Cairo University
Faculty of Engineering
Department of Computer Engineering

Revelio

We reveal secrets.

A Graduation Project Report Submitted to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree of

Bachelor of Science in Computer Engineering

Presented by

Ahmed Ayman Ahmed
Ammar Mohamed Sobhi
Mohamed Akram Abdelfattah
Omar Ahmed Mohamed

Supervised by

Prof. Hoda Baraka

11 Jul. 2023

Abstract

The Spread of fake media generated by advanced AI techniques in recent years has been a serious problem in our societies. The tremendous progress of deep learning techniques made it easy to generate quite realistic fake media.

Our main objective is to detect and prevent the spread of fake videos that can be used in news, adult content, financial fraud, blackmailing, politics, and more.
We use various Image Processing and Machine Learning techniques to determine whether the person in the video is authentic or manipulated.

Our system analyzes the video through four main techniques which tackle different abnormalities found in manipulated videos which are: Lips movements analysis, Dynamic Texture analysis, Frequency domain analysis, and finally Self Blending images technique.

Our system manages to detect fake videos with an accuracy at least 95%, with providing different analysis insights including the type of manipulation that may have been used.

We provide an easy-to-use website for users to submit their videos for analysis and API to be integrated with various platforms.

الملخص

انتشار الصور والمقاطع الزائفة في السنوات الأخيرة كان مشكلة كبيرة في مجتمعاتنا، حيث أدى تقدم تقنيات التعلم العميق إلى إنشاء محتوى زائف واقعي للغاية.
هدفنا الرئيسي هو الكشف عن ومنع انتشار الفيديوهات المزيفة التي يمكن استخدامها في الأخبار، والمقاطع المشينة، والاحتيال المالي، والابتزاز، والسياسة، وغيرها.
نستخدم في مشروعنا تقنيات متنوعة لمعالجة الصور والتعلم الآلي لتحديد ما إذا كان الشخص في المقطع حقيقيًا أم زائفًا.
يحلل نظامنا المقطع من خلال أربع تقنيات رئيسية تتعامل مع الاضطرابات المختلفة التي يمكن ملاحظتها في المقاطع الزائفة وهي: تحليل حركة الشفاه، وتحليل النسيج الديناميكي، وتحليل المجال الترددي وأخيرًا تقنية خلط الصور ذاتيا.
يتمكن نظامنا من كشف الفيديوهات المزيفة بدقة على الأقل 95%، مع توفير تحليل تفصيلي للمقطع يحتوي معلومات مختلفة منها توقع الطريقة التي تم تزييف بها المقطع.
نقدم أيضًا موقع سهل الاستخدام يمكن من خلاله رفع المقطع لتحليله وتوفير إمكانية لربط خدماتنا مع المنصات المختلفة.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Professor Hoda Baraka for her invaluable support, mentorship, and guidance throughout the duration of this project.

Also, we do not want to forget our families, friends for their invaluable support during our beloved journey through college, and our professors who taught us and pushed us toward progress, we have no words to express our feelings for all of them.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ANOVA	Analysis of Variance
AUC	Area Under the Curve
CDF	Deepfake Classifier
CF	Compression Factor
CF2	Face2Face Classifier
CFSS	Coarse-to-Fine Shape Searching
CFSW	FaceSwap Classifier
CNT	NeuralTextures Classifier
DF	Deepfake
DFT	Discrete Fourier Transform
F2F	Face2Face
FDDB	Face Detection Data Base
FF++	FaceForensics++ dataset
FSW	FaceSwap
HOG	Histogram of Oriented Gradients
LBP	Local Binary Pattern
LDP	Local Derivative Pattern
LDP^2	The second-order directional
LDP-TOP	Local Derivative Pattern on Three Orthogonal Planes
MS-TCN	Multi Scale Temporal Convolutional Network
NT	NeuralTextures
OR	Original
ORB	Oriented Fast and Rotated BRIEF
RBF	Radial Basis Function
ROI	Region of interest
SAM	Sharpness-Aware Minimization
SBI	Self-Blended Images
SIFT	Scale Invariant Feature Transform
TCN	Temporal Convolutional Network
VRAM	Video RAM

List of Symbols

$\rightarrow$	Direct Mode
$\leftarrow$	Inverse Mode
$\leftrightarrow$	Bidirectional Mode
α	Angular Directions {0°, 45°, 90°, 135°}
β	Similarity function sensitivity parameter in CFSS Algorithm
θ	Threshold of features in decision stump in Face Detection
ϕ	Radial angle between x-axis to spatial frequency radius
ϕ_{en}	Parameter for model complexity in Efficient Net
ω_k	Radius to the spatial frequency
B	Bottom-Half of The Face
D	Time Window
F	Full Face Information
I	Image Pixel Intensity
K	Number of regressors per stage in CFSS Algorithm
L	Number of stages in CFSS Algorithm
N	Total size of candidate shapes in CFSS Algorithm
N_s	Sample Size in CFSS Algorithm
p	Polarity of decision stump in Face Detection
P_r	Probability distribution in CFSS Algorithm
SSB	Sum of Squares Between groups
SSW	Sum of Squares Within group
T	Number of Iterations of AdaBoost Algorithm
T	Top-Half of The Face in LDP

Contacts

Team Members

Name	Email	Phone Number
Ahmed Ayman Ahmed	ahmedayman1420@email.com	+201147771061
Ammar Mohamed Sobhi	ammarmohamed13@gmail.com	+2 01116185647
Mohamed Akram Abdelfattah	mohamed1999akram@gmail.com	+2 01032972946
Omar Ahmed Mohamed	omar.ahmed314@hotmail.com	+2 01100086995

Supervisor

Name	Email	Number
Prof. Hoda Baraka	Hodabaraka60@gmail.com	+2 01005411083

This page is left intentionally empty.

Chapter 1: Introduction

The current era can be described as the era of AI generation in technology, the huge progress made in the last few years in Deep Learning, Generative Adversarial Networks, and Computational Power have opened a wide new world where you can generate literally any type of content, whether it is text, audio or visual.

The applications on this have taken over the trend, with ChatGPT[1] or Google Bard[2] you do not only get answers to any question you ask but you can also generate short stories or poems, while in other applications you can turn your imagination with just a few words into images or videos, and other applications where you can generate a song with the voice of your favorite singer.

Many useful applications can benefit significantly from the advancing technology specially in the fields of entertainment, however, the risks and cons of such technologies have been concerning, this can be fun until we reach a moment we cannot distinguish between the reality and the falsity.

One major concern was the ability to swap the identity of a person in a video which is famously known as “DeepFake”, the impacts of this have been huge as such videos can have harmful consequences by being used in fake news, politics, adult content, blackmailing, financial fraud and more, this led to huge academic interest in detecting manipulated or fake videos because it became indistinguishable with a human eye.

Our project focuses on the analysis of videos to detect possible face manipulation and verify the reality or the falsity of the video.

[1] https://chat.openai.com/
[2] https://bard.google.com/

1.1. Motivation and Justification

The harmful impacts of impersonating and using the identities of people in fake videos made the task of detecting it a very urgent need.

As discussed earlier these fake videos can be used in media and journalism to spread fake news that can have political consequences, blackmailing people by producing fake videos of them, generating fake adult content, financial fraud, falsifying evidence in trials and many more uses that can have very bad consequences.

Recently we have heard about these consequences in Egypt with suicide cases reported for blackmailing by fake videos[3], other examples include impersonating cryptocurrency company FTX founder for financial fraud[4] and spreading a video between Ukrainian soldiers for Zelenskyy calling to lay down their arms.

Current video manipulation tools produce high quality fake content that became very difficult to detect visually, this urged the researchers to focus on detecting the manipulated videos.

Tech giants such as Facebook, Amazon and Microsoft collaborated to generate a dataset and competition for Deepfake detection methods[5] and governmental institutions such as DARPA have created and funded a project called MediFor[6] that focuses on detecting fake content, all of that have encouraged research to work on this topic.

1.2. Project Objectives and Problem Definition

The main objective of the project is to create a reliable system that is resilient to various face manipulation methods and detect it with high accuracy and provide in-depth analysis of the abnormalities found in the video using our main techniques such as Frequency domain analysis, Dynamic Texture analysis, Lips movement analysis or Self-Blended Images.
Using different analysis techniques makes our system more reliable in detecting fake videos.

[3] https://shorturl.at/dCKL8
[4] https://shorturl.at/nvGPV
[5] https://ai.facebook.com/datasets/dfdc/
[6] https://www.darpa.mil/program/media-forensics

1.3. Project Outcomes

A reliable system that is resilient to various face manipulation methods and detect it with high accuracy and produce the analysis result, provided in a user-friendly web application where the user can upload a video for analysis, and an easy-to-use API to be integrated with various platforms.

1.4. Document Organization

In Chapter 2 we show our market analysis and target customers, in Chapter 3 we do a literature survey about the face manipulation detection techniques and give some background information such as types of face manipulations used. In Chapter 4 we discuss the system architecture and pipeline and provide details for each module and sub-module, we have six main modules which are Face Detection, Facial Landmarks Detection, Lips Movement Analysis, Frequency domain analysis, Dynamic Texture Analysis, and Self-Blended Images.
In Chapter 5 we show our system testing and verification and finally in Chapter 6 we demonstrate our future work and conclusions.

Chapter 2: Market Feasibility Study

Face manipulation detection methods are popular in research with high need to solve the problems we discussed before. However, the availability of products is still too low, we found only two competitors that have the same scope as our project, which we will discuss briefly in this chapter.

2.1. Targeted Customers

Our intended solution used to prevent the fake videos from spreading out, and make anyone check the video validity, so our product targets mainly social media platforms, media agencies and governmental institutes the system can be used before them as a service at when the video being uploaded then make a check and the decision returns to the company whether to delete it or flag it as a fake source and so on.

2.2. Market Survey

2.2.1. Deepware

Deepware[7] first recognized the danger while their parent company Zemana[8] researched methods to develop an AI-based antivirus engine. Later, in mid-2018, they started their research on deepfake detection as the deepware AI team.
They expect destructive use of deep fakes, particularly as phishing attacks, to materialize very soon.
They have open-sourced their current deepware scanner, along with the research topics as they suggest that the communities must collaborate to stop deepfake videos.

[7] https://deepware.ai/
[8] https://zemana.com/

2.2.2. DuckDuckGoose DeepDetector 2021

DuckDuckGoose[9] founded in 2020 aiming to enhance the market of deepfake detection systems, they are developing insightful deep fake detection software. Their software is not only able to classify a certain image or video as real or fake, but also provides insight into why that content has been classified as manipulated.
They target various customers that intend to protect themselves from deepfake including journalism or video-conferencing tools.
DuckDuckGoose provide various services with pricing based on the number of videos to be analyzed and the type of analysis whether it is just binary or detailed.

2.3. Business Case and Financial Analysis

We expect that the need for deepfake detection tools will grow exponentially in the upcoming years along with the huge advance of face manipulation techniques.
Our current competitors made their pricing confidential, and negotiable based on the client.
We plan to use a pay-as-you-go pricing plans that depends just on the amount of usage and the type of analysis needed, this gives us an advantage over competitors due to flexibility, minimized risk for clients as they do not have to invest for long-term, and cost-efficiency as clients will have to pay only for their actual usage.

We plan to use a free tier as well to attract clients.

Plan	**Free**	**Pay-as-you-go**
Usage	100 minutes/month	Unlimited
Price	Free	$0.5 per 1 Minute
Methods	Only one analysis method	All four analysis methods
Output	Binary (Real or Fake) probability	All analysis insights in-depth

Table 2.1: Pricing plans of our services.

[9] https://www.duckduckgoose.ai/

Over time number of users is expected to increase by marketing and by simply using the service and earning their trust. Many journalism agencies will be interested in our service, even in Egypt we will target some fact checker media agencies like who might be interested in our services.

We also expect to gain more investments through time with the growth of our brand.

	Aug-23	Sep-23	Oct-23	Nov-23	Dec-23	Jan-24	Feb-24	Mar-24	Apr-24	May-24	Jun-24	Jul-24
Starting Balance	$0	-$8,600	-$17,000	-$8,100	-$3,200	$11,700	$31,500	$71,300	$111,100	$150,900	$180,700	$210,500
Investments	$5,000	$0	$10,000	$10,000	$0	$0	$0	$0	$0	$0	$0	$0
Service Usage	$0	$200	$5,000	$5,000	$20,000	$30,000	$50,000	$50,000	$50,000	$50,000	$50,000	$50,000
Sponsors	$1,000	$2,000	$5,000	$5,000	$5,000	$10,000	$10,000	$10,000	$10,000	$10,000	$10,000	$10,000
Total inflows	$6,000	$2,200	$20,000	$20,000	$25,000	$40,000	$60,000	$60,000	$60,000	$60,000	$60,000	$60,000
Models Training	$2,000	$0	$0	$0	$0	$5,000	$0	$0	$0	$0	$0	$0
Salaries	$10,000	$10,000	$10,000	$10,000	$10,000	$10,000	$20,000	$20,000	$20,000	$30,000	$30,000	$30,000
Servers Hosting	$100	$100	$100	$100	$100	$200	$200	$200	$200	$200	$200	$200
Infrastructure	$2,000	$0	$0	$0	$0	$5,000	$0	$0	$0	$0	$0	$0
Marketing	$500	$500	$1,000	$5,000	$0	$0	$0	$0	$0	$0	$0	$0
Total outflows	$14,600	$10,600	$11,100	$15,100	$10,100	$20,200	$20,200	$20,200	$20,200	$30,200	$30,200	$30,200
Net cash flow	-$8,600	-$8,400	$8,900	$4,900	$14,900	$19,800	$39,800	$39,800	$39,800	$29,800	$29,800	$29,800
Closing balance	-$8,600	-$17,000	-$8,100	-$3,200	$11,700	$31,500	$71,300	$111,100	$150,900	$180,700	$210,500	$240,300

Table 2.2: Cash flow analysis of first 12 months of our project.

Chapter 3: Literature Survey

In this chapter we will discuss the main techniques found in the research for face manipulation detection problem, however we will give some background information on the existing methods for face manipulation and discuss the main dataset we used for our project which is FaceForensics++ [1].

3.1. Background on Face Manipulation techniques

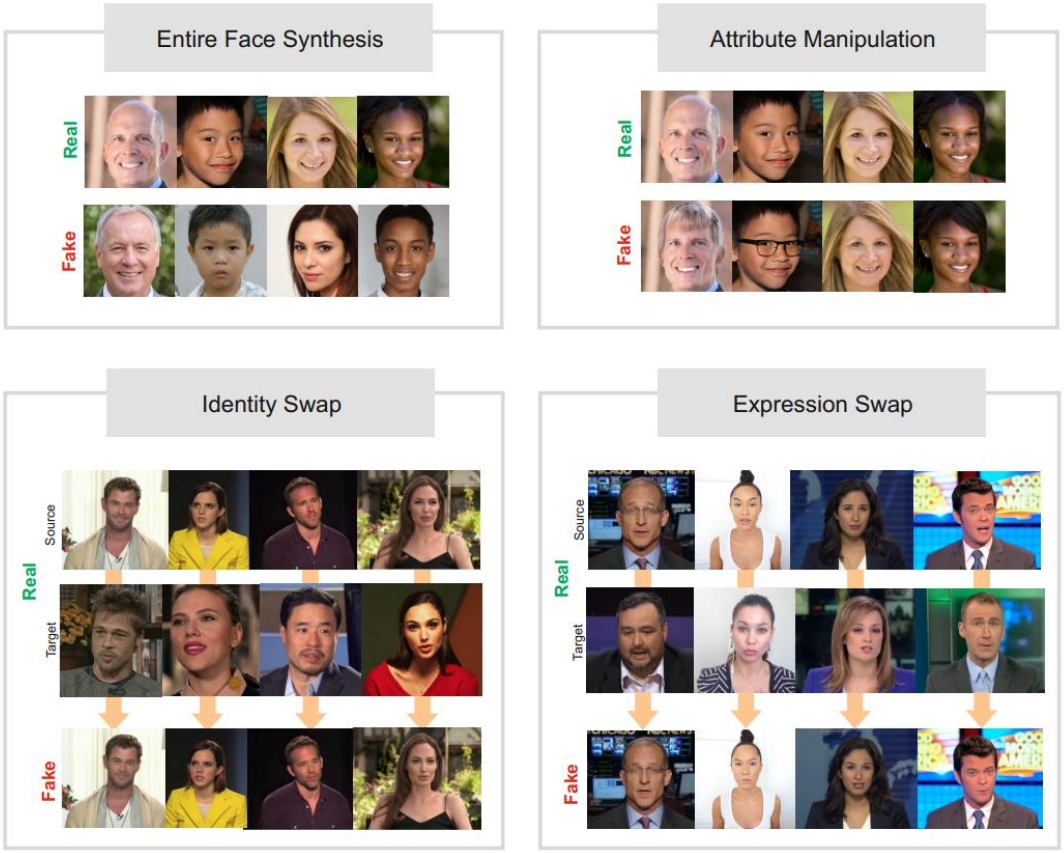

Figure 3.1: Different categories of face manipulation methods. [2]

According to a recent survey [2] manipulation approaches can be classified into four separate methods which are Entire Face Synthesis, Attribute Manipulation, Identity Swap and Expression swap.

- **Entire Face Synthesis**: is mainly created by GAN where a completely AI generated face image is created. This can be used in many applications, but its main concern is creating non-existent identities or fake social media accounts.

- **Attribute Manipulation**: This technique focuses on editing some features of the input face such as changing hairstyle or adding glasses and so on. The main application of this is just editing photos or enhancing them or using it in applications such as FaceApp.

- **Identity Swap**: This method is probably the most concerning method, which is replacing the face of a person in a video with another person. This is one of the main methods we are targeting in our project as it can be used in many harmful situations. The main techniques in this method are DeepFakes[10] and FaceSwap[11] which are used in FaceForensics++ dataset discussed in the next section.

- **Expression Swap**: This is another concerning method where in this method you can apply the face expressions of a person in a video to another person in another video. In this approach you can edit what a person says in a real video he made which makes it harder to detect. The two main methods here are NeuralTextures [3] and Face2Face [4] which are found in the FaceForensics++ dataset as well.

Through our project we focus detecting on the last two manipulation methods which are Identity Swap and Expression Swap because they are the most concerning and have the most harmful impacts.

3.2. Background on FaceForensics++ dataset

The FaceForensics++ [1] dataset is a comprehensive benchmark dataset designed for the evaluation and development of deepfake detection methods. It aims to address the increasing concerns and challenges associated with the proliferation of deepfake technology. This dataset consists of a large collection of manipulated face videos created using various state-of-the-art deepfake generation methods, as well as a set of real face videos. FaceForensics++ is structured into two main categories, real (1000 videos) and fake (4000 videos). The fake is structured into four main categories each representing a different manipulation method:

- Deepfakes: This category includes videos generated using deepfake techniques and models. These videos involve the replacement of one person's face with another. The deepfake videos are created by blending the

[10] https://github.com/deepfakes/faceswap
[11] https://github.com/MarekKowalski/FaceSwap

facial features of a source person into the appearance of a target person, resulting in realistic but synthetic videos.

- Face2Face [4]: This category contains videos manipulated using the Face2Face approach. Face2Face technology uses facial tracking and mapping techniques to transfer the facial movements and expressions of one person onto another person in real-time.

- FaceSwap: This category comprises videos created with the FaceSwap technique. FaceSwap involves swapping the faces of two individuals in a video, resulting in videos where the identities of the individuals have been exchanged.

- NeuralTextures [3]: This category includes videos manipulated using the NeuralTextures approach. NeuralTextures generates convincing deepfake videos by blending the textures of a target face with the facial attributes and expressions of a source face.

The FaceForensics++ dataset also provides videos with several compression factors (Raw, C23, C40) which can make it more challenging to detect any manipulations or alterations that might have been made to the videos. However, these compressed videos can still be used as a benchmark dataset for developing and testing deep learning models and algorithms for detecting deepfake videos. The videos in the FaceForensics++ dataset is compressed for two main reasons:

- To mimic real-world scenarios: In practice, deepfake videos may be compressed to various degrees depending on the platform or medium used to share them. For example, videos shared on social media platforms are often heavily compressed to save bandwidth and reduce loading times. By providing compressed versions of the videos in the dataset, we can create a more realistic test set that better simulates the conditions under which deepfake videos are often shared.

- To challenge automated detection algorithms: Compression can introduce artifacts into the video that can make it more challenging to detect deepfakes using current automated methods. By including compressed versions of the videos in the dataset, researchers can create a more difficult test set that

better represents the challenges faced by automated detection algorithms used to detect deepfake videos.

Figure 3.2: Same frame with different compression factors [1]

The FaceForensics++ dataset with compression factor C23 is provided directly through Kaggle but the full version of the dataset is available through the download script after filling a google form. The download script gives many options:

- All Datasets: Original, DeepFakeDetection_original, Deepfakes, DeepFakeDetection, Face2Face, FaceShifter, FaceSwap and NeuralTextures.
- Compression: raw, C23 and C40
- Type: video, masks, and models
- Servers: EU, EU2 and CA

From these options, we downloaded (original, Deepfakes, FaceSwap, Face2Face and NeuralTextures) videos with compression factors (C23, C40) through EU2 server.

3.3. Comparative Study of Previous Work

The academic interest in deepfake and face manipulation detection have increased massively over the last few years with various encouragement from tech giants and large institutes like Facebook, Amazon, MIT, DARBA and more by creating various competitions and creating datasets.
We will show some of the best approaches split into Deep Learning based methods and Classical based methods.

Method	Description	Performance on FF++
Xception [1]	A method based on ImageNet model that is finetuned for binary classification.	AUC = 99.3% (HQ) AUC = 99.8% (RAW)
CNN-aug [5]	Generating synthesized fake dataset using different generator models and training on ProGAN model.	AUC = 99.1% (HQ) AUC = 99.8% (RAW)
Patch-based [6]	Train a CNN based on local patches rather the whole face.	AUC = 97.2% (HQ) AUC = 99.9% (RAW)
SBI [7]	Generating fake videos by inducing artifacts to real ones and train on EfficeintNet	AUC = 99.6% (RAW)
LipForensics [8]	Using ResNet18 + MSTCN model pretrained on Lip reading	AUC = 99.7% (HQ) AUC = 99.9% (RAW)

Table 3.1: Comparing some SOFA Deep Learning-based methods on face manipulation detection methods.

Method	Description	Performance on FF++
Dynamic Texture Analysis [9]	Using Local Derivative Pattern features to extract temporal texture features and train on SVM	AUC = 95%
Frequency Domain Analysis [10]	Analyzing frequency domain of video frames using azimuthal integration features and train on SVM.	Accuracy = 90%
Matern et. al [11]	Extracting some visual features and train on MLP	AUC=78%

Table 3.2:Comparing some classical ML methods on face manipulation detection methods.

3.4. Implemented Approach

We chose to implement four different detection approaches to make our system resilient to various manipulation methods and increase the ability of detecting various abnormalities.
We chose the most promising and best performance approaches while making sure they all have special different characteristics.

The four approaches are:

- Lips Movement Analysis, which focuses on analyzing lips movements only using a very strong Temporal Convolutional Network model pretrained on Lipreading task to capture precise mouth movements, LipForensics [8] had the best results on FF++ dataset.

- Dynamic Texture Analysis [9], which focuses on analyzing the frames texture of the video by extracting a temporal feature which is LDP, which gives the ability to analyze the video texture-wise.
- Frequency Domain Analysis [10], this method analyzes the frequency domain of the video which gives us another dimension to detect the abnormalities from.
- Self-Blended Images [7], the main advantage of this method is using only real videos and generate the artifacts on it, thus makes it general to various manipulation techniques, then use a strong EfficientNet for classification.

By combining these four methods of completely different approaches we get a very strong system.

Chapter 4: System Design and Architecture

At this chapter we introduce our system design and how we approach the idea as system, and will discuss the following:

- The entire system design.
- Machine Learning core methods.
- Pipeline of the system.
- Modules implementations and Results.
- Design Constraints.

4.1. Overview and Assumptions

For an input video we conduct our analysis through four main approaches as mentioned before, these approaches vary from classical machine learning techniques and deep learning techniques each trying to tackle different abnormalities to make our system resilient to different types of manipulations.
Each approach is discussed in detail in the following sections including implementation details, training, and results.

Our assumption is that the video will contain only one person to be analyzed, however our system can be easily scaled to analyze multiple persons in the same video.

4.2. System Architecture

Our architecture consists of six main modules, two of them are used for preprocessing which are Face Detector and Landmarks Detector, and four of them are the main analysis methods we use which are Lips Movement Analysis, Dynamic Texture Analysis, Frequency Domain Analysis and Self-Blended Images method.

The block diagram shows the main flow of analysis for an input video, firstly we need to detect the exact position of the face for all frames of the video to be analyzed, for Lips Movement analysis we also need the landmarks to conduct analysis on Mouth Region only, each analysis method produces its analysis results that will all be combined to show the authenticity of the input video.

