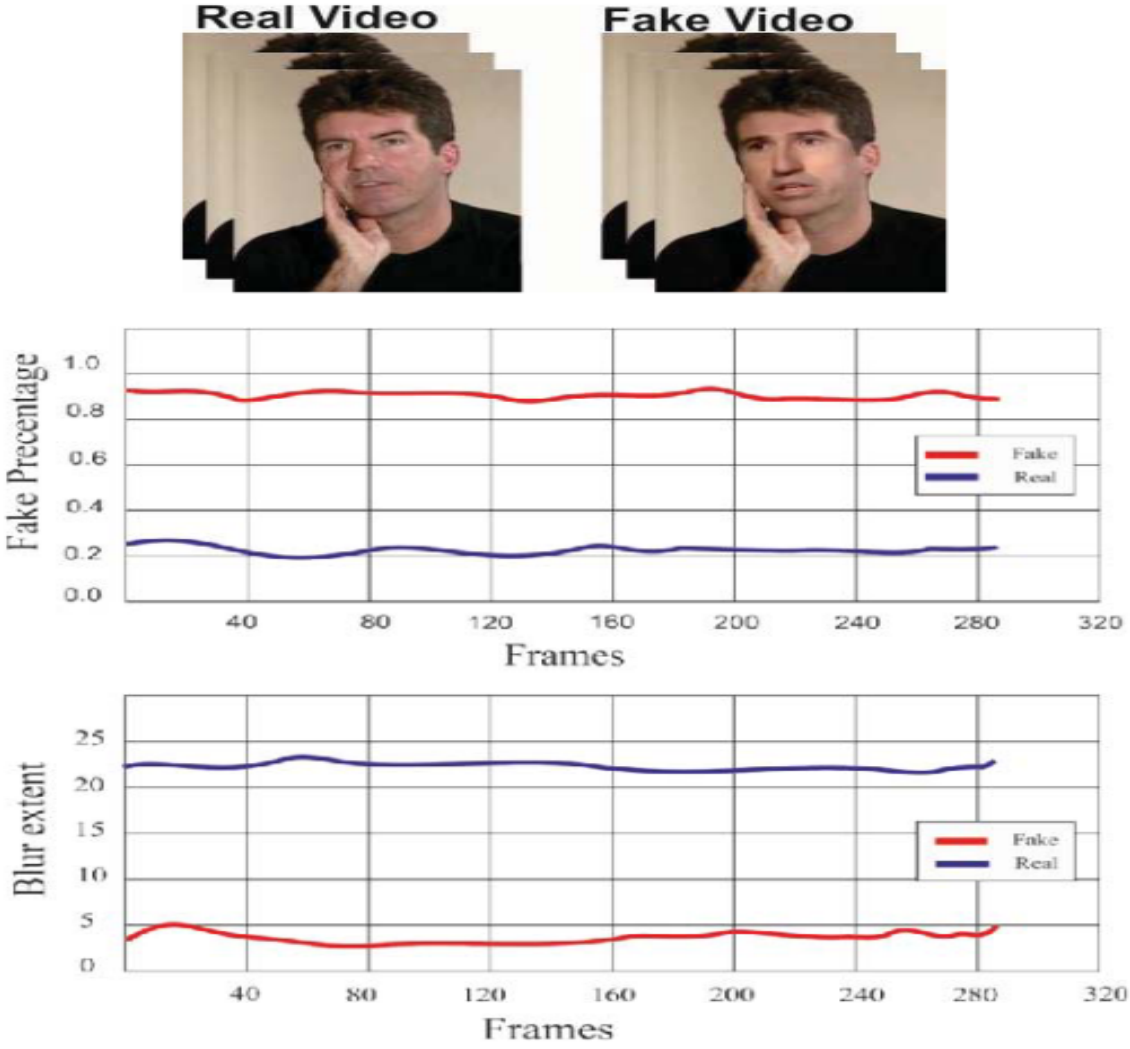


Figure 2.6: Example of the proposed method on one of the DeepFake generated videos from the "UADFV" datase

2.3 DeepFake Source Detection via Interpreting Residuals with Biological Signals

Deep Fakes are AI generated videos which look real but are actually fake.The development of Deep fake was meant for Positive Intent and people used it for Negative intent. The misuse of these led to so many catastrophe across the globe,These are developed using CNN(Convolutional Neural Network) in which the comparison takes place between real and fake videos. The Deep Fake was generally intended for positive use like in the famous movie fast and furious the recreation of Paul Walker after his death but people started using it for negative intendent like Pornography,Poltical issues etc.[2]

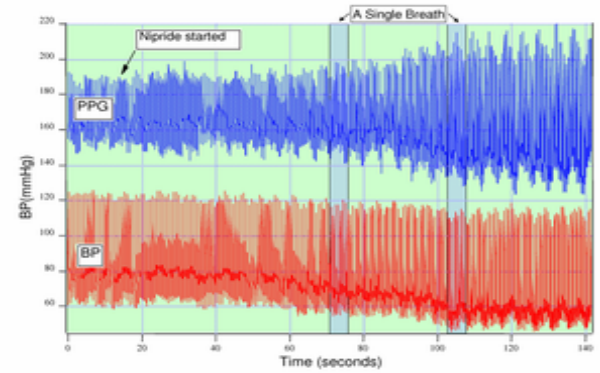


Figure 2.7: PPG

A photoplethysmogram (PPG) is an optically obtained plethysmogram that can be used to detect blood volume changes in the microvascular bed of tissue.

- **PPG and PPG Cells**

  Biological Signals are present in all Human beings.A photoplethysmogram (PPG) is an optically obtained plethysmogram that can be used to detect blood volume changes in the microvascular bed of tissue.Anatomical actions such as heartbeat ,blood flow,or breathing creating subtle changes that are not visible to the eye but still detectable computationally.

  In our present scenario there is no generative network for creating Deep fake with consistent PPG. A PPG is often obtained by using a pulse oximeter which illuminates the

skin and measures changes in light absorption; each person has a different signature with respect to the signals when measured against noise.

They extracted 32 raw PPG signals from different locations in the face, from a window of frames, from a video of windows ,they then encode the signals along with their spectral density into a spatiotemporal block, which is so-called PPG cell.This cell is then fed into to the an off-the-shelf neural network to recognise the signatures of the distinct residuals of the source generative models.At last combine per sequence predictions into a per video prediction using average log of odds .Their key finding emerges from the fact that we can interpret these biological signals as fake heart beats that contain a signature transformation of the residuals per model.

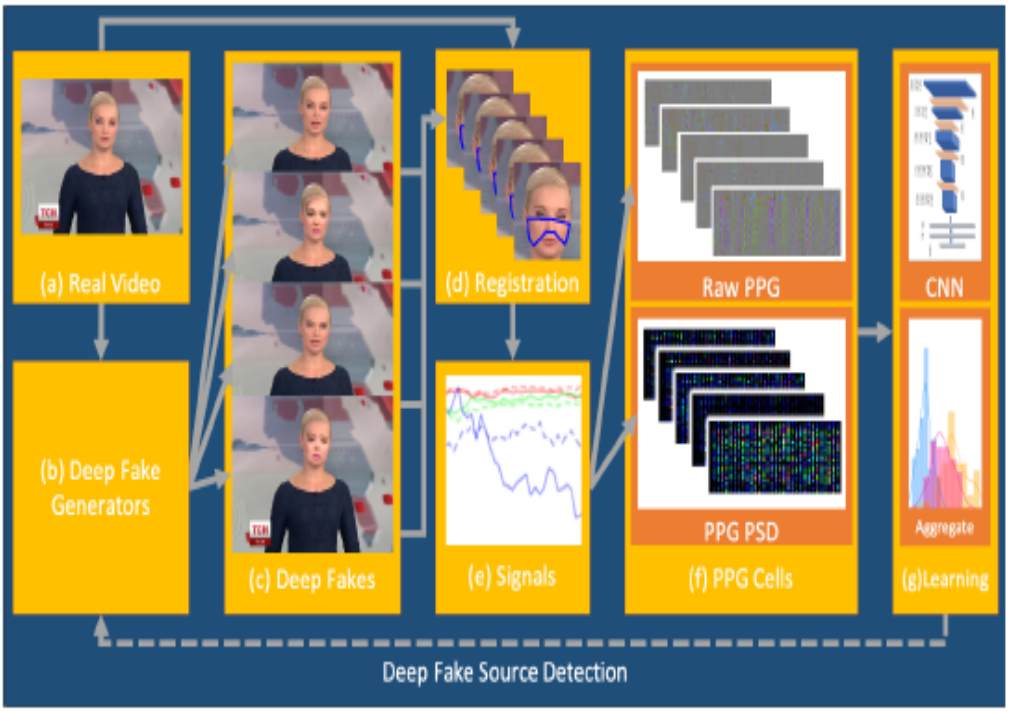


Figure 2.8: Working Overview

- **WORKING:**

1. **Defining PPG**:

The first step is to capture the characteristics of biological signals consis-

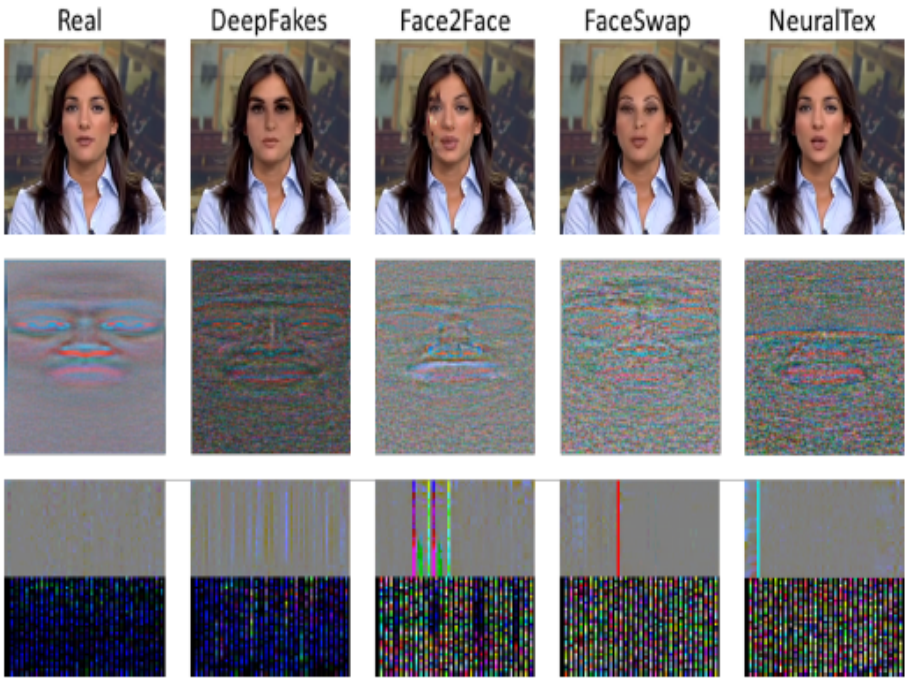


Figure 2.9: PPG cells comparison

tently,for that define a novel spatio-temporalblock, called the PPG cell.The PPG cells combine several raw PPG signals and their power spectra, extracted from a fixed window. The generation of PPG cells starts with finding the face in every frame using a face detector. In case the window contains multiple faces, we process signals individually and aggregate the results in the final step.

2. **Extraction of ROI**:

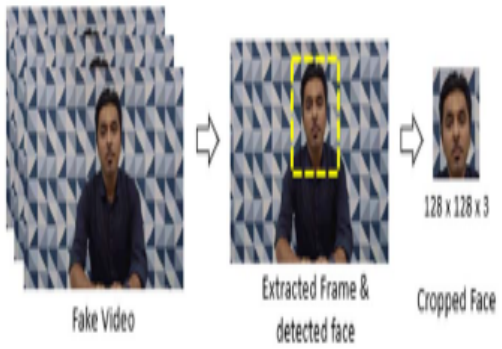


Figure 2.10: Extraction of ROI

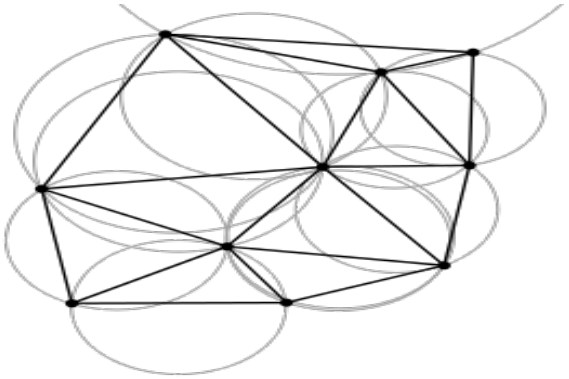


Figure 2.11: Delaunay triangulation

The second step is to extract regions of interests (ROI)from the detected faces that have as much stable PPG signals as possible.Biological signals are sensitive to facial movements, illumination variations, and facial occlusions.In order to extract these areas robustly, They used the face region between eye and mouth regions,maximizing the skin exposure,As the PPG signals from different face regions are correlated with each other, locating the ROIs and measuring their correlation become a crucial step to enhance the detection.

3. **ALIGNING OF ROI:**

The third step involves aligning these nonlinear ROIs to a rectangular image,We employ Delaunay triangulation,followed by a nonlinear affine transformation per triangle to transform each triangle into the rectified image.

4. **CALCULATING PPG**

In the fourth step, They divided each image into 32 equal size squares and calculate the raw Chrom-PPG signal per square in a fixed window with the size of w frames, without interruptions in face detection Then, They calculated the Chrom-PPG in the rectified image since it produces more reliable PPG signals.Each window have w times 32 raw PPG values.They reorganized these into a matrix of 32 rows and w columns,forming the base of the PPG cells as shown in Figure top half of bottom rows .The bright columns correspond to significant motion or illumination changes where the PPG signal deviates abruptly.

