SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SANTA CLARA, CALIFORNIA

Project Report
On

"Deepfake Detection"

Project Guide:

Prof. Ming-Hwa Wang

Submitted by:

Aman Mishra (W1600017)
Kevin Lan (W1628780)

Group No. - 02

ABSTRACT

With the rapid penetration of the Internet into every part of our daily life, it is agreed that it will be an important media for future communication, perhaps even more important than the television.

This product is a self-contained product made to facilitate the users with the facility to detect which video amongst the 2 is a real or fake one. This can be very helpful the society to control and reduce blackmailing and sharing of obscene content.

We extract the feature points from the images in training dataset using FAST and get the feature point descriptors using BRIEF. Then using DLIB face detector to detect face region and regions inside the face. We group the feature points based on the region that they are falling in. Then the feature point descriptors are aggregated to train the random forest classifier.

ACKNOWLEDGEMENT

A major project is a golden opportunity for learning and self-development. We consider our self very lucky and honored to have so many wonderful people lead us through in completion of this project.

First and foremost we would like to thank **Prof. Ming-Hwa Wang** who gave us an opportunity to undertake this project.

Our grateful thanks to **Prof. Wang** for his guidance in our project work, who in spite of being extraordinarily busy with academics, took time out to hear, guide and keep us on the correct path. We do not know where we would have been without his help.

Prof. Wang monitored our progress and arranged all facilities to make life easier. We choose this moment to acknowledge their contribution gratefully.

Name of Students

Aman Mishra (W1600017)
Kevin Lan (W1628780)

TABLE OF CONTENT

Abstract……………………………………………………………………………………………..………ii

Acknowledgement…………………………………………………………………………………..……..iii

Table of Content…………………………………………………………...........................iv

List of Figures……………………………………………………………………………………….……...v

List of Tables………………………………………………………………………………….……….…vi

Chapter-2: **Project Introduction**

2.1: Objective

2.2: What is the problem?

2.3: Why this project is related to this class

2.4: Why other approach is no good

2.5: Why our approach is better

2.6: Area or scope of investigation

Chapter-3: T**heoretical bases and literature review**

3.1: Definition of the problem

3.2: Theoretical background of the problem

3.3: Related research to solve the problem

3.4: Advantage/disadvantage of those research

3.5: Our solution to solve this problem

3.6: Where your solution different from others

3.7: Why your solution is better

Chapter-4: **Hypothesis**

4.1: Goals

Chapter-5: **Methodology**

5.1: How to generate/collect input data

5.2: How to solve the problem

5.2.1: Algorithm design

5.2.2: Language used

5.2.3: Tools used

5.3: How to generate output

5.4: How to test against hypothesis

5.5: How to proof correctness

Chapter-6: **Implementation**

6.1: Code

6.2: Design document and flowchart

Chapter-7: Data analysis and discussion

7.1: Output generation

7.2: Output analysis

7.3: Compare output against hypothesis

7.4: Abnormal case explanation

7.5: Static regression

7.6: Discussion

Chapter-8: Conclusions and Recommendations

8.1: Summary and conclusions

8.2: Recommendations for future studies

Chapter-9: Bibliography

Chapter-10: Appendices

10.1: Program source code with documentation

10.2: Input/output listing

10.3: Other related material

CHAPTER-2: PROJECT INTRODUCTION

2.1: OBJECTIVE

The internet is filled with fake face images and videos synthesized by deep generative models. These realistic DeepFakes pose a challenge to determine the authenticity of multimedia content.

As the democratization of creating realistic digital humans has positive implications, there is also positive use of Deepfakes such as their applications in visual effects, digital avatars, snapchat filters, creating voices of those who have lost theirs or updating episodes of movies without reshooting them. However, the number of malicious uses of Deepfakes largely dominates that of the positive ones.

The development of advanced deep neural networks and the availability of large amount of data have made the forged images and videos almost indistinguishable to humans and even to sophisticated computer algorithms. The process of creating those manipulated images and videos is also much simpler today as it needs as little as an identity photo or a short video of a target individual. Less and less effort is required to produce a stunningly convincing tempered footage.

These forms of falsification create a huge threat to violation of privacy and identity, and affect many aspects of human lives. It is even more challenging when dealing with Deepfakes as they are majorly used to serve malicious purposes and almost anyone can create Deepfakes these days using existing Deepfake tools.

Hence, finding the truth in digital domain therefore has become increasingly critical and therefore arise the need for a good Deepfake detection algorithm which has a good efficacy in catching the malicious content.

2.2: WHAT IS THE PROBLEM?

Nowadays, people are facing an emerging problem of AI-synthesized face swapping videos, widely known as the DeepFakes. This kind of videos can be created to cause threats to privacy, fraudulence and so on. Sometimes good quality Deepfake videos recognition could be hard to distinguish with people eyes.

There are three most dangerous ways of using face swapping algorithms: face-swap, in which the face in a video is automatically replaced with another person's face; lipsync, in which only the mouth region of face is changed and people on video are made to say something that they had never said (for example, a video where former USA President Obama is altered to say things like "President Trump is a total and complete dip-****."); and the most dangerous – puppet master, in which target person's face is animated by person, sitting in front of camera.