4.2.1. Block Diagram

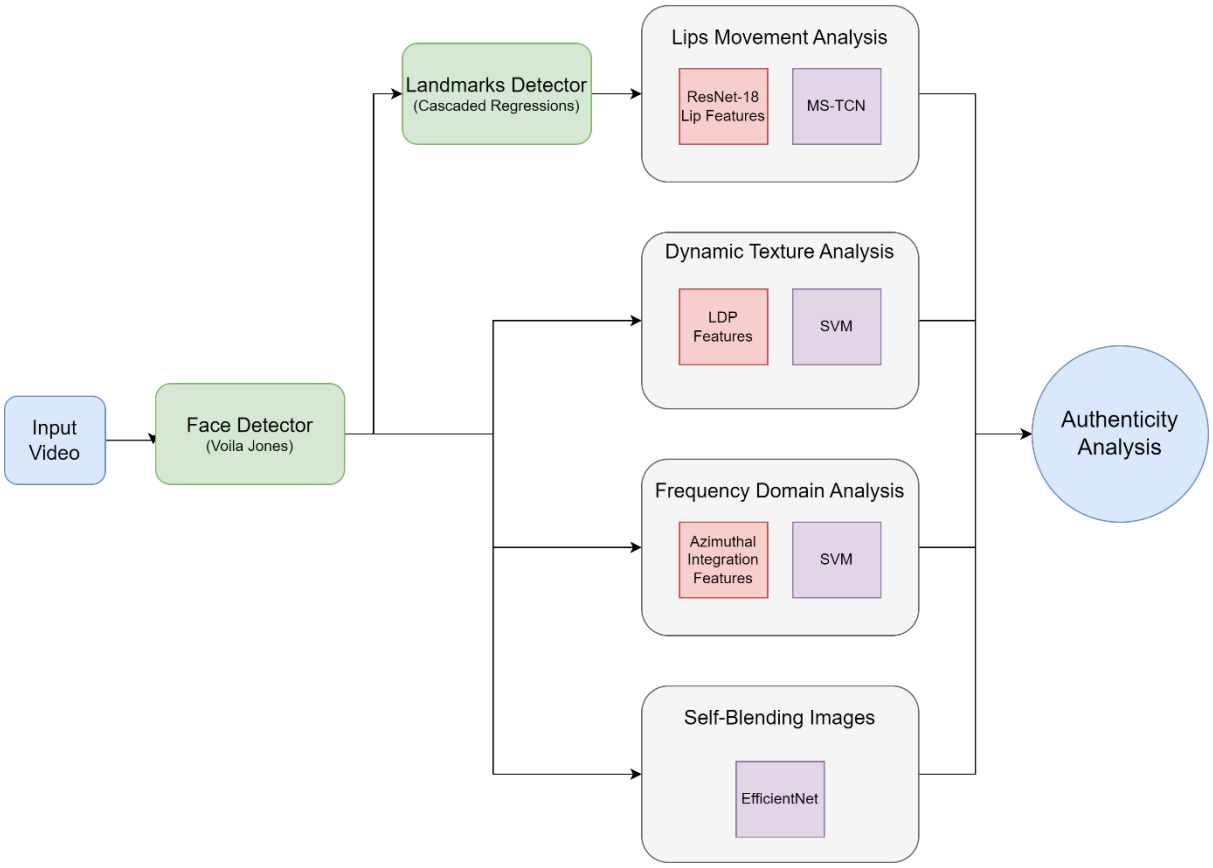

Figure 4.1: The figure represents our full pipeline main modules, starting from face detection and cropping the video frames around the face, to landmarks detection used in Lips Movement Analysis for detecting mouth region, and the other three analysis methods all using different features and classifiers with different analysis results.

4.3. Face Detector

4.3.1. Functional Description

The purpose of this module is to detect bounding boxes around faces in an input image. It is important to our project as our focus is on the manipulations made on faces i.e., Fake faces with altered facial expressions and speech movements like lips movement.

It is also an important step for landmark detection which is used in many of the fake detection methods. As fake videos change facial landmarks change how lips move and how facial expressions appear, thus creating new videos for some images but with fake expressions and movement of lips, eyes, etc.

Various methods were developed for this task. Some of these are based on deep learning approaches like retina-face which uses ResNet-50 (or any version of it) to classify and detect faces in an image.

The method we use for this module is the boosted cascade of simple features, which was first presented in the Viola-Jones paper [12]. It is not specific to face detection as it can be used for general object detection, but its focus is on faces. It is still in use now in cameras and libraries in programming languages (like open-cv in python) still use it.

The method uses many simple features extracted from a dataset of face images and non-face images, and it tries to choose which features best discriminate between faces and non-faces.

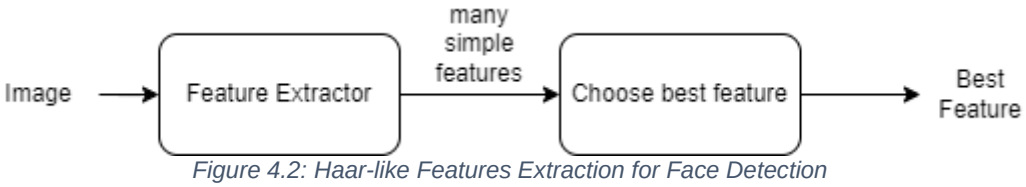

Figure 4.2: Haar-like Features Extraction for Face Detection

It then builds a boosted classifier using decision stumps on these features (using the AdaBoost algorithm) which builds a weighted sum of the classifications of each of these features.

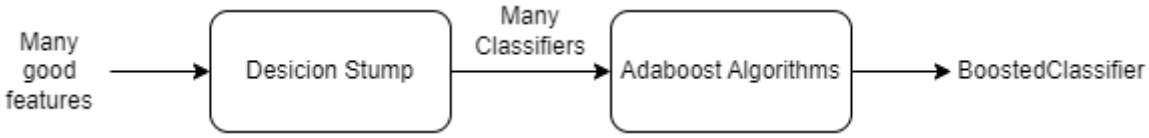

Figure 4.3: Adaboost to combine weak classifiers.

Then a cascade of these boosted classifiers is built where each layer in the cascade filters some non-faces till detecting whether an image is a face or not.

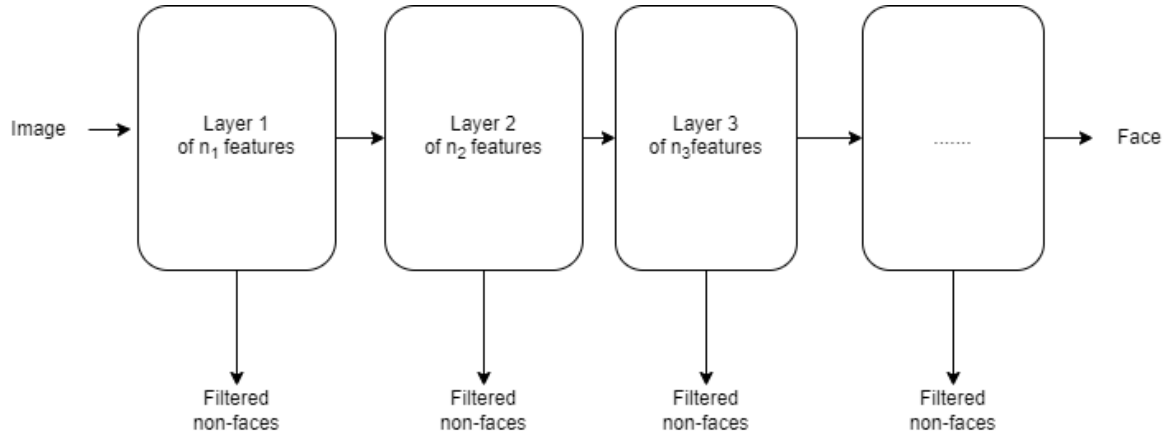

Figure 4.4: Cascade of boosted classifiers.

For a general image, the image will be divided into multiple sub-windows, and each sub-window is detected to be a face or non-face. Non-faces are filtered and are not passed to subsequent layers in the cascade (thus making the cascade faster). And harder examples keep going to later layers till detecting a face in the image. Then overlapping rectangles are combined to get a bounding box around the face.

This method takes a lot of time for training. And we observe that it can be broken down into independent similar modules where each one finds the one feature that minimizes the error, this is done for many simple features (162,336 features for 24x24 images) so we use GPUs and PyTorch to make feature extraction and the training process faster.

PyTorch is mainly used for deep learning as deep learning involves matrix multiplication where weights are multiplied by inputs while flooding the network and while going from one layer to another. However, PyTorch also has an interface (API) like that of NumPy where we can perform vector and matrix operations like sum, dot product, sorting, finding minimum, and so on using GPUs. This makes it easier to implement the tasks and modules than CUDA programming.

Microsoft Azure was used for training the final model using cloud resources (GPU, RAM, ...)

4.3.2. Modular Decomposition

4.3.2.1 Building Dataset

There are many datasets out there with annotated faces. We use the FDDB dataset [12]. It is a faces-in-the-wild dataset that contains 5171 faces present in 2845 images. The coordinates of bounding boxes are also provided (annotations). These annotations have been manually made, so they contain some space (padding) around the face.

We need our dataset to have faces and non-faces where images of both have the same size (i.e., 24x24 or 19x19, …). So, we find these faces, crop them, and resize them to the same size. We then scan the images to find sub-windows with different sizes that do not intersect with faces to make them background images (non-faces).

While we could have just used the given annotated faces, the added padding due to manual annotation can give shifted faces and this is not suitable for the algorithm we have as it finds features discriminating faces and non-faces based on these features' locations. Also, we humans can detect faces that are not looking straight toward the camera (like looking to the left or right). So, these images can give outliers where we do not want these examples to be included in the face images.

Thus, we use the pre-trained retina face to detect bounding boxes around the faces with high confidence (above some threshold like 90% confidence) such that it is more accurate than the annotated ones. But padding of annotated faces can be useful for staying away from the face when finding background images, so we use annotations to be away from faces for finding non-face images.

While trying different architectures and testing the implementation, we use a dataset where faces and non-faces are resized to 19x19 for faster feature extraction, training, and fewer resources needed as one 19x19 image results in 63,960 features. However, for the final model, we use 24x24 images which have 162,336 features.

Retina face gives us 4,523 face images. And finding backgrounds gives us. 28,121 non-face images. We randomly choose twice the number of faces as non-faces (9,046 non-faces) as we want to focus on non-faces (most of any image input to the model is background). We might even have increased the number of background images for more variety of backgrounds, but we just stuck to double the number of faces as non-faces. Thus, we have 13,569 images in the dataset. We split 80% as training and 20% as testing. So, we have 3618, 905 training and testing faces and 7236, 1810 training and testing non-faces respectively.

[12] http://vis-www.cs.umass.edu/fddb/

4.3.2.2 Feature Extraction

Features used in this method are simple features called Haar-like features. They consist of either 2, 3, or 4 rectangles where each has a start and end, and the pixel values inside rectangles are either added or subtracted according to the following shapes:

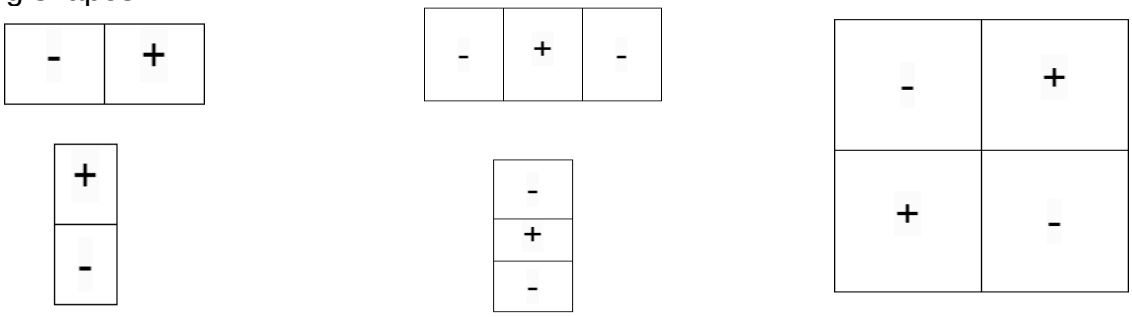

Table 4.1 Used 2 - rectangle, 3-rectangle, and 4-rectangle features.

To make computations faster for features, an integral image is calculated, where instead of pixel values at some location, the integral image stores the sum of all pixels that are to the above left to it. This can be achieved by using `cumsum` in PyTorch.

To extract these features for multiple images using GPUs, we first use the "Fancy Indexing" feature that is present in NumPy and PyTorch to extract many values at the same time. So, we describe the features as starting indices and ending indices of the rectangle and do these for all rectangles in the current feature. So, a 2-rectangle feature will be described by two elements each describing the start and end of a rectangle.

So, for 2-rectangle features, we have a positive rectangle and a negative rectangle for each feature. We can access the integral images that are stacked like this: `A[:, idx1, idx2]`, where idx1 and idx2 are row and column indices respectively. Idx1 and idx2 can be arrays of indices to get multiple features for each image (resulting in a matrix of shape (n images, n features) as all feature values will be stacked. An example of how the sum of rectangles is calculated using rectangle indices is shown in Figure 4.5 Rectangle area using integral images

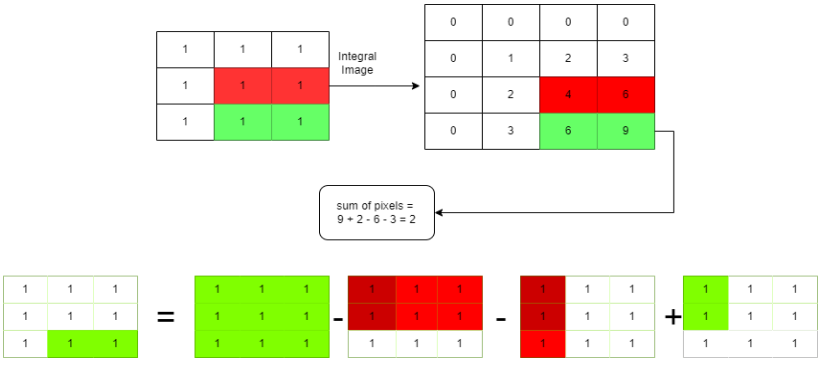

Figure 4.5 Rectangle area using integral images

Percentile of features:

Because we have so many features, and we have limited resources, we can take a subset of the features to use in training and compare between models, datasets, preprocessing training time, etc. We need this subset of features to be the effective ones in the predicted class.

This is done using the ANOVA test to get f-values for each feature then p-values for them using f-distribution and get the best (lowest p-values) k% of features to use in training. There are libraries in programming languages that give this percentile directly. In python, there is `sklearn. feature_selection. SelectPercentile`

The ANOVA test tests how much the means of different classes are different. So, the null hypothesis is that there is not much difference between the means of different groups. While the alternative hypothesis tells that there is a significant difference between the means of different classes. A difference between means of different classes means that this feature is good at discriminating between different classes.

For a list of examples, we want to find the f-value for some feature, we can do the following:

$$Fvalue(X: set\ of\ examples, y: labels\ of\ each\ example, F: feature\ extractor):$$
$$N: number\ of\ examples$$
$$n_c: number\ of\ examples\ \ class\ c$$
$$X \leftarrow F(X)$$
$$SSB \leftarrow 0: sum\ of\ squares\ between\ groups$$
$$SSW \leftarrow 0 : sum\ of\ squares\ within\ groups$$
$$\mu_{all} = \frac{\sum X[i]}{N}$$
$$for\ each\ class\ c\ in\ unique(y):$$
$$\mu[c] \leftarrow \frac{\sum X[i] \times I(X[i]\ c)}{n_c}$$
$$SSB \leftarrow SSB + n_c * (\mu[c] - \mu_{all})^2$$
$$SSW \leftarrow SSW + \sum_{i=0}^{i=n_c} (X[i] \times I(X[i]\ c) - \mu[c])^2$$
$$Fvalue \leftarrow \frac{SSB}{SSW}$$
$$return\ Fvalue$$

Table 4.2 – Calculating the F-value for some features over a set of examples.

Then, we find the p-value from F-distribution using this f-value and using:

$$\nu_1 = n_{classes} - 1,$$
$$\nu_2 = n_{samples} - n_{classes}$$

We then take k% of the features with the lowest p-values denoting these features have high effects on the output class.

Python's selectpercentile was used for comparing GPU implementation with sklearn's implementation of AdaBoost as well as comparing different datasets. The final model that was trained on azure used all features. Not a percentile of them.

4.3.2.3 Decision Stump

After extracting features from the dataset, we have a matrix of shape (n_features, n_samples) where each row denotes a specific feature calculated for all examples. We also have a weight for each example that tells how much to focus on that example. The weights are necessary by the AdaBoost algorithm.

Using the dataset, we can build a classifier $h_j(x)$ for each feature j (each row) where it is based on a threshold θ and a polarity p.So, given an example (x), we have:

$$h_j(x) = \begin{cases} 1 \; if \; p_j . f_j(x) \leq p_j . \theta_j \\ 0 \quad otherwise \end{cases}$$

We need to find the best θ and p that result in the least $\sum w_i \times I(y_i \neq h_j(x_i))$. Meaning we need to find the θ, p resulting in the least sum of weights of misclassified samples. It is obvious that if p is positive, then it is $f(x) \leq \theta$ and if it is negative, $f(x) > \theta$

We observe that if we choose one example as the threshold; meaning all examples having values higher than it (or lower than it is depending on p), they are faces, otherwise non-faces, we just need to find the example that separates the dataset best for that specific feature.

This task is repetitive and independent for all features, and that is where GPUs are used. We want to use PyTorch's functions (like sum, argsort, argmin, etc.) to find the threshold, polarity, and sum of weights of misclassified examples, and find the feature with minimum error.

First, we sort each row (feature values for each example) by the feature value and sort their corresponding weights, and labels by feature values. We then find the accumulative sum of weights of misclassified samples for that feature from the left and right. We sum those two accumulative weights (left and right). The position of the minimum weight is where the threshold should be.

If the error (sum of weights of misclassified examples from left and right) $\epsilon > 0.5$, we just reverse the polarity so that the $\epsilon < 0.5$. This is because the sum of the weights to 1. And if the classifier gets say, 80% of the examples as faces although they are non-faces, then just reverse its classification (reverse polarity). This makes $\epsilon \leftarrow 1 - \epsilon$

We thus have an error corresponding to each feature, we find the index of the feature that has the least error and consider it the classifier for that iteration in AdaBoost. In the next section, we will see how to use this classifier and how to update weights.

Table 4.3 – Decision Stump for choosing the best feature. shows a pseudo-code for this:

$Classifier(X, y, w)$:

$X: matrix\ of\ shape\ (n\ features \times n\ samples)$
$y: labels\ for\ each\ example$
$w: weights\ of\ each\ example$
$F_i(x): means\ executing\ the\ function\ accross\ axis\ i$

$idx \leftarrow argsort_1(X)$
$y, w \leftarrow repeat_0(y), repeat_0(w)$
$X, y, w = X[idx], y[idx], w[idx]$
$w_l, w_r \leftarrow cumsum_1(w), cumsum_1(reversed(w))$
$\epsilon \leftarrow w_l + w_r$
$p \leftarrow -1$
$\epsilon, p \leftarrow 1 - \epsilon, -p \quad if \quad \epsilon > 0.5$
$idx_{best} \leftarrow argmin_1(\epsilon)$
$\theta \leftarrow X[idx\_best]$
$idx_{best\ feature} \leftarrow argmin_0(\epsilon)$
$\epsilon_{best} \leftarrow \epsilon[idx_{best\ feature}]$
$\theta, p \leftarrow \theta[idx_{best\ feature}], p[idx_{best\ feature}]$
$return\ build\_classifier(\theta, p), \epsilon_{best}$

$build\_classifier(\theta, p)$:

$$h_j(x) = \begin{cases} 1\ if\ p_j . f_j(x) \leq p_j . \theta_j \\ 0 \quad otherwise \end{cases}$$

$return\ h(x)$

Table 4.3 – Decision Stump for choosing the best feature.

We divide the X into batches such that we do not exceed GPU memory. So, if we got better resources (GPU with more VRAM), we just increase batch size.

This GPU implementation made training time a lot faster. Comparing with sklearn's AdaBoostClassifier+DecisionTree(max_depth=1) as the base classifier (Which uses CPU): the GPU implementation was 8.5 times faster using 5000 as the batch size (more GPU VRAM can make this even faster)

The chosen features are the ones that best discriminate between faces and non-faces. For example, eyes and mouths tend to be darker. Thus, higher values in these locations push forward to making the classification for this image as a face.

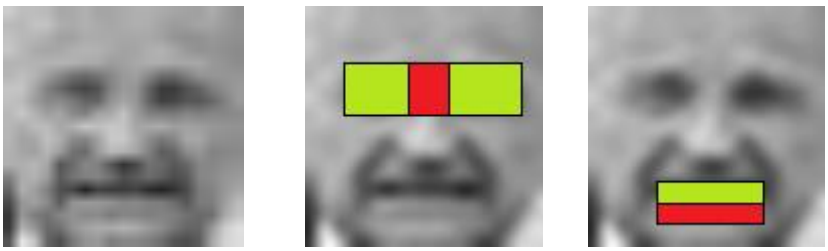

Table 4.4 Haar-like Features choose regions with high values for faces

4.3.2.4 Adaboost Classifier

This algorithm is used to combine a set of weak classifiers to form a strong one. It creates a weighted sum of the outcomes of each of the weak classifiers. The weak classifiers here are the decision stumps built in the previous section. The weight of a classifier will be proportional to its strength. A strong classifier here means it achieved low error on the dataset.

It is an iterative algorithm, running for T times and generating T classifiers. The more we increase T, the more it will focus on hard examples in the training data. Thus, it can get 100% accuracy on the training data. And this makes it sensitive to outliers. But this can be solved by omitting later classifiers are they are independent and later ones seem to contribute more to overfitting (instead of 200, can take 80) which is equivalent to regularization.

It gives a weight to each example, initially, all weights are equal. Then it keeps updating the weights after each iteration. The weights must sum to 1. Focusing on hard examples that were misclassified, increases their weights, while easier examples have less weight in subsequent iterations.

$AdaBoost(X, y, T)$:
    $y_i \{0, 1\}$
    $X: matrix\ of\ features\ (n\ features, n\ samples)$
    $T: number\ of\ iterations$

    $w_p, w_n \leftarrow \frac{1}{2\times\sum I(y_i=1)}, \frac{1}{2\times\sum I(y_i=0)}$
    $w \leftarrow y * w_p + (1 - y) * w_n$
    $for\ t\ between\ 1\ and\ T$:
        $w \leftarrow \frac{w}{\sum w}$
        $h_t(x), \epsilon = Classifier(X, y, w)$
        $\alpha_t \leftarrow ln\left(\frac{1-\epsilon}{\epsilon}\right)$
        $y_{pred} \leftarrow h_t(X)$
        $for\ i\ between\ 1\ and\ length(y)$:
            $if\ y_i = y_{pred}$:
                $w[i] \leftarrow w[i] \times e^{-\alpha_t}$
$Smaller\ w[i]\ if\ correct$
            $else$:
                $w[i] \leftarrow w[i]$

    $H(x) = \sum_{t=1}^{t=T} \alpha_t \times h_t(x) > \frac{1}{2}\sum \alpha_t$
    $Confidence(x) = \sum_{t=1}^{t=T} \alpha_t \times h_t(x)$
    $return\ H(x), Confidence(x)$

Table 4.5 Pseudo-Code for AdaBoost Classifier

The base classifier here was decision stump, but maybe if we used deeper decision trees this would make it better to capture relations between multiple features as well. For example, if $f1$ and $f2$ are high together, it is a face, and so on. But this would make its implementation on the GPU harder. In this case, though,

sklearn's AdaBoostClassifier+DecisionTree would be useful. But the decision stump still gives good results. Just maybe it would need fewer classifiers then.

4.3.2.5 Cascade Classifier

Here we build consequent layers where each layer is a classifier built with AdaBoost. Earlier layers have a smaller number of features and later layers have more features (more complex). An example for layers: instead of 100 classifiers in one layer, they will be divided into something like: [5, 5, 10, 10, 20, 50]

The purpose is to filter out obvious non-faces with simpler classifiers without the need for more computations; if detected as non-face, the image does not propagate to later layers. This is because most any image is non-face (background). This makes the classifier faster as only faces reach the final layers of it. Figure 4.4: Cascade of boosted classifiers. Shows how the cascade works.

In training it, each layer is trained on its number of features. Non-faces are not passed to later layers in the cascade as they have already been learned by earlier layers, thus saving training time. Also, new non-faces that the model so far detects as faces (false positives), are passed to subsequent layers.

4.3.2.6 Results

For the final model, we get 96% accuracy on the test set using 200 features in one layer. The dataset is 24x24 and we are not using percentile for training. We didn't use more layers in the cascade as this achieved less accuracy on the test set, it achieved 86% recall (found faces) while the one-layer model achieved 96% recall. We didn't apply normalization to the images as this didn't make a difference.

The training was done using Microsoft azure. The batch size was set to 10,000 as the GPU had 12 GB Ram and the node had 56 RAM. The training finished in 36 minutes. Training accuracy reached 100% and testing accuracy was 96% as was mentioned. The model detects the face in a 250x250 image in an average time of 0.6 seconds.

4.3.2.7 Face Detector

The image is divided into sub-windows of different scales. The minimum size is 24x24 and there is a stride between each sub-window. Sub-windows are scaled iteratively till being bigger than the image or reach a maximum size. The smaller the stride and the scale factor, the more precise the detection is.

Each sub-window is classified as non-face or face. If cascade is used, non-faces are not passed to subsequent layers. If one face in the image is to be detected, we can compute the confidence instead of AdaBoost's classification and find maximum confidence to be the one face that is detected.

Sub-windows that intersect with each other are combined in one bounding box and they are considered as one face.