5. **FINAL STEP:**

The final step adds the information from the frequency domain to the PPG cells to calculate the power spectral density of each raw PPG value in the window and scale it to w size then concatenate the power spectra to the bottom to generate PPG cells with 64 rows and w columns.To analyze the contribution of the spectral information, we conduct experiments on PPG cells both with and without this last step and compare their accuracies.The projection of residuals of deep fake generators into the biological signal domain creates a unique pattern that can be utilized for source detection.GAN residuals can be approximated by consistent noise in fake images.

Apply temporal non-local means denoising on the aligned face in one frame from each video in FF then accumulate and normalize the difference of original and denoised images, and subtract the noise of the real images from each corresponding fake residual to obtain the middle row in Figure containing the ”fingerprint” per generator.For the real class, demonstrate the overall noise accumulation.The colors of PPG-PSD correspond to different frequencies in the spectra of these residuals, and some of these frequencies are actually visible in the residual accumulation images. The main observation follows this correlation between the residuals and PPG cells: residuals create unique variations in the “deep fake heart beats” per model.

2.3.1 Implementation

The system is implemented in python utilizing Open-Face library for face detection, OpenCV for image processing, and Keras for neural network implementations ,Training and Testing NVIDIA GTX 1060 GPU with tractable training times .The most computationally expensive part of the system is the extraction of PPG cells from large datasets, which is a one time process per video.The process is to analysis, results, and some ablation studies. Unless otherwise noted, we set our testbed as the FF dataset with the same 70%-vs-30% split – 700 real videos and 4*700 deep fakes for training, and 300 real videos and 4*300 deep fakes for testing.

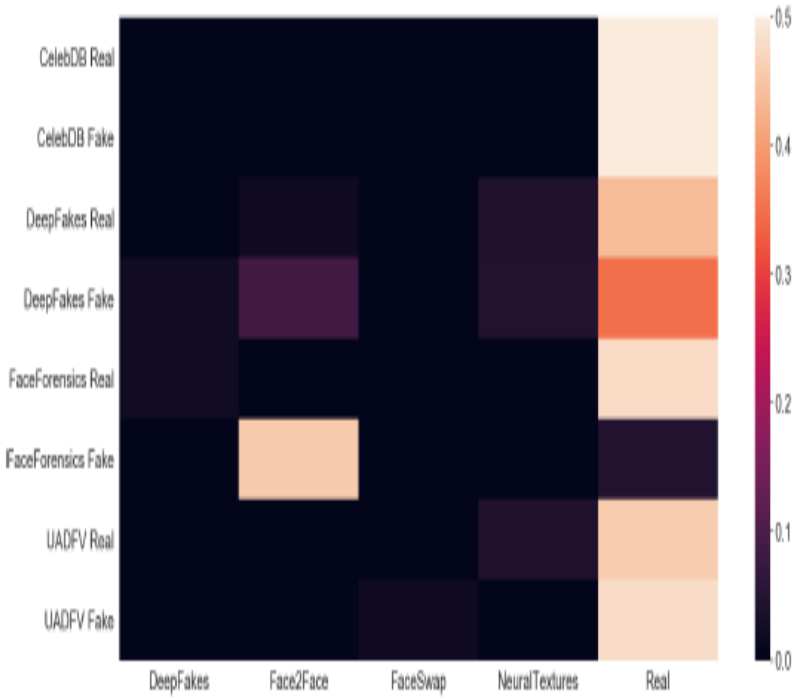


Figure 2.12: Unseen dataset classification

2.3.2 Extension to new models

1. **DATASET:** With respect to the data source the datasets can be of two types:

   (1) datasets with single model generation

   (2) datasets using multiple generative sources.

   some of the common examples for deepfake datasets are UADFV dataset contains 48 real and 48 fake videos generated by FakeAPP,DeepfakeTIMIT dataset have 650 deep fake videos generated using faceswap-GAN where as vidtimit videos are used as originals. FaceForensics dataset congregates 1,004 videos from the internet with their deep fake versions created by Face2Face, resulting in 2,008 videos. Celeb-DF dataset collects 590 real videos of famous actors, with 5,639 deep fake versions generated by an improved synthesis process A typical dataset generated by multiple generative methods is the com-

monly used FaceForensics++ (FF) dataset, which includes 1,000 real videos and 4,000 fake videos, generated by four generative models – FaceSwap, Face2Face, Deepfakes, and Neural Textures.

1.1 **DATA ACQUISITION:**

To acquire the PPG signals, the ROI of the videos source generated were recorded while in different background and lighting setup, this will enable us to publish the dataset publicly in order to be utilized by the research community while ensuring that no privacy or security regulations are being breached.

2.3.3 Neural Network- Convolutional Neural Network

Since CNN is more emphasized here, details of CNN is described below.

Convolutional Neural Network:

One of the most popular algorithm used in computer vision today is Convolutional Neural Network or CNN. Convolutional Neural Networks have a different architecture than regular Neural Networks. Regular Neural Networks transform an input by putting it through a series of hidden layers. Every layer is made up of a set of neurons, where each layer is fully connected to all neurons in the layer before. Finally, there is a last fully-connected layer — the output layer — that represent the predictions. Convolutional Neural Networks are a bit different. First of all, the layers are organised in 3 dimensions: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension. CNN is composed of two major parts:

1. Feature Extraction: In this part, the network will perform a series of convolutions and pooling operations during which the features are detected. If you had a picture of a zebra, this is the part where the network would recognize its stripes, two ears, and four legs.

2. Classification: Here, the fully connected layers will serve as a classifier on top of these extracted features. They will assign a probability for the object on the image being what

the algorithm predicts it is.

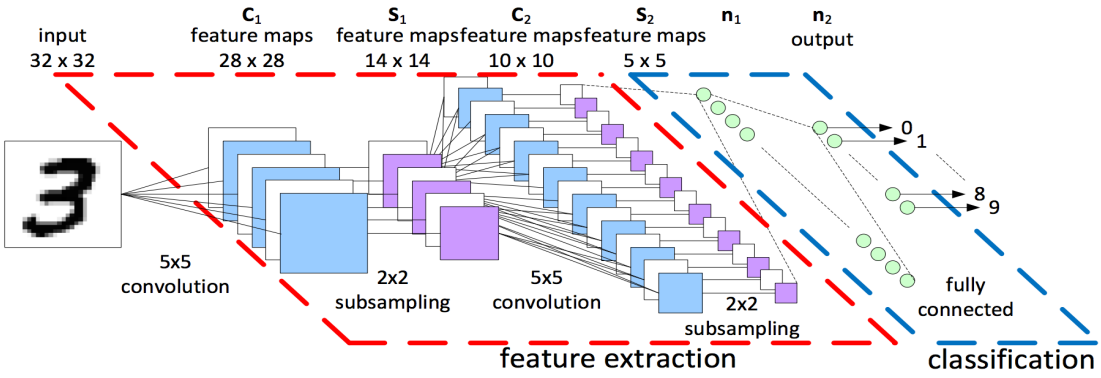


Figure 2.13: Convolutional Neural Network

There are squares and lines inside the red dotted region which we will break it down later. The green circles inside the blue dotted region named classification is the neural network or multi-layer perceptron which acts as a classifier. The inputs to this network come from the preceding part named feature extraction. Feature extraction is the part of CNN architecture from where this network derives its name. Convolution is the mathematical operation which is central to the efficacy of this algorithm. Lets understand on a high level what happens inside the red enclosed region. The input to the red region is the image which we want to classify and the output is a set of features.

Feature Extraction: Convolution

Convolution in CNN is performed on an input image using a filter or a kernel. To understand filtering and convolution you will have to scan the screen starting from top left to right and moving down a bit after covering the width of the screen and repeating the same process until you are done scanning the whole screen. For instance if the input image and the filter look like following: The filter (green) slides over the input image (blue) one pixel at a time starting from the top left. The filter multiplies its own values with the overlapping values of the image while sliding over it and adds all of them up to output a single value for each overlap until the entire image is traversed.

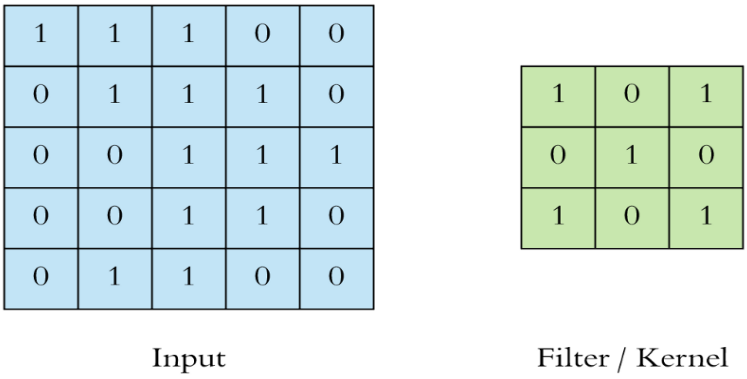


Figure 2.14: Feature Extraction

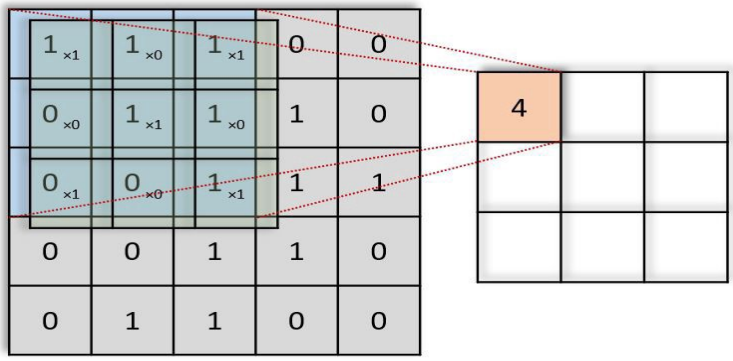


Figure 2.15: Feature Extraction

(1×1+0×1+1×1)+(0×0+1×1+1×0)+(1×0+0×0+1×1)=4 Similarly we compute the other values of the output matrix. Note that the top left value, which is 4, in the output matrix depends only on the 9 values (3x3) on the top left of the original image matrix. It does not change even if the rest of the values in the image change. This is the receptive field of this output value or neuron in our CNN. Each value in our output matrix is sensitive to only a particular region in our original image.

Feature Extraction: Padding

There are two types of results to the operation — one in which the convoluted feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding or Same Padding in the case of the latter. In above example our padding is 1.

Feature Extraction: Example

Lets say we have a handwritten digit image like the one below. We want to extract out only the horizontal edges or lines from the image. We will use a filter or kernel which when convoluted with the original image dims out all those areas which do not have horizontal edges:

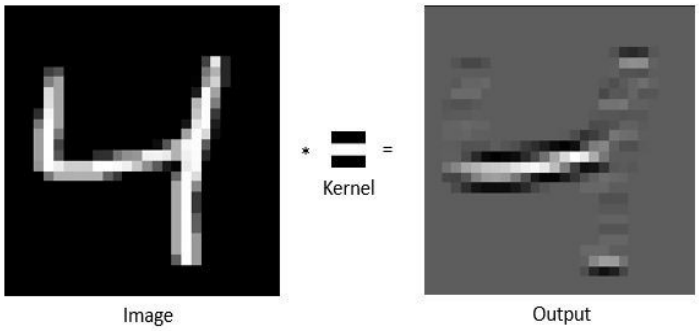


Figure 2.16: Horizontal Filter

Notice how the output image only has the horizontal white line and rest of the image is dimmed. The kernel here is like a peephole which is a horizontal slit. Similarly for a vertical edge extractor the filter is like a vertical slit peephole and the output would look like:

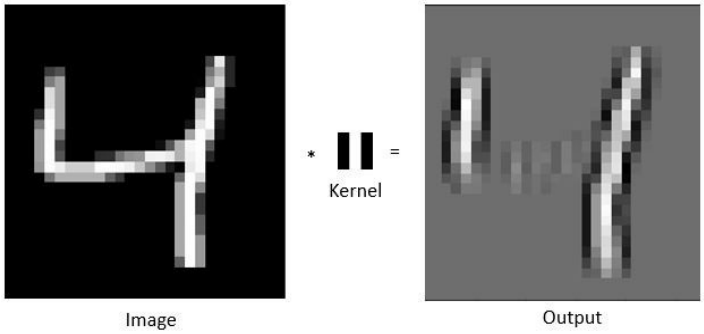


Figure 2.17: Vertical Filter

Feature Extraction: Non-Linearity

After sliding our filter over the original image the output which we get is passed through another mathematical function which is called an activation function. The activation function usually used in most cases in CNN feature extraction is ReLu which stands for Rectified Linear

Unit. Which simply converts all of the negative values to 0 and keeps the positive values the same:

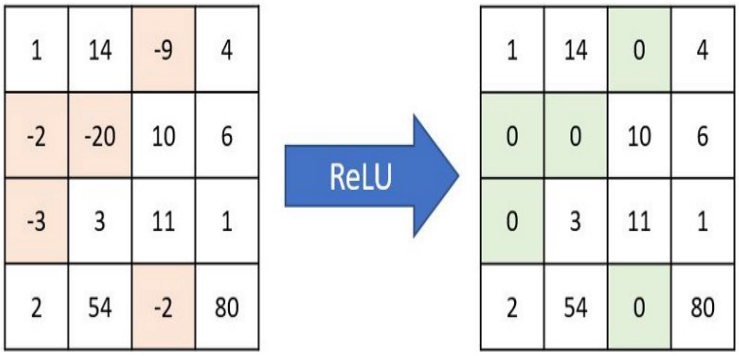


Figure 2.18: Feature Extraction with ReLu

After passing the outputs through ReLu functions they look like:

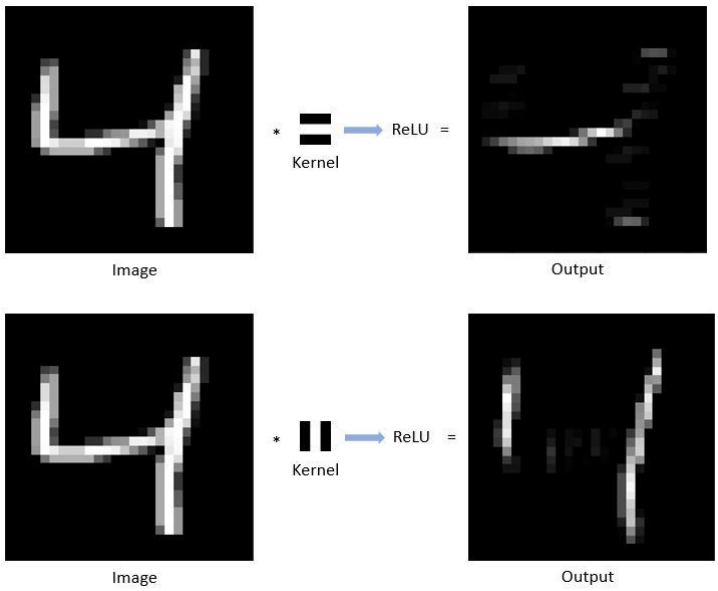


Figure 2.19: Input after filtering with ReLu

Feature Extraction: Pooling

After a convolution layer once you get the feature maps, it is common to add a pooling or a sub-sampling layer in CNN layers. Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the

computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. Pooling shortens the training time and controls over-fitting. There are two types of Pooling- Max pool and Average pool. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

Classification — Fully Connected Layer (FC Layer):

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space. Example of CNN network:

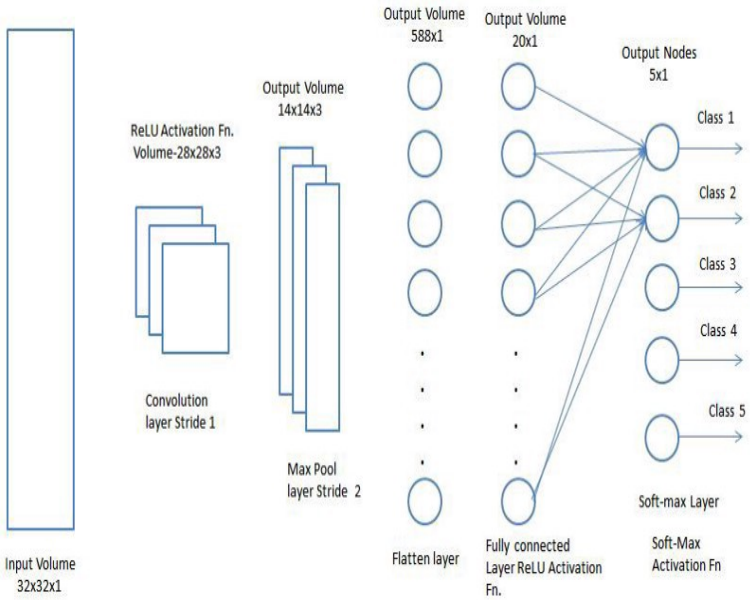


Figure 2.20: Fully Connected Model

Now that we have converted our input image into a suitable form, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification technique.

2.3.4 Methodology

The three main objectives we are attempting to achieve are, firstly, we are interested in investigating the performance of the binary classification aspect on the problem and comparing the results with the literature. Secondly, observing and evaluating the outcome of the multi-class identification classification aspect of the problem and attempting to show the best algorithm out of the three algorithms implemented using different evaluation metrics. Finally, we aim to answer the question of which algorithm has the lowest training and evaluation time. In order to fulfill the three objectives above, different algorithms are compared and evaluated based on their accuracy, F1 score, precision and recall metrics. In order to understand the metrics, concept of confusion matrix needs to be explained.

2.3.5 Confusion Matrix

Confusion Matrix is a table with 4 different combinations of predicted and actual values.

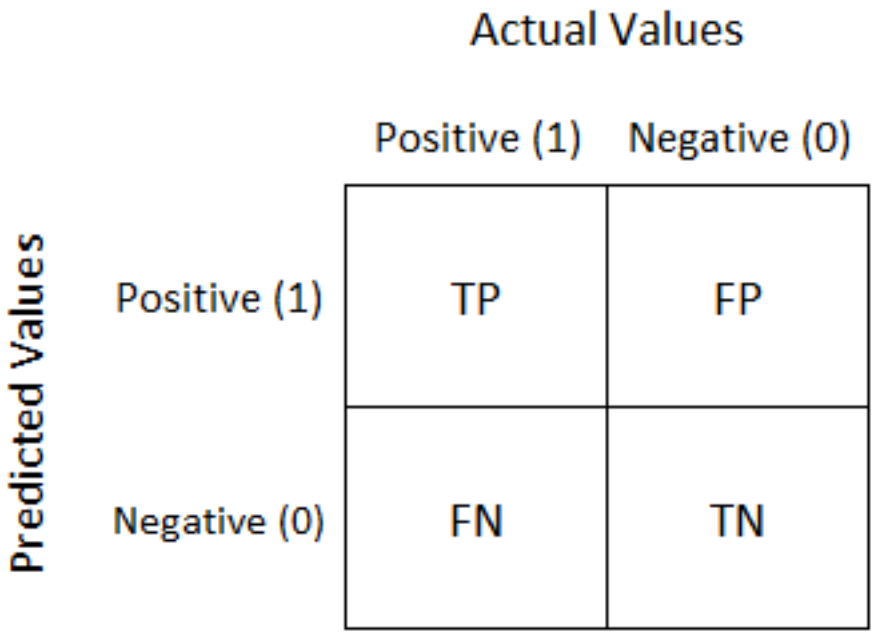


Figure 2.21: Confusion Matrix

1. **True Positive (TP):** A true positive is an outcome where the model correctly predicts the positive class.

2. **True Negative (TN):** A true negative is an outcome where the model correctly predicts the negative class.

3. **False Positive (FP):** A false positive is an outcome where the model incorrectly predicts the positive class.

4. **False Negative (FN):** A false negative is an outcome where the model incorrectly predicts the negative class.

Metrics

1. **Accuracy:** It is the ratio of number of correct predictions to the total number of input samples. It works well only if there are equal number of samples belonging to each class.

Accuracy= (TP + TN)/(TP + TN + FP + FN)

2. **Precision:** It is the number of correct positive results divided by the number of positive results predicted by the classifier.

Precision= TP/(TP + FP)

3. **Recall:** It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

Recall= TP/(TP + FN)

4. **F1 Score:** F1 Score is used to measure a test's accuracy. F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is

(it does not miss a significant number of instances).

F1 Score= (2*Precision*Recall)/(Precision + Recall)

In order to ensure that every algorithm is performing at its optimum, we have carefully chosen the steps below to define the termination condition of the training phase:

1. Executing the algorithm with a very large number of training steps.

2. At an interval of 100 steps, the trained model is tested on the validation-set and the accuracy is calculated and recorded.

3. We compare the new accuracy of the validation-set with the best accuracy achieved so far.

4. If the accuracy did not improve over three successive validation tests, we test the trained model on the testing set and report the observed results.

2.3.6 Results

Source Classification Accuracy

To better evaluate our video source classification, we analyze how uniquely each generative model is detected using the biological signals as a modulator for residuals. This analysis supports our claim of different generative models having signature patterns projected to the biological signal space.

2.4 Classification of Real Fake Images Using One-Class Variational Encoder

Deepfake videos are AI generated videos that look real but are actually fake. The great threat of deepfake is that it is impossible for us to identify whether it is fake or not using our eyes as it is much perfect.Using such videos and images it is easy for malicious abuser to create arbitrary fake news and fool and mislead the public. Generally Binary Classification methods are used for classification of real and fake images. But it requires large dataset of real and fake images for training the model. It is challenging to collect sufficient fake images in advance. When new deepfake generation methods are introduced, only little deepfake dataset are available for training the model. So the output of such models won't be accurate. This method which requires only real images for training so that data scarcity limitation can be solved and gives output with 97.5 percent accuracy.

Dataset used here is FaceForensics++ which is comprised of 5 types of deepfake data as well as normal data, as our baseline dataset.

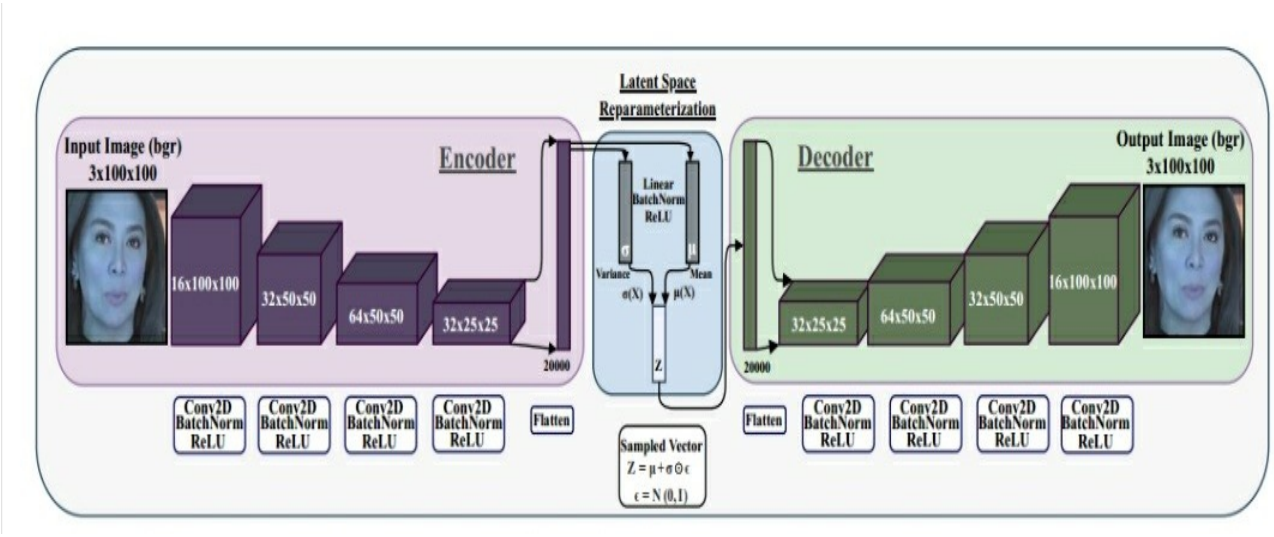


Figure 2.22: Architecture Diagram

This is the architecture diagram of One-Class Variational Autoencoder[6]. It consists of encoder and decoder. At Encoder side image is given as input and we are scaling it at each stage. Convolutional layers are used and batch normalization with ReLU activation is

applied.A distribution is returned by this encoder and latent space reparameterization is done to that distribution i.e. mean and variance is found and Z is calculated using equation Z = (x) × + μ (x), N (0, I). Z is given as input to decoder and again convolutional layers and batch normalization are applied. Thus reconstructed image is produced.Then RMSE value is calculated using the equation

$$rmse = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(X_i' - X_i)^2},$$

where X is the original input and X' is the reconstructed output.If it is real image, its RMSE values will be low and if it is fake, RMSE values will be high.

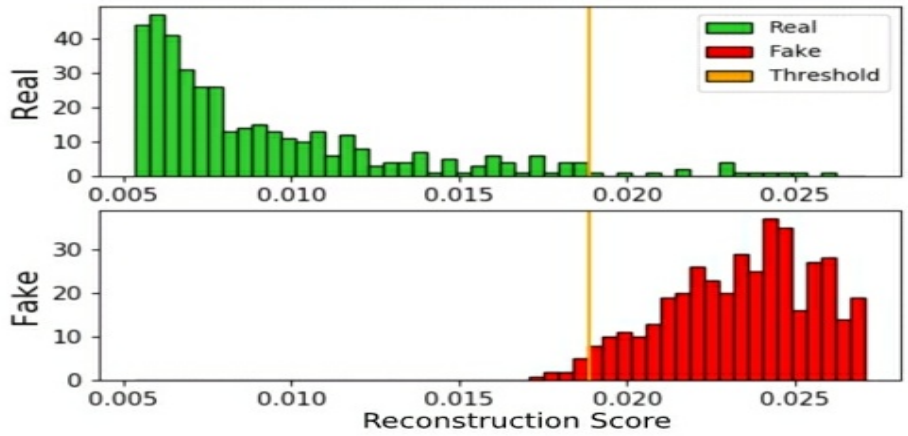
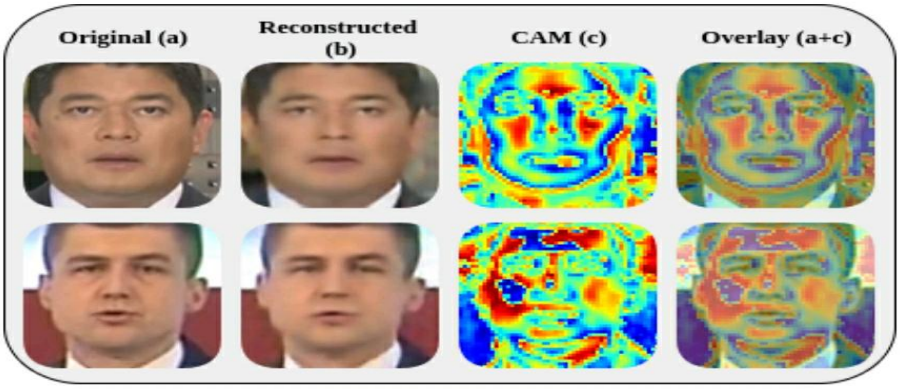
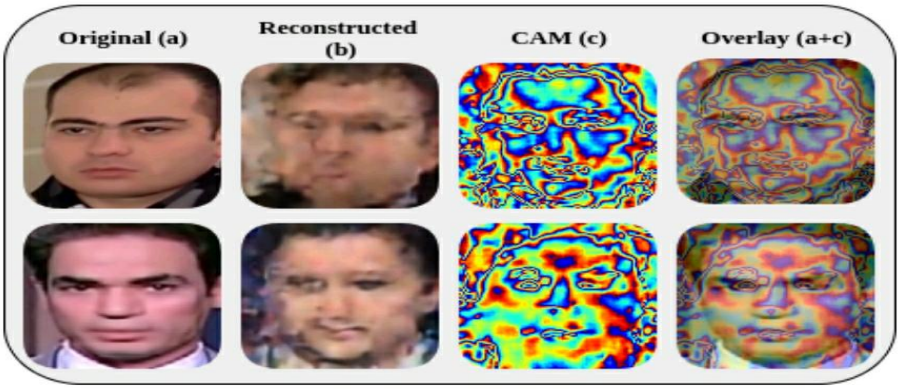


Figure 2.23: Histogram

Inorder to compare and visualize the learned features between real and faking images, we can employ GRADCAM (Gradient-weighted Class Activation Mapping). It highlights important region in the image for predicting the concept. Here we can clearly observe that OC-FakeDect produces intense activation around the face areas of real images, Face features such as nose, the forehead and the cheeks are clearly localized via class activation mapping. But in fake images it produces disordered and dispersed activation patterns. It complicates the perception of essential facial region compared to that in real images.