2.3: WHY THIS IS A PROJECT RELATED TO THIS CLASS?

Data mining is the process of finding anomalies, patterns and correlations within large data sets to predict outcomes. Using a broad range of techniques, you can use this information to increase revenues, cut costs, improve customer relationships, reduce risks and more.

Hence this project is related to data mining in all possible ways because it involves finding various features, patterns and anomalies in the input image/video and then predict outcome on the basis of those findings that whether or not the input image/video is fake or original.

2.4: WHY OTHER APPROACH IS NO GOOD?

Currently, many methods for Deepfake detection are based on deep learning. Convolutional neural networks and recurrent neural networks are often used for the task. While these can give good results, they are computationally expensive to train to the point where they achieve said good results. Importantly, given the fact that Deepfakes are getting easier and easier computationally to produce, deep convolutional neural networks may not always be desirable tools for Deepfake detection, especially if less computationally expensive methods also achieve good results.

One promising approach is Deepfake detection via using classifiers on feature points and feature point descriptors. Recent research has shown that using classifiers like SVM and random decision forests on metrics computed from feature point and feature point descriptors can lead to good results. These methods are much less expensive.

Our approach will draw on the approach described in “FFR FD: Effective and Fast Detection of DeepFakes Based on Feature Point Defects.” We believe that, while their approach has led to some good results, there are ways to improve said results. Notably, their approach is motivated by the difference in count of feature points between real and Deepfake images. However, their actual results show that in many datasets taking the average of descriptor vectors, and hence losing information about the number of feature points, leads to better results. Thus, depending on their metric (whether they take FFR_FD or FFR_FD_avg), they always lose something.

2.5: WHY YOU THINK YOUR APPROACH IS BETTER?

Our approach will aim to improve on the above described deficiencies. To maintain computational cheapness, we will also reduce the size of the data using feature point detection and description. We will also use machine learning classification with SVMs and random forests, rather than using any neural networks.

To fix the issue we perceive with the FFR_FD metric, we propose the following change: we can average the feature point descriptors, to preserve the distinguish ability of the descriptors themselves, and append the count of the number of feature points in each region to the ends of the row vectors. This way, despite taking an average, we also allow the classifiers to make decisions based on the number of feature points, which have been shown to differ between real and Deepfake images.

To improve distinguish ability; we will also modify feature point detector algorithms to better suit our specific task. Namely, it is not always necessary for our dataset to maintain things like rotational or scale invariance of feature point detection.

Finally, given that we know that various classifiers each can have good performance, we will ensemble several classifiers and see whether that can improve performance over a single classifier.

2.6: AREA OR SCOPE OF INVESTIGATION

The area of investigation in this project would be to determine if a particular input video is a original video or an artificially created by the means of machine learning algorithms.

CHAPTER-3: THORETICAL BASES AND LITERATURE REVIEW

3.1: DEFINITION OF THE PROBLEM

In a narrow definition, Deepfakes (stemming from “deep learning” and “fake”) are created by techniques that can superimpose face images of a target person onto a video of a source person to make a video of the target person doing or saying things the source person does. This constitutes a category of Deepfakes, namely faceswap. In a broader definition, Deepfakes are artificial intelligence-synthesized content that can also fall into two other categories, i.e., lip-sync and puppet-master. Lip-sync Deepfakes refer to videos that are modified to make the mouth movements consistent with an audio recording. Puppet-master Deepfakes include videos of a target person (puppet) who is animated following the facial expressions, eye and head movements of another person (master) sitting in front of a camera.

While some Deepfakes can be created by traditional visual effects or computer-graphics approaches, the recent common underlying mechanism for Deepfake creation is deep learning models such as auto encoders and generative adversarial networks, which have been applied widely in the computer vision domain. These models are used to examine facial expressions and movements of a person and synthesize facial images of another person making analogous expressions and movements. Deepfake methods normally require a large amount of image and video data to train models to create photo-realistic images and videos. As public figures such as celebrities and politicians may have a large number of videos and images available online, they are initial targets of Deepfakes.