Table 4.6 - Examples of results of face detector

4.3.3. Design Constraints

Here are some problems with the detector:

- The detector needs faces to be directly looking at the camera with minimum tilt or rotation.
- The architecture (number of cascaded layers) is an important factor in the performance of the final classifier as the more we increase layers, the better the detection but also the slower the model.
- False positives seem to be a problem with the final classifier although achieving good results on the test set (which means it did learn). A solution might be to increase the threshold of the AdaBoost. Another solution might be to put various types of backgrounds from different datasets in the training set. The solution used is to find the one face with the highest confidence, but this makes using cascade harder. This should not be problematic as most videos in fake datasets have 1 face.
- For limited memory and limited VRAM of GPU, we cannot train on large datasets or train on the full features, so, we train on a subset of features. If we need to solve this and use all features, we will need to get batches from files on the disk instead of from RAM.
- AdaBoost algorithm is sensitive to outliers, especially when increasing T. We might try to remove outliers features before training. Or find a maximum T that no layer would exceed. We can omit later classifiers as they seem to contribute more to overfitting (performing regularization).

4.3.4. Other Description of Face Detector

Failed trials:

• Simple dataset

Training on a simple dataset[13]. This dataset consists of 2,429 faces and 4,548 non-faces for the training set. And consists of 472 faces, and 23,573 non-faces for the test set. All images are 19x19. It was used for training at the beginning. And training accuracy was good (95%) but as the test set is highly imbalanced, it is obvious we need an f1 score. F1 score was just 25%. Which means it does overfit. Looking at the dataset,

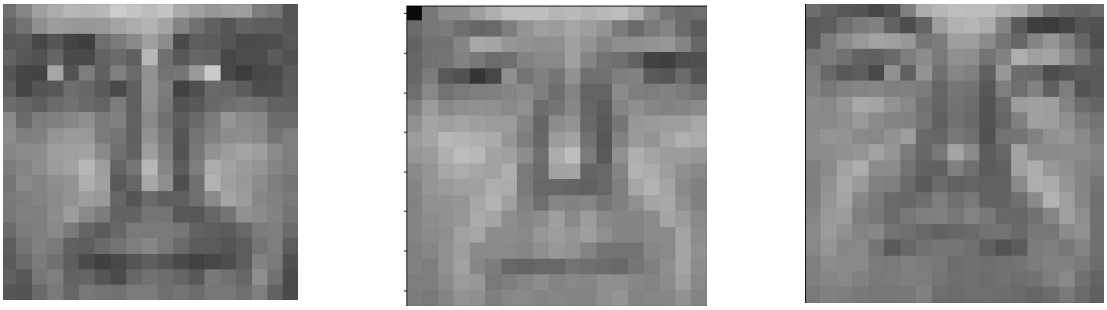

Table 4.7 Examples of easy faces causing overfitting.

Faces are so ideal and straightforward to the camera and properly cropped. So, we had to search for an alternative. And thus used FDDB, detected faces in it using the retina face. which was like the following:

Table 4.8 Examples for more general faces - for making the model more robust.

This made testing the f1 score become 96% - tested on a smaller number of non-faces (double the number of faces).

We kept using background faces for this simple dataset. But then generating background faces from FDDB resulted in better results (97.6% at some point)

• More features (350 instead of 200)

This resulted in 99.98% training accuracy but 94% on the testing set which seems to be overfitting. So, 200 features were used in subsequent trials.

[13] http://cbcl.mit.edu/software-datasets/FaceData2.html

• Normalization

Looks like it is subject to the specific dataset it was trained on as trying it on a different dataset, yielded bad bounding boxes. Maybe needs to merge different datasets as the training set and testing set to be more general.

• Cascade

Instead of 200 features in one layer, we tried 20 features for 10 layers. This resulted in 95% training accuracy and 95% testing accuracy. But the recall was bad in this case (86%) whereas the 200 features had 96% recall and 99.88% training accuracy.

• Local GPU vs Google Colab vs Microsoft Azure

For the previous simple dataset that had 6977 total training examples and that was 19x19 and taking 10% of features, the computation was quite feasible on local resources (GPU VRAM=4GB, RAM=8GB).

But for the bigger dataset (size=10,826) Google Colab was used (12.7 GB RAM + 15 GB VRAM) this even allowed for calculating the 20% percentile instead of 10%. This is because local resources were struggling.

For the final model 24x24 on the full dataset and full features, Microsoft azure was used (56 GB RAM + 12 GB VRAM)

• sklearn

Training using AdaBoostClassifier+DecisionTree(max_depth=1) was equivalent to our implementation. On Google Colab, training of 200 features and 20% percentile of features on the 10,826 examples dataset, took 38 minutes, while the GPU implementation took 4.5 minutes for training. However, sklearn's AdaBoost got 97.75% accuracy while the GPU implementation got 97.6% accuracy. This means both implementations of training are not the same, but they are very similar.

For inference, sklearn needed a matrix for the whole features and this took 90 seconds for one 250x250 image to find faces in it. But as we have more flexibility in our implementation, we can choose which features to extract for inference, and thus it took between 1 and 1.5 seconds to find the faces.

Other Trials

•4297 features in the paper

In the original paper, it was mentioned that a cascade was built using 4297 features. So just to know how much time it would take to train it, we trained the simple dataset (19x19) on the local machine using 10% of the features, and it took about 45 minutes to complete the training on a local machine.

Future work:

- Implement decision trees on GPU as the base classifier for AdaBoost.
- Embed different datasets and focus on non-faces (more diverse examples) for training.
- Optimizations in inference.
- Try training without outliers (remove examples far from the median)

4.4. Facial Landmarks Detector

4.4.1. Background

As explained in the system architecture, Facial Landmarks Detector is used to be able to track certain parts of the face, Lips Movements for example, or used to align the cropped face frames, and thus help in the analysis methods in the next stage.

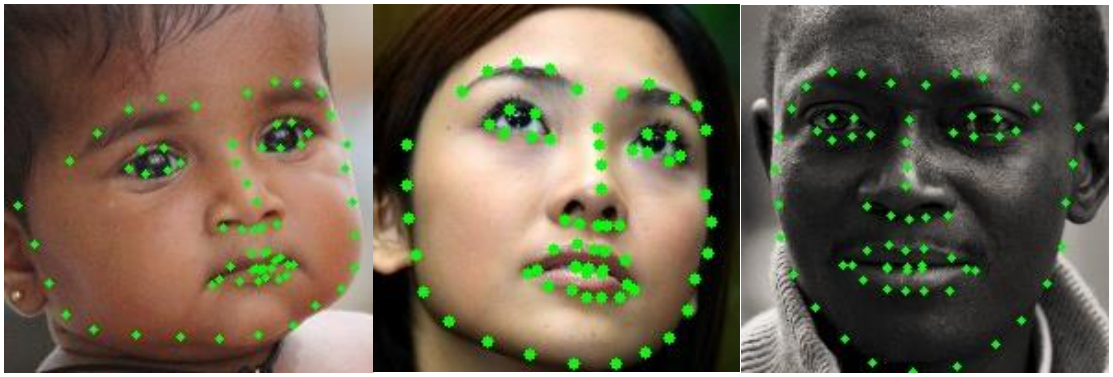

Figure 4.6: Some examples for facial landmarks detected using our implementation of landmarks detector.

Facial Landmarks or Face Alignment had been ubiquitous in research especially due to its wide uses in Computer Vision applications and facial analysis, like face recognition, emotion analysis, facial animations, virtual reality and more.

Various methods and algorithms have been used in the topic of face alignment throughout the years, however according to a recent survey [13], we can divide the methods into three main types: Holistic Methods, Constrained Local Methods and Regression based methods.

Algorithms	Appearance	Shape	Performance	Speed
Holistic Method	whole face	explicit	poor generalization/good	slow/fast
Constrained Local Method (CLM)	local patch	explicit	good	slow/fast
Regression-based Method	local patch/whole face	implicit	good/very good	fast/very fast

Table 4.9: Comparison of three main categories of landmark detection algorithms [13]

Using this information, we can notice that Regression-based methods are the most favorable, not only they are faster, but they perform better as well.

The core idea of Regression based methods is to find a direct mapping between the image features and the landmarks locations, and they are further divided into three types: Direct Regression, Cascaded Regression and Deep Learning based Regression.

Direct Regression learns a direct mapping between the image features and the landmarks in one step without any initialization, while cascaded regression starts from an initial prediction and apply a series of regressors that tries to take a step closer to the correct landmarks' locations, and finally the Deep Learning based methods use CNN to extract the features from the image.

We chose to implement a Cascaded Regression method based on the Coarse-to-Fine Shape Searching algorithm [14] which achieved the best results on multiple benchmark datasets.

4.4.2. Functional Description

Facial Landmark Detectors in general are given a face image as an input and it outputs the positions of the landmarks of the face. By applying this in the context of our system, a video to be analyzed is used as an input, firstly it gets through our Face Detector module that will output a series of frames of cropped faces, we apply our Landmark Detector on each frame to output a series of landmarks positions for each frame.

As mentioned in section 4.4.1, the exact method we used is based on CFSS Algorithm (Coarse-to-Fine Shape Searching) due to its great results and fast inference.

The output shape of landmarks is based on the standard shape used by IBUG [15] to annotate all benchmark datasets.

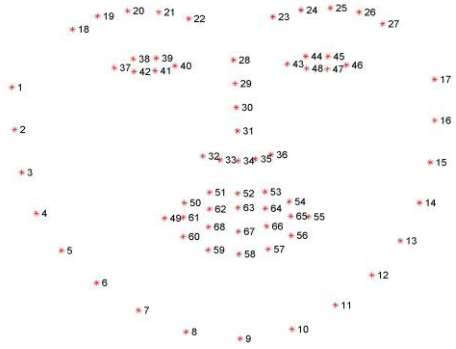

Figure 4.7: Standard 68 landmarks format used in research and applications. [15]

The main idea of CFSS algorithm is to start from an initial prediction, then by applying a series of regressors trained on features locally extracted from the image based on landmarks positions as X and the offset between the ground truth and the current prediction as Y, each regressor take a step closer to the ground truth until the final regressor, more details of the algorithm will be described in the next section.

4.4.3. Modular Decomposition

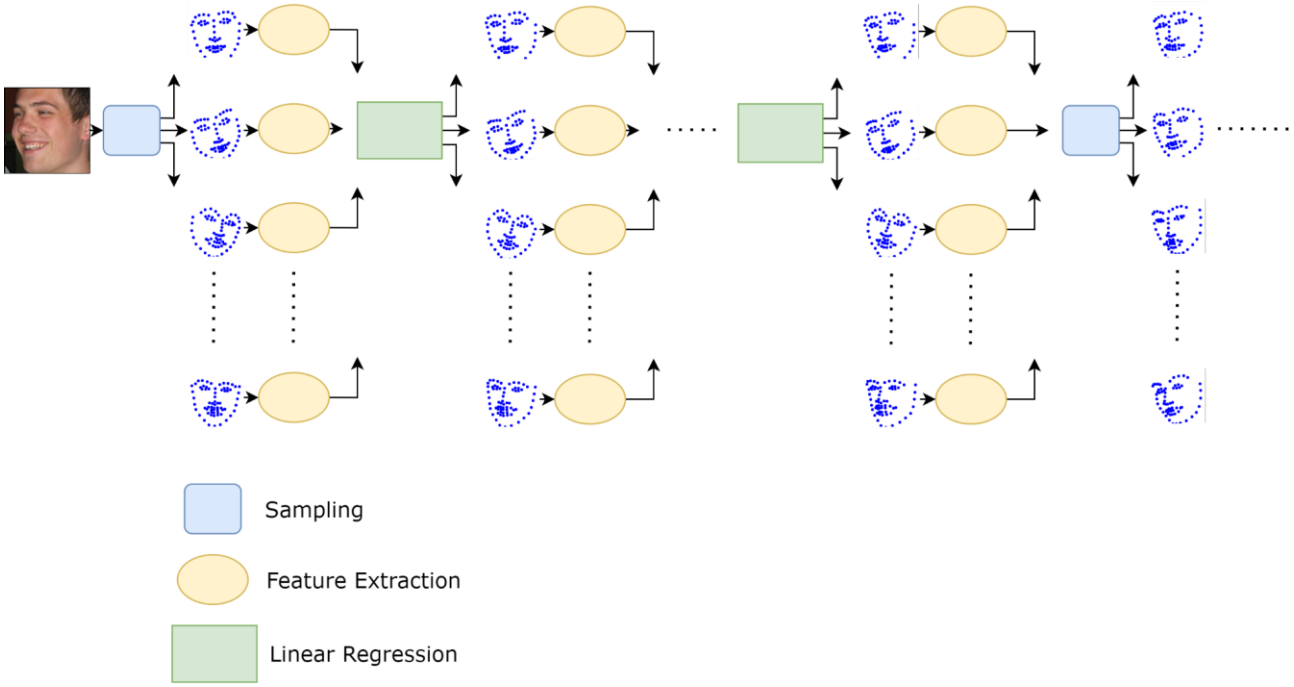

Figure 4.8: A Figure that represents Coarse-to-Fine Shape Searching algorithm, it shows one stage of the pipeline training process. where we try to reduce the searching space each stage making it closer to the ground truth.

The CFSS [14] algorithm can be summarized in Figure 4.8, for each image in the dataset, samples of size Ns are drawn from candidate shapes (the dataset) based on a probability distribution Pr, initially the distribution is a normal distribution on all images. Then a dataset of length NxNs is generated from sampling, each data point is features extracted around each landmark of the sampled shape, and the label a vector of size 136 which is the differences between ground truth landmarks and the sampled shape. For each stage this is repeated for K regressors, then finally a new distribution is calculated and the whole stage is repeated with another K regressors for a definite number of total stages.

We can notice that the algorithm can have many various ranges of factors that affect it, not only the hyperparameters but also how the exact steps are implemented, how the preprocessing is made, how the sampling is made, what size of images are used, what features are used and so on.
Throughout this project we have tried different combinations of parameters, settings, and implementations that we generated over 100 different models and reports to get the optimal settings.

In the upcoming sections we will discuss every step in detail, the preprocessing, the types of features used, the exact training algorithm in detail, and finally the chosen final model and how the inference is made.

4.4.3.1 Pre-processing

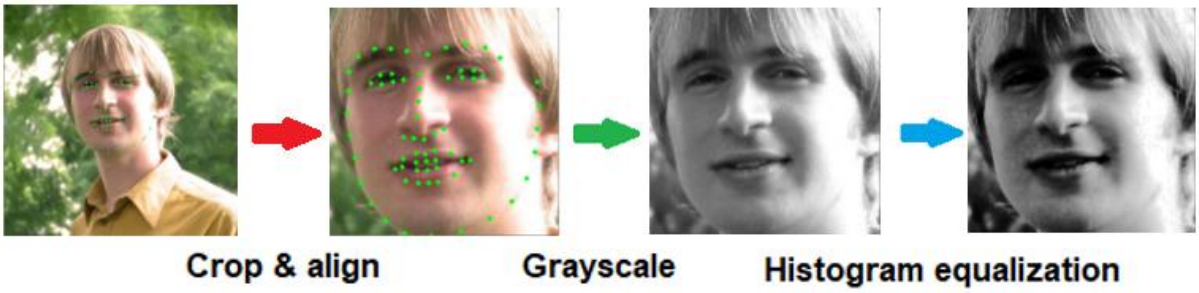

Figure 4.9: The preprocessing pipeline needed for landmark detection consisting of three stages, cropping, gray scale transformation and histogram equalization.

Preprocessing is an important step before applying the algorithm. The main goal of preprocessing is to center and align the face images before the training process and reduce the effect of different lightings, colors, brightness, exposure, and resolutions.

For training we need to make three steps shown in Figure 4.9.9 for preprocessing: Crop & Align, Grayscale and Histogram Equalization, for inference only the last two steps are used since the Face Detector will crop the exact image of the face.

The first step in preprocessing is to crop and align the face image, this is due to the fact that Facial Landmarks Detectors input is always the cropped face image, as its main goal is to detect the landmarks positions in the face and the pose not to detect the face image itself, so normally

Algorithm: Crop & Align images

```
1.  Procedure CropAndAlign(Dataset Data, Target Shape T, Padding P)
2.    Set Padding Top PT, Right PR, Left PL, Bottom PB = Padding P
3.    For row in Data:
4.      Get Image , Landmarks from Row
5.      Get Bounding Box of Landmarks Xmin,Ymin, W, H
6.      Update the bounding box with Paddings (7-10):
7.      Set Xmin = Xmin – W * PL
8.      Set Ymin = Ymin – H * PT
9.      Set Xmax = Xmax + W * PR
10.     Set Ymax = Ymax + W * PR
11.     Update bounding box to be inside image(12-14):
12.       Xmin, Xmax = max(Xmin, 0), min(Xmax, ImageWidth)
13.       Ymin, Ymax = max(Ymin, 0), min(Ymax, ImageWidth)
14.     Update W, H = Xmax – Xmin, Ymax-Ymin
15.     Update bounding box to be square(16-20):
16.     IF Width W Greater Than Height H:
17.       Update Xmin, Xmax to match Height
18.     Else:
19.       Update Ymin, Ymax to match Width
20.     End IF
21.     Crop Image with bounding box
22.     Resize Image to Target Shape
23.     Update Image
24.     Update Landmarks to match Cropped and Resized Image
25.   End For
26.   Return Data
27. End Procedure
```

Table 4.10: Algorithm used for cropping the dataset images for training landmarks detector.

Face Detectors are used before landmarks detectors, so that the input is only the face image.
Since we are working on benchmark “in the wild” datasets, where they are extracted randomly from images on the internet, they have different sizes, settings, zooming, lighting, etc.
So, the first step is to crop images around the face using the given annotations in the dataset and then finally resize to a given target size.
The Algorithm in Table explains the Cropping and Alignment process, given as an input the full dataset of images with annotations, the target size of the image, and the crop paddings on each side the algorithm updates the dataset with cropped face images.
The algorithm starts by defining a bounding box around the given landmarks of an image, updates it with the given paddings, then decreases the bigger dimension to be a square, finally the image is cropped and resized, and annotations are updated accordingly.
The chosen parameters for implementation to get optimal results were (200,200) target size, top padding of 0.3 and 0.1 for the rest. Higher padding on top led to better results in general as it takes into account the forehead area.

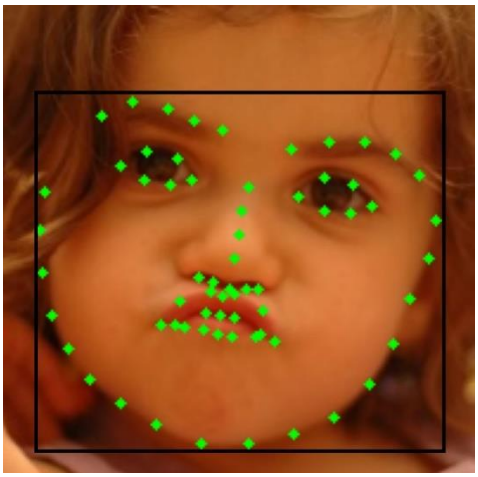

Figure 4.10: Representation of padding used around face area for cropping the dataset, where top padding is higher to take full face.

The second and third step in Preprocessing is to transfer the image to grayscale and apply histogram equalization, and this step is important to reduce the effect of different illuminations, brightness, and colors.
The algorithm was tested with and without histogram equalization step and it led to better generalization and better accuracy when applying the equalization.

4.4.3.2 Feature Extraction

As we have seen in the CFSS algorithm, Feature Extraction is used in the regressors to map between the image features and the offset of errors between the ground truth and the current prediction, various feature descriptors that are commonly used in Computer Vision can be used in this case like LBP, HOG, SIFT, BRIEF, SURF, ORB, etc., however for our purpose we need fast features for faster inference as well as being strong features, and being able to perform fast computations is important because we apply feature extraction for a definite area around each landmark before applying each regressor.
Research papers suggested that in earlier stages we can use fast feature descriptors then use strong features in the later stages, throughout our project we tried different combinations of features, mainly we tried ORB, HOG and SIFT and for the final chosen model only HOG and SIFT are used. We have implemented our own version of HOG and compared it to HOG implemented by Scikit Image[14] and both achieved similar results in training.

4.4.3.2.1 HOG Implementation

Histogram of Oriented Gradients (HOG) [16] is a very common feature descriptor for computer vision and image processing applications, it has been widely adopted due to its simplicity, efficiency, and effectiveness in detecting objects with well-defined shapes and edges.

Figure 4.11: Visualization of HOG features extracted from the image.

[14] https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_hog.html#id1

HOG Algorithm consists of four major steps shown in Table , the first step is to compute the gradient image in x and y, Sobel filters are used to compute the first derivative of the input image.

The second step is by computing a histogram of gradients over each cell, where each image is divided into different cells determined by cell pixel size, the histogram is calculated by dividing the gradient angle over bins ranging from -180 to 180.
The third step is to normalize histograms over blocks, where each block contains multiple overlapping cells, where a block stride is 1.
Finally, the normalized histograms are concatenated together to form the feature vector extracted from the input image.

Algorithm: Calculate HOG Feature Descriptor

```
1.  Procedure CalculateHog(Image, PixelsPerCell, CellsPerBlock, Orientations)
2.    Gx, Gy = Sobel(Image)
3.    Get Gradients and Angles from Gx, Gy
4.    Split Image to Cells
5.    For cell in cells:
6.      Calculate Histogram of Cell
7.    End For
8.    For block in blocks:
9.      Normalize Histograms of cells in block
10.   End For
11.   Get feature vector from Concatenated normalized blocks
12.   Return feature vector
13. End Procedure
```

Table 4.11: HOG Feature Extraction Algorithm

4.4.3.3 Training Algorithm

Algorithm: Training Facial Landmarks Detector

1. **Procedure** Train(Training Dataset $\{I^i, x^i\}^N$)
2. Set Candidate Shapes S to be the training set shapes x
3. Set Uniform distribution Pr(0) of size (NxN) as uniform for all images
4. **For** l in range 1..L
5. Sample NxNs shapes from Candidate Shapes using Pr(l-1) where each sample is x^{ij} (1)
6. Train K_l Regressors on sampled shapes of size NxNs (2)
7. Update Sampled Shapes with trained Regressors
8. **IF** l < L
9. Set initial weights vector for samples of each training image as uniform $w^i(0) = e/Ns$
10. Construct affinity matrix for each image samples (3)
11. **For** t in range 1..T
12. Update weights $w^i(t)$ (4)
13. **End For**
14. Compute average samples for each training image $\bar{x}^i(l)$ using $w^i(l)$
15. Update Probability Distributions Pr using new centers $\bar{x}^i(l)$ (5)
16. **End IF**
17. **End For**
18. **End Procedure**

Table 4.12: Summarization of CFSS Training Algorithm showing 5 main stages: sampling, training regressors, affinity matrices calculation, updating weights, and finally updating probability distributions.

In this section we will explain the general training algorithm of Coarse-to-fine Shape Searching method, as explained earlier, the main idea of the algorithm is to start from an initial prediction then try to take steps towards the ground truth by searching in a narrower search space.
The search space for each training image indexed by i is defined by (Pr^i, $\bar{x}^i$) a Probability Distribution Pr and center $\bar{x}$ of search space.
Firstly, the algorithm starts by setting the candidate shapes as the training set shapes, or the given annotations for the training set, and also set the distribution as uniform distribution.
The algorithm contains 5 major steps that will be explained in the coming sub sections which are sampling, training regressors, constructing affinity matrices, updating weights, and finally updating the probability distributions.
These steps are repeated for a chosen number of stages L that is usually a small number 2 or 3, where each stage contains multiple regressors usually 3 as well.

4.4.3.3.1 Sampling

The first step is to sample Ns samples from N candidate shapes for each image using the probability distribution Pr, initially since it starts with uniform distribution and random samples are firstly chosen as shown in Figure 4.12 from the initial diagram.

But going through stages the probability distribution is updated such that the closest shapes have the highest probability to be sampled and thus we can narrow the search space and as we can see in the figure by the next iteration, we can see that the sampled shapes have a closer pose to the real face.

The idea behind resampling instead of following up to the same regressed shapes is to prevent models from being trapped in local optima and thus improve the training process.

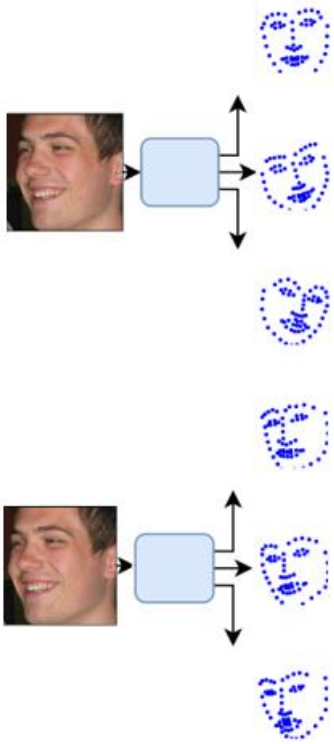

Figure 4.12: Representation of sampling process in CFSS algorithm, initially as seen in the upper figure the shapes are random and not necessarily close to the ground truth, by going through stages, in the lower half the samples became much closer to the pose.

After sampling we now have a dataset of size NxNs instead of N and this helps in generalization, however it was noticed choosing suitable Ns can affect the model's performance greatly, low Ns leads to worse generalization and simply we will not have enough searching space for each image, high Ns would make it difficult to find enough samples closer to the ground truth and have narrower search space in late iterations and make it difficult for regressors to converge, so suitable Ns can range from 5 to 15.

By default, sampling should be done without replacement, however during our trials to get narrower search space at later iterations specially when training in a small dataset we tried to use sampling with replacement then add a small random error to avoid having repeated samples, this did not lead to better results and was discarded at the end.

4.4.3.3.2 Training Regressors and Feature extraction

This step is the core of the algorithm, it is needless to say that Linear Regressions are used due to its simplicity and having a closed form that makes the training and inference very quick compared to other models, but that makes it harder to find

optimal solution, thus multiple regressors in needed in Cascaded Regression algorithms where each regressor takes a step towards the optimal result.