(a) Real face images.



(b) Fake face images.

Performance summary based on accuracy for all methods on the Deepfakes, NeuralTextures, FaceSwap, Face2Face and Deepfake-datasets

Model	NT	DF	FS	F2F	DFD
OC-AE	54.60	49.20	48.20	47.80	66.90
OC-FakeDect1	95.30	86.20	84.80	70.70	98.00
OC-FakeDect2	**97.50**	88.40	86.10	71.20	**98.20**
MesoNet [25]	40.67	87.27	61.17	56.20	N/A
Xcep. Net [25]	80.67	**96.36**	**90.29**	**86.86**	N/A

Figure 2.24: performance table

Here MesoNet and Xcep.Net are two class based methods. Here we can see OC-FakeDect scored highest accuracy in Neural Textures and Deepfake Dataset. But for Deepfake, Faceswap and Face2Face dataset Xcep.Net scored high accuracy, but the disadvantage is that it requires both dataset of real and fake for training. If enough dataset of fake images are not available for training, results won't be this much accurate. But in one class based methods, only real images are required for training and in those methods, OC FakeDect method scores highest accuracy Better performance is only on the NT dataset and DFD dataset compared to XceptionNet, which is the current state-of-the-art method. So it needs performance improvement in other datasets as well.OC-based detection is indeed a viable option for the development of a generalized deepfakes detector, without the need for any fake images during the training phase. This cost effective method helps in preventing the usage of deepfakes in creating political distress,blackmailing, fake terrorism events, etc. and identity and privacy of person can be protected.

2.5 Classification of DeepFake Using Mouth Features

Technological innovation especially in handheld devices with high quality cameras which are available on latest smartphones in combination with the spread of AI tools, models and apps have resulted in a huge number of videos of world famous celebrities and political leaders that have been used to mislead, To convey fake news for either political gains or in order to ridicule certain people which lead to catastrophe.With the growing of billions of digital images and videos daily across the several social media platforms together with the latest of deep learning apps allow us to create a fake video within a matter of minutes. Videos can be taken from huge repositories that are available on the internet, and with a few things in deep learning apps, it is very easy to create genuine-looking DeepFake videos. The potential impact of a fake video of a world leader can be a catastrophic and can have far-reaching implications for the world's economy and political stability of the country and can threaten the global peace, another side the use of these in generation of Pornographic videos..[5]

In order to prevent this,They used a deep-fake detection model with mouth features (DFT-MF), using deep learning approach to detect Deepfake videos by isolating, analyzing and verifying lip/mouth movement.They used CNN deep learning algorithm to classify deepfake videos. MoviePy, which an open-source application for editing and cutting videos, to cut the video based on certain words in which the mouth appears open and the teeth are visible. This is different from all other algorithms, that extract all images from the video and then attempt to identify the facial region within the extracted images.

1. **Creation Of DeepFake Videos**

DeepFake is a video that has been constructed to make a person appear to say or do something that they never said or did which looks like real but are actually Fake. The first Deepfake was on early 2018,through the utilization of generative adversarial networks(GANs), which have led to the development of tools like Open Face, Swap Face2Face and Fake App that can generate videos from a large volume of images with only the requirement of minimum manual editing. Table 1 contains some of Common tools that Used to Create Deepfake videos. The main aim of deepfake algorithm is to allows users to /transpose/replace the face of one person in a video with the face of another in a,realistic possible manner which is impossible to identify. To build deepfake videos, the deepfaker's need to assign two important GAN algorithm components,namely: the encoder network that will help to achieve a dimensional reduction by encoding the data starting from the input layer until it reduces the number of variables. The second one is the decoder network which reduces variables to create a new output very similar to the original as of illustrated in figure 1.

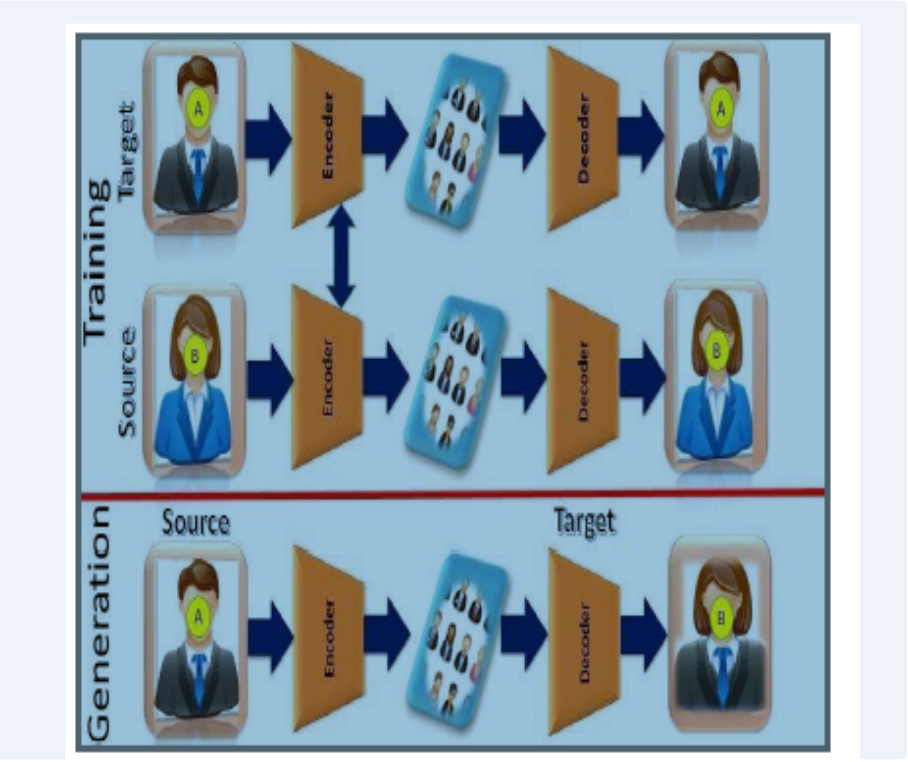


Figure 2.25: Creation of DeepFake

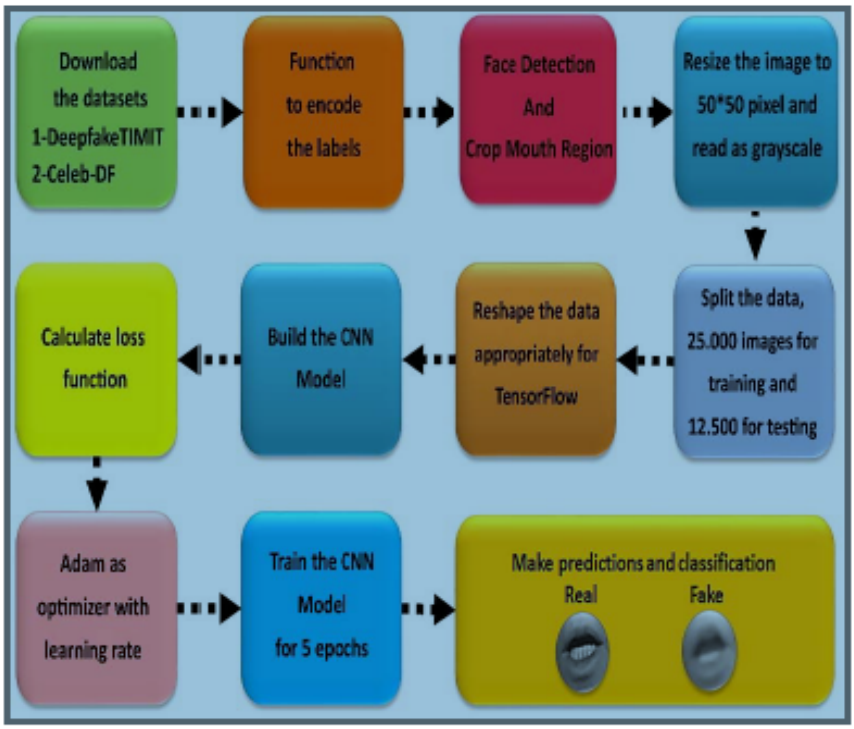


Figure 2.26: Working steps

- **Different Phases in DeepFake model**:

1. DataCollection

   In this stage, the information will be collected from relevant sources to create a new dataset that contains a combination of deepfake and real videos. They used the Deepfake Forensics (Celeb-DF) dataset and the Deepfake Vid-TIMIT dataset to extract all frames from the video before filtering irrelevant frames.

2. Pre-processing

   Prior to performing analysis on the image frames, some pre-processing is required. Face detection is one of the most essential steps of this work to enable us to filter out image frames (or parts thereof) that do not contain faces . TO this end, the Dlib classifier will be used to detect face landmarks and eliminate all unnecessary frames in the video.

3. Mouth Cropping

The DFT-MF model focuses on area surrounding the mouth,especially the teeth; therefore, the mouth area will be croppedfrom a face in the frame. Working on a typical image frame ofa face, the facial landmark detector inside the Dlib library is used to estimate the location of 68 (X, Y)-coordinates that graph to specific facial structures, these coordinates can be visualized as follows:

• The mouth can be located through points (49, 68).

• The right eyebrow through points (18, 22).

• The left eyebrow through points (23, 27).

• The right eye using points (37, 42).

• The left eye using points (43, 48).

• The nose is defined with points (28, 36).

• The jaw via points (1, 17).

The Dlib face landmark detector will return a shape object containing the face bounding box at 68 (x, y)-coordinates of the facial landmark regions in the image figure 3 illustrates that and at the points (49,68) the mouth can be located. This area will be used by DFT-MF model to crop the mouth region based on the ratio between each two-point upper lips and the lower lips.

The next step is to exclude all frames that contains closed mouth by calculating distances between lips of the person. This is because of an image with a closed mouth has no fake value as nothing is being uttered in that frame. We will be tracking the open mouth, which the teeth with reasonable clarity so as to obtain high accuracy and increase efficiency of the model, figure 4 illustrates the idea.

Figure 2.27: Face landmarks

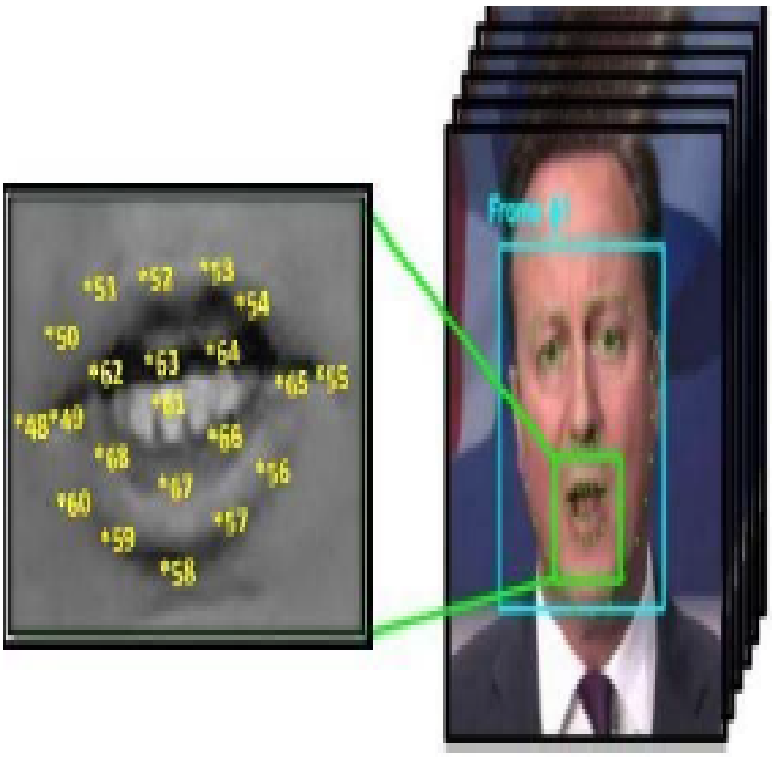


Figure 2.28: Mouth Cropped

4. DeepFake Video Classification

The test data is split into fake and real videos for training 25000 Frames: 12500 frames were labeled as Real and 12500 frames were labeled as Fake videos. As seen in figures

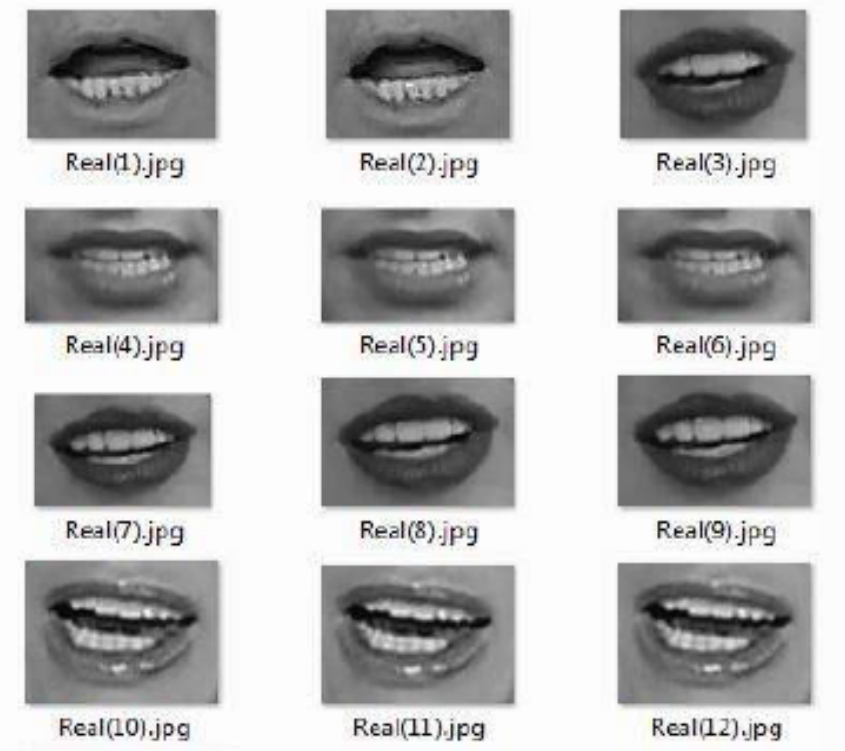


Figure 2.29: Mouth Samples

Figure 2.30: Mouth samples with teeth

The DFT-MF model uses AI concepts like deep learning,supervised learning ,Convolutional Neural Network (CNN), to classify videos to check whether video is fake or not based on a (threshold) number of the fake frames that are identified in the entire video Based on calculate three variables word per sentence, speech rate and frame rate.