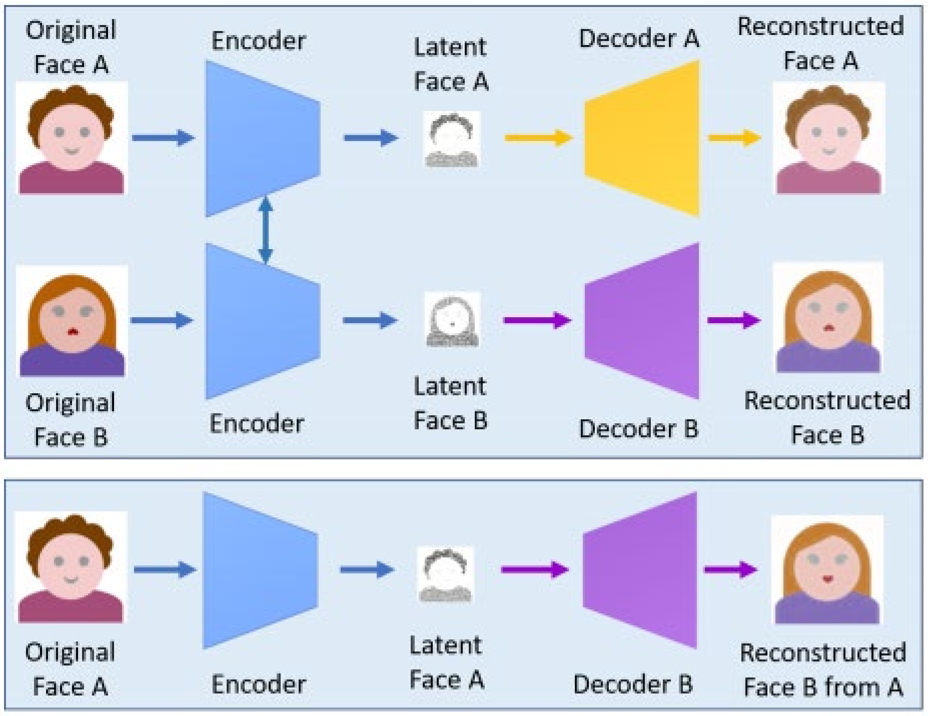
It is threatening to world security when Deepfake methods can be employed to create videos of world leaders with fake speeches for falsification purposes. Deepfakes therefore can be abused to cause political or religion tensions between countries, to fool public and affect results in election campaigns, or create chaos in financial markets by creating fake news. It can be even used to generate fake satellite images of the Earth to contain objects that do not really exist to confuse military analysts, e.g., creating a fake bridge across a river although there is no such a bridge in reality. This can mislead troops who have been guided to cross the bridge in a battle.

3.2: THEORETICAL BACKGROUND OF THE PROBLEM

DEEPFAKE CREATION: Deepfakes have become popular due to the quality of tampered videos and also the easy-to-use ability of their applications to a wide range of users with various computer skills from professional to novice. These applications are mostly developed based on deep learning techniques. Deep learning is well known for its capability of representing complex and high-dimensional data. One variant of the deep networks with that capability is deep auto encoders, which have been widely applied for dimensionality reduction and image compression.

To swap faces between source images and target images, there is a need of two encoder-decoder pairs where each pair is used to train on an image set, and the encoder's parameters are shared between two network pairs.

This strategy enables the common encoder to find and learn the similarity between two sets of face images.

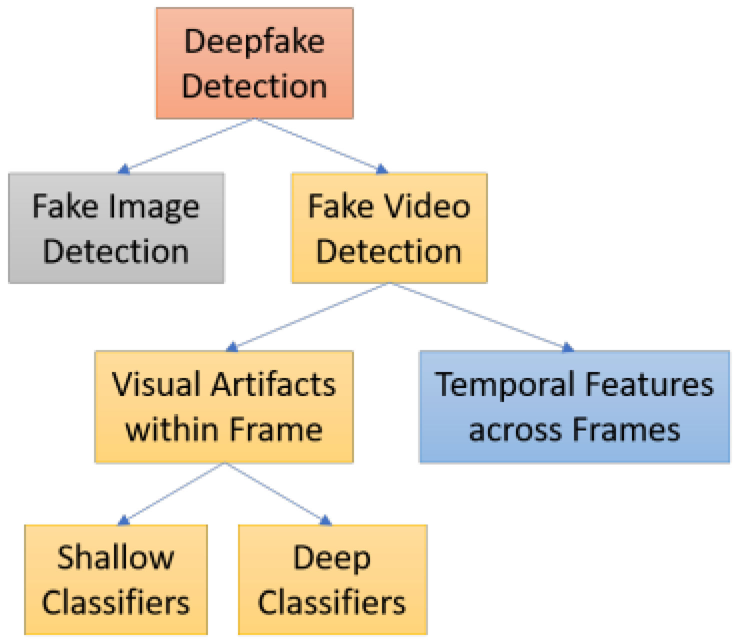


(Figure 1)

This Figure shows a Deepfake creation proces where the feature set of face A is connected with the decoder B to reconstruct face B from the original face A. This approach is applied in several works such as DeepFaceLab, DFaker, and DeepFake TensorFlow.

DEEPFAKE DETECTION:

Deepfake detection is normally deemed a binary classification problem where classifiers are used to classify between authentic videos and tampered ones. This kind of methods requires a large database of real and fake videos to train classification models.