These equations summarize cascaded regressions algorithms:

$for\ k\ in\ 1..K$

$$r_k = argmin(R) \sum_{i=1}^{N} \sum_{j=1}^{Ns} \left|\left| x^{i*} - x_k^{ij} - r(\phi(x_k^{ij})) \right|\right| + \Phi(r)$$

$$x_{k+1}^{ij} = x_k^{ij} + r_k(\phi(x_k^{ij}))$$

As we noticed in the algorithm we have L iterations, each iterations contain K regressors, where each regressor is trained on local extracted features around landmarks $\phi(x_k^{ij})$ and NxNs samples offset from the original Image ground truth x^{i*} as the label, the output of each regressor updates the sampled shapes to be input to the next regressor.
To avoid overfitting and improve regularization Ridge Regression is used with the parameter $\Phi(r)$ as the regularization parameter.
As mentioned before, different feature descriptors can be used, but we can notice through this equation how it is important for both training and inference to choose faster computation features, then stronger features in later stages to improve prediction.
We tried BRIEF and ORB features but as mentioned before HOG and SIFT were chosen for better performance, we will discuss all details about implementation and different trials in section 4.4.3.4.
Another important notice is that different feature descriptors will lead to different length feature vectors (mostly very large) as we extract a feature vector for each landmark, so at 68 landmarks we produce for every sample a feature vector of length Fx68 where F is the feature vector extracted from the patch around each landmark, this suggests that dimensionality reduction is needed and thus principal component analysis is used to reduce the dimensionality of features extracted to avoid the dimensionality curse problem.
Finally, it is very important to notice that regularization parameter changes based on the type of feature, thus hyperparameters tuning was made according to each feature type.

4.4.3.3.3 Constructing affinity matrices.

After regressing the shapes closer to the ground truth, we need to update the probability distribution Pr, and this step is the first step towards updating it.

The idea behind this step is that after regressing the sampled shapes, there will be a portion of sampled shapes that is closer to each other than the others and these samples are also by default closer to the ground truth, so the idea here is to give more weight to samples that are more similar to each other.
So, we construct a symmetric matrix of size NsxNs with zeros at the diagonal for each image that describes how close each sample is to the others as described in this equation.

$$a_{pq} = exp(-\beta \left|\left| x_K^{ip} - x_K^{iq} \right|\right|), p \neq q$$

β in the similarity equation is determined via hyperparameters tuning.

4.4.3.3.4 Updating weights.

Using affinity matrix A calculated in the previous step, we need now to assign weights for each sample through an iterative process, initially all the weights are equal to 1/Ns, then through an optimization technique known as replicator dynamics [17] the weights are updated by this equation.

$$w^i(t+1) = \frac{w^i(t) \circ (Aw^i(t))}{w^i(t)^T (Aw^i(t))}$$

This equation is calculated for T iterations, normally T=10.
Using this equation samples that are closer to each other will have a higher weight, using this weight we can now calculate $\bar{x}^i$ as a weighted average of samples which will help in updating the probability distribution in the next step.

4.4.3.3.5 Updating probability distribution.

The expected output of the probability distribution step is a narrower search space for each image by giving high weights to closer candidate shapes to the ground truth. We used another similarity function which is the Gaussian similarity function between the candidate shapes and mean shape $\bar{x}$ according to this equation.

$$Pr \propto exp(-\frac{1}{2}(s - \bar{x})^T \Sigma^{-1}(s - \bar{x}))$$

This is then normalized to produce a probability distribution for each image and be used in the sampling process in the next step.
Research papers suggested using another factor for updating the distribution learned by discriminative models to map between local features and the error offset between each facial sub region and the mean shape, but we ignored this factor to reduce the complexity of the model.

4.4.3.4 Results and Trials

4.4.3.4.1 Inference

For inference, after cropping the face using a Face Detector and applying the same preprocessing mentioned in section 4.4.3.1, inference happens by starting with an initial prediction, usually the mean shape of the dataset, then regressing it through the trained regressors with the corresponding chosen feature extractors as can be seen in the following figure.

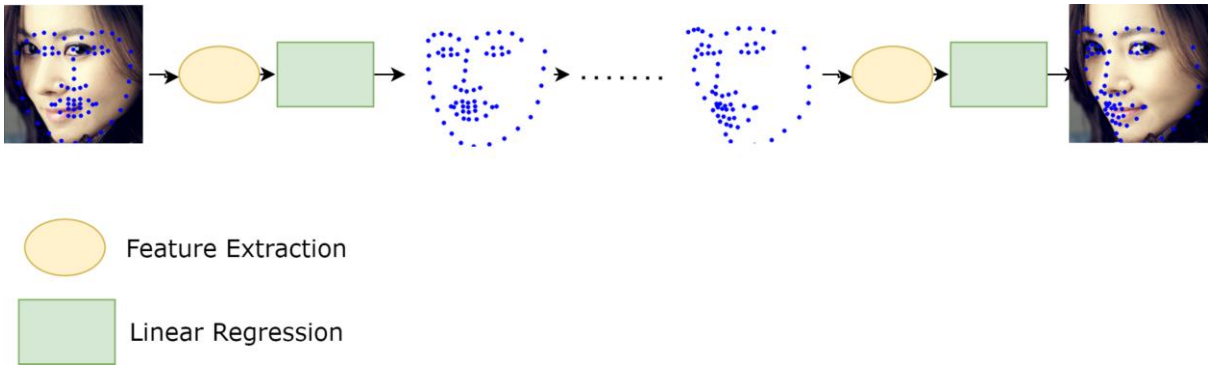

Figure 4.13: Inference in CFSS algorithm where we start from the initial shape and go through all regressors until final result.

4.4.3.4.2 Evaluation criteria

The most popular standard way to calculate the error is the inter-ocular distance normalized error, it is simply calculated by normalizing the Euclidean distance between each corresponding landmark in the predicted shape and the ground truth by the distance between the centers of the pupils of left eye and right eye, the main advantage of this is to make the comparison fair for different image sizes and different face poses.

$$normalizedError = \frac{1}{N}\sum_{i=1}^{N}\frac{||d_i - g_i||}{||g_{le} - g_{re}||}$$

Where N represents the length of the landmarks, d represents the predicted shape, g represents the ground truth, g_{le} is the left pupil center, g_{re} is the right pupil center. However, since not all datasets annotate the pupils, it can be estimated as the center of each eye region.
Over the full test set the mean normalized error is used.

4.4.3.4.3 Datasets

Facial images datasets can be divided into “controlled” datasets and “in the wild” datasets, the latter is preferably used as benchmark datasets because it captures the diversity and complexity of real-world scenarios, such that there is no limit to the images used in the dataset.

Most popular benchmark datasets for Facial Landmarks detection problems are Helen [18], AFW [19], LFPW [20] and 300w [15]
We used all of these datasets for both training and testing our models.

One problem was that datasets can come with different annotations standards and even different lengths, this problem was solved by iBUG [15] who re-annotated all these datasets using a standard annotation showed previously in Figure 4.7 to provide a standard way for annotations and make the comparisons easier and more fair.

4.4.3.4.4 Final Model results and implementation

Figure 4.14: Some results of our landmark detector from test sets or random images from the internet showing great diversity of colors, resolutions, poses, expressions, and even masked faces.

Our final model implementation that had the highest accuracy used the following parameters:

L=3, we used three stages or three iterations for the training process

K=3, each iteration has three regressors, so in total 9 regressors.
S=5, The sample size.
β**=1**, the sensitivity parameter used for similarity function in calculating affinity matrix.
T=10, Number of iterations for updating the weights.
FeaturesUsed = [HOG, HOG, SIFT] the features used per each stage, where each K regressors in a single stage are trained on the corresponding feature extractor.
alphas = [10000, 5000, 40000] the L2 regularization parameters for each stage, where each K regressors in each stage are trained on the corresponding regularization parameter, initially we used single regularization parameter per feature not per stage, but trying to change regularization parameters per stage improved the final results, it was noticed that higher regularization was needed with SIFT due its high tendency to overfitting and for being a stronger feature.
pca_sift_components = 2176, The number of components used in reducing the dimensions of SIFT features, SIFT produced a feature vector of length 8704 with 95% of variance in 1600-2000 dimensions depending on each regressor, so 2176 was chosen to standardize the length of feature vector across all regressors.
ImageSize = (200,200), the standard image size used, at higher image sizes, larger feature vectors will be needed and that makes training more difficult, low-resolution image decreases the ability to extract the needed features which leads to underfitting.

The following tables will show our final model results and its comparison to some of the leading algorithms. Note that the type columns describe the type of algorithm C: Constrained Local Methods, DR: Deep Learning, R: Regression based.

Dataset	**Normalized Error**
Helen [18]	6.38
300w [15]	10.27
LFPW [20]	6.15
Combined datasets	6.89

Table 4.13: Final model normalized error results on various benchmark datasets.

Helen			LFPW		
Method	**Type**	**Error**	**Method**	**Type**	**Error**
FPLL [19]	C	8.16	FPLL [19]	C	8.29
DRMF [21]	C	6.70	DRMF [21]	C	6.57
Revelio	**R**	**6.38**	RCPR [22]	R	6.56
RCPR [22]	R	5.93	**Revelio**	**R**	**6.15**
CFAN [23]	DR	5.53	CFAN [23]	DR	5.44
Original CFSS [14]	R	4.63	Original CFSS [14]	R	4.87

Table 4.14: Comparison between our Facial Landmarks Detector and other leading methods

4.4.3.4.5 Trials

As we have mentioned before many parameters can affect the results including the cropping of face images from the original datasets, throughout our project we conducted about 100 different trials each testing different settings and parameters, in this section we will discuss only some of the trials that led to the selection of the final model.

For early trials we used only Helen dataset to reduce the complexity of training.

we used a reporting method to report details about the models, at first, we tracked time of training, but it was not important as most models take between 15 minutes and 60 minutes at worst, of course the biggest factor affecting the time complexity of training is number of stages L and feature type, later we focused on comparing the error performance, stability of training and overfitting.

For choosing regularization parameter alpha for HOG feature we can see that lower regularization leads to steeper decrease in error at earlier regressors, however more stable steps at higher regularization were better for the final result as it reduced the overfitting.

Dataset	**alpha=1000**	**alpha=5000**	**alpha=10000**
Helen	6.98	6.58	6.33
LFPW	6.82	6.66	6.63

Table 4.15: Comparing different alphas on HOG stages.

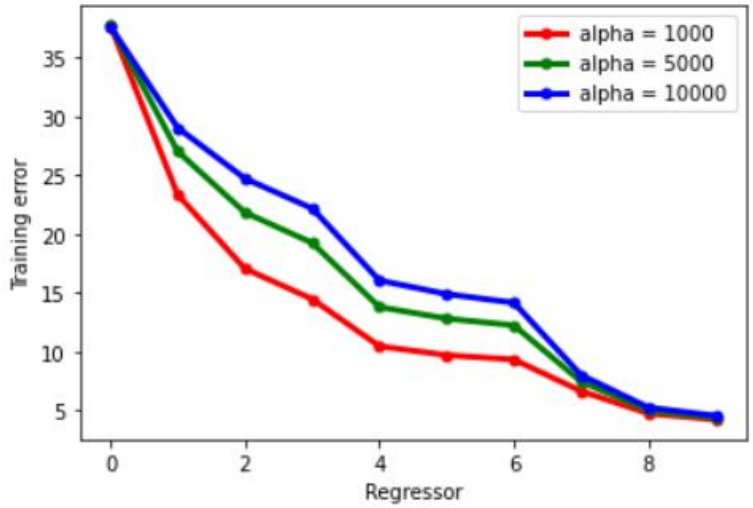

Figure 4.15: Comparing the effect of alphas on training accuracies through regressors, high regularization leads to less overfitting.

Using alpha of hog = 1000 achieved better results with only 2 stages was better than higher alpha but comparing it to alpha=10000 with 3 stages it performed worse, but of course lower stages lead to slightly faster inference.

Dataset	**alpha=1000, L=2**	**alpha=10000, L=3**
Helen	6.94	6.33
LFPW	6.68	6.63

Table 4.16:Comparing low regularization and low number of stages to higher regularization and higher number of stages.

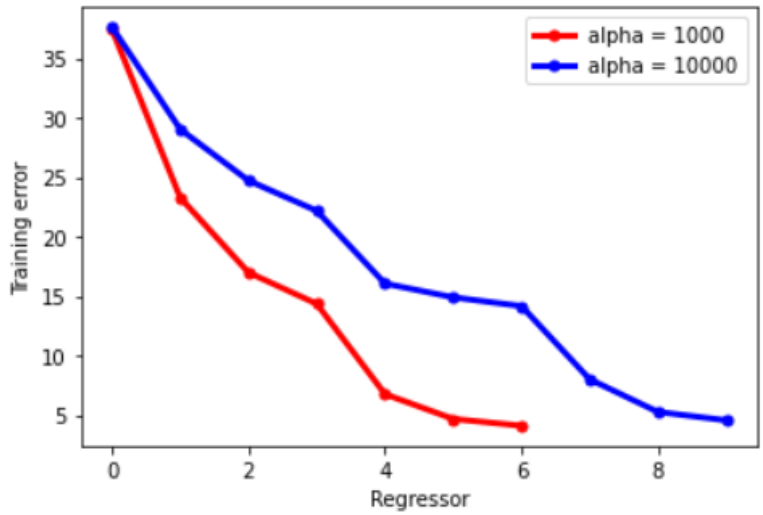

Figure 4.16: Less number of stages and regularization leads to faster convergence but leads to overfitting.

For choosing different alphas for the same feature extractor, we realized that this could improve performance because at earlier stages we need smooth movements, at second stage we can reduce the regularization parameter for more fine movements, so we started at 10000 then 5000 for HOG and finally 40000 with SIFT.

Dataset	10k, 10k, 40k	10k, 5k, 40k	10k, 3k, 40k
Helen	6.41	6.38	6.39
LFPW	6.20	6.15	6.33

Table 4.17: Some trials showing using different alphas for the same feature (hog) at different stages.

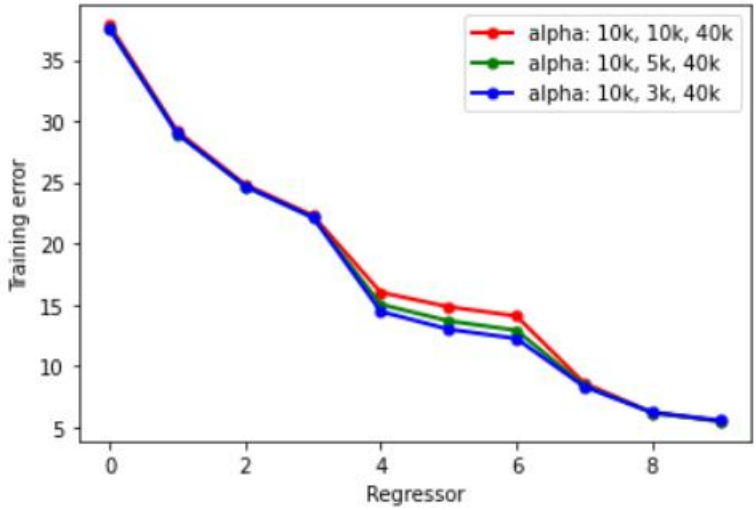

Figure 4.17: Some trials with different alpha at the second stage can affect the final result.

The following table also shows some comparisons of different combinations of features used at training stages, it was noted that ORB and SIFT at early stages tends to overfit even at high regularization parameters, and using HOG at early stages improves training significantly and makes the training more stable.

Feature Combinations	Final training Error	Helen normalized error
SIFT, SIFT	3.06	17.8
ORB, SIFT	3.00	11.59
ORB, ORB	5.61	17.48
HOG, SIFT	4.08	6.94

Table 4.18: Comparisons of different features trials

4.4.4. Design Constraints

One common constraint of 2D facial landmarks detectors is the cropping of the face before being input to the system, it has been noticed that cropping the face in a way that is not in the same standard of training can have a very bad effect on results.

This figure shows how dlib[15] for example performs after using their face detector or cropping the face in another way.

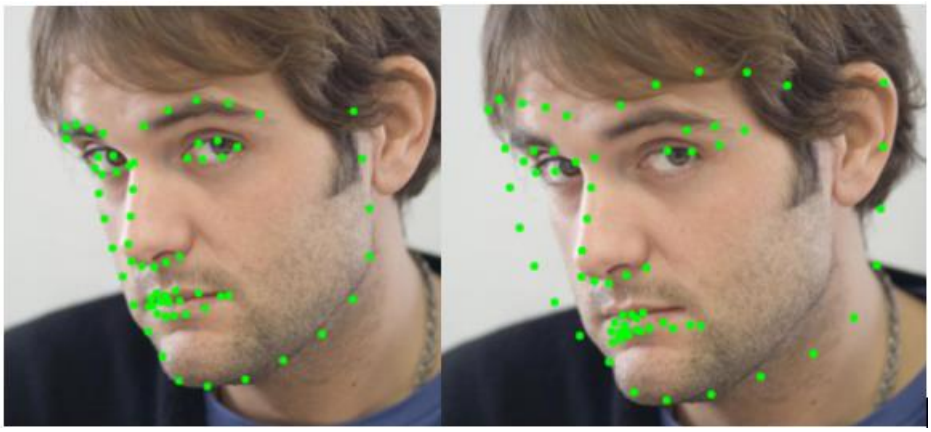

Figure 4.18: Showing the effect of different cropping on dlib landmark detector result, on the left the cropping is by dlib face detection, on the right a slightly different manual cop was made.

This can limit the performance of a Facial Landmarks Detector to how well the face detector is performing, also it is very hard to get the crop standard always because this may depend on the pose of the image, how much area of the face is actually appearing on the image, the resolution and many more.

One way to solve this is by training the model on various crop standards but this can also affect the performance of the model.

Another problem that Facial Landmarks Detectors have is when using Cascaded Regressions methods the result may depend on the initialization of the predicted landmarks, generally the mean shape of the dataset is used as initialization, but the performance can improve if the mean shape is closer to the real face and that is why when the face pose is straight it has better results.

[15] https://pypi.org/project/dlib/

4.5. Dynamic Texture Analysis

4.5.1. Functional Description

The module's main function is to analyze the spatio-temporal texture dynamics of an input video to classify the real or manipulated sequences. This is achieved through using Local Derivative Pattern on Three Orthogonal Planes (LDP-TOP) that has been specifically designed for texture analysis in video sequences. LDP-TOP extends the original LBP technique by considering local derivatives on three orthogonal planes (horizontal, vertical, and diagonal) instead of just a single plane, enabling a more comprehensive characterization of the dynamic texture patterns. In this module, we targeted both spatial and time domain of the video sequence. This yields relatively large feature representations that can be learned through simpler classifiers, such as linear SVMs.

4.5.2. Modular Decomposition

4.5.2.1 Pre-processing

First, video frames are extracted from each video of the dataset and do the following on each frame.

- Face detection: after extracting video frames, dlip library is used to detect the region of interest (ROI). Then we resized the face to (128, 128) and converted to grayscale. After project integration, we used our face detection model instead of python dlip.

- Temporal partition: it is a crucial step in video analysis and processing that involves dividing a video into smaller segments or temporal windows based on time intervals. Once the video frames are converted to grayscale, overlapping temporal windows are isolated. These windows are defined by a duration parameter "d," representing the length of each window in seconds.

- Area selection: At this stage, we introduce the capability to select a specific area of the face for feature analysis. This allows us to determine the importance and relevance of different facial regions in the chosen feature representation. The objective is to understand which regions of the face contribute most significantly to the analysis. In our tests, we consider three distinct cases, denoted by uppercase letters: the top-half (T), the bottom-half (B), or the full-face information (F). These cases refer to the specific region or subset of the face that will be used for subsequent analysis. For the "top half" case, only the upper portion of the face is selected. This includes the forehead and upper part

of the eyes. In the "bottom-half" case, only the lower portion of the face is considered for feature analysis. This involves the nose, mouth, and chin. Finally, the "full face information" case encompasses the entire face region. All facial features and regions are taken into account for feature analysis. By testing and comparing these different cases, we gain insights into the relevance and contribution of specific facial regions to the chosen feature representation.

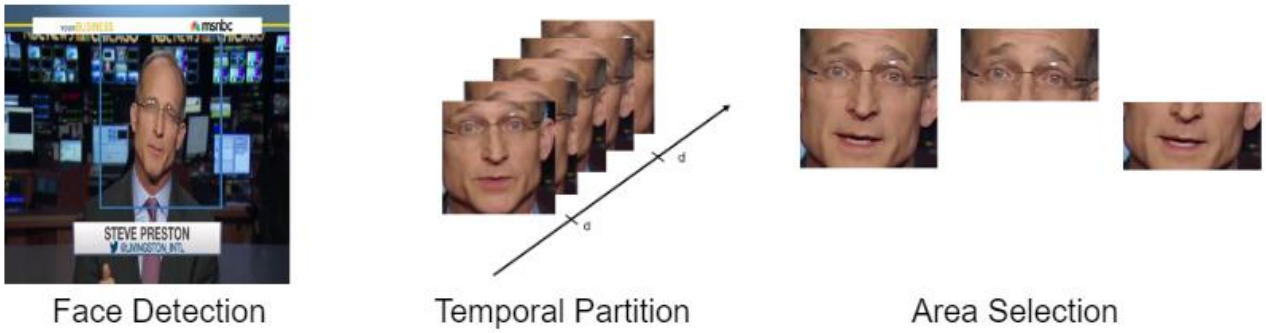

Figure 4.19 pre-processing pipeline

The output of the preprocessing stage for each video is a 4D array with a size of (H, W, K, N). Here, (H, W) represents the height and width of each frame (it could be top-half, bottom-half or full face), which in this case is (128, 128) due to frame resizing. K is the result of multiplying the video's frame rate by the time window (d), which gives the total number of frames within that time window. N is calculated by dividing the total video length by the time window (d), representing the number of video partitions.

4.5.2.2 Feature Extraction

4.5.2.2.1 Local Binary Pattern

Local Binary Pattern (LBP) is a widely used and effective texture descriptor in image processing and computer vision. It is particularly useful for tasks such as texture classification, face recognition, and object detection. LBP encodes the local structure and texture information of an image by comparing the pixel values of a central pixel with its neighboring pixels. The LBP algorithm works as follows: for each pixel in an image, a binary code is assigned based on the relative intensity values of the neighboring pixels. Starting from the central pixel, the neighboring pixels are compared to the central pixel, and if the neighbor's intensity is greater than or equal to the central pixel's intensity, a value of 1 is assigned; otherwise, a value of 0 is assigned. This process is performed for each pixel, resulting in a binary pattern for the entire image.

Figure 4.20 output image of LBP

4.5.2.2.2 Local Derivative Pattern

We aim to exploit both spatial and temporal domains in the analysis of video sequences. To this purpose, we considered the Local Derivative Pattern features (LDP), a feature extraction technique widely used in image processing and computer vision for tasks such as texture analysis, face recognition, and object detection. The LDP, a generalization of the widely used Local Binary Pattern (LBP), is a pointwise operator applied to 2D arrays of pixels that encodes diverse local spatial relationships. As suggested in [24], we considered the second-order directional LDPs with direction α, where α ∈ {0°, 45°, 90°, 135°}. The second-order directional LDPs are an extension of the traditional Local Derivative Pattern (LDP) method that incorporates second-order derivative information and directional filtering for capturing local contrast and texture information in images. The second-order directional LDPs consider both gradient magnitude and gradient orientation in the analysis of image patches. The gradient magnitude measures the rate of change of intensity, while the gradient orientation indicates the direction of the intensity variation. These measures are calculated based on the second-order derivatives.

Given a 2D array of pixels, the LDP^2(α) at the location(h, w) is an 8-bit vector as shown in Table , the LDP^2(α) are extracted for every pixel of the 2D array and their 2^8-bin histogram is computed; this is replicated for the four different directions {0°, 45°, 90°, 135°}, and the histograms are concatenated, so the output feature vector length from 2D array will be $2^8 * 4 = 1024$.

To exploit both spatial and temporal domains in the analysis of video sequences [9], proposes to extend the computation of LDP histograms to 3D arrays. This is done by sequentially considering the three central 2D arrays along each dimension that intersect orthogonally as shown in Figure 4.21 three orthogonal planes and again concatenating the obtained histograms, yielding the so-called LDP-TOP features. The output feature vector length form LDP-TOP will be $2^8 * 4 * 3 = 3072$, where 4 are the directions and 3 are the orthogonal planes.