Speech rate is used to describe how much words spoken per minute (wpm) in the video.

- **Conclusion**

They used CNN deep learning mechanism to classify deepfake videos using MoviePy for editing and cutting videos, to cut the video based on certain words in which the mouth appears open and the teeth are visible. They eliminated all images that containing closed mouth for easy identification.

The DFT-MF model was built to detect deepfake videos by using mouth as the biological signal.The datasets that were used contain both fake and real videos for the Celeb-DF and Deepfake-TIMIT then deep learning (CNN) was applied to classify fake videos depending on the features that will be taken from the mouth as a biological signal.

CHAPTER 3

PROBLEM STATEMENT

Deep Fakes are AI generated fake videos that look real but are actually fake.The word deepfake combines the terms “deep learning” and “fake,” and is a form of artificial intelligence.Deepfakes are falsified videos made by means of deep learning. The danger of that is the technology can be used to make people believe something is real when it is not.It started out as fun but like any technology, it is being misused. In the beginning, these videos could be identified by human eyes. But due to the development of machine learning,it became easier to create deep fake videos. These videos can affect a person’s integrity. It became necessary to identify these fake videos from real videos.

CHAPTER 4

PROJECT MANAGEMENT

4.1 Introduction

Project management is the discipline of planning, organizing, securing, managing, leading, and controlling resources to achieve specific goals. A project is a temporary endeavor with a defined beginning and end (usually time-constrained, and often constrained by funding or deliverables), undertaken to meet unique goals and objectives, typically to bring about beneficial change or added value. The temporary nature of projects stands in contrast with business as usual (or operations), which are repetitive, permanent, or semi-permanent functional activities to produce products or services. In practice, the management of these two systems is often quite different, and as such requires the development of distinct technical skills and management strategies.

In our project we are following the typical development phases of an engineering project

1. Initiation

2. Planning and Design

3. Execution and Construction

4. Monitoring and Controlling Systems

5. Completion

4.1.1 Initiation

The initiating processes determine the nature and scope of the project. The initiating stage should include a plan that encompasses the following areas :

1. Analysing the business needs/requirements in measurable goals
2. Reviewing of the current operations
3. Financial analysis of the costs and benefits including a budget
4. Stakeholder analysis, including users, and support personal for the project
5. Project charter including costs, tasks, deliverables, and schedule

4.1.2 Planing and design

After the initiation stage, the project is planned to an appropriate level of detail (see example of a flow-chart). The main purpose is to plan time, cost and resources adequately to estimate the work needed and to effectively manage risk during project execution. As with the initiation process, a failure to adequately plan greatly reduces the project's chances of successfully accomplishing its goals.

- Determining how to plan
- Developing the scope statement
- Selecting the planning team
- Identifying deliverables and creating the work breakdown structure
- Identifying the activities needed to complete those deliverables
- Developing the schedule

- Risk planning

4.1.3 Execution

Executing consists of the processes used to complete the work defined in the project plan to accomplish the project's requirements. The execution process involves coordinating people and resources, as well as integrating and performing the activities of the project in accordance with the project management plan. The deliverables are produced as outputs from the processes performed as defined in the project management plan and other frameworks that might be applicable to the type of project at hand.

4.1.4 Monitoring & controlling

Monitoring and controlling consists of those processes performed to observe project execution so that potential problems can be identified in a timely manner and corrective action can be taken, when necessary, to control the execution of the project. The key benefit is that project performance is observed and measured regularly to identify variances from the project management plan.

4.2 System Development Life Cycle

The Systems development life cycle (SDLC), or Software development process in systems engineering, information systems, and software engineering, is a process of creating or altering information systems, and the models and methodologies that people use to develop these systems. In software engineering, the SDLC concept underpins many kinds of software development methodologies. These methodologies form the framework for planning and controlling the creation of an information system.

The SDLC phases serve as a programmatic guide to project activity and provide a flexible but consistent way to conduct projects to a depth matching the scope of the project. Each of the SDLC phase objectives is described in this section with key deliverables, a description of recommended tasks, and a summary of related control objectives for effective management. The project manager must establish and monitor control objectives during each SDLC phase while executing projects. Control objectives help to provide a clear statement of the desired result or purpose and should be used throughout the entire SDLC process.

4.2.1 Spiral Model

We have used the Spiral model in our project. The Spiral model incorporates the best characteristics of both- waterfall and prototyping model. In addition, the Spiral model also contains a new component called Risk Analysis, which is not there in the waterfall and prototype model. In the Spiral model, the basic structure of the software product is developed first. After the basic structure is developed, new features such as user interface and data administration are added to the existing software product. This functionality of the Spiral model is similar to a spiral where the circles of the spiral increase in diameter. Each circle represents a more complete version of the software product. The spiral is a risk-reduction oriented model that breaks a software project up into main projects, each addressing one or major risks. After major risks have been addressed the spiral model terminates as a waterfall model. Spiral iteration involves six steps:

1. Determine objectives, alternatives and constraints.

2. Identify and resolve risks.

3. Evaluate alternatives.

4. Develop the deliverables for the iteration and verify that they are correct.

5. Plan the next iteration.

6. Commit to an approach for the next iteration.

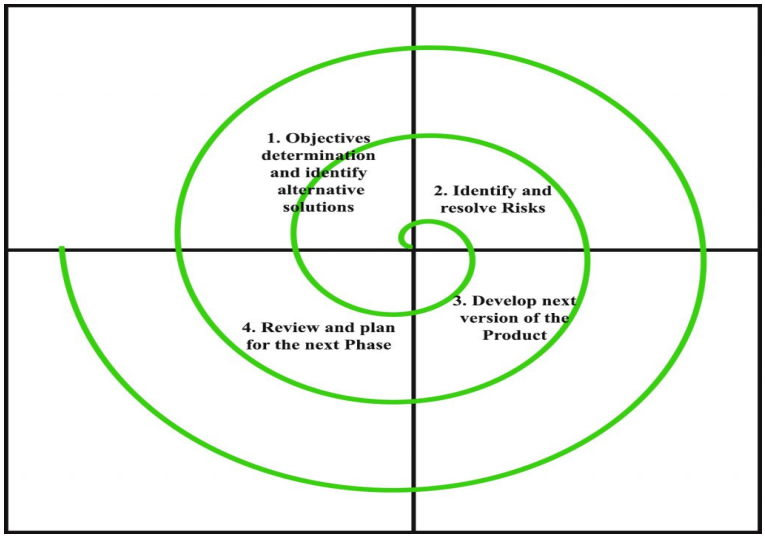


Figure 4.1: Spiral Model

CHAPTER 5

METHODOLOGY

5.1 System Requirements & Specifications

5.1.1 Windows 10

Windows 10 is a series of personal computer operating systems produced by Microsoft as part of its Windows NT family of operating systems. It is the successor to Windows 8.1 and was released to manufacturing on July 15, 2015, and to retail on July 29, 2015. Windows 10 receives new builds on an ongoing basis, which are available at no additional cost to users. Mainstream builds of Windows 10 are labeled version YYMM with YY representing the year and MM representing the month of release. For example, the latest mainstream build of Windows 10 is Version 1809. There are additional test builds of Windows 10 available to Windows Insiders. Devices in enterprise environments can receive these updates at a slower pace, or use long-term support milestones that only receive critical updates, such as security patches, over their ten-year lifespan of extended support.

5.1.2 Python 3.6.2

Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code.

Python runs on Windows, Linux/Unix, Mac OS X, OS/2, Amiga, Palm Handhelds, and Nokia mobile phones. Python has also been ported to the Java and .NET virtual machines. Python is distributed under an OSI-approved open source license that makes it free to use, even for commercial products.

5.1.3 TensorFlow

Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high performance C++. TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

5.1.4 Jupyter Environment

JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones.

5.2 Proposed System

Modules

5.2.1 Data Acquisition Module

A large number of real and fake images are collected. In-order to make this model as accurate as possible we'll split the real images and fake images. So that both training and testing data set will have both images. Thisensures that the dataset is diverse and that the model can be used in various settings and can classify real and fake videos more accurately.

In order to evaluate the effect of different methodological choices, we prepared three standard data sets for training and prediction:
D1: The standard training gives the model an insight into different types of scenarios where fake images can be detected and identified..
D2:The prediction set or validation set includes the rest 30 % of the training set.
D3: The test set will be the overall leftover 20 % of the dataset, which we set aside for the final evaluation.
The forecasts will be evaluated on future data (D4 - test set).

5.2.2 Image Enhancement Module

Data may contain noise. These noise reduces the quality of images and makes their interpretation and analysis really difficult, so it is necessary to filter images to remove the noise in order to prevent as much as possible important features of the data.It is important to remove this noise before these images are used for deepfake detection because they may cause variations in our model and training. So In this module we will remove that unwanted noise.

In this module videos are broken down into frames.Frames are resized into required size for processing. Frames are then converted into squared image. All the images will be in BGR format by default, they will be converted into RGB format using cvt.Color function. Face extraction is done on each frames using blazeface face detection model. If same face appear more than one time, it is avoided. Overlapping faces are discarded. Faces which acquire more than a particular percentage of the frame are large crps. These large crops are discarded because these may be false positives. Blazeface is a fast, light-weight face detector. Besides the bounding box, blazeface also predicts 6 keypoints for facial landmarks (2eyes, 2ears, nose,

mouth)

5.2.3 Deepfake detection Module

In this module we read videos as input and we predict the probability of video being fake. If the prediction score is greater than 0.5, the video is classified as fake and if prediction score is less than 0.5, the video is classified as real.

We are using a pretrained Xception neural network model to classify the video as real or fake. Xception model is a neural network model that is trained over half a million images. During training, input data are loaded using data loader function. Then we read metadata which define which videos are for training and which are for validation. Videos are converted frames. From each frames face region is extracted. Each face is then transformed using albumentation into various forms (rotation, flip, brightness, contrast, blur, compression, normalisation, etc). Then we extract facial landmarks/features(2x eye, 2x ear, nose, mouth). Then we train the model for a particular number of epochs.We have achieved a best accuracy of 94.9% during 10th epoch with a lowest loss of 15%.

Then we pass the test videos. These videos are converted to frames using frame extractor. If frames cannot be retrieved from a video, it is discarded and an error message is printed. The captured frames are stored as a numpyarray. Then the captured frames are passed into face extractor. original frames are copied to other variables for future use.Then the frames are converted into tiles. Tiles are resized into required size. Then the tiles are stored as a batch. Overlapping tiles are discarded. Face detection is applied on each frame. Each extracted face will be in BGR format by deafult. We have to convert it into RGB format. So we use Cvt.Color function to conver image from BGR format to RGB format. Then we add padding to the extracted face. From the face we extracted the facial features/landmarks (2x eye, 2x ear, nose, mouth) and pass these features to the Xception model. Features are compared with the model using sigmoid function and we will get a value between 0 and 1 as result. If the prediction value is grater than 0.5, we classify the videos as fake and if it is less than 0.5 we classify the video as real.