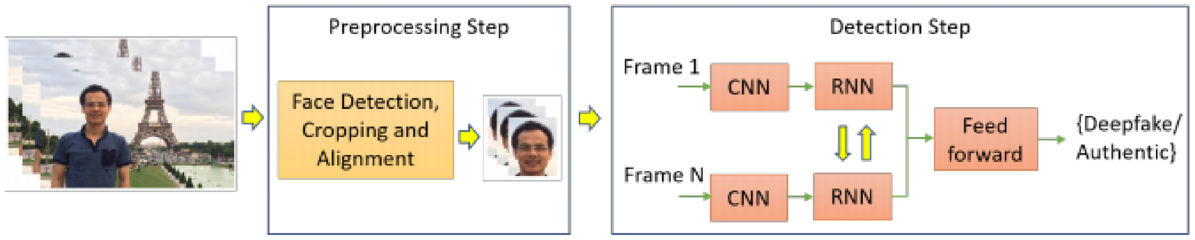


(Figure 2)

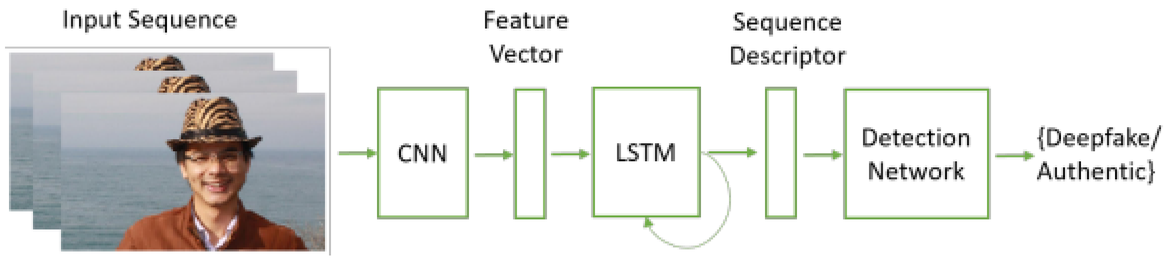
We can group it into two major categories: fake image detection methods and fake video detection ones (Figure 2). The latter is distinguished into two smaller groups: visual artifacts within single video frame-based methods and temporal features across frames-based ones.

A) FAKE IMAGE DETECTION: It is a two-phase deep learning method for detection of Deepfake images. The first phase is a feature extractor based on the common fake feature network (CFFN). Discriminative features between the fake and real images, i.e. pair wise information, are extracted through CFFN learning process. These features are then fed into the second phase, which is a small CNN concatenated to the last convolutional layer of CFFN to distinguish deceptive images from genuine.

B) FAKE VIDEO DETECTION USING TEMPORAL FEATURES ACROSS VIDEO FRAMES: As Video manipulation is carried out on a frame-by-frame basis so the generated Deepfake videos contain intra-frame inconsistencies and temporal inconsistencies between frames.



(Figure 3)



(Figure 4)

A temporal-aware pipeline method that uses CNN and long short term memory (LSTM) to detect Deepfake videos is used. CNN is employed to extract frame-level features, which are then fed into the LSTM to create a temporal sequence descriptor. A fully-connected network is finally used for classifying doctored videos from real ones based on the sequence descriptor.

C) FAKE VIDEO DETECTION USING VISUAL ARTIFACTS WITHIN VIDEO FRAME: In this the approach is to normally decompose videos into frames and explore visual artifacts within single frames to obtain discriminant features. These features are then distributed into either a deep or shallow classifier to differentiate between fake and authentic videos.

3.3: RELATED RESEARCH TO SOLVE THE PROBLEM

Early work identified physical behavior patterns, such as inconsistent head poses [3], unnatural eye blinking [4], and correlations between facial expressions and head movements [5]. However, these artifacts were fixed in second-generation DeepFake datasets, resulting in limited detection performance. Recent work has also exposed DeepFakes based on biological signals [6, 7, 8].

Detection methods based on deep neural networks (DNNs) have become mainstream. For example, a two-stream CNN was used, Meso-4 focused on the mesoscopic properties of images, a capsule structure based on VGG19 was used, ResNet was used to capture faces warping artifacts, and classic Xception was used to detect fake faces. Because videos have temporal features, some researchers have combined CNNs with RNNs for classification. With their powerful feature extraction capabilities, DNN-based methods have achieved some success, but they still have limitations against advanced DeepFakes. Learning-based methods have been further studied to address this issue. For example, FakeSpotter monitors neuron behavior to detect fake faces. More recently, researchers have combined useful modules or important features. Dang et al. utilized an attention mechanism to improve detection ability. Similarly, a vision transformer was used for detection. Gram-Net and InTeLe explore the texture information of images to improve robustness. A method combining an attention mechanism and texture features was proposed. Instead of designing large, complex neural networks, we efficiently extract features for effective DeepFake detection. To improve generalization ability, Cozzolino et al. proposed to learn an embedding based on an auto encoder. Wang et al. trained ResNet with a multi-class ProGAN dataset and showed that appropriate preprocessing and post processing could improve generalization. Face X-ray observes the blending boundaries between faces and the background to detect swapped faces; its framework adopts HRNet. The unsampling strategies of deep generative models introduce artifacts in the frequency domain, inspiring many spectrum-based detection methods. However, detection based only on the frequency spectrum leads to unsatisfactory performance and generalization. Frequency-domain artifacts can be reduced by training with spectrum regularization, focal frequency loss, or a spectrum discriminator. FakePolisher performs shallow reconstruction and can reduce artifact patterns. This calls for the discovery of the more fine-grained feature defects of DeepFakes to provide effective DeepFake detection.