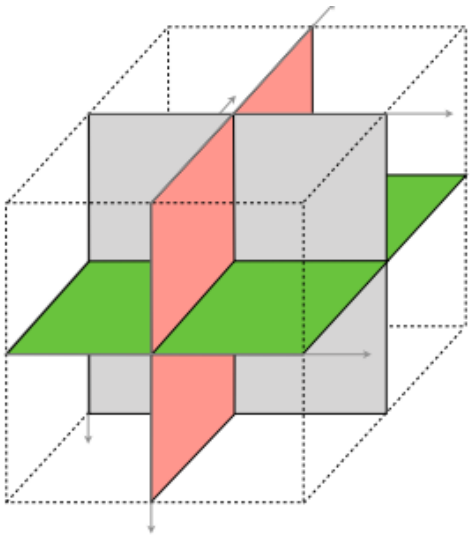

Figure 4.21 three orthogonal planes [9]

h^-, w^-	h^-, w	h^-, w^+
h, w^-	h, w	h, w^+
h^+, w^-	h^+, w	h^+, w^+

Figure 4.22 3×3 neighborhood

$LDP^2(\alpha)$ Algorithm:

```
1.  FUNCTION F(a, b):
2.      IF a * b > 0 THEN RETURN "0"
3.      ELSE IF a * b <= 0 THEN RETURN "1"
4.  # ========== ----- ========== #
5.  FUNCTION I(picture, h, w, theta):
6.      IF theta = 0 THEN RETURN picture[h, w] - picture[h, w + 1]
7.      ELSE IF theta = 45 THEN RETURN picture[h, w] - picture[h - 1, w + 1]
8.      ELSE IF theta = 90 THEN RETURN picture[h, w] - picture[h - 1, w]
9.      ELSE IF theta = 135 THENRETURN picture[h, w] - picture[h - 1, w - 1]
10. # ========== ----- ========== #
17. FUNCTION LDP^2_α(picture, h, w):
18.     angles = [0, 45, 90, 135]
19.     FOR EACH theta IN angles DO
20.         eigth_bit_binary = [F(I(picture, h, w, theta), I(picture, h-1, w-1, theta)), F(I(picture, h, w,
        theta), I(picture, h-1, w, theta)), F(I(picture, h, w, theta), I(picture, h-1, w+1, theta)), F(I(picture,
        h, w, theta), I(picture, h, w+1, theta)), F(I(picture, h, w, theta), I(picture, h+1, w+1, theta)),
        F(I(picture, h, w, theta), I(picture, h+1, w, theta)), F(I(picture, h, w, theta), I(picture, h+1, w-1,
        theta)), F(I(picture, h, w, theta), I(picture, h, w-1, theta))]
```

Table 4.19: LDP^2 (α) Algorithm

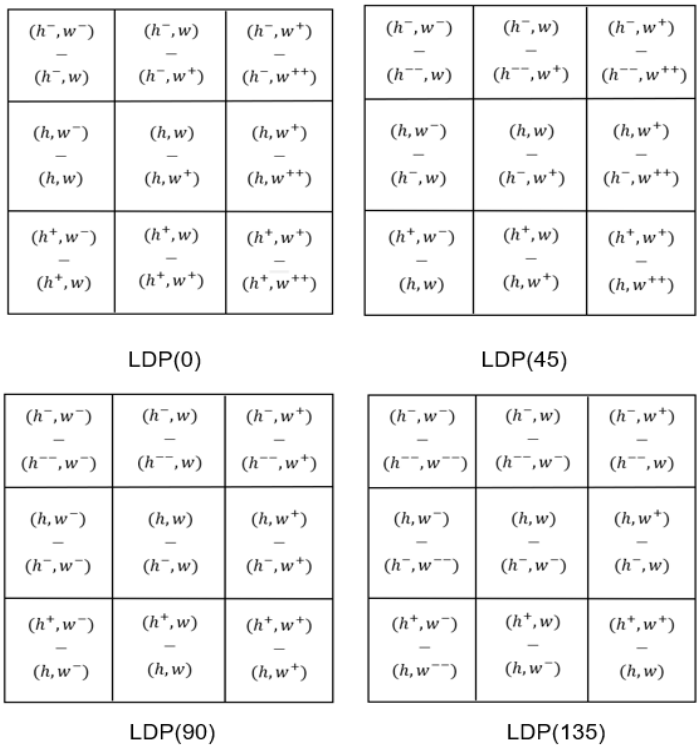

Figure 4.24 The output of the I function in four directions.

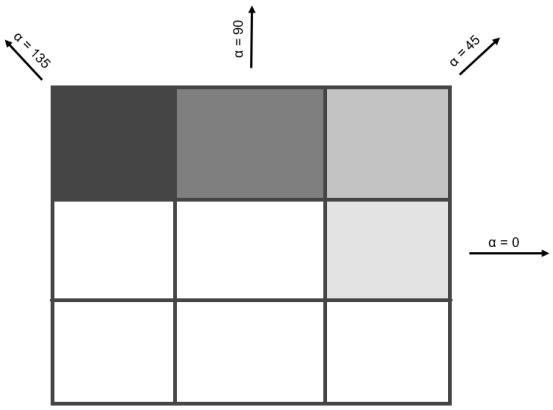

Figure 4.23 α Directions

In order to explore potential peculiarities in the way the temporal information is captured by LDPs, paper proposed to run the feature extraction in three different temporal modes:

- Direct mode (→): S is processed forward along the temporal direction.
- Inverse mode (←): S is processed backward along the temporal direction starting from the last frame and ending with the first frame.
- Bidirectional mode (↔): S is processed in both directions and histograms are concatenated, so the output feature vector length will be doubled.

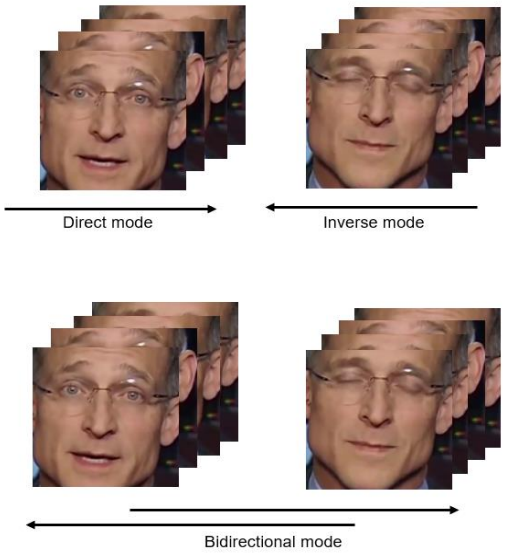

Figure 4.25 Temporal modes

4.5.2.3 Training

After getting the feature vector from LDP-TOP, it's time for training. The feature vectors computed from each temporal sequence, so LDP-TOP may generate four or five feature vectors from a single video based on its length.
The training process involves a set of real and manipulated videos, that we indicate as TRr (labeled as 0) and TRm (labeled as 1), respectively and all of them are used as inputs for training the classifier. The dataset comes with a standard split of videos for training and testing. In order to enable a fair comparison with other recently proposed approaches, we also considered the same training and testing set, yielding the sets TRD with |TRD| = 860 and TSD with |TSD| = 140, where D ∈ {OR, DF, F2F, FSW, NT}. Different subsets will be combined according to the experimental scenario considered.

4.5.2.3.1 Training Model

The suggested classifier in the paper is Support Vector Machines (SVM). Support Vector Machines (SVM) are often used in deepfake detection problems because of their ability to handle high-dimensional data and their effectiveness in separating classes. Here are a few reasons why SVMs are commonly applied in deepfake detection:

- Non-linear classification: SVMs can efficiently classify data that is not easily separable by a linear decision boundary. Deepfake detection involves complex and non-linear patterns, and SVMs can capture these patterns effectively.

- Robustness against overfitting: SVMs have a regularizing parameter that helps prevent overfitting, which is crucial when dealing with limited training data in deepfake detection.

- Ability to handle high-dimensional feature spaces: Deepfake detection often involves analyzing large amounts of image or video data. SVMs can handle high-dimensional feature spaces and are not adversely affected by the "curse of dimensionality" that can impact other algorithms.

- Margin-based learning: SVMs aim to find a decision boundary that maximizes the margin between classes, leading to improved generalization performance. This provides robustness against noise and can help distinguish between real and fake samples in deepfake detection.

- Support for both binary and multiclass classification: Deepfake detection can involve classifying samples into multiple categories. SVMs can be extended to handle multiclass classification problems using techniques such as one-vs-one or one-vs-rest.

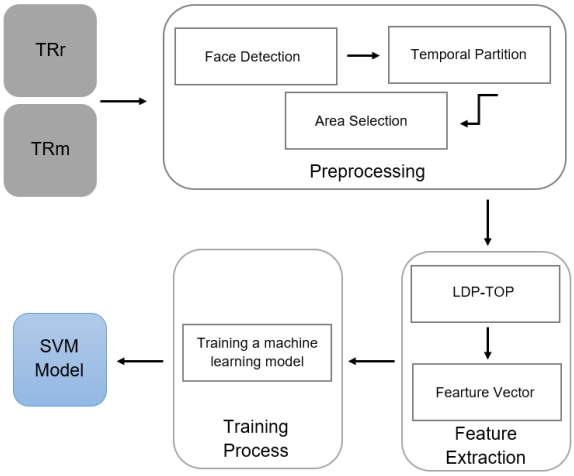

Figure 4.26 Training pipeline

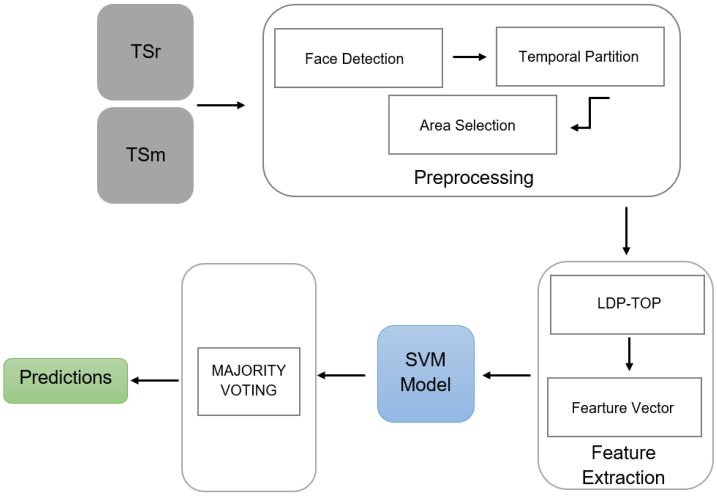

Figure 4.27 Testing Pipeline

4.5.2.3.2 Training Machines

Due to the enormous dataset and the large number of models necessary to cover the analysis in the paper as shown in Table 4.20 training time for each model, training all these models on a single machine was impractical. Fortunately, several cloud providers offer online model training services.

For our project, we chose Azure as our cloud provider. Azure provides a specialized package for students known as the Azure for Students program. This program grants students' access to a wide range of Azure services and resources. We utilized this program to train our models and obtain the necessary analysis. Training machines are found in Table 0.1 Training Machines.

4.5.2.3.3 Training Time

We listed the training time of each model to demonstrate the significant amount of time taken to obtain the required analysis.
The training time of models varies depending on several factors, including the complexity of the model, the size of the dataset used for training and the computational resources available.

Model	Kernal	Dataset Version	Training Data (videos)	Training Time in minutes
Kickoff	RBF	Cf23	500	5172
Deepfakes	RBF	Cf23	1720	3.36
Deepfakes	Linear	Cf23	1720	120.45
Face2Face	RBF	Cf23	1720	100.23
Face2Face	Linear	Cf23	1720	4156.1
FaceSwap	RBF	Cf23	1720	4.13
FaceSwap	Linear	Cf23	1720	201.06
NeuralTextures	RBF	Cf23	1720	12.33
NeuralTextures	Linear	Cf23	1720	6935.11
Multi-Classifier	RBF	Cf23	4300	123.05
Multi-Classifier	Linear	Cf23	4300	11213.51
Deepfakes	RBF	Cf40	1720	100
Deepfakes	Linear	Cf40	1720	10864
Face2Face	RBF	Cf40	1720	180
FaceSwap	RBF	Cf40	1720	7
FaceSwap	Linear	Cf40	1720	7132

Table 4.20 training time for each model

4.5.2.4 Experimental results

The next sections present the experimental tests conducted in order to validate the proposed method in practical scenarios.

4.5.2.4.1 CF23 Experimental results

In this section, models trained on the version of the dataset subject to a light compression (H.264 with constant rate quantization parameter equal to 23). We have tested the feature representation and classification framework in several experimental scenarios, which are described in detail in the next subsections.

4.5.2.4.1.1 Kickoff model

The main purpose of this model is to validate the feature representation and classification framework. This model is exceptional as it has been trained on 500 real videos and 500 manipulated videos (125 videos from each technique). Additionally, we used a list of hyperparameters (as shown in Table 4.21 Kickoff model hyperparameters) in this model to determine the parameters that yield the highest accuracy. We tested the performance of our model on an equal number of training data, consisting of 500 real videos and 500 manipulated videos (125 videos from each technique). The resulting accuracy is not bad but not good, as shown in Table 4.22 Kickoff model accuracy.

The kickoff model approves the following:

- We can rely on LDP-TOP as a feature representation.
- The best hyperparameters are (C=0.1, kernel='rbf', gamma=0.1).
- Face2Face and NeuralTextures are more complex techniques than Deepfakes and FaceSwap.

C	Kernal	Gamma
0.1	Linear	1
1	**Rbf**	**0.1**
5	Poly	0.01
10	Sigmoid	0.001
20		
30		
40		
50		
60		
70		
100		
200		

Table 4.21 Kickoff model hyperparameters

Algorithm Version	Real	Manipulated	Cross-Dataset
(F, →)	85.54%	73.68%	79.87%

Table 4.22 Kickoff model accuracy

4.5.2.4.1.2 Single-technique scenario

In this section, we evaluated the performance of our approach in distinguishing between original videos and videos that have undergone specific manipulation techniques. The primary objective was to showcase the capabilities of each classifier when tested with its corresponding dataset. As a result, we obtained an SVM classifier for each manipulation technique, namely CDF, CF2F, CFSW, and CNT. The videos in these sets were inputted into the training pipeline described in Figure 4.26 Training pipeline.

During this phase, we present the results obtained by utilizing the full-face facial area (F) and the forward temporal mode (→). Additionally, we trained the classifier using two different kernels, namely Linear and RBF. This resulted in a total of 8 classifiers, two for each manipulation technique, allowing us to observe their variations and interactions. Results are depicted as bar plots in Figure 4.28 Classification accuracy per manipulation technique in terms of accuracy and full numerical results are reported in Table 4.23 Classification accuracy on the single-manipulation scenario.

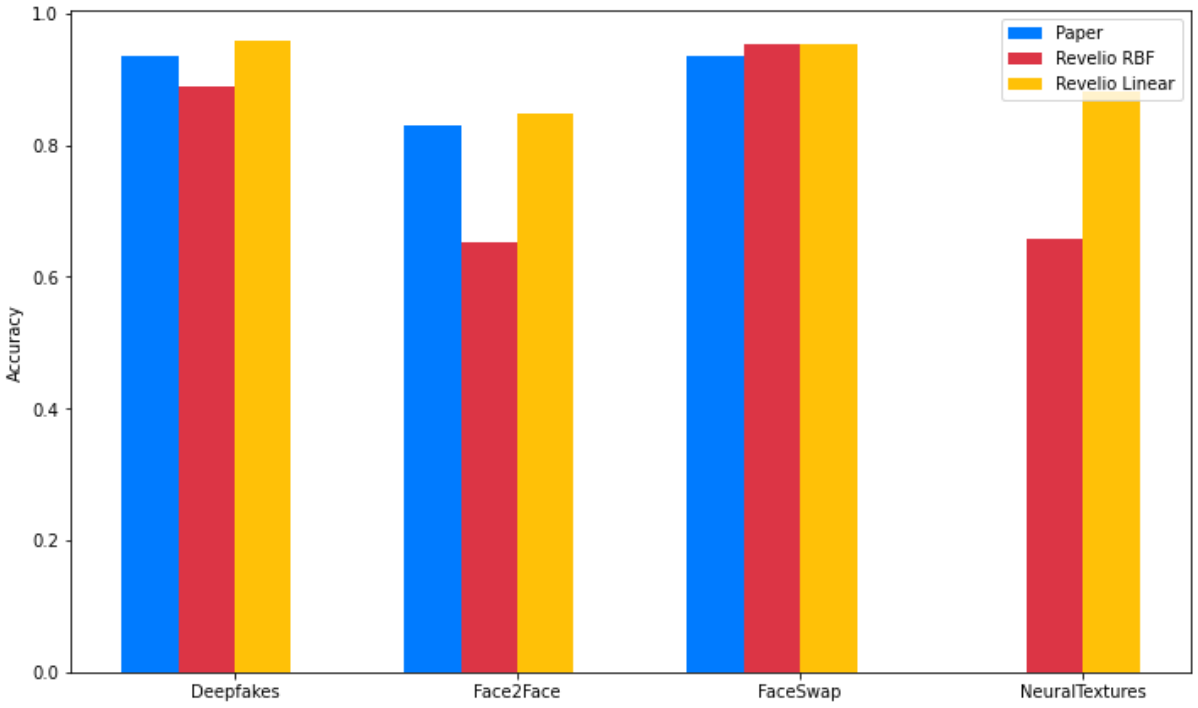

Figure 4.28 Classification accuracy per manipulation technique

Model	Deepfakes	Face2Face	FaceSwap	NeuralTextures	Cross-Dataset
Paper	93.57%	82.86%	93.57%	Not Covered	90.00%
Revelio RBF	88.92%	65.35%	93.35%	65.71%	78.3325%
Revelio Linear	95.71%	84.64%	95.35%	88.21%	90.97%

Table 4.23 Classification accuracy on the single-manipulation scenario

As indicated in Table 4.23 Classification accuracy on the single-manipulation scenario, the accuracies of Face2Face and NeuralTextures are lower than those of Deepfakes and FaceSwap. This implies that Face2Face and NeuralTextures typically result in more challenging manipulations to detect.
Additionally, the accuracy of Revelio RBF is lower than that of both the paper and Revelio Linear, likely due to using the default value of gamma. I believe that performing a grid search to obtain the best hyperparameters would increase the accuracies of Revelio RBF. The primary reason we did not employ grid search is the extended training time required by the models. As indicated in Table 4.20 training time for each model, certain models require significant time for training on the given data, leaving insufficient time for grid search.
In summary, the best results in terms of accuracy are achieved with Revelio Linear, which demonstrates an average accuracy of 90.97% across datasets.

4.5.2.4.1.3 Multiple-technique scenario

We now turn our attention to the scenario where manipulation techniques are combined. Specifically, we consider the more realistic case where the test video can be either real or manipulated using any technique from the dataset.

4.5.2.4.1.3.1 Combining binary classifiers.

The first method to achieve multiple-technique classification involves combining the outcomes of classifiers trained on individual manipulation techniques. This approach also enables us to estimate the specific manipulation technique used in the event of a positive detection.
More specifically, we propose assigning each test video a label $\in \{0, 1\}$ by combining the outputs of the binary classifiers CDF, CF2F, CFSW, and CNT. This results in four predicted labels: p(DF), p(F2F), p(FSW), and p(CNT). A video is classified as manipulated if one of the four detectors returns the label 1. Additionally, in the case of p = 1, the maximum value among the scores is selected as an indicator of the specific manipulation technique used to create the video.
The accuracy of multiple technique classification using binary classifiers can be observed in Table 4.24 The accuracy of multiple technique classification using binary . Additionally, we have reported the false positive rate, which represents the fraction of original videos erroneously classified as manipulated, and the false negative rate, reflecting the fraction of manipulated videos erroneously classified as original in the table.

False Positive Rate	False Negative Rate	Accuracy
27.1%	4.46%	91%

Table 4.24 The accuracy of multiple technique classification using binary classifiers.

4.5.2.4.1.3.2 multi-class classifier

Instead of combining binary classifiers to achieve multiple-technique classification, we will train a single classifier on the entire dataset, which includes real videos and videos with various manipulation techniques. This approach allows us to obtain a multi-class classifier capable of performing multiple-technique classification.

The paper states, 'We have observed that training a single binary classifier on the entire dataset leads to poor results. This can possibly be attributed to the linearity of the classifier used, which appears to be inadequate for accurately separating the two classes through a hyperplane in the feature space. Instead of emphasizing the requirement for a single classifier to achieve perfect separation, we propose combining the outcomes of classifiers trained in individual manipulation techniques.'

The results obtained from the multi-class classifier do not align with those presented in the paper. We achieved an accuracy of approximately 90% using the linear version of the multi-class classifier. Thus, indicating that the classifier is capable of identifying the hyperplane that effectively separates the different manipulation techniques. Results are depicted as bar plots in Figure 4.29 multi-class accuracy in terms of accuracy and full numerical results are reported in Table 4.25 multi-class accuracy.

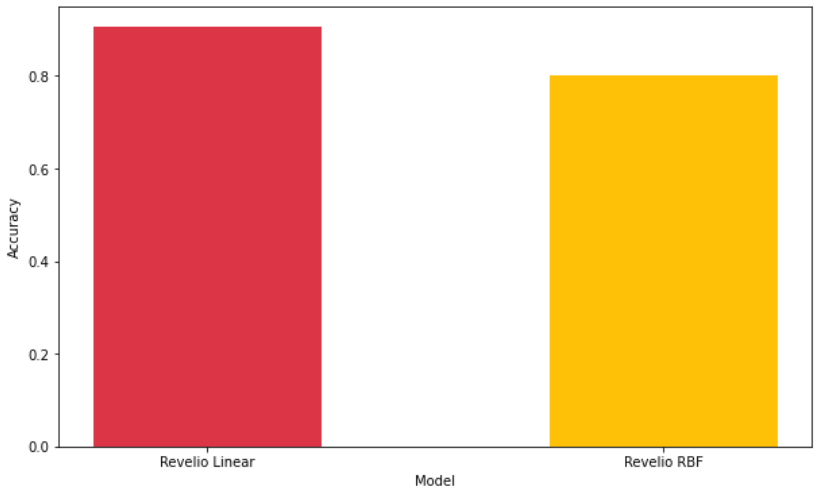

Figure 4.29 multi-class accuracy

Model	Accuracy
Revelio Linear	90.57%
Revelio RBF	80.14%

Table 4.25 multi-class accuracy

4.5.2.4.2 CF40 Experimental results

In this section, models trained on the version of the dataset subject to a high compression with cf = 40. Compressing videos can potentially affect the performance of deepfake detection algorithms.

The artifacts and distortions caused by video compression may impact the performance of deepfake detection algorithms. Deepfake detection methods often rely on analyzing patterns, inconsistencies, or artifacts that are indicative of manipulation. If the compression process alters or removes these patterns or introduces additional artifacts, it can make the detection task more challenging or less accurate.

To get the analyses of single technique scenario, we trained the classifiers on dataset with cf = 40 using two different kernels, namely Linear and RBF. This resulted in 5 classifiers, two for each manipulation technique. Results are depicted as bar plots in Figure 4.30 Classification accuracy per manipulation technique in case of strong video compression in terms of accuracy and full numerical results are reported in Table 4.26

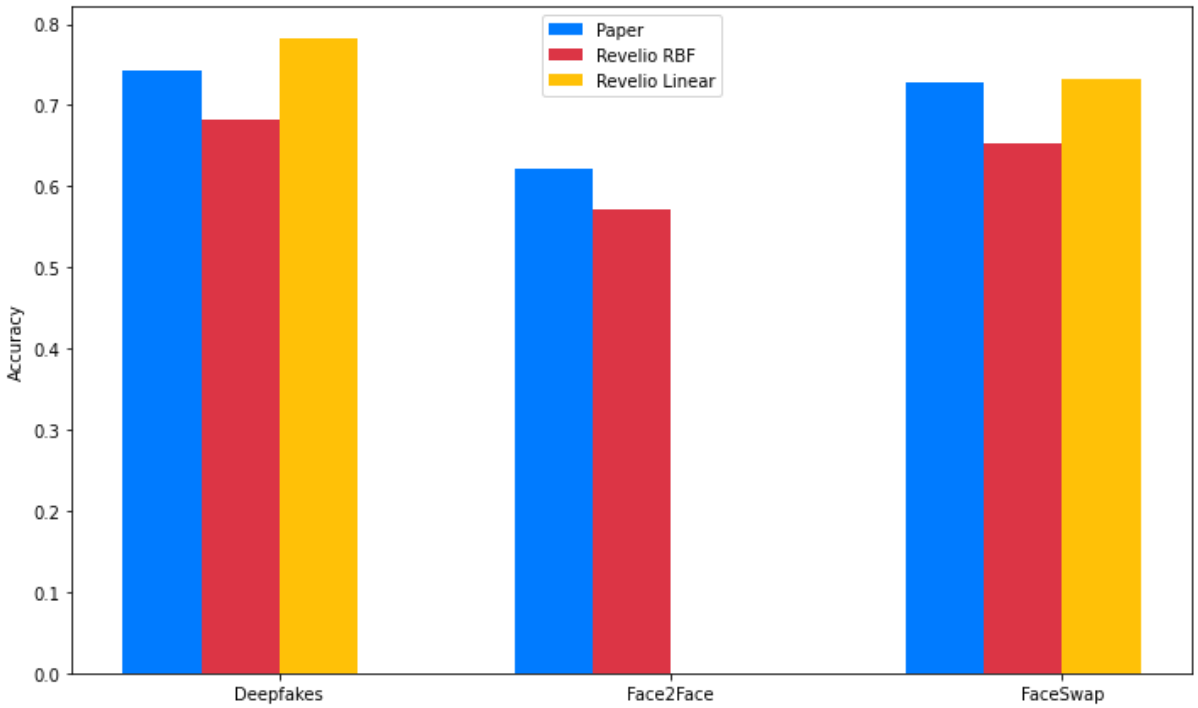

Figure 4.30 Classification accuracy per manipulation technique in case of strong video compression

Classification accuracy on the single-manipulation scenario in case of strong video compression.

Model	Deepfakes	Face2Face	FaceSwap	Cross-Dataset
Paper	74,29%	62,14%	72,86%	69,76%
Revelio RBF	68.21%	57.14%	65.35%	63.56%
Revelio Linear	78.21%	Not Covered	73.21%	75.71%

Table 4.26 Classification accuracy on the single-manipulation scenario in case of strong video compression

To get the analysis of multiple technique classification, we combined the output of binary classifiers that trained on dataset with cf = 40. The accuracy of multiple technique classification using binary classifiers can be observed in Table 4.27 The accuracy of multiple technique classification using binary classifiers in case of strong video .

False Positive Rate	False Negative Rate	Accuracy
34.57%	15.35%	75.23%

Table 4.27 The accuracy of multiple technique classification using binary classifiers in case of strong video compression.