5.3 Data Flow Diagrams

5.3.1 Data Flow Diagram- Level 0

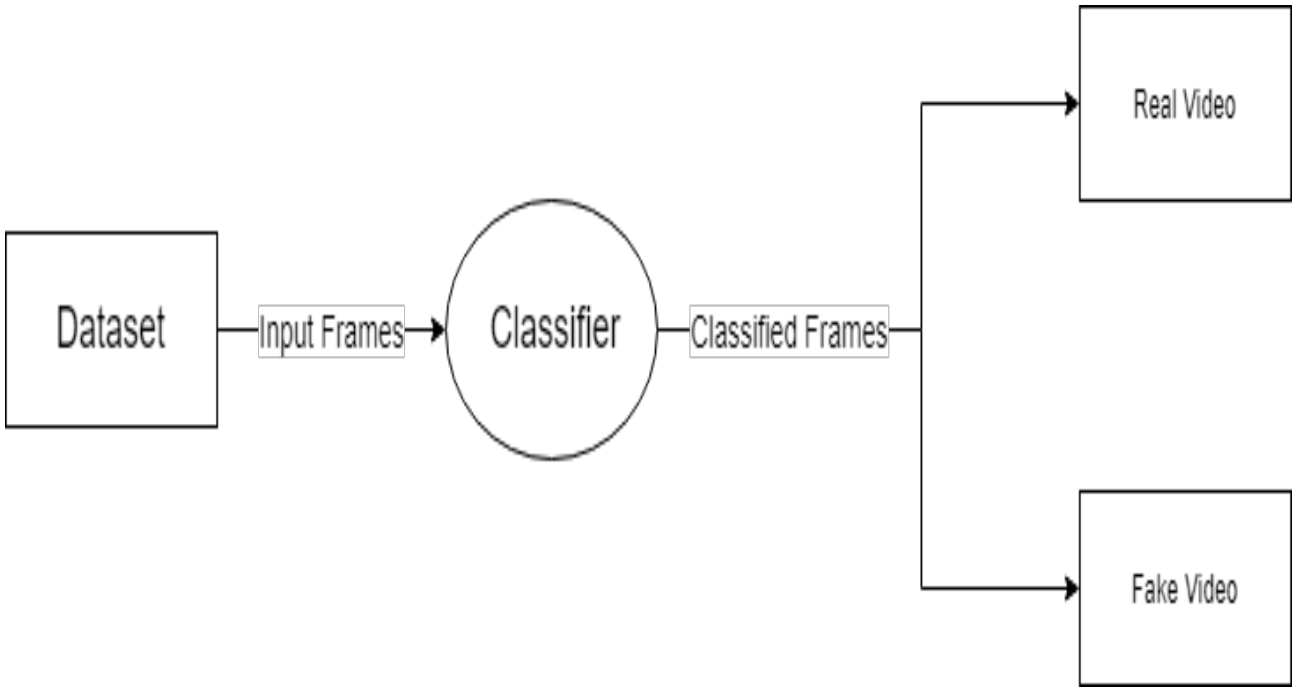


Figure 5.1: DFD level 0

5.3.2 Data Flow Diagram- Level 1

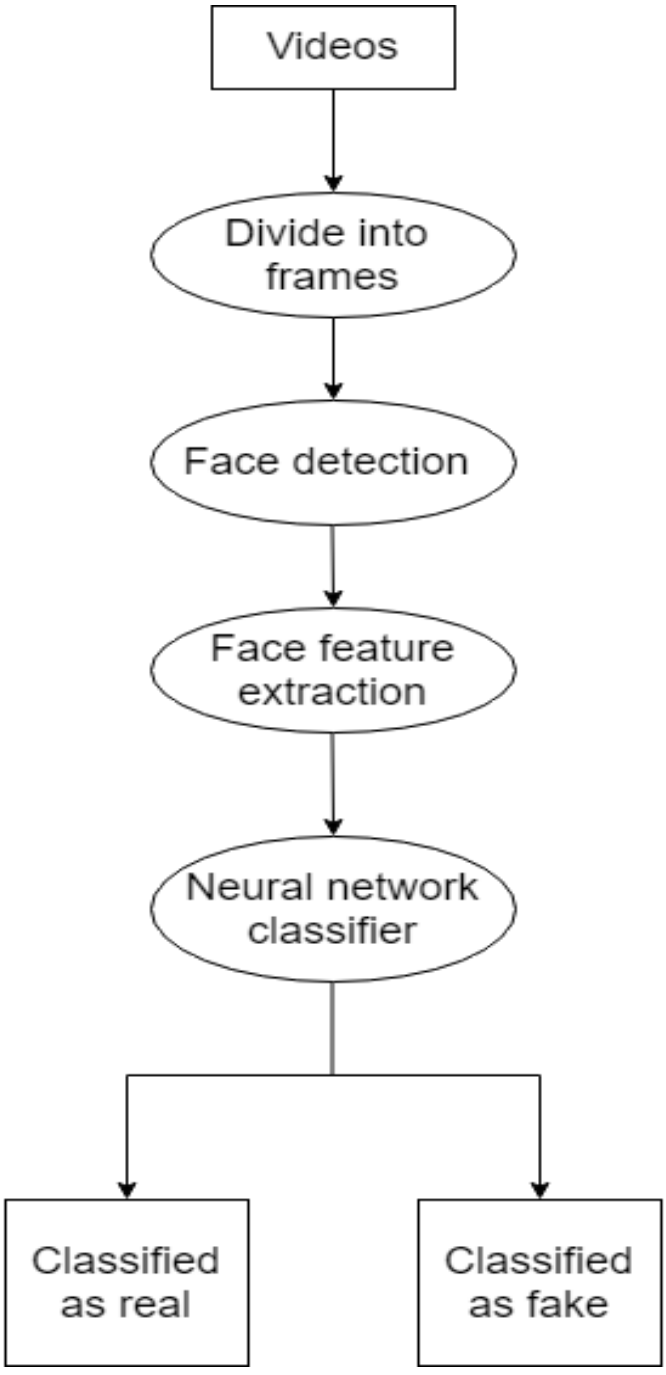


Figure 5.2: DFD level 1

5.3.3 Data Flow Diagram- Level 2

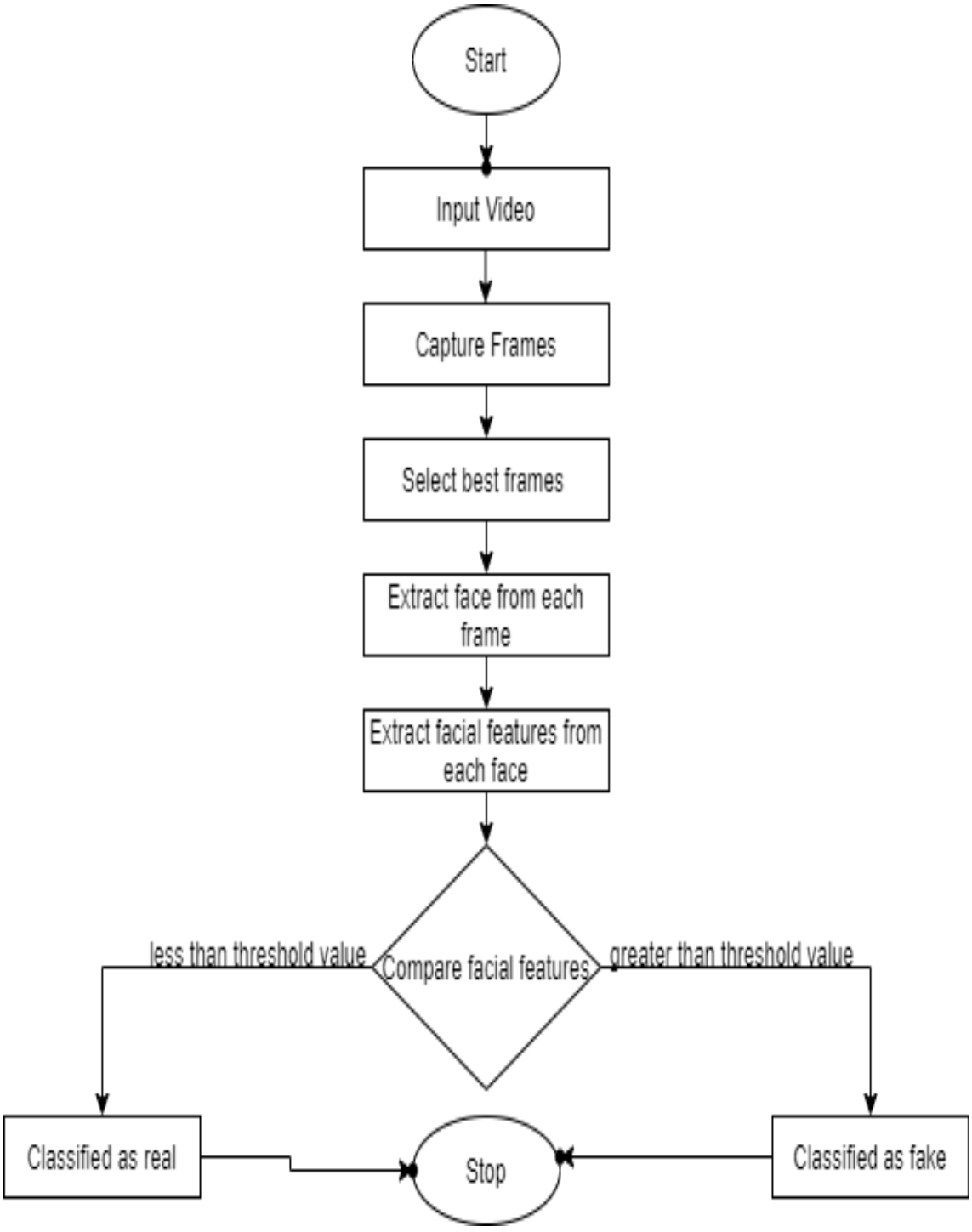


Figure 5.3: DFD level 2

5.4 Flow Chart

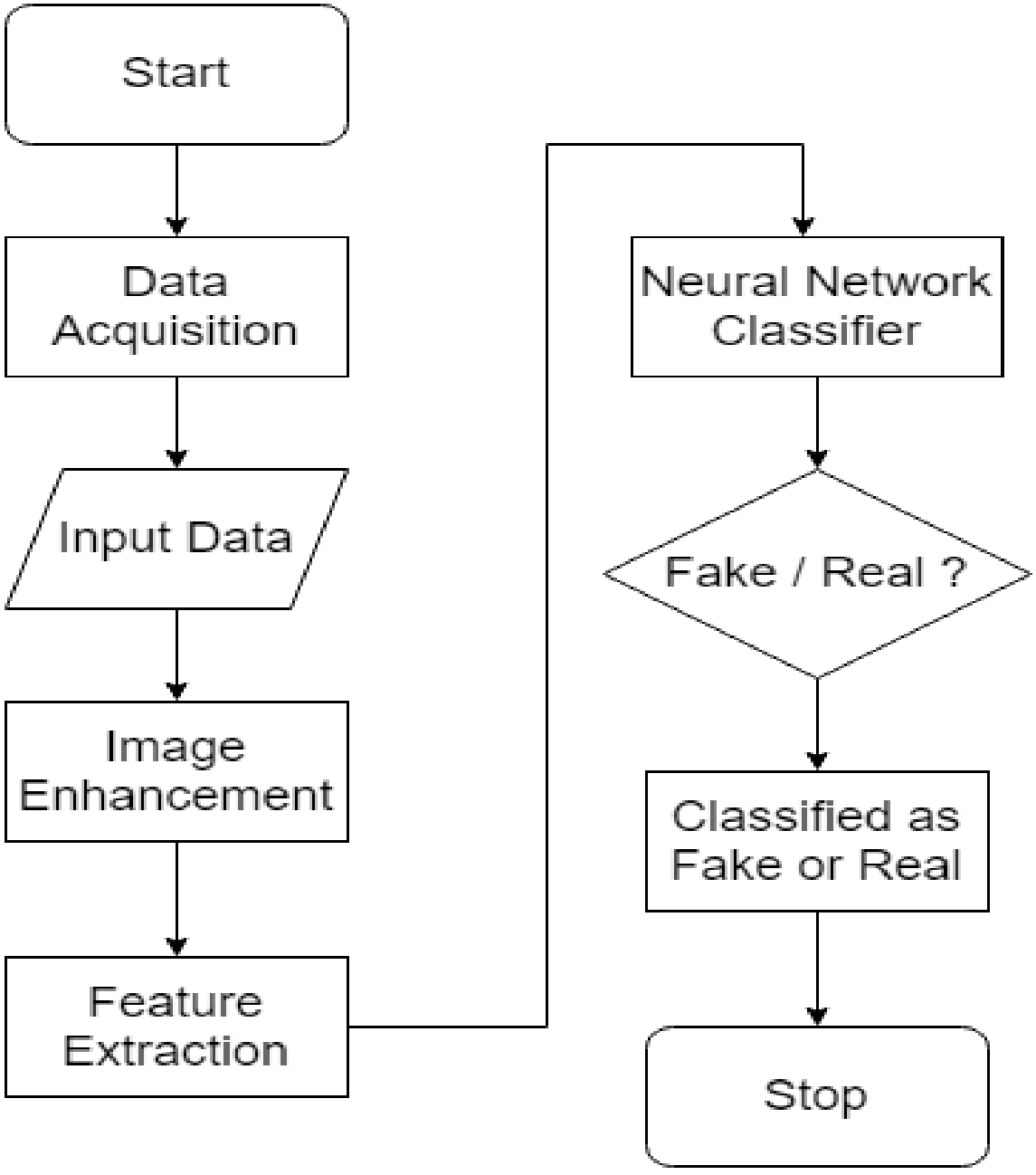


Figure 5.4: Flow Chart

5.5 Architecture

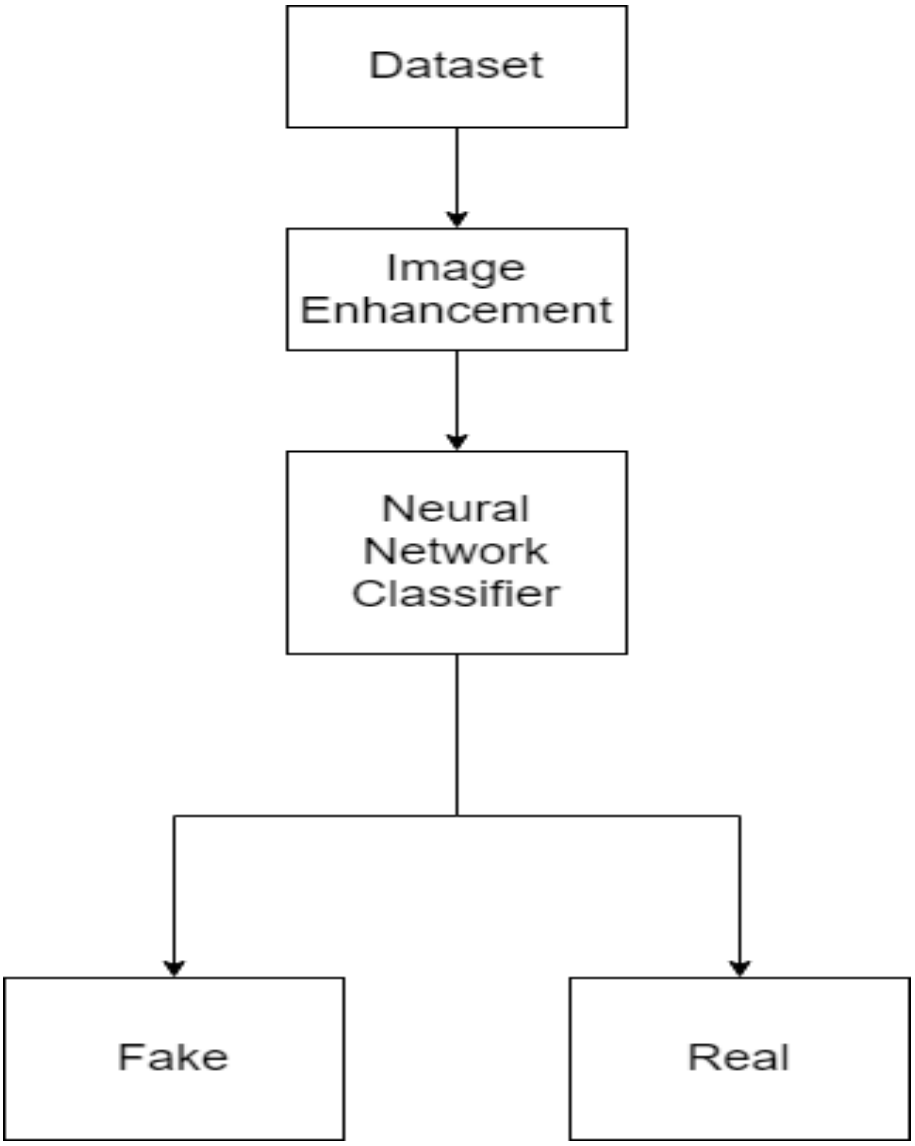


Figure 5.5: Architecture

CHAPTER 6

RESULTS

6.1 Training Result

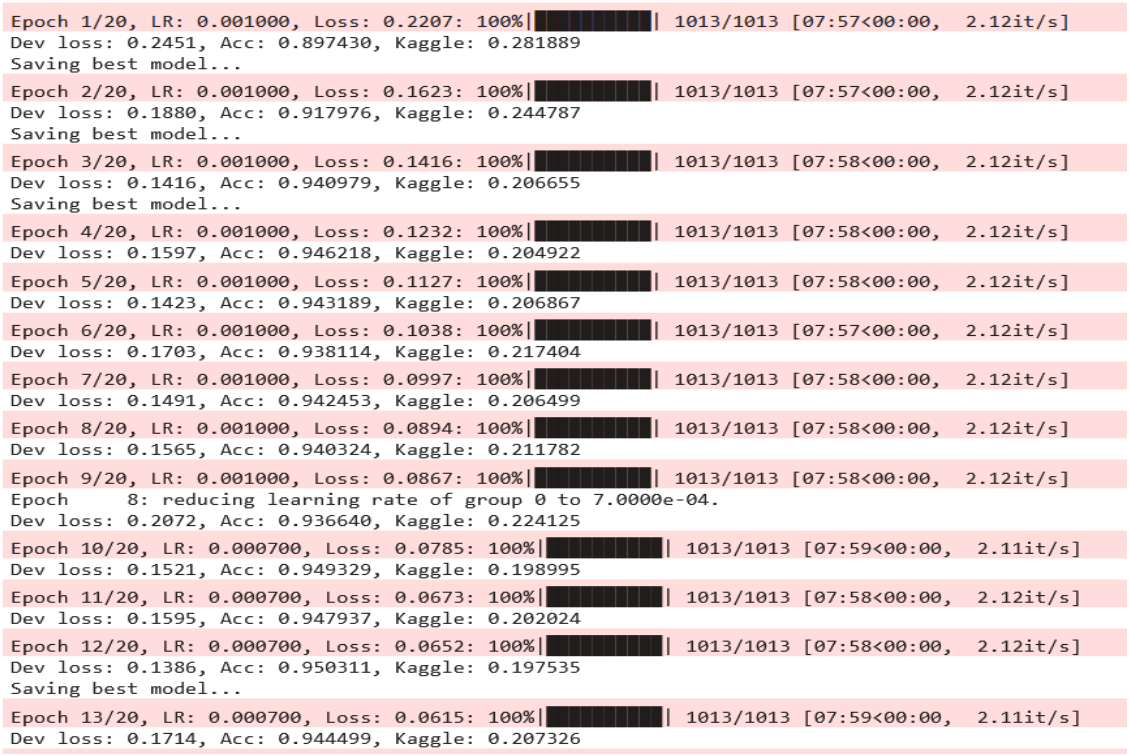
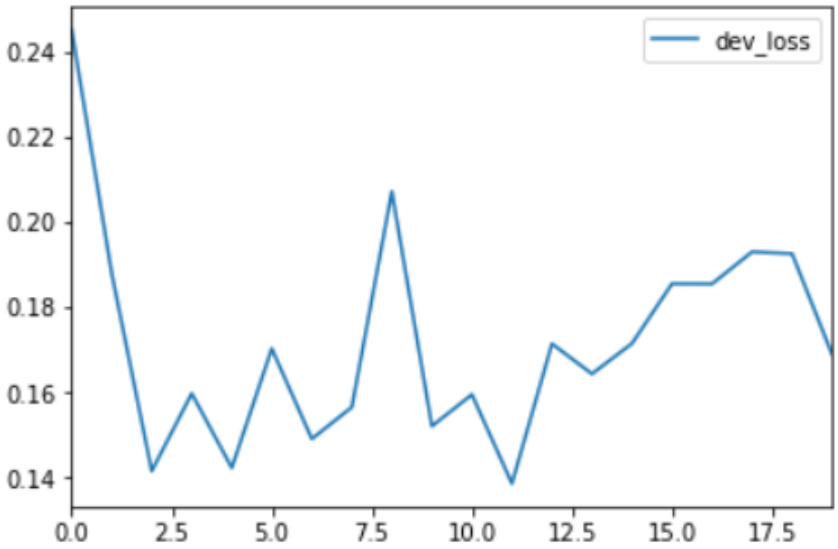


Figure 6.1: epoch results

6.2 Validation Result



X-axis –> epoch and Y-axis –> loss

Figure 6.2: Validation Loss graph.

6.3 Output

```
submission_df_xception.head()
```

	filename	label	result
0	aassnaulhq.mp4	0.971394	FAKE
1	aayfryxljh.mp4	0.009612	REAL
2	acazlolrpz.mp4	0.871641	FAKE
3	adohdulfwb.mp4	0.005082	REAL
4	ahjnxtiamx.mp4	0.748403	FAKE

Figure 6.3: output

CHAPTER 7

CONCLUSION AND FUTURE WORKS

The proposed method will identify whether a video has been manipulated or not. It will identify whether an image is fake or real. It will classify data as fake and real. By this we can prevent misusing of present technologies. We can make internet a lot safer. We can protect people's integrity.

In future we can try to improve this method so that it could work with images of various resolutions. We can also try to improve the accuracy rate.

REFERENCES

[1] S Agatonovic-Kustrin and R Beresford. Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, 22(5):717–727, 2000.