3.4: ADVANTAGE/DISADVANTAGE OF THOSE RESEARCHES

CNN and RNN approaches, and deep neural network approaches in general, are very computationally expensive. Despite getting good results, they can also fail to generalize to videos and images outside of the dataset. For example, on the Kaggle Deepfake detection challenge, the solutions which performed the best on the public dataset were not necessarily the ones with the best performance on the hidden test set.

Non deep learning methods are less expensive computationally, but may require more designing and testing to achieve good results.

3.5: YOUR SOLUTION TO SOLVE THIS PROBLEM?

Our solution is to train multiple classifiers on a new metric, and take the ensemble of these models to predict whether a video is fake or real. For each video, we get the frames, then extract faces from the frames. From the frames, we detect facial feature points using an algorithm that prioritizes distinguish ability over variance. Then, we average the feature point descriptors, and append the number of feature points detected to the descriptor vector, to create our new metric. We then train our classifiers on these vectors.

3.6: WHERE YOUR SOLUTION DIFFERENT FROM OTHERS?

Compared to other research papers, we are classifying based on a new metric, which aims to preserve as much important information about the image's features as possible. We also will ensemble models, rather than using a single classifier.

Feature point selection

Random forest feature importance:

columns:

```
[-44.2418509  -44.98661973 -44.39700824 -44.09917644 -44.49016882
 -44.7081636  -44.30495787 -44.91533713 -45.6317193  -44.66948455
 -44.23551159 -44.88836695 -44.51546431 -44.97866405 -45.28289765
 -44.70964204 -44.97911605 -44.89927267 -45.23547959 -44.41096512
 -44.08209888 -44.82559261 -45.4840814  -44.83819125 -45.48893895
 -45.18011409 -44.36566779 -45.28099968 -45.03726818 -44.78122571
 -45.38329366 -44.61870843 -43.2514237 ]
```

rows:

```
[-182.92803201 -185.85245647 -190.50400684 -186.39647454 -185.08483651
 -188.01474309 -179.07428785 -179.34263362]
```

Logistic regression feature coefficients

columns:

```
[0.85136601 0.3569569  0.53390835 0.60816597 0.32013999 0.6203025
 0.73895418 0.46805727 0.57515975 0.7028464  0.81821467 0.65264439
 0.69214676 0.59530065 0.47938812 0.69954491 0.62609315 0.37878863
 0.56878615 0.80717729 0.38123539 0.33507235 0.50334526 0.87642186
 1.38518389 0.76138493 0.73434561 0.77158684 0.6314829  0.7230825
 0.92787694 0.93237134 0.74626874]
```

rows:

```
[4.47047647 2.5267772  2.74948715 2.30159538 2.16884456 2.02064425
 1.8990638  3.66671179]
```

Final estimator coefficients:

```
[[ 2.49478079 -0.1942041   6.24533777  0.16343052]]
```

We compare the geometric means of the feature importance for each row and column of our metric. The feature importance is computed by taking the reduction in the criterion due to the feature in the random forest training process. Since the feature importance is normalized, we take the geometric rather than the arithmetic mean.

We also take the arithmetic mean of the logistic regression feature coefficients for each row and column, to compute another rough estimate of feature importance.

Finally, we take the coefficients of the upper level logistic regression classifier for our stacking classifier, which predicts the final result based on the predictions of the lower level estimators. From these coefficients we select the most promising lower level estimators, which turn out to be random forest and SVM.

3.7: WHY YOUR SOLUTION IS BETTER?

As discussed above, we believe that our metric preserves more information, while adding a very minor computational load (one extra column in the matrix). We also believe that a custom feature point detector will work better than general ones. Finally, we believe that an ensemble of classifiers will perform better than single classification algorithms.

CHAPTER-4: HYPOTHESIS

4.1: GOALS

Our goal is to show that our method can achieve similar performance to state of the art neural network methods on various datasets, such as Efficient Net based methods that were the highest performing methods in the Kaggle challenge. We also want to show that we can outperform FFR_FD.

However, if our methods fail to achieve very good performances, we would like to further explore why certain changes we made were or were not beneficial to end results.

CHAPTER-5: METHODOLOGY

5.1: HOW TO GENERATE/COLLECT INPUT DATA

For this project we need a large dataset of real and fake videos. And we will be taking this dataset from the following places:

1) Deepfake Detection Challenge Dataset from Facebook AI
2) Celeb Deepfake forensics master Dataset
3) Kaggle Deepfake Dataset

5.2: HOW TO SOLVE THE PROBLEM

When we talk about Deepfake detection, obvious things that can tell us about video/photo “fakeness” are as follows:

• Too smooth skin, lack of skin details – this indicators are consequence of one problem in DeepFake algorithms: low resolution of synthesized faces. But sometimes detection can be very hard, especially because of makeup on one of two faces. Original DeepFake algorithm generates faces of 64x64 pixels so we usually need to resize them. Now, some of the algorithms can produce 128x128 or even 256x256 faces but even such sizes can be not enough for good DeepFake video.