4.5.3. Design Constraints

One of the primary constraints for deepfake detection models is video compression, which can potentially impact the performance of deepfake detection algorithms in the following ways:

- Loss of visual quality: Video compression algorithms aim to reduce the file size by removing or reducing redundant information in the video frames. As a result, the compressed video may exhibit lower quality, introducing compression artifacts and distortions. These artifacts can obscure or alter subtle manipulation indicators, making it more challenging for deepfake detection models to identify the presence of manipulation.

- Increased noise and artifacts: Compression techniques can introduce additional noise and video artifacts, such as blocking artifacts, banding, or color shifts. These artifacts can potentially interfere with the detection model's ability to accurately analyze and classify video frames, reducing the effectiveness of the detection process.

Figure 4.31 same frame with cf=23 & cf=40

As seen in section 4.5.2.4.2, when using a dataset with cf=40 instead of cf=23, the accuracy of the single technique scenario cross-dataset decreased from 90.97% to 75.71%, and the accuracy of the multiple technique classification decreased from 91% to 75.23%. Thus, the effect of video compression on system accuracy is evident.

The second constraint of deepfake detection is poor generalization across different manipulation techniques. Deepfake detection models need to be capable of detecting various types of deepfake manipulations, but they are usually only able to detect the manipulation techniques they were trained on. If we test the model with a technique that it was not trained on, I believe the model's accuracy will not be as good as its accuracy on trained techniques.

4.6. Frequency Domain Analysis

4.6.1. Overview

In recent publications that introduced GANs(Generative Adversarial Networks) or VAEs(Variational Autoencoders) all of these are machine learning models used to generate imagery images given a random vector or some information describe the needed image, going with GANs, it's divided into two adversarial networks Generator and Discriminator, as known that the generator's main functionality is to generate the images from low dimensionality vector which is known as up-sampling or up-convolution which causes the main problem, as up-sampling stage failed to reproduce the frequency domain like real images have.
As this method depends entirely on frequency domain, we're going to use Discrete Fourier Transform method for image processing to extract it, and in turns, this will be processed also to extract the total signals power but by another way called azimuthal integration, and this considered our main feature vector for inference process.

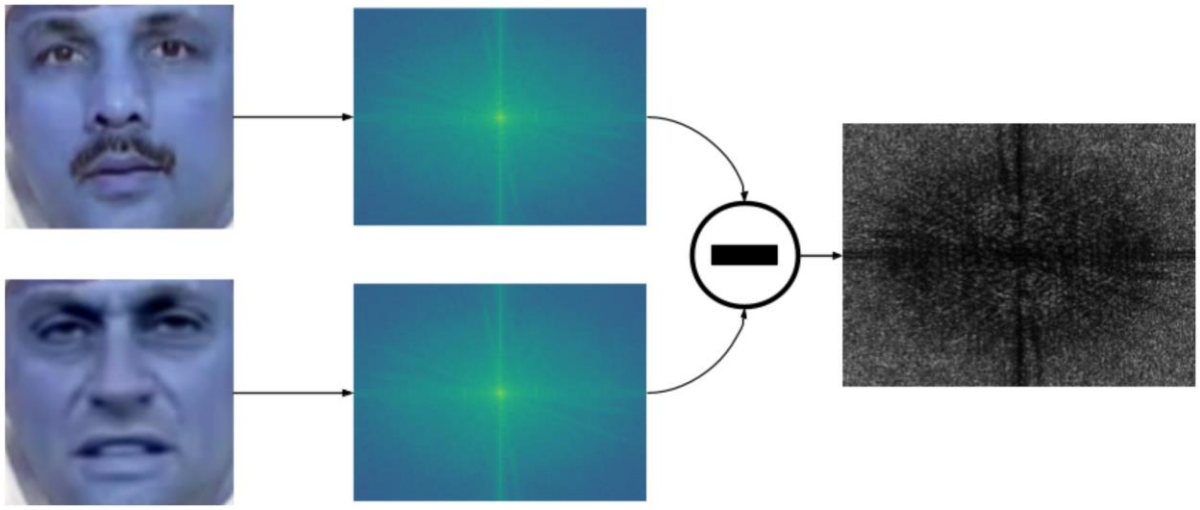

Figure 4.32: Figure shows the difference between two frequency domains one for real image(top) and fake one(bottom) and difference at the right.

From the above figure it shows the results of two images one is real and the other is fake of the same image, at the middle are the frequency domain analysis for both, which not obvious that there's a difference between them with our naked eye.
By taking the difference shown on the right, we notice that there's a difference in the set of frequencies especially at the higher one although there's a difference too at the middle, but this is due to different aspects of faces.
But our main concern is the first which generation failed to reproduce mainly.
In the following sections the approach will be explained in detail.

4.6.2. Functional Description

In this module we aim to extract what is called the azimuthal integration over the frequency domain for the target video, to be used as a feature vector for whether it's fake or not.
As shown in the figure below at the module pipeline:

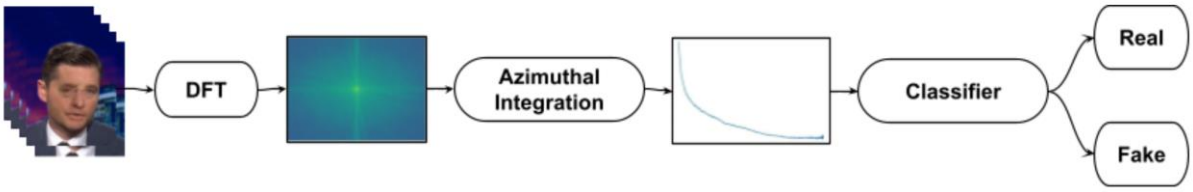

Figure 4.33: Frequency Domain Module Pipeline

we cut out our input video into frames and should be a well-detected face using a face recognition system, as any additional material such as a little observed background will affect our results, not just background, but unfortunately any slight deviation out of constraints described in section 4.6.4. will lead to inaccurate results. then using DFT submodule extract the frequency domain for each video frame, and use them as input to Azimuthal integration submodule, at this point we must take the average for all azimuthal lines we got from frames, that result for just one vector for each video to train on.
Finally, use this feature vector after some modifications as an input to our classifier.

4.6.3. Modular Decomposition

At this section we'll divide our module into submodules:

- Preprocessing
- DFT
- Azimuthal Integration
- Data Visualization
- Classification

Note: for the following modules the dotted lines indicate that these modules applied to each frame within the video and the results will be kept for the next stage.

4.6.3.1 Preprocessing

At this step the video will be divided into frames, for our dataset the videos are at 30 fps at average so 25 frames will be taken from each video just for feature extraction convenience.

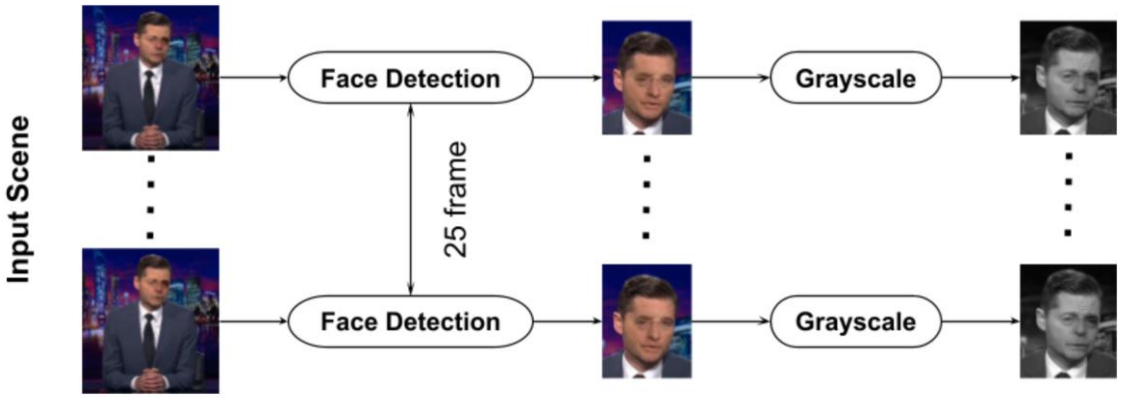

Figure 4.34: Frequency Domain Preprocessing Pipeline

As shown in the figure above the video frames forms scenes will be passed into face detection system to extract just the face we will working with, try to get slightly little background as much as we can depending on the detection system we have, as much facial image we have, the more accuracy we will gain as it consists of frequency domain for the target face only, as more background we have, the more of unrelated set of frequencies we will have, as each frame of scene definitely has different background, so let us concentrate on only the main face frequencies and finally will be converted into grayscale.

4.6.3.2 DFT

DFT[16] is an important part in this chain of our modules, DFT is responsible for converting our grayscale image into its finest first frequency components that construct the image, following up with the following equation used for conversion.

$$F(I)(k,l) = \sum_{m=1}^{M-1}\sum_{n=1}^{N-1} e^{-2\pi i.\frac{jk}{M}}\, e^{-2\pi i.\frac{jl}{N}}\, I(m,n),$$
$$for\ k = 0,\ldots,M-1, l = 0,\ldots,N-1$$

First, we have an image of size *M x N. By* applying the previous equation, we will have a matrix of the same size as that of the image and each element of that matrix will contain a complex value representing the frequency component at that pixel.

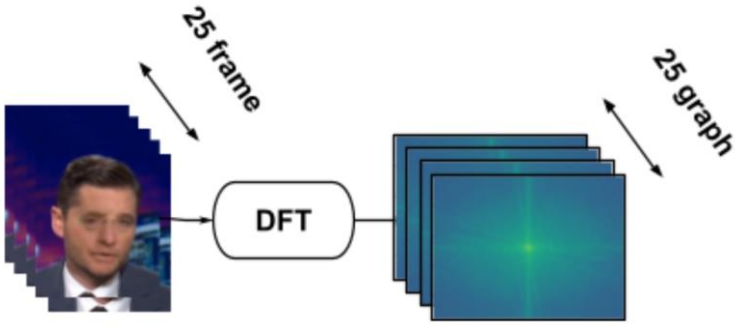

Figure 4.35: Frequency Domain DFT Module 25 frames into DFT and 25 graphs out

then, by taking the magnitude of each complex value,

$$Mag(k,l) = \left||F(I)(k,l)|\right|$$

because of magnitude step, it will produce a much higher values, in addition to much lower ones producing high difference as shown in following figure,

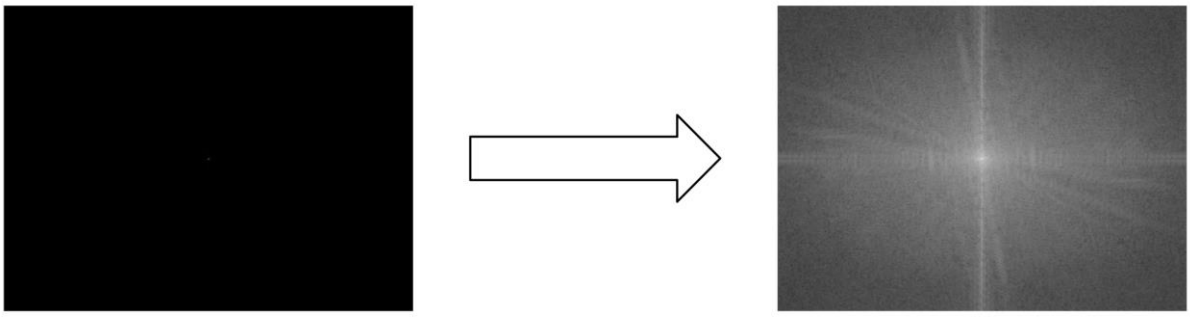

Figure 4.36: The effect of log scaling on the frequency domain, the image on left without scaling and right after scaling.

[16] https://en.wikipedia.org/wiki/Discrete_Fourier_transform

so, we should have normalization step using log scale,

$$F(I)(k,l) = 20 * log\,(Mag(k,l) + 1)$$

From the previous equation we're able to make our frequency values more reliable and convenient for the next step.

4.6.3.3 Azimuthal Integration

Azimuthal integration is the final step at frequency domain analysis, the purpose of this one is to calculate the radial power along the spatial frequency axis resulting from the previous step.

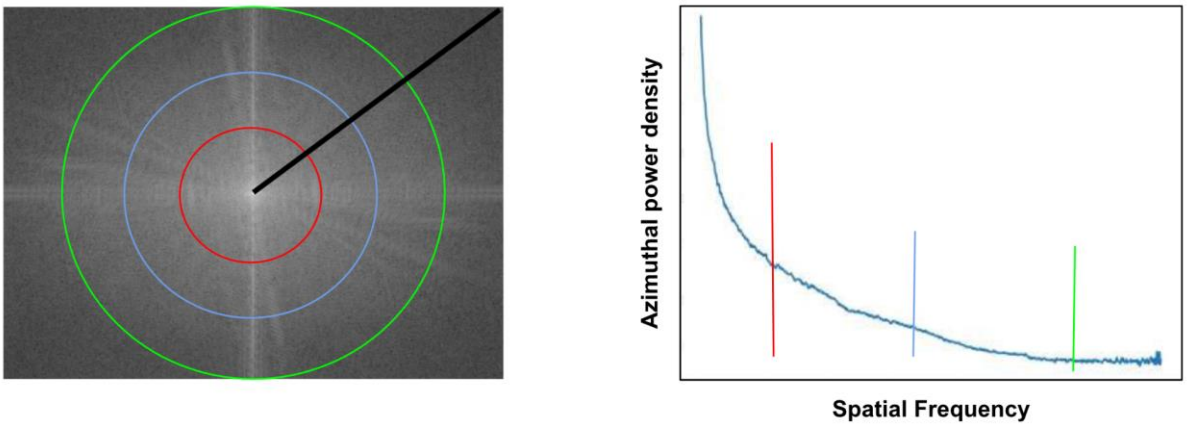

Figure 4.37: Azimuthal Integration process

from the previous figure that shows the process of the integration, at the right figure, it's shown that at each point in the spatial frequency represented by the black line at the left figure, the total power density is calculated at this point due to this equation,

$$AI(\omega_k) = \int_0^{2\pi} Mag(F(\omega_k.cos(\phi)\,, \omega_k.sin(\phi)))^2\, d\phi$$
$$for\ k = 0,\ldots,M/2 - 1$$

Where ω_k is the radius from the center to the target frequency, ϕ is the radial angle from x-axis to the radius vector, then by taking the square of the magnitude of the current pixel we now calculate the power element at this point, then completing the entire radial integration to get the total power at this frequency.
as shown in the previous figure, the red circle represents the total points must be summed at this frequency, and that's called azimuthal integration as we integrate over the $2\,\pi$.
The following part explains the previous process in detail.

4.6.3.3.1 Integration part

This part discusses the integration process in detail, as known from the previous works as azimuthal integration is the summation of the power by a discrete radius, as shown in the following figure:

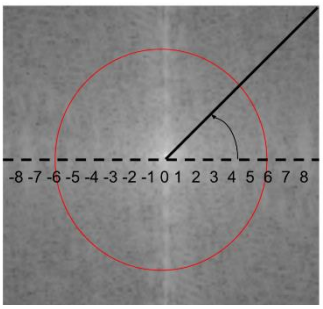

Figure 4.38: The process of integration of the radial power

The current radius is 6 so the summation will go from 0 to 2π to sum all the values lie on the current circle's circumference.
So, this process is being done for all the points with the same distance from the center, so we say explicitly it's a circle.

but there is a slight problem with the previous way, as the current radius from the center is multiplied by $cos(\phi)$, $sin(\phi)$ then cast them into integers to get the pixel location, it may cause one pixel to be added up to different radiuses as the two pixels definitely have different radiuses but due to the rounding part, they end up to the same pixels.

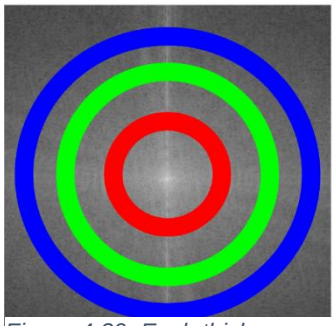

Figure 4.39: Each thick circumference represents bin size for a punch of equidistant pixels.

So, we solve this by ranking each pixel by its radius then summing all subsets of pixels that have the same radius, as shown in the following figure the thick circle represents the range of equidistance pixels, so they are considered to be in one group.

4.6.3.3.2 Power density part

After calculating the power at each radius, first it takes a big part of memory so considering an optimization step which calculates the power density for each collection of equidistant points.

as we have at each radius a punch of points, so after the summation we divide by the total points within this radius to get the power density.

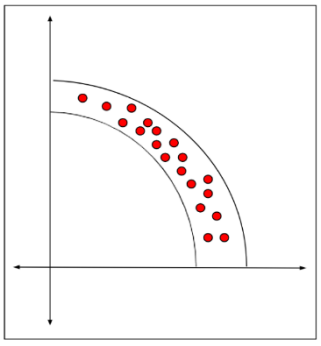

Figure 4.40: Example of the number of pixels that have the same radius and are used to calculate the power density within each bin.

4.6.3.3.3 Interpolation part

At this part we're going to standardize the azimuthal feature vector we have, due to the variety of our input frames whether in size or resolution, they produce different frequency image sizes, thus different feature vector size so we must use fix size 300 and this one is flexible may be change w.r.t the running environment, and absolutely preferred the fixed size to be more bigger than the actual image vector size, so as not to lose any information.
And this step is done using linear line interpolation equations.

$$y = y_1 + (x - x_1)\frac{(y_2 - y_1)}{(x_2 - x_1)}$$

At the previous equation, it's used to embed new value by taking the average of two existing ones, and so on till the complete interpolation.
Then the final part is to take all the feature vectors for all the video frames and average them all.
In turns, at the previous preprocessing part the modules cut the video into just 25 frames in order to take the most common features for those frames, as the more frames we take the more information we lost for this video due to the averaging process.

4.6.3.4 Data Visualization

At this part the features extracted visualization will be discussed, so to get better understanding of the nature of the features we have.
First our system trained on 4 main sub datasets, Deepfakes, Face2Face, FaceSwap, and NeuralTextures.

All the following graphs show each dataset vs the original one, by considering the mean feature vector for the original one with each mean feature vector of the datasets.

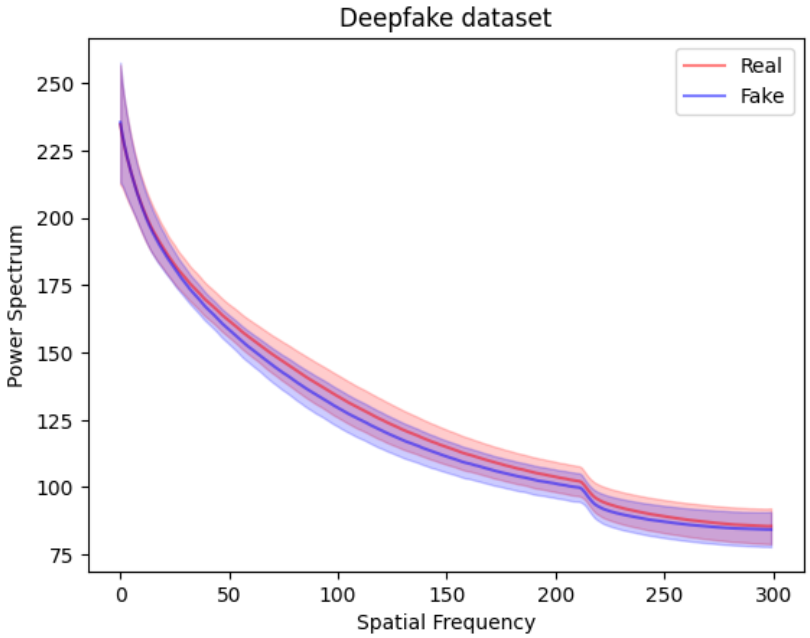

Figure 4.41: Example shows the mean and std of real data vectors and the fake one at Deepfake Dataset

At the previous figure the range or the variance of each dataset is represented by the shaded area around the mean, as it's obvious there's a little difference in the middle area from 50 to 200 for this dataset.

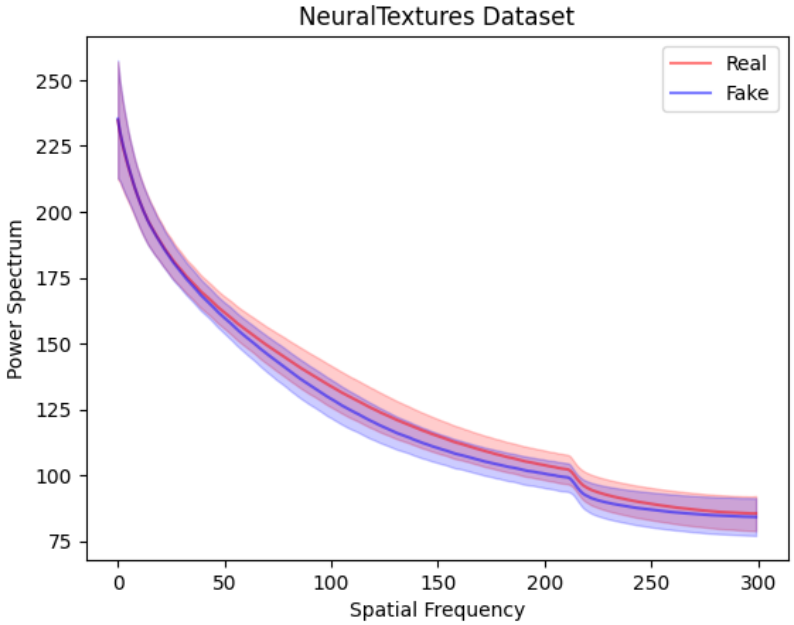

Figure 4.42: Example shows the mean and std of real data vectors and the fake one at Neural Textures Dataset

This figure is too similar to the Deepfake dataset, due to the quality of producing the video, which makes it a little harder to detect this kind of video using this technique.

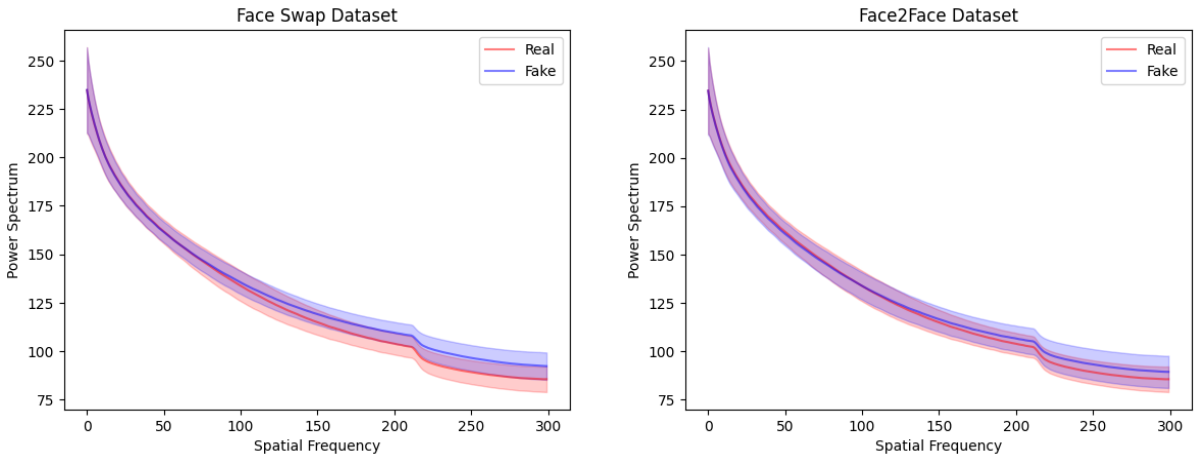

Figure 4.43: Example shows the mean and std of real data vectors and the fake one at Face Swap Dataset [left] and Face2Face [right]

At the previous figures, Face Swap and Face2Face may be considered too similar specifically at the last part from 150 to 300, so at the training, they are reliable for this method.

4.6.3.5 Classification

At this step we trained each dataset with the original one using SVM classifiers with rbf kernel for all of them and the steps go as following:

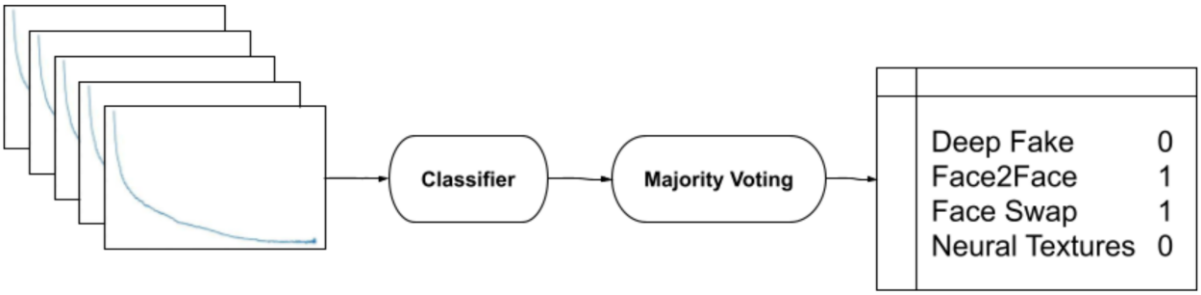

Figure 4.44: This Example shows the Classification Pipeline

The above figure shows the pipeline of the classification process as it goes as following:

- The video is cropped based on a face detection system, so we now have a video to face only.
- The video is then divided into separate frames.
- Extract the feature vector for each frame, then we have a list of features of size **num_of_frames * feature_vector_size.**
- All feature vectors are then predicted using the four trained models, so we have for each model a list of predicted class for each frame.
- Take the majority vote for each model over all the frames to see if the full video is fake or not.
- As shown in the last diagram of figure, each model has a probability that this video is fake and belongs to this manipulation method.

The following table shows the resulting accuracies for each model, on data splitted 80%, 20% for training and testing.