[2] Umur Aybars Ciftci, Ilke Demir, and Lijun Yin. How do the hearts of deep fakes beat? deep fake source detection via interpreting residuals with biological signals. *arXiv preprint arXiv:2008.11363*, 2020.

[3] Reagan L Galvez, Argel A Bandala, Elmer P Dadios, Ryan Rhay P Vicerra, and Jose Martin Z Maningo. Object detection using convolutional neural networks. In *TENCON 2018-2018 IEEE Region 10 Conference*, pages 2023–2027. IEEE, 2018.

[4] D. Güera and E. J. Delp. Deepfake video detection using recurrent neural networks. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, 2018.

[5] Mousa Tayseer Jafar, Mohammad Ababneh, Mohammad Al-Zoube, and Ammar Elhassan. Forensics and analysis of deepfake videos. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 053–058. IEEE, 2020.

[6] H. Khalid and S. S. Woo. Oc-fakedect: Classifying deepfakes using one-class variational autoencoder. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2794–2803, 2020.

[7] Pavel Korshunov and Sébastien Marcel. Deepfakes: a new threat to face recognition? assessment and detection. *arXiv preprint arXiv:1812.08685*, 2018.

[8] Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron Courville, and Yoshua Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*, 2015.

[9] Mohammed Akram Younus and Taha Mohammed Hasan. Effective and fast deepfake detection method based on haar wavelet transform. In *2020 International Conference on Computer Science and Software Engineering (CSASE)*, pages 186–190. IEEE, 2020.