• Color mismatch between the synthesized face and the original face - this indicator can be used in human DeepFake recognition, but sometimes such mismatches can be very tricky to detect by eyes. But not for good program.

• Visible parts of original face or temporal flickering - when face swapping algorithm got improper choice of the face region we can see artifacts of the original face or even whole original face flickering. May be it is just one frame of the whole one-hour video. But we should check this frame more precisely.

• Head position – this indicator can appear due to the problem, described above.

• Artifacts on small moving parts – due to resolution limits, DeepFake algorithm cannot produce small moving parts with good quality. That's why we can sometimes see artifacts on hairs, eyebrows, eyelashes or some small skin defects.

• Eye blinking rate – indicator that was very useful in the very beginning of the face swapping algorithms popularity. Due to small datasets of photos and very small amount of eye-closed pictures there DeepFake couldn't produce an eye-blinking face and so blinking rate reduces. Now new versions of algorithms solved such problem, so it's not very helpful anymore.

• Face warping artifacts – one of the best indicators of fake videos, generated by algorithms with low resolution face output (64x64 or 128x128). After such small picture synthesized it should be transformed affinity. So some artifacts can be seen clearly. As another plus of such indicator is that we don't need Deepfake datasets to train model. We can just use face detection algorithms and make some affine transformations to it. Face warping artifacts indicator may be the best choice right now, but when new face swapping algorithm and technologies appear and higher quality face pictures will be synthesized it can become useless.

• Person's patterns of behavior – can be useful, when we talk about the puppet-master and lip-sync techniques of Deepfakes. We can take usual person's behavior, get some patterns of it and try to identify similarity of usual and video behavior. It is, may be, the best indicator of fake videos, but it’s very hard to use such indicator on photos and it can detect only fakes with person whose behavior patterns were taken.

5.2.1: ALGORITHM DESIGN

First, we divide our datasets into training and test sets. Then, we divide the videos in our dataset into frames. Next, we extract faces from these frames.

We will then use feature point detection and description algorithms to get feature points and their descriptors from the extracted faces. We then divide the face into regions, and create our metric by averaging the feature point descriptors of the whole face, then the descriptors in each region, and finally concatenating everything. We also append the count of feature points detected to the ends of each averaged descriptor vector.

Finally, we will experiment with ensembling Logistic Regression, SVM, random forest, and other classifiers trained on our data. We will experiment with ways to determine the class of the video from the classification of the individual frames.

5.2.2: LANGUAGE USED

We have chosen Python as the programming language to work upon this project because of the following factors:

Machine learning and AI, as a unit, are still developing but are rapidly growing in usage due to the need for automation. Artificial Intelligence makes it possible to create innovative solutions to common problems, such as fraud detection, personal assistants, spam filters, search engines, and recommendations systems.

The demand for smart solutions to real-world problems necessitates the need to develop AI further in order to automate tasks that are tedious to program without AI. Python programming language is considered the best algorithm to help automate such tasks, and it offers greater simplicity and consistency than other programming languages. Further, the presence of an engaging python community makes it easy for developers to discuss projects and contribute ideas on how to enhance their code.

5.2.3: TOOLS USED

We have used following tools in our project:

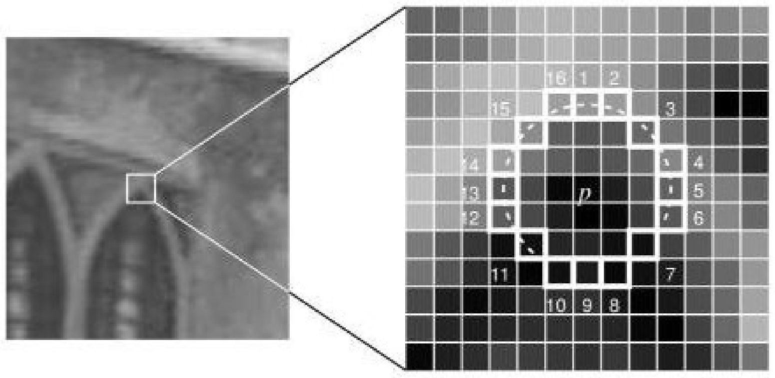
1) Numpy
2) Matplotlib
3) Jupyter Notebook
4) OpenCV
5) Pandas
6) MTCNN
7) Glob2
8) Skimage

5.3: HOW TO GENERATE OUTPUT

We are using FAST and BRIEF algorithms to generate the output.

<u>FAST (Features from Accelerated and Segments Test):</u>

The algorithm is explained below:



12 point segment test corner detection in an image patch. The highlighted squares are the pixels used in the corner detection. The pixel at p is the center of a candidate corner. The arc is indicated by the dashed line passes through 12 contiguous pixels which are brighter than p by more than the threshold.