Table 4.28: SVM accuracies for Frequency Domain Analysis

Dataset	Accuracy
Deep Fake	68.17%
Face Swap	86.25%
Face2Face	75.75%
Neural Textures	69.50%

4.6.4. Design Constraints

For the model to be reliable, it should follow some constraints as following:

- For the complex scenes the face must be detected first, as the approach won't work well at the full scene, so try to cut the face part with a little background as much as you can.
- Don't make any resizing or resampling for the image, as it distorts the frequency domain, you can do it for the frequency domain itself, note as many face detectors try to resize the image try to avoid them.
- Use Square image as much as you can as non-square images distort the process.
- Unfortunately, this approach won't work well if the video is compressed to a large extent. Try to use the raw data as much as you can.

4.7. Lips Movement Analysis

Despite the huge progress of various face manipulation techniques, most generated videos have slightly notable artifact patterns that can be found in Lips Movements. These artifacts can be noticed in the abnormal frequency in movement of lips, not closing the lips well or artifacts in teeth, etc.
The quick movements of lips between each frame makes the job of generating fake content harder, but using advanced Machine Learning techniques we can detect the abnormalities by analyzing the lips movements, this was first suggested by the LipsForensics [8] method.

4.7.1. Functional Description

The module takes as an input the video frames cropped around the face using the face detector module, along with the facial landmarks in every frame detected by the landmarks detector module, the module uses the landmarks to crop a defined region around the mouth for each frame and uses these sequence of mouth region frames to analyze it using a pre-trained deep learning model that is finetuned to detect the abnormalities in mouth movements and finally classify the video whether it is real or fake.

The original model was pre-trained on the task of lipreading [25] which makes it so powerful in describing and detecting precise mouth and lips movements, and this gives it a huge advantage in detecting abnormalities in the lips movements.

4.7.2. Modular Decomposition

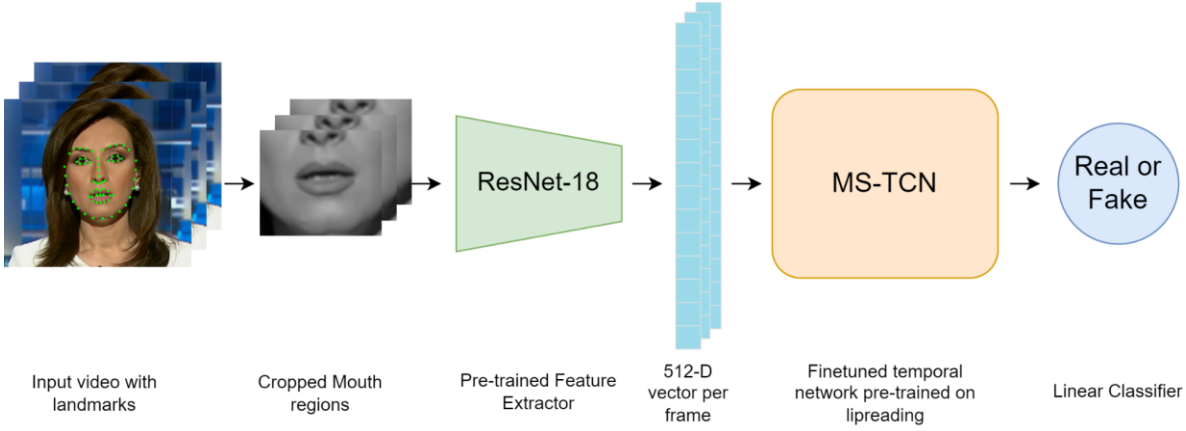

Figure 4.45: Pipeline of Lips Movements Analysis method, starting from the original input video with corresponding facial landmarks, mouth regions are cropped and fed into ResNet-18 feature extractor then MS-TCN followed by a linear classifier for final output.

The module can be separated into three main sub-modules, which are pre-processing, feature extraction and the classification model.

As mentioned before, we use a model that was pre-trained on lipreading, and that makes it sensitive to lips movements, we build on this fact to detect artifacts in lips movements.

We freeze the first part of the model which is the ResNet-18 model and use it as a feature extractor and fine-tune the multi-scale temporal convolutional network (MS-TCN) part to output the classification of the video whether it is real or fake.

4.7.2.1 Pre-processing

The first stage in the pipeline is to align the mouth regions in the frames of the input video using the facial landmarks.
To standardize the cropping of the mouth regions, the region of interest is defined as a square; its center is the mean point of the mouth landmarks, its size is twice the width of the mouth.
The cropped frames are then transformed to grayscale then resized to 96x96 then centrally cropped again to size 88x88 which is the input frame size used by the original pre-trained ResNet-18 model for feature extraction.

For training this step was very computationally expensive as we worked on 5000 videos each containing at least 300 frames, we needed to detect the face then detect the facial landmarks for each frame.
To reduce the computations, we only detected the face regions every 5 frames using the fact that face movements are not likely to be that quick while we had to detect the landmarks every frame as our module depends on analyzing lips movements so we had to make sure they are well aligned as mouth movements can be very quick.

Preprocessing the whole training set took more than 15 hours while multiprocessing on 8 CPU cores.

4.7.2.2 Feature Extraction

The original pre-trained deep learning model pipeline consisted of two parts which are the ResNet-18 model and the MS-TCN temporal network model. For our task we separated the two parts where the ResNet-18 part is freezed and used only as a feature extractor, while the MS-TCN is then fine-tuned to produce a binary classification instead of a vector that represents the predicted lipreading.

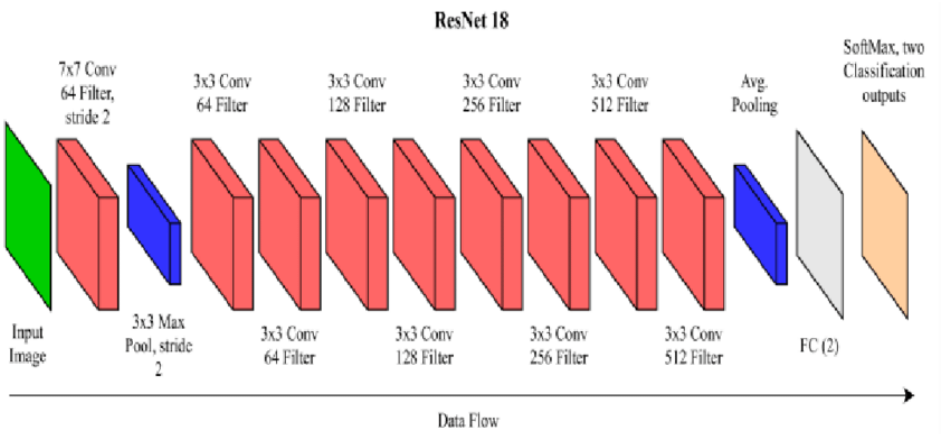

Figure 4.46: ResNet-18 original architecture [26], small changes were made for the task of feature extracting for Lip-reading task.

A standard ResNet-18 architecture is used except that the first convolution layer is replaced with a 3D convolution of kernel 5x7x7, and a global spatial average pooling is applied after the last convolution.

The input to the ResNet-18 Feature Extractor is BxTxWxH, where B is the batch size, T is the time steps or number of frames. In practice T is 25 while the frame size is 25x25.
The model output is BxTxC, where C represents the feature vector for each frame, in our case it is 512.

4.7.2.3 MS-TCN

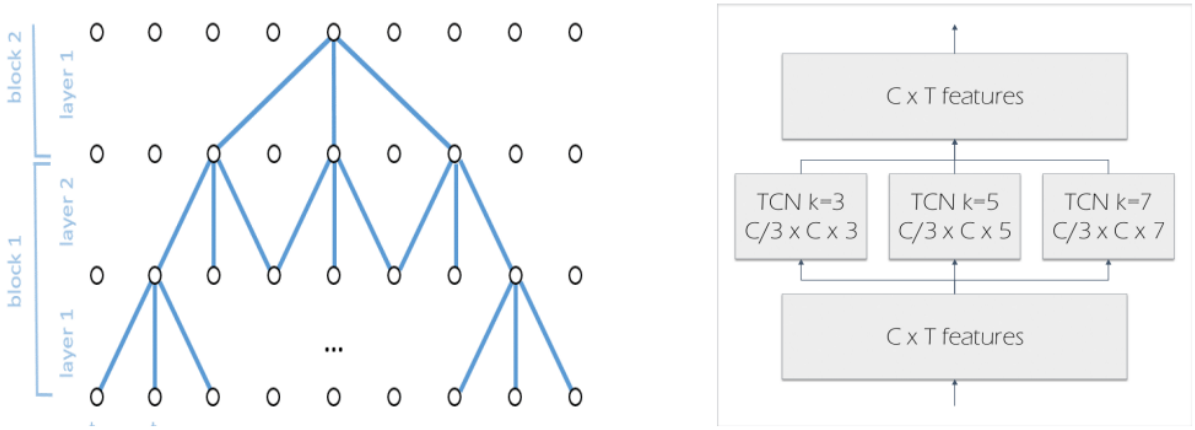

Figure 4.47: Representations of MS-TCN used for Lip-reading [25], on the left is a TCN representation where causality feature is not used, on the right it shows using MS-TCN with different kernel sizes.

As mentioned before the temporal convolutional network MS-TCN is used as a classifier by fine-tuning it, instead of originally output a feature vector representing the text predicted, the final linear layer was changed to output only a single value then sigmoid was used to output binary probability.

Temporal Convolutional networks have been widely used in many applications due to having advantages over LSTM and RNN including having stable gradient propagation compared to LSTM that can have the exploding or vanishing gradient problem, being computationally efficient allowing more parallel execution and more.

TCN allows causal analyses such that each input depends only on the previous timesteps information, however, the proposed MS-TCN architecture used for the original Lip-reading task uses a non-causal implementation of TCN, as the whole sequence is classified at once as shown in the figure.

One of the main advantages of TCN is the ability to extract information from long sequences by increasing the receptive field with low number of weights and computations, and this is allowed by using dilated convolutions, in the original architecture for each block i, a stride of 2^{i-1} is used.

One of the main features of the architecture is that it combines both short-term and long-term information using a multi-scale temporal convolutional network, where multiple kernel sizes are used on multiple temporal convolutions as shown in the figure then concatenate the outputs of each temporal convolution.

4.7.3. Design Constraints

As noticed that method's performance can be affected greatly by the face and landmarks detection models performance. which in turn can be affected by the resolution and the compression of the videos, so the error can propagate from the landmark detectors of the model.
Another constraint is that the model focuses only on the mouth region, and in case there were clearer artifacts in different regions it will go unnoticed, however as discussed earlier most fake videos still have a problem in perfecting the mouth movements to resemble a real one.
However, we have tested our model on a medium level of compression and showed very good results that will be discussed in the following section.

4.7.4. Training and Results

The dataset we used for training was the FaceForensics++ [1] with version c23 which was described in section 3.2, We trained four models on the four types of manipulations versus the original videos, for inference the video will be tested on all

of them, and thus provide informative analysis on which manipulation method may have been used on the video.
The videos were split into sequences of 25 frames each, for inference a voting system between all sequences for each video is used. In total, each model was trained on more than 10000 original sequences and 10000 fake sequences, where each sequence is of size 25x512.

All four models were trained on the following hyperparameters.
epochs = 10
lr = $2x10^{-4}$ (Except DF and F2F $5x10^{-4}$)
batch size = 32
The following results show our model's performance and comparisons to some of the leading models on FF++ dataset with HQ level (c23) where the last 140 videos for each category are used for the test set.

We use both Accuracy and AUC metric for comparison and use the Video-Level metrics where the video is split into sequences and a voting on all the sequences are taken to the final result on the video.

Average training time of each model took between 15 and 20 minutes on 8GB GPU.

Metric	**DF (HQ)**		**FS (HQ)**		**NT (HQ)**		**F2F (HQ)**		**Average**
Accuracy % (Video-level)	92.5		99.3		92.1		97.9		95.45
AUC % (Video-level)	97.7		100		98.3		99.7		98.9
Confusion Matrix (Video-level)	124	16	139	1	131	9	139	1	
	5	135	1	139	13	127	5	136	

Table 4.29: Results of the model trained on the four manipulation methods of FF++, showing accuracy, AUC and Confusion Matrix (where first row represents fake videos) all calculated on Video-Level.

Method	**Accuracy %**	**AUC %**
Xception [1]	97	99.3
CNN-aug [5]	96.9	99.1

Revelio	**95.45**	**98.9**
Patch-based [6]	92.6	97.2
Face X-ray [27]	78.4	97.8

Table 4.30: Comparison of our trained models with various Deep Learning based methods, the results are the average of the four manipulation methods of FF++ all with compression c23.

The table shows some comparisons regarding hyperparameters and choosing the learning rate using final video-level accuracy for comparison.

Learning rate	**DF**	**FS**	**NT**	**F2F**
1e-4	89.6	97.1	88.2	97.1
2e-4	88.2	**99.3**	**92.1**	97.5
5e-4	**92.5**	98.2	92.0	**97.9**
1e-5	85.7	95.7	87.9	91.1

Table 4.31: Comparisons of Learning rates trials used on the four models.

4.8. Self-Blended Images

4.8.1. Functional Description

This is a module for deep fake detection that creates synthetic data by repeating present artifacts in fake videos. Then use deep learning to learn how to discriminate between the two classes. The idea is that deepfakes are created by source and target images, where there is the original face, and then fake expressions are added to it producing artifacts from the differences between source and target images.

Examples of these artifacts are differences in statistical measures in both images, like one image is brighter than the other, or there is a mismatch in colors of source and target images. There can also be a mismatch in the sizes of the two faces.

The method of focusing on artifacts between source and target was introduced in [28], where they find two similar faces from the dataset (similar landmarks) and blend them to create a new fake sample from two original samples then feed this to a classifier. But this seemed to be overfitting on the dataset used for training as it was in danger of being specific to the choice of the source and target images.

This method is present in [7], where instead of blending two different images, it blends the same image with a modified version of itself, where random transformations are applied to the modified version. Transformations used are random shifts in color, size, and brightness to create artifacts between source and target images. The method operates on frame-level rather than video level. So, it deals with each video frame not with how frames evolve.

As it does only use original videos to create fake samples, it does not train on fake samples created by specific manipulation techniques. And thus, it is robust to these techniques and does not focus on some of them while being terrible to others. Though, it is slightly better to train on original samples similar to the domain where fake testing images are sampled, for example, if fake videos considered are taken from YouTube, it would be better to train on YouTube original videos and so on.

Both original and their corresponding changed samples (considered fake) are then fed to any classifier to discriminate between real and fake. The model used for this is the pre-trained model EfficientNet-b4 which was trained on the ImageNet dataset. For each epoch in training, random transformations are applied to original images (so it is not like creating a new dataset). This makes it more robust as it is like data augmentation.

4.8.2. Modular Decomposition

4.8.2.1 Preparing the dataset.

The dataset used is FaceForensics++ [1] where only original images are used for both training and validation. And manipulated videos are only used for testing. As this method operates on frames, 32 frames are extracted first from videos. Frames are chosen across the video with equal time intervals between each frame. Faces' bounding boxes and landmarks are stored on disk so that they are not computed at every retrieval of the frame in training.

81 facial landmarks are used so that they can capture the forehead of the face. They will be used later to create a mask around the face and use it for blending source and target images. The face images are not cropped exactly around the face, but there is some padding within the image.

4.8.2.2 Creating synthetic data.

Using the original images, forgery artifacts are reproduced by applying transformations on the original image and blending the two images using a mask built using landmarks. Other random transformations are applied to the original images. Both types of transformations are different between epochs to ensure robustness to unseen fake data.

Let's first consider transformations made on original images that are used for data augmentation and making the model robust to unseen data. First, the image is horizontally flipped with a probability of 0.5, and its landmarks, and face's bounding box are flipped as well. Random shifts for RGB color, Hue, Saturation, Value, Brightness, and Contrast, are also applied with a probability of 0.3. Also, image compression is robust against compression techniques as most videos online are compressed. These transformations are applied only while training to be robust against unseen data.

Now for generating fake-like samples, some transformations are applied to the source and target before blending. First, either the source or the target gets random shifts as in the previous phase. Then artifacts are created by applying some transformations to the source image before blending source and target. These transformations are random translation, scaling, and elastic transformations.

This is done to focus on mismatches between source and target images like small shifts in position between the two faces or slight differences in size between them. Also, elastic transformations (deformations, smoothing) are applied to the source image. These transformations are applied to the source image and to the mask that will be used to blend both images.

A gaussian kernel is applied then to the mask so that it blends the two faces smoothly without sudden changes at the boundary. Then the mask is used to blend both images

$$mask \leftarrow gaussianBlur(mask)$$
$$mask \leftarrow mask * random\ ratio$$
$$fake\ img \leftarrow mask * source + (1 - mask) * target$$

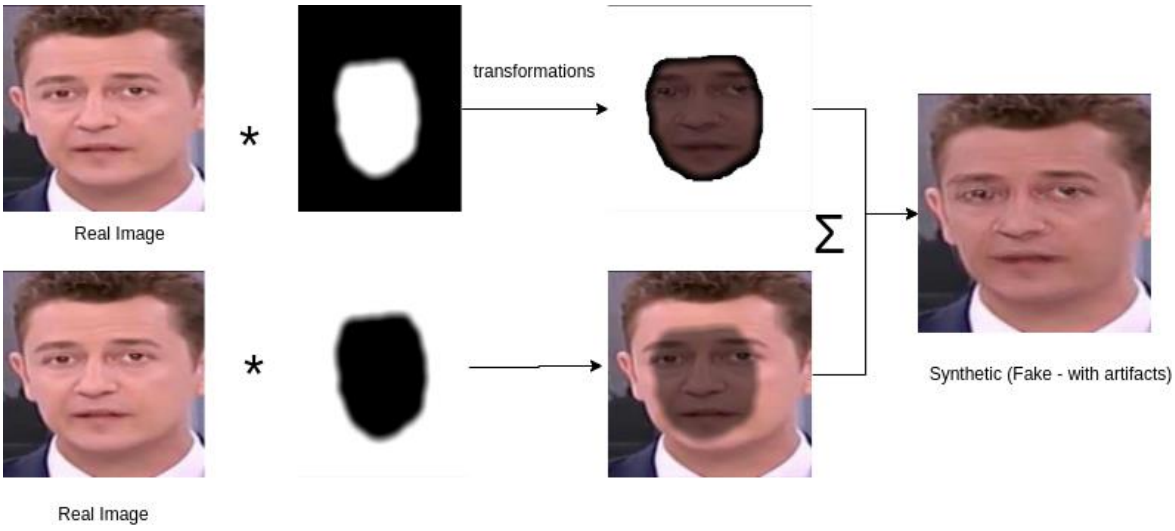

Figure 4.48 - Blending the original image and a transformed version of it to get a fake-like image.

4.8.2.3 Training

For each video, 8 frames are stored with their landmarks and bounding boxes around faces. These are then fed to the pre-trained Efficientnet-b4. Training is run for several epochs; each epoch produces new manipulated images from the original ones.

The training was held on 100 examples of the dataset with validation on 50 examples. Transformations that were applied to make the model robust while training are not applied in validation. Validation is not done on FaceForensics++'s manipulated videos but rather with generated data that is self-blended like in training.

Training runs for 36 epochs with batch size =16, learning rate = 0.001, momentum=0.9. The optimizer used is the SAM optimizer. And loss is calculated using cross-entropy loss.

The best 5 models achieving the highest AUC on validation data are stored. And the model used for testing is the last one of these 5 (the one that has the highest number of epochs)

4.8.2.3 Testing

For each video in the test set, 16 frames are chosen along the video. For some frames, fakeness is calculated for each face in it, then maximum fakeness in faces is considered the fakeness of the frames. Then the average of the frames scores is taken as the score of the whole video.

4.8.2.3 Results

The following table contains AUC achieved by training only on 100 examples and validating on 50 for 36 epochs:

Deepfakes	Face2Face	FaceSwap	NeuralTextures	Total
93.67%	85.93%	78.58%	75.1%	83.32%

Table 4.32 - Results of SBI [36 epochs, 100 training, 50 validation, 140 testing]

4.8.3. Design Constraints

- The dataset is large so preprocessing and training take time and resources.
- batch size = 32 might give better results for training but it needs a GPU that has about 40 GB VRAM
- The dataset used for training is compressed (c23) which gives worse results. For better results, we could have trained and tested on the raw dataset, but its size is huge (in terabytes)
- The method doesn't capture artifacts across frames (over time).

4.8.4. Other Description of Module

EfficientNet-b4

EfficientNet-b4 is from the family of Efficient Net that was trained on the ImageNet dataset and gave state-of-the-art results. Its basic idea is that given a basic block of CNNs, we want to scale that block to be more powerful. We want this scaling to be dependent on the resources available. This is through a parameter ϕ_{en} that controls the amount of scale applied to the basic block.

Scaling is done in width, height, and depth (3 dimensions) while other neural network architectures scale up to 2 dimensions. Networks from b0 to b7 vary in required resources. So, b0 is the simplest (basic block) and can run on low-power devices such as embedded systems. While b7 is the largest one that requires more resources and is more powerful, making it more suitable for more complex tasks.

Chapter 5: System Testing and Verification

Throughout this chapter we will discuss the testing and verification performed on our various modules and showing the final results.

5.1. Testing Setup

Since our project is based on different Machine Learning models the main setup is testing the accuracies of our models on benchmark datasets, we will provide the testing results in the coming sections.

5.2. Testing Plan and Strategy

We focused on modules testing as each module was tested well before integration and by providing an easy tested interface for integration, the integration job was much easier.

5.2.1. Module Testing

5.2.1.1 Face Detector Testing

For feature extraction, we compare our implementation using PyTorch with another implementation with bare python code using random data. This is to make sure that the outputs of both are the same and to compare the speed at which the module runs. The same thing is also applied for the decision stump classifier.

Comparing bare Python and PyTorch, the decision stump was 60 times faster using PyTorch than bare Python. It was also 10 times faster than a NumPy implementation. The strong classifier (AdaBoost) was 8.5 times faster than sklearn's implementation.

For results of the model, testing a 200-features model on the test set yielded 96% accuracy. The test set contains 905 faces and 1810 non-face images.

5.2.1.2 Facial Landmarks Detection Testing

Dataset	Normalized Error
Helen [18]	6.38
300w [15]	10.27
LFPW [20]	6.15
Combined datasets	6.89

Table 5.1: Landmarks Detector normalized error results on various benchmark datasets.

In Table 5.1 we see the final performance using the standard metric used for Landmarks Detector, which is the inter-ocular distance, As explained earlier in section 4.4.3.4.4, Our algorithm used for Landmarks Detector can have various parameters that affect the performance of the model, so we used a reporting method to track all trials parameters, results and training progress.

```python
def __initializeReporting(self, resultsOutputPath):
    '''
    Initializes reporting, creates folder for results and report file
    report is used for each training trial to analyze results
    '''
    if not os.path.exists(resultsOutputPath):
        os.makedirs(resultsOutputPath)
    numModels = len(os.listdir(resultsOutputPath))
    self.resultsOutputPath = resultsOutputPath+'/model'+str(numModels)
    os.makedirs(self.resultsOutputPath)
    reportName = self.resultsOutputPath+'/report.txt'
    self.report = open(reportName, "w")
    self.report.write("Models path: "+self.resultsOutputPath+'\n')
```

Figure 5.1: Function used that initializes the reporting method to track training in Landmarks Detector

```
Models path: models/model84
Parameters:
L = 3
K = 3
sampleSize = 5
alpha = 10000
alpha_sift = 40000
Features Used = [<FeatureType.HOG: 1>, <FeatureType.HOG: 1>, <FeatureType.SIFT: 2>]
B = 1
T = 10
trainLength = 3648
imagesPath = combined/trainset/

Models path: models/model84
X_bar path: mean_shape.npz

Iteration 1
Iteration 1 Error after sampling: 0.3757338583652888
K: 1
Alpha: 10000.0
Error after iteration l=1, regressor k=1: 0.2894977384857812
K: 2
Alpha: 10000.0
Error after iteration l=1, regressor k=2: 0.24650582944518784
K: 3
Alpha: 10000.0
Error after iteration l=1, regressor k=3: 0.22108217068704092
Iteration 2
Iteration 2 Error after sampling: 0.18023413907562516
K: 1
Alpha: 5000.0
Error after iteration l=2, regressor k=1: 0.15017892527210278
K: 2
Alpha: 5000.0
Error after iteration l=2, regressor k=2: 0.13666347809298046
K: 3
```

Figure 5.2: Example of report produced tracking a trial for Landmarks Detection

5.2.1.3 Lips Movement Analysis Testing

We tested our Lips Movement Analysis model on FF++ [1] dataset and the following results summarize the performance, similar to Facial Landmarks Testing reporting method we produced reports tracking various models' trials.

Metric	DF (HQ)		FS (HQ)		NT (HQ)		F2F (HQ)		Average
Accuracy % (Video-level)	92.5		99.3		92.1		97.9		95.45
AUC % (Video-level)	97.7		100		98.3		99.7		98.9
Confusion Matrix (Video-level)	124	16	139	1	131	9	139	1	
	5	135	1	139	13	127	5	136	

Table 5.2: Results of the model trained on the four manipulation methods of FF++, showing accuracy, AUC and Confusion Matrix (where first row represents fake videos) all calculated on Video-Level.

5.2.1.4 Dynamic Texture Analysis Testing

Unit testing for a deepfake detection model plays a crucial role in ensuring its robustness and reliability. Deepfake detection models are designed to identify manipulated or synthesized media content, which requires a high degree of accuracy. Unit testing helps verify the functionality of individual components or units within the detection model.