Appendix A

Code Screenshot

```
# **Frame_Extractor**

import os
import cv2
import numpy as np

class VideoReader:

    def __init__(self, verbose=True, insets=(0, 0)):

        self.verbose = verbose
        self.insets = insets

    def read_frames(self, path, num_frames, jitter=0, seed=None):
      # Reads frames that are always evenly spaced throughout the video.
      #     jitter: if not 0, adds small random offsets to the frame indices;
      #         this is useful so we don't always land on even or odd frames
      #     seed: random seed for jittering; if you set this to a fixed value,
      #         you probably want to set it only on the first video

        assert num_frames > 0

        capture = cv2.VideoCapture(path)
        frame_count = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))
        if frame_count <= 0: return None

        frame_idxs = np.linspace(0, frame_count - 1, num_frames, endpoint=True, dtype=np.int)
        if jitter > 0:
            np.random.seed(seed)
            jitter_offsets = np.random.randint(-jitter, jitter, len(frame_idxs))
            frame_idxs = np.clip(frame_idxs + jitter_offsets, 0, frame_count - 1)
```

```
        result = self._read_frames_at_indices(path, capture, frame_idxs)
        capture.release()
        return result

    def read_random_frames(self, path, num_frames, seed=None):
       #Picks the frames at random.

        assert num_frames > 0
        np.random.seed(seed)

        capture = cv2.VideoCapture(path)
        frame_count = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))
        if frame_count <= 0: return None

        frame_idxs = sorted(np.random.choice(np.arange(0, frame_count), num_frames))
        result = self._read_frames_at_indices(path, capture, frame_idxs)

        capture.release()
        return result

    def read_frames_at_indices(self, path, frame_idxs):
      #Reads frames from a video and puts them into a NumPy array
      #Returns:
            # - a NumPy array of shape (num_frames, height, width, 3)
            # - a list of the frame indices that were read
      assert len(frame_idxs) > 0
      capture = cv2.VideoCapture(path)
      result = self._read_frames_at_indices(path, capture, frame_idxs)
      capture.release()
      return result
```

```
    def _read_frames_at_indices(self, path, capture, frame_idxs):
      #capture frames at indices for processing

        try:
            frames = []
            idxs_read = []
            for frame_idx in range(frame_idxs[0], frame_idxs[-1] + 1):
                # Get the next frame, but don't decode if we're not using it.
                ret = capture.grab()
                if not ret:
                    if self.verbose:
                        print("Error grabbing frame %d from movie %s" % (frame_idx, path))
                    break

                # Need to look at this frame?
                current = len(idxs_read)
                if frame_idx == frame_idxs[current]:
                    ret, frame = capture.retrieve()
                    if not ret or frame is None:
                        if self.verbose:
                            print("Error retrieving frame %d from movie %s" % (frame_idx, path))
                        break

                    frame = self._postprocess_frame(frame)
                    frames.append(frame)
                    idxs_read.append(frame_idx)

            if len(frames) > 0:
                return np.stack(frames), idxs_read
            if self.verbose:
                print("No frames read from movie %s" % path)
            return None
```

```
        except:
            if self.verbose:
                print("Exception while reading movie %s" % path)
            return None

    def read_middle_frame(self, path):
      #Reads the frame from the middle of the video

        capture = cv2.VideoCapture(path)
        frame_count = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))
        result = self._read_frame_at_index(path, capture, frame_count // 2)
        capture.release()
        return result

    def read_frame_at_index(self, path, frame_idx):
      #this is the more efficient way to read single frame from a video

        capture = cv2.VideoCapture(path)
        result = self._read_frame_at_index(path, capture, frame_idx)
        capture.release()
        return result

    def _read_frame_at_index(self, path, capture, frame_idx):
        capture.set(cv2.CAP_PROP_POS_FRAMES, frame_idx)
        ret, frame = capture.read()
        if not ret or frame is None:
            if self.verbose:
                print("Error retrieving frame %d from movie %s" % (frame_idx, path))
            return None
        else:
            frame = self._postprocess_frame(frame)
            return np.expand_dims(frame, axis=0), [frame_idx]
```

```
        def _postprocess_frame(self, frame):
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            if self.insets[0] > 0:
                W = frame.shape[1]
                p = int(W * self.insets[0])
                frame = frame[:, p:-p, :]

            if self.insets[1] > 0:
                H = frame.shape[1]
                q = int(H * self.insets[1])
                frame = frame[q:-q, :, :]

            return frame
```

```
# **FaceExtractor**


import os
import cv2
import numpy as np
import torch


class FaceExtractor:

    def __init__(self, video_read_fn, facedet):
      # video_read_fn: a function that takes in a path to a video file
      #           and returns a tuple consisting of a NumPy array with shape
      #           (num_frames, H, W, 3) and a list of frame indices, or None
      #           in case of an error
      #       facedet: the face detector object

        self.video_read_fn = video_read_fn
        self.facedet = facedet

    def process_videos(self, input_dir, filenames, video_idxs):
      # For the specified selection of videos, grabs one or more frames
      #   from each video, runs the face detector, and tries to find the faces
      #   in each frame.

        target_size = self.facedet.input_size
        videos_read = []
        frames_read = []
        frames = []
        tiles = []
```

```
        resize_info = []

        for video_idx in video_idxs:
            # Read the full-size frames from this video.
            filename = filenames[video_idx]
            video_path = os.path.join(input_dir, filename)
            result = self.video_read_fn(video_path)


            if result is None: continue

            videos_read.append(video_idx)

           # Keep track of the original frames (need them later).
            my_frames, my_idxs = result
            frames.append(my_frames)
            frames_read.append(my_idxs)
           # Split the frames into several tiles. Resize the tiles to 128x128
            my_tiles, my_resize_info = self._tile_frames(my_frames, target_size)
            tiles.append(my_tiles)
            resize_info.append(my_resize_info)
            # Put all the tiles for all the frames from all the videos into
             # a single batch.

        batch = np.concatenate(tiles)
        # Run the face detector. The result is a list of PyTorch tensors,
        # one for each image in the batch.
        all_detections = self.facedet.predict_on_batch(batch, apply_nms=False)

        result = []
        offs = 0
```

```
        for v in range(len(tiles)):
            num_tiles = tiles[v].shape[0]
            detections = all_detections[offs:offs + num_tiles]
            offs += num_tiles
            detections = self._resize_detections(detections, target_size, resize_info[v])
            # Because we have several tiles for each frame, combine the predictions
            # from these tiles. The result is a list of PyTorch tensors,
            num_frames = frames[v].shape[0]
            frame_size = (frames[v].shape[2], frames[v].shape[1])
            detections = self._untile_detections(num_frames, frame_size, detections)
            # The same face may have been detected in multiple tiles, so filter out
            # overlapping detections. This is done separately for each frame.
            detections = self.facedet.nms(detections)

            for i in range(len(detections)):
               # Crop the faces out of  frame
                faces = self._add_margin_to_detections(detections[i], frame_size, 0.2)
                faces = self._crop_faces(frames[v][i], faces)
                # Add additional information about the frame and detections
                scores = list(detections[i][:, 16].cpu().numpy())
                frame_dict = { "video_idx": videos_read[v],
                               "frame_idx": frames_read[v][i],
                               "frame_w": frame_size[0],
                               "frame_h": frame_size[1],
                               "faces": faces,
                               "scores": scores }
                result.append(frame_dict)

        return result
```

```
def _tile_frames(self, frames, target_size):
  # Splits each frame into several smaller, partially overlapping tiles
  #   and resizes each tile to target_size

    num_frames, H, W, _ = frames.shape
    split_size = min(H, W)
    x_step = (W - split_size) // 2
    y_step = (H - split_size) // 2
    num_v = 1
    num_h = 3 if W > H else 1

    splits = np.zeros((num_frames * num_v * num_h, target_size[1], target_size[0], 3), dtype=np.uint8)

    i = 0
    for f in range(num_frames):
        y = 0
        for v in range(num_v):
            x = 0
            for h in range(num_h):
                crop = frames[f, y:y+split_size, x:x+split_size, :]
                splits[i] = cv2.resize(crop, target_size, interpolation=cv2.INTER_AREA)
                x += x_step
                i += 1
            y += y_step

    resize_info = [split_size / target_size[0], split_size / target_size[1], 0, 0]
    return splits, resize_info
```

```
    def _resize_detections(self, detections, target_size, resize_info):
        projected = []
        target_w, target_h = target_size
        scale_w, scale_h, offset_x, offset_y = resize_info

        for i in range(len(detections)):
            detection = detections[i].clone()
            for k in range(2):
                detection[:, k*2    ] = (detection[:, k*2    ] * target_h - offset_y) * scale_h
                detection[:, k*2 + 1] = (detection[:, k*2 + 1] * target_w - offset_x) * scale_w

            # keypoints are x,y
            for k in range(2, 8):
                detection[:, k*2    ] = (detection[:, k*2    ] * target_w - offset_x) * scale_w
                detection[:, k*2 + 1] = (detection[:, k*2 + 1] * target_h - offset_y) * scale_h

            projected.append(detection)

        return projected
```

```
def _untile_detections(self, num_frames, frame_size, detections):
  # With N tiles per frame, there also are N times as many detections.
  #   This function groups together the detections for a given frame; it is
  #   the complement to tile_frames().

    combined_detections = []
    W, H = frame_size
    split_size = min(H, W)
    x_step = (W - split_size) // 2
    y_step = (H - split_size) // 2
    num_v = 1
    num_h = 3 if W > H else 1

    i = 0
    for f in range(num_frames):
        detections_for_frame = []
        y = 0
        for v in range(num_v):
            x = 0
            for h in range(num_h):
                detection = detections[i].clone()
                if detection.shape[0] > 0:
                    for k in range(2):
                        detection[:, k*2    ] += y
                        detection[:, k*2 + 1] += x
                    for k in range(2, 8):
                        detection[:, k*2    ] += x
                        detection[:, k*2 + 1] += y

                detections_for_frame.append(detection)
                x += x_step
                i += 1
            y += y_step
        combined_detections.append(torch.cat(detections_for_frame))

    return combined_detections

def _add_margin_to_detections(self, detections, frame_size, margin=0.2):
  #Expands the face bounding box
  #The face detections often do not include the forehead, which
  # is why we use twice the margin for ymin.
  # Returns a PyTorch tensor of shape (num_detections, 17)

    offset = torch.round(margin * (detections[:, 2] - detections[:, 0]))
    detections = detections.clone()
    detections[:, 0] = torch.clamp(detections[:, 0] - offset*2, min=0)            # ymin
    detections[:, 1] = torch.clamp(detections[:, 1] - offset, min=0)              # xmin
    detections[:, 2] = torch.clamp(detections[:, 2] + offset, max=frame_size[1])  # ymax
    detections[:, 3] = torch.clamp(detections[:, 3] + offset, max=frame_size[0])  # xmax
    return detections

def _crop_faces(self, frame, detections):
  # Copies the face region(s) from the given frame into a set
    # of new NumPy arrays

    faces = []
    for i in range(len(detections)):
        ymin, xmin, ymax, xmax = detections[i, :4].cpu().numpy().astype(np.int)
        face = frame[ymin:ymax, xmin:xmax, :]
        faces.append(face)
    return faces
```

```
    def remove_large_crops(self, crops, pct=0.1):
      # Removes faces from the results if they take up more than X%
        # of the video. Such a face is likely a false positive.

        for i in range(len(crops)):
            frame_data = crops[i]
            video_area = frame_data["frame_w"] * frame_data["frame_h"]
            faces = frame_data["faces"]
            scores = frame_data["scores"]
            new_faces = []
            new_scores = []
            for j in range(len(faces)):
                face = faces[j]
                face_H, face_W, _ = face.shape
                face_area = face_H * face_W
                if face_area / video_area < 0.1:
                    new_faces.append(face)
                    new_scores.append(scores[j])
            frame_data["faces"] = new_faces
            frame_data["scores"] = new_scores

    def keep_only_best_face(self, crops):

        for i in range(len(crops)):
            frame_data = crops[i]
            if len(frame_data["faces"]) > 0:
                frame_data["faces"] = frame_data["faces"][:1]
                frame_data["scores"] = frame_data["scores"][:1]
```

```
import os, sys, time
import cv2
import numpy as np
import pandas as pd

import torch
import torch.nn as nn
import torch.nn.functional as F

%matplotlib inline
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
```

```
[ ] !pip install /content/drive/MyDrive/deepfake/xception/pytorchcv-0.0.55-py2.py3-none-any.whl
```

```
[ ] test_dir = "/content/drive/MyDrive/deepfake/dataset/test_videos"
    test_videos = sorted([x for x in os.listdir(test_dir) if x[-4:] == ".mp4"])
    frame_h = 5
    frame_l = 5
    len(test_videos)
```

```
400
```

```
[ ] gpu = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    gpu
```

```
device(type='cuda', index=0)
```

```
[ ] import sys
    sys.path.insert(0, "/content/drive/MyDrive/deepfake/blazeface")
```

```
[ ] from blazeface import BlazeFace
    facedet = BlazeFace().to(gpu)
    facedet.load_weights("/content/drive/MyDrive/deepfake/blazeface/blazeface.pth")
    facedet.load_anchors("/content/drive/MyDrive/deepfake/blazeface/anchors.npy")
    _ = facedet.train(False)
```

```
[ ] frames_per_video = 64 # originally 4

    video_reader = VideoReader()
    video_read_fn = lambda x: video_reader.read_frames(x, num_frames=frames_per_video)
    face_extractor = FaceExtractor(video_read_fn, facedet)
```

```
[ ] input size = 150
```

```
[ ] from torchvision.transforms import Normalize

    mean = [0.485, 0.456, 0.406]
    std = [0.229, 0.224, 0.225]
    normalize_transform = Normalize(mean, std)
```

```
def resize_image(img, size, resample=cv2.INTER_AREA):
    h, w = img.shape[:2]
    if w > h:
        h = h * size // w
        w = size
    else:
        w = w * size // h
        h = size

    resized = cv2.resize(img, (w, h), interpolation=resample)
    return resized


def make_square_image(img):
    h, w = img.shape[:2]
    size = max(h, w)
    t = 0
    b = size - h
    l = 0
    r = size - w
    return cv2.copyMakeBorder(img, t, b, l, r, cv2.BORDER_CONSTANT, value=0)
```

```
from pytorchcv.model_provider import get_model
model = get_model("xception", pretrained=False)
model = nn.Sequential(*list(model.children())[:-1]) # Remove original output layer

class Pooling(nn.Module):
  def __init__(self):
    super(Pooling, self).__init__()

    self.p1 = nn.AdaptiveAvgPool2d((1,1))
    self.p2 = nn.AdaptiveMaxPool2d((1,1))

  def forward(self, x):
    x1 = self.p1(x)
    x2 = self.p2(x)
    return (x1+x2) * 0.5

model[0].final_block.pool = nn.Sequential(nn.AdaptiveAvgPool2d((1,1)))
class Head(torch.nn.Module):
  def __init__(self, in_f, out_f):
    super(Head, self).__init__()

    self.f = nn.Flatten()
    self.l = nn.Linear(in_f, 512)
    self.d = nn.Dropout(0.5)
    self.o = nn.Linear(512, out_f)
    self.b1 = nn.BatchNorm1d(in_f)
    self.b2 = nn.BatchNorm1d(512)
    self.r = nn.ReLU()
```

```
  def forward(self, x):
    x = self.f(x)
    x = self.b1(x)
    x = self.d(x)

    x = self.l(x)
    x = self.r(x)
    x = self.b2(x)
    x = self.d(x)

    out = self.o(x)
    return out

class FCN(torch.nn.Module):
  def __init__(self, base, in_f):
    super(FCN, self).__init__()
    self.base = base
    self.h1 = Head(in_f, 1)

  def forward(self, x):
    x = self.base(x)
    return self.h1(x)

net = []
model = FCN(model, 2048)
model = model.cuda()
model.load_state_dict(torch.load('/content/drive/MyDrive/deepfake/xception/model_50epochs_lr0001_patience5_factor01_batchsize32.pth')) # new, updated
net.append(model)
```

```
def predict_on_video(video_path, batch_size):
    try:
        # Find the faces for N frames in the video.
        faces = face_extractor.process_video(video_path)

        # Only look at one face per frame.
        face_extractor.keep_only_best_face(faces)

        if len(faces) > 0:
            # NOTE: When running on the CPU, the batch size must be fixed
            # or else memory usage will blow up. (Bug in PyTorch?)
            x = np.zeros((batch_size, input_size, input_size, 3), dtype=np.uint8)

            # If we found any faces, prepare them for the model.
            n = 0
            for frame_data in faces:
                for face in frame_data["faces"]:
                    # Resize to the model's required input size.
                    # We keep the aspect ratio intact and add zero
                    # padding if necessary.
                    resized_face = resize_image(face, input_size)
                    resized_face = make_square_image(resized_face)

                    if n < batch_size:
                        x[n] = resized_face
                        n += 1
                    else:
                        print("WARNING: have %d faces but batch size is %d" % (n, batch_size))
```

```
                    #n += 1

            if n > 0:
                x = torch.tensor(x, device=gpu).float()

                # Preprocess the images.
                x = x.permute((0, 3, 1, 2))

                for i in range(len(x)):
                    x[i] = normalize_transform(x[i] / 255.)
#                   x[i] = x[i] / 255.

                # Make a prediction, then take the average.
                with torch.no_grad():
                    y_pred = model(x)
                    y_pred = torch.sigmoid(y_pred.squeeze())
                    return y_pred[:n].mean().item()

    except Exception as e:
        print("Prediction error on video %s: %s" % (video_path, str(e)))

    return 0.5
```

```
[ ]
    def predict_on_video_set(videos, num_workers):
        def process_file(i):
            filename = videos[i]
            y_pred = predict_on_video(os.path.join(test_dir, filename), batch_size=frames_per_video)
            return y_pred

        with ThreadPoolExecutor(max_workers=num_workers) as ex:
            predictions = ex.map(process_file, range(len(videos)))

        return list(predictions)
```

```
[ ] model.eval()
    predictions = predict_on_video_set(test_videos, num_workers=4)
```

+ Code + Text

```
prediction_value = []
for value in predictions:
    if value > .5:
        prediction_value.append('FAKE')
    else:
        prediction_value.append('REAL')
```

```
[ ] submission_df_xception = pd.DataFrame({"filename": test_videos, "label": predictions,"result":prediction_value})
    submission_df_xception.to_csv("/content/drive/MyDrive/deepfake/output/result.csv", index=False)
```

```
[ ] submission_df_xception.head()
```

```
submission_df_xception.head()
```

	filename	label	result
0	aassnaulhq.mp4	0.971394	FAKE
1	aayfryxljh.mp4	0.009612	REAL
2	acazlolrpz.mp4	0.871641	FAKE
3	adohdulfwb.mp4	0.005082	REAL
4	ahjnxtiamx.mp4	0.748403	FAKE