- Select a pixel ***p*** in the image which is to be identified as an interest point or not. Let its intensity be ***Ip***.
- Select appropriate threshold value ***t***.
- Consider a circle of 16 pixels around the pixel under test. (This is a Bresenham circle of radius 3.)
- Now the pixel ***p*** is a corner if there exists a set of ***n*** contiguous pixels in the circle (of 16 pixels) which are all brighter than ***Ip + t***, or all darker than ***Ip - t***. (The authors have used ***n***= 12 in the first version of the algorithm)

- To make the algorithm fast, first compare the intensity of pixels 1, 5, 9 and 13 of the circle with ***Ip***. As evident from the figure above, at least three of these four pixels should satisfy the threshold criterion so that the interest point will exist.
- If at least three of the four-pixel values — ***I1, I5, I9, I13*** are not above or below ***Ip + t***, then ***p*** is not an interest point (corner). In this case reject the pixel ***p*** as a possible interest point. Else if at least three of the pixels are above or below ***Ip + t***, then check for all 16 pixels and check if 12 contiguous pixels fall in the criterion.
- Repeat the procedure for all the pixels in the image.

There are a few limitations to the algorithm. First, for ***n***<12, the algorithm does not work very well in all cases because when ***n***<12 the number of interest points detected are very high. Second, the order in which the 16 pixels are queried determines the speed of the algorithm. A machine learning approach has been added to the algorithm to deal with these issues.

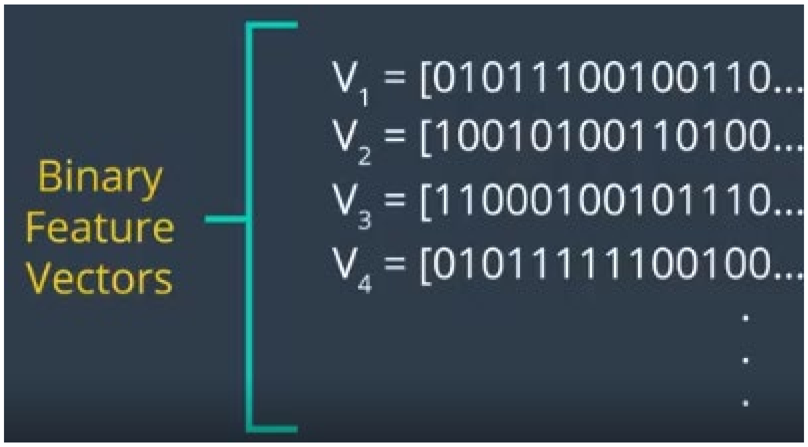
Machine Learning Approach

- Select a set of images for training (preferably from the target application domain)
- Run FAST algorithm in every image to find feature points.

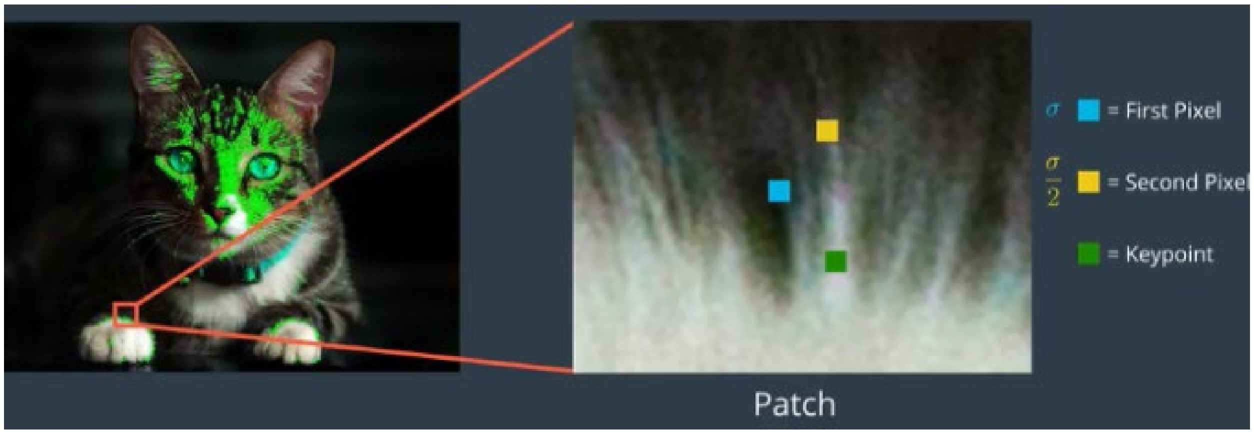
- For every feature point, store the 16 pixels around it as a vector. Do it for all the images to get feature vector ***p***.
- Each pixel (say ***x***) in these 16 pixels can have one of the following three states:
- Depending on these states, the feature vector ***P*** is subdivided into 3 subsets ***Pd, Ps, Pb***.
- Define a variable ***Kp*** which is true if ***p*** is an interest point and false if ***p*** is not an interest point.
- Use the ID3 algorithm (decision tree classifier) to query each subset using the variable Kp for the knowledge about the true class.
- The ID3 algorithm works on the principle of entropy minimization. Query the 16 pixels in such a way that the true class is found (interest point or not) with the minimum number of queries. Or in other words, select the pixel x, which has the most information about the pixel ***p***. The entropy for the set ***P*** can be mathematically represented as:
- This is recursively applied to all the subsets until its entropy is zero.
- The decision tree so created is used for fast detection in other images.