During unit testing, different aspects of the model are thoroughly examined. This includes testing the integrity of the data preprocessing pipeline, evaluating the performance of feature extraction algorithms, and validating the effectiveness of the classification models. Each component is tested in isolation to ensure that it functions as expected and returns correct results.

5.2.1.4.1 Feature Extraction Testing

During this stage, we tested the different components of feature extraction algorithm to ensure they produce the expected output.

Tested components:

- Function (F): one of the components of the feature extraction algorithm, used to analyze how patterns change in an image. We tested this component by inputting test cases and verifying if the output matches the expected results.

- Function (I): an important component of the feature extraction algorithm is used to measure the change in intensity between two adjacent pixels in specific directions (0, 45, 90, 135 degrees) to analyze how patterns change in an image. We tested the I function by inputting test cases and verifying if the output matches the expected results.

- Function (ldp_pixel): component of the feature extraction algorithm is used to obtain the eight binary bits of a specific pixel. LDP_pixel works by assigning binary values (0 or 1) to each neighboring pixel based on its intensity. By encoding these binary values in a clockwise or counterclockwise order, a binary pattern is generated for the center pixel. This process is repeated for every pixel in the image, resulting in a new image where each pixel holds its corresponding LDP value. We tested the ldp_pixel function by inputting test images and verifying the output patterns.

- Function (LDP_TOP): The main component of the feature extraction algorithm is used to calculate LDP on the orthogonal planes. It takes frames as input and outputs the feature vector, which is used to train the SVM model. We tested LDP_TOP function by checking the length of feature vector = 3072.

5.2.1.4.1 Model Testing

Model testing in deepfake detection is a critical phase to evaluate the performance and effectiveness of the developed detection models. Testing involves assessing the model's ability to accurately classify media content as either genuine or manipulated. This process helps validate the model's generalization capabilities and its performance on unseen data.
During model testing, a diverse set of test data is used to assess the model's robustness and resilience against various types of deepfake techniques. The test data can include manipulated videos or original videos that were not used during the model's training phase. This ensures that the model is evaluated on real-world scenarios and can detect new or unknown deepfake methods.

Different evaluation metrics are employed to measure the model's performance during testing, but we will show the accuracy of each model.

Model	Kernal	Dataset Version	Training Data (videos)	Accuracy
Kickoff	RBF	Cf23	500	79.87%
Deepfakes	RBF	Cf23	1720	88.92%
Deepfakes	Linear	Cf23	1720	95.71%
Face2Face	RBF	Cf23	1720	65.35%
Face2Face	Linear	Cf23	1720	84.64%
FaceSwap	RBF	Cf23	1720	93.35%
FaceSwap	Linear	Cf23	1720	95.35%
NeuralTextures	RBF	Cf23	1720	65.71%
NeuralTextures	Linear	Cf23	1720	88.21%
Multi-Classifier	RBF	Cf23	4300	80.14%
Multi-Classifier	Linear	Cf23	4300	90.57%
Deepfakes	RBF	Cf40	1720	68.21%
Deepfakes	Linear	Cf40	1720	78.21%
Face2Face	RBF	Cf40	1720	57.14%
FaceSwap	RBF	Cf40	1720	63.56%
FaceSwap	Linear	Cf40	1720	75.71%

Table 5.3 Dynamic Texture Models accuracies

5.2.1.5 Frequency Domain Analysis Testing

Table 5.4: SVM accuracies for Frequency Domain Analysis

Dataset	**Accuracy**
Deep Fake	68.17%
Face Swap	86.25%
Face2Face	75.75%
Neural Textures	69.50%

Table 5.5: Confusion Matrix for the Frequency domain model

	T/P	**T/N**	**F/P**	**F/N**
Deep Fake	127	145	68	59
Face Swap	185	160	22	33
Face 2 Face	170	133	37	60
Neural Textures	130	148	77	45

5.2.1.6 Self-Blended Images Testing

The model trained for 36 epochs, on 100 examples was tested on the test set. The results were as stated in Table 4.32 - Results of SBI [36 epochs, 100 training, 50 validation, 140 testing]; the total AUC was 83.3.

5.2.2. Integration Testing

Since our modules were well tested, we made sure that every module provides a well-tested simple function that takes the suitable input and output, for integration every module was simply integrated with very few lines of code, this made the integration testing much easier was had to perform only end-to-end testing by trying a video inputs and get the expected analysis outputs of the four different analysis methods.

5.3. Testing Schedule

Our module testing was performed parallel to working on the module such that each submodule was tested after being finished and each full module was tested after being finished as well, then full integration testing was performed at the end.

5.4. Comparative Results to Previous Work

Since we have created four different analysis models, we will compare the performance of our implementation results with some state-of-the-art methods in this table.
Note that these results are average results on FF++ dataset on c23 (HQ).

Method	**Accuracy %**	**AUC %**
Xception [1]	97	99.3
CNN-aug [5]	96.9	99.1
Revelio – Lips Movement	**95.45**	**98.9**
Patch-based [6]	92.6	97.2
Revelio – Dynamic Texture Analysis	**91**	**-**
Face X-ray [27]	78.4	97.8
Revelio - SBI	-	**83.32**[17]

[17] Note that it was trained only on 100 videos due to large memory requirements by SBI.

Revelio – Frequency Domain Analysis	**75**	**-**
Zhou et. al [29]	-	70

Table 5.6: Comparison of our four analysis methods to previous face manipulation methods on FF++ (HQ).

Chapter 6: Conclusions and Future Work

At the end we introduced our main project **Revelio** a system to detect deepfake videos that threatened our community and caused destabilization, we showed how it is too important to have such a system right now and gave evidence about popular accidences.

Also, we have introduced our main working dataset and system design which in turn will be divided into sub modules like face and landmarks detection, self-blended images, lips movement analysis, dynamic textures analysis and frequency domain analysis.
And finally shows the point of strength of our system and how it works.

In this chapter we will go thoroughly through what we have learned, experience we got and what we are going to do in future work.

6.1. Faced Challenges

In this section we are going to talk about the challenges we faced throughout our development process.

6.1.1 Time and computational power

This problem is considered the most serious one we have faced, as our dataset is mainly videos which need a high computational power whether for preprocessing, feature extraction or training.
As an example, at the dynamic textures analysis module, it takes days to train only on 2000 video with 20s on average.
The way we solved this problem is to train on cloud using their prepared high computational power system to train with which decreased our training time by half.

6.1.3 Classification computational power

We have two preprocessing models, the face and landmark detection, in addition to four main machine learning models, so the pipeline goes as the following, the two

preprocessing modules has to work in sequential as they depends on each other, and they both take the most time as face detection and landmarks detection needs to be performed on each frame, some solutions for this include to update the detected results by skipping two or three frames assuming the face did not move a lot in such a small time, then the results of detectors will be used as input for the four models, on average it takes about 2 minutes per 15s video with one person, so we consider to run the four models in parallel as they are independent.

6.2. Gained Experience

We started thinking about this project although we did not have the complete vision about the tools we used and the approaches we took, then we started to gain experience whether with courses or papers and get to know how to communicate with each other so well, so we concluded that as:

- Applying on Machine Learning is essential to understand the theory, so throughout this project we understood various aspects of Machine Learning and gained a lot of experience that can only be gained through applying on real projects.
- We increased our knowledge massively in various fields such as Computer Vision and Image processing and got to know more about advanced techniques used in the research.
- We got to know more about various Machine Learning tools used in real projects such as Torch.
- Learning how to structure a large Machine Learning project and pipeline was very important throughout this project and apply the best clean code and clean architecture patterns.
- Gaining knowledge about academic research and how to read and write a meaningful academic paper through reading various papers was another important experience for us.
- Learning about some techniques like Transfer Learning and how to take advantage of previous work to enhance your projects.
- We learned more about the communication architectures used with our web service.
- We learned more about concurrent programming to run our training models faster.
- Gain more experience about deepfake and how it was generated.

6.3. Conclusions

Revelio is a face manipulation detection system used to detect various face manipulation methods. We have managed to produce a very strong system that is able to detect face manipulation with very high accuracy. Throughout this book we explained the complete process of building the system and how it was built, we conclude that the face manipulation detection is still not an easy problem due to many circumstances such as the realistic fake videos produced that make it too hard to detect, and the ability to produce realistic fake videos will keep improving, so we have to keep up with the advancing technologies which is a big challenge and the problem will still be under the research.

6.4. Future Work

Revelio is mainly a web service built upon machine learning models, so we are targeting the scalability of the system beside the accuracy we may get, so at the following sections we discuss our future work thoroughly.

6.4.1 Dataset

Our current dataset is standard FaceForensics++ [1] which actually contains 1000 original video and 4000 fake video from different categories. So, we target to train on more big and various datasets like DFDC[18] which is a large dataset created by Facebook, Amazon, MIT and more contains 124k videos manipulated by 8 different manipulation algorithms.
Training on such large datasets requires huge computational power and resources.

6.4.2 Models

We are currently working with two classical methods and another two deep learning ones, which makes the former have the higher accuracy, so we are targeting to train more deep learning models for their ability to detect more complex features.

6.4.3 Training platforms

We've actually devoted too much time to train on our machines, such a trial in the dynamic textures analysis took days to converge, the same job took much less time when migrated to azure, especially when working with a data like videos and here

[18] https://ai.facebook.com/datasets/dfdc/

we are planning to transfer our work like preprocessing and training into cloud platform, for its ability to provide us with more processing power than we have on our devices, in addition to we can work use services like spark or Hadoop to parallelize our work in distributed manner like synapse analytics service.

6.4.4 website architecture

Our main service now is running on single processor meaning that the process of analyzing the video to detect whether it's fake or not beside giving the analysis takes a lot of time although the process is running in parallel, it actually takes a about 2 minutes to complete just 15 second clip with just handling the request for one user, so we are working on migrating into distributed systems, that may help a lot in case of the handling more requests, and even more within the analysis process itself.

References

[1] A. e. a. Rossler, "Faceforensics++: Learning to detect manipulated facial images.," *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV),* pp. 1-11, 2019.

[2] R. e. a. Tolosana, "Deepfakes and beyond: A survey of face manipulation and fake detection.," *Information Fusion 64,* pp. 131-148, 2020.

[3] J. M. Z. a. M. N. Thies, "Deferred neural rendering: Image synthesis using neural textures.," *Acm Transactions on Graphics (TOG) 38.4,* pp. 1-12, 2019.

[4] J. e. a. Thies, "Face2face: Real-time face capture and reenactment of rgb videos.," *Proceedings of the IEEE conference on computer vision and pattern recognition.,* 2016.

[5] S.-Y. e. a. Wang, "CNN-generated images are surprisingly easy to spot... for now.," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition,* 2020.

[6] L. e. a. Chai, "What makes fake images detectable? understanding properties that generalize.," *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVI 16. Springer,* 2020.

[7] K. S. a. T. Yamasaki, "Detecting Deepfakes with Self-Blended Images," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, New Orleans, LA, USA, 2022.

[8] A. e. a. Haliassos, "Lips don't lie: A generalisable and robust approach to face forgery detection.," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.,* 2021.

[9] M. C. P. a. G. B. Bonomi, "Dynamic texture analysis for detecting fake faces in video sequences.," *Journal of Visual Communication and Image Representation 79.,* 2021.

[10] R. M. K. a. J. K. Durall, "Watch your up-convolution: Cnn based generative deep neural networks are failing to reproduce spectral distributions.," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.,* 2020.

[11] F. C. R. a. M. S. Matern, "Exploiting visual artifacts to expose deepfakes and face manipulations.," *IEEE Winter Applications of Computer Vision Workshops (WACVW).,* 2019.

[12] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Kauai, HI, USA, 2001.

[13] Y. a. Q. J. Wu, "Facial landmark detection: A literature survey.," *International Journal of Computer Vision 127,* pp. 115-142, 2019.

[14] S. e. a. Zhu, "Face alignment by coarse-to-fine shape searching.," *Proceedings of the IEEE conference on computer vision and pattern recognition.,* 2015.

[15] E. A. G. T. S. Z. M. P. C. Sagonas, "300 faces In-the-wild challenge: Database and results.," *Image and Vision Computing (IMAVIS), Special Issue on Facial Landmark Localisation "In-The-Wild". ,* 2016.

[16] N. a. T. B. Dalal, "Histograms of Oriented Gradients for Human Detection," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition.,* 2005.

[17] M. a. M. P. Pavan, "Dominant sets and pairwise clustering.," *IEEE transactions on pattern analysis and machine intelligence 29.1,* pp. 167-172, 2006.

[18] V. e. a. Le, "Interactive facial feature localization.," *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part III 12. Springer Berlin Heidelberg, 2012..*

[19] X. Z. a. D. Ramanan, "Face detection, pose estimation, and landmark localization in the wild," *IEEE Conference on Computer Vision and Pattern Recognition,* 2012.

[20] D. W. J. D. J. K. a. N. K. P. N. Belhumeur, "Localizing parts of faces using a consensus of exemplars," *CVPR 2011, Colorado Springs,* 2011.

[21] A. e. a. Asthana, "Robust discriminative response map fitting with constrained local models.," *Proceedings of the IEEE conference on computer vision and pattern recognition.,* 2013.

[22] X. P. P. P. a. P. D. Burgos-Artizzu, "Robust face landmark estimation under occlusion.," *Proceedings of the IEEE international conference on computer vision.,* 2013.

[23] J. e. a. Zhang, "Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment.," *Computer Vision–ECCV 2014: 13th European Conference, Springer.*

[24] Baochang Zhang, Yongsheng Gao, Sanqiang Zhao, and Jianzhuang Liu., "Local derivative pattern versus local binary pattern: Face recognition," *IEEE,* 2010.

[25] P. e. a. Ma, "Towards practical lipreading with distilled and efficient models.," *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE.,* 2021.

[26] M. E. a. Z. D. Atik, "Deep learning-based 3D face recognition using derived features from point cloud.," *Innovations in Smart Cities Applications Volume 4: The Proceedings of the 5th International Conference on Smart City Applications. Springer International Publishing,* 2021.

[27] L. e. a. Li, "Face x-ray for more general face forgery detection.," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.,* 2020.

[28] L. Li, et al, "Face X-Ray for More General Face Forgery Detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2020.

[29] P. Z. e. al., "Two-Stream Neural Networks for Tampered Face Detection.," *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops,* 2018.

Appendix A: Development Platforms and Tools

In this section we are talking about the hardware platforms we used, software platforms and tools, and the external services we used.

A.1. Hardware Platforms

We used our personal PCs for training on both CPU and GPU.
We also used Microsoft Azure for some models training like Dynamic Texture Analysis SVM models.

The following table describes some of the resources used for training.

Name	Specifications	Cost
PC1	4 cores, 24GB RAM, 256GB storage	-
PC2	8 cores, 16GB RAM, INTEL 8GB GPU	-
PC3	8 cores, 16GB RAM, AMD 4GB	-
PC4	4 cores, 8GB RAM	-
Standard_D4_v3	4 cores, 16GB RAM, 100GB storage	0.19/hr$
Standard_E4ds_v4	4 cores, 32GB RAM, 150GB storage	0.29/hr$
Standard_E8s_v3	8 cores, 64GB RAM, 128GB storage	0.50/hr$
Standard_NC6	6 cores, 56GB RAM, 380GB storage	0.90/hr$
Standard_E16s_v3	16 cores, 128GB RAM, 256GB storage	1.01/hr$

Table 0.1 Training Machines

A.2. Software Tools

A.2.1 Programming Languages

A.2.1.1 Python

We used python for the whole project.

A.2.2 Libraries and Frameworks

A.2.2.1 NumPy

NumPy is an efficient computational library is python used for matrices operations and vectorization of a function and do it in C and Fortran efficiency manner.
We used it in:

- Matrices operations

A.2.2.2 OpenCV

OpenCV is a computer vision library that provides real-time optimized computer vision and also supports most machine learning operations.
We used it in:

- Video frames preprocessing like converting into grayscale, and even read video frames.

A.2.2.3 PyTorch

The main framework that we used for Deep Learning models. Also used for implementing Adaboost.

A.2.2.4 SciKit

SciKit was used for different purposes such as training models SVM and Ridge Regression.

A.2.2.5 ReactJs

React is a front-end web development framework used to design fast and reliable web applications.
We used to design our system front-end web application.

A.2.2.6 Flask

Flask is minimal python back-end framework used to design fast and easy back end application.

Appendix B: Use Cases

Our platform can be used to analyze videos to detect any face manipulation and produce analysis of different results produced from our different analysis methods. The platform can be used by a user through the website by simply uploading a video for analysis.
Other users can use our APIs to integrate with different platforms such as:

- Social media to analyze videos before being posted.
- Media agencies that analyze videos for journalism purposes.
- Governmental institutes for political purposes.
- Video-conferencing tools to detect live meeting deepfakes.

Appendix C: User Guide

We provide a user-friendly website for analysis. We will show steps through the following list of screenshots.

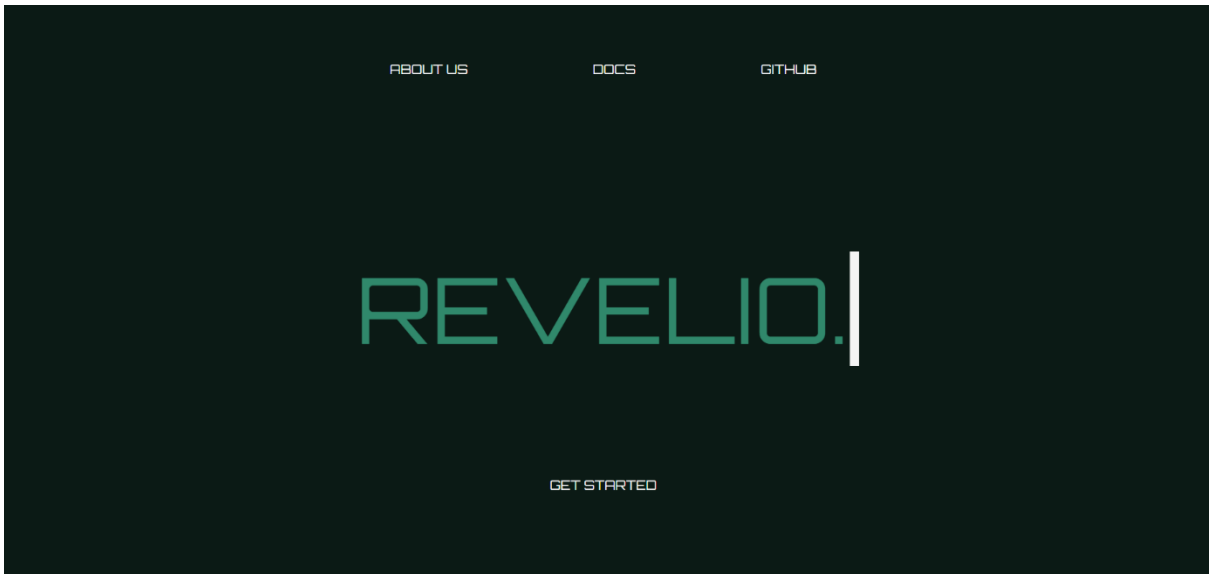

Figure C.0.1: Our main landing page

User can then click on Get Started, and with a simple drag and drop we can upload the video for analysis.

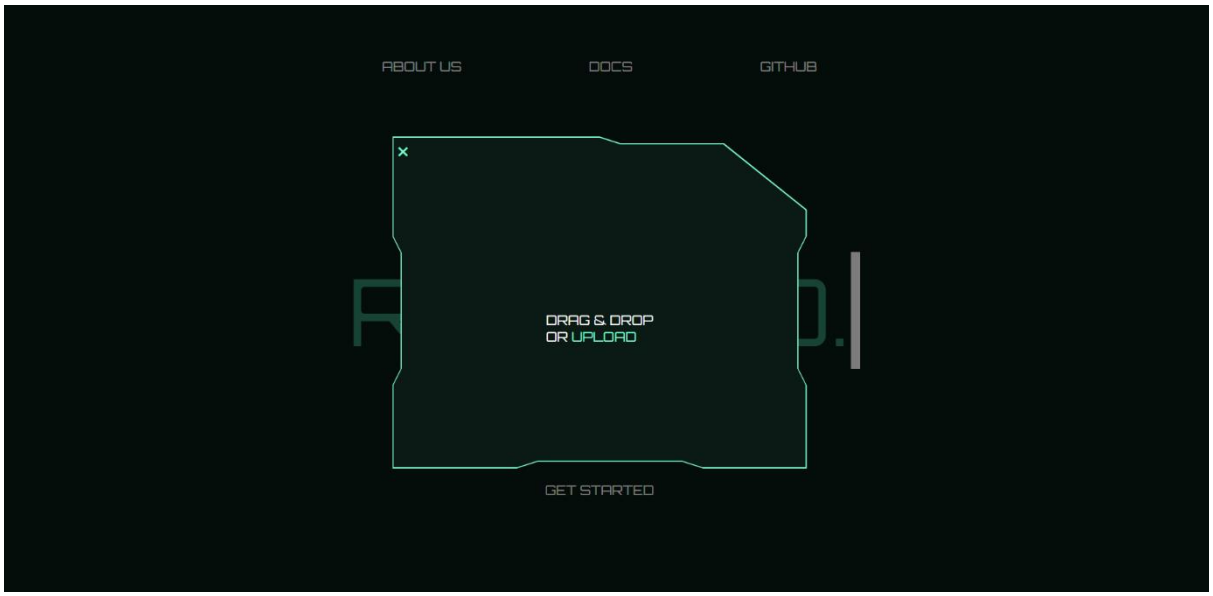

Figure C.0.2: Simple Drag and Drop to upload video for analysis.

Figure C.0.3: Our system is currently analyzing the video.

Users can now see full analysis of our four different methods with bar charts showing the probability of the video being fake and the type of manipulation used.

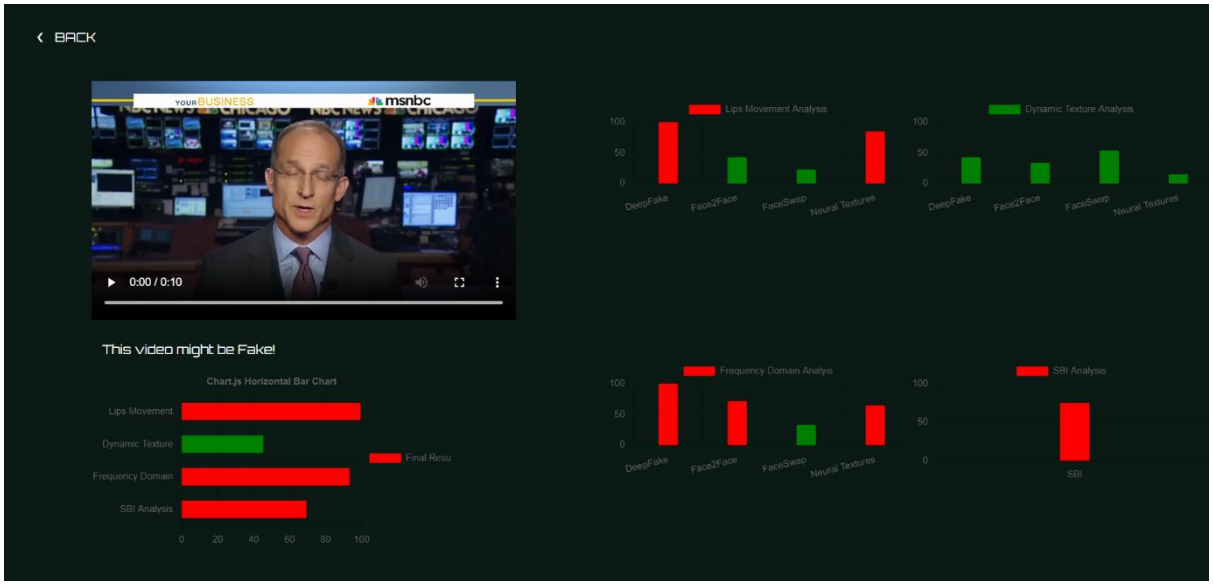

Figure C.4: Final analysis result.

Appendix D: Feasibility Study

D.1 Technical Feasibility

Since some of our models were proven to be very strong in detecting manipulation methods, we will not have a technical problem for the project. However, as mentioned before we might need to train on larger and harder datasets. It is also important to spend on infrastructure and reliable servers to perform analysis as it requires heavy GPU computational power.
Another important point is that we have to keep up with the technology, every day the face manipulation methods keep improving and we have to keep improving our techniques to be able to detect it, so a strong academic background is needed to keep developing.

D.2 Financial Feasibility

To make a stable project we will need to invest in a strong infrastructure, talented Machine Learning engineers and a team of researchers to stay up to date with the technology as well as spend enough money on marketing to attract clients.
Step by step there is room for growth but having an investor that provides enough capital is essential to kick off our project in the market.
However, the investor needs to be convinced that investing in our project is a good idea, so providing a well performed market analysis will be essential.

D.3 SWOT Analysis

Strengths: • Resilient to different manipulation methods. • Clear Analysis. • Easy to use and understand.	**Weakness:** • Providing visual analysis
Opportunities: • Huge market need without enough providers. • The social awareness on fake content is growing.	**Threats:** • Continuous development of manipulation methods.

Table D.2: SWOT Analysis of our project

SWOT analysis is important to understand the risks we might face in the future and see how to fix our weaknesses.

We understand that the market need for our project is growing without enough competitors, this gives us huge opportunity to be one of the pioneers in the field relying on our strengths of having resilient system to different manipulation methods and the insightful analysis we provide, however we need to address our weaknesses and improve the analysis with visual analysis of artifacts, we also need to prepare for the threat of continuous development of manipulation methods and keep up with the advancing technologies.