<u>BRIEF (Binary robust independent elementary feature):</u>

Brief takes all key points found by the fast algorithm and convert it into a binary feature vector so that together they can represent an object. Binary features vector also know as binary feature descriptor is a feature vector that only contains 1 and 0. In brief, each key point is described by a feature vector which is 128–512 bits string.


Binary Feature Vectors
V_1 = [01011100100110...
V_2 = [10010100110100...
V_3 = [11000100101110...
V_4 = [01011111100100...

Brief start by smoothing image using a Gaussian kernel in order to prevent the descriptor from being sensitive to high-frequency noise. Than brief select a random pair of pixels in a defined neighborhood around that key point. The defined neighborhood around pixel is known as a patch, which is a square of some pixel width and height. The first pixel in the random pair is drawn from a Gaussian distribution centered on the key point with a stranded deviation or spread of sigma. The second pixel in the random pair is drawn from a Gaussian distribution centered on the first pixel with a standard deviation or spread of sigma by two. Now if the first pixel is brighter than the second, it assigns the value of 1 to corresponding bit else 0.



Again brief select a random pair and assign the value to them. For a 128-bit vector, brief repeat this process for 128 times for a key point. Brief create a vector like this for each key point in an image. However, BRIEF also isn't invariant to rotation so orb uses rBRIEF (Rotation-aware BRIEF). ORB tries to add this functionality, without losing out on the speed aspect of BRIEF.

5.4: HOW TO TEST AGAINST HYPOTHESES

In our hypothesis, we believed that we could outperform FFR_FD by averaging and adding a column for the counts of feature points in regions, along with stacking classifiers. This proved to be the case. However, we also believed that we would be able to achieve state of the art performances, akin to deep learning based methods of deepfake detection. For this dataset, this goal was unfortunately not achieved.

CHAPTER-6: IMPLEMENTATION

6.1: CODE

We have 2 python files namely compute_metric.py and testing_models.py**.**

compute_metric.py:

```
import numpy as np

import os

import json

import re

import cv2 as cv

import dlib

from imutils import face_utils

from numba import jit

from numba import cuda

import sklearn

from sklearn.ensemble import RandomForestClassifier

import tqdm


metadatas = {}

img_paths = []


fast = cv.FastFeatureDetector_create()

brief = cv.xfeatures2d.BriefDescriptorExtractor_create()
```

```
detector = dlib.get_frontal_face_detector()

predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')\


unzero = np.vectorize(lambda x: x if x > 0 else 1)


def detect_face(img):

   gray = None

   if len(img.shape) == 3:

      gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

   else:

      gray = img

   faces = detector(gray, 1)

   return faces, gray


@jit

def rect_contains(rect, point):

   return rect[0] < point[0] < rect[0] + rect[2] and rect[1] < point[1] < rect[1] + rect[3]


@jit

def add_to_row(metric, row, vector):

   metric[row, :] += vector
```

```
@jit
def create_metric(size):
    return np.zeros((8, size))


@jit
def take_avg(matrix, column):
    column = unzero(column)
    matrix /= column


def get_label(filepath):
    numbers = re.findall('[0-9]+', filepath)
    number = int(''.join(numbers)[0:2])
    key = filepath.split("\\")[3][:-4] + '.mp4'
    return 0 if metadatas[number][key]['label'] == 'REAL' else 1


for dirname, _, filenames in os.walk('archive'):
    for filename in filenames:
        if "metadata" in filename:
            numbers = re.findall('[0-9]+', filename)
            number = int(''.join(numbers))
            os.path.join(dirname, filename)
            with open(os.path.join(dirname, filename)) as f:

                metadatas[number] = json.load(f)


        else:

            img_paths.append(os.path.join(dirname, filename))


labels = list(map(get_label, img_paths))


def create_data(indices, avg=False, extra_column=False, rows=range(7)):

    data = []

    for i in tqdm.tqdm(indices):

        ip = img_paths[i]


        img = cv.imread(ip, 0)

        fp = fast.detect(img, None)

        fp, des = brief.compute(img, fp)

        descriptor_size = brief.descriptorSize()

        metric = create_metric(descriptor_size)

        counts_column = np.zeros((8, 1))

        faces, gray = detect_face(img)

        if len(faces) == 0:

            '''for j, p in enumerate(fp):
```

```
            des_vector = des[j, :]

            metric += des_vector

            counts_column += [1]

        if avg:

            take_avg(metric, counts_column)

        data.append(metric.flatten())'''

        continue

    shape = predictor(gray, faces[0])

    shape = face_utils.shape_to_np(shape)


    for l, (name, (j, k)) in enumerate(face_utils.FACIAL_LANDMARKS_IDXS.items()):

        if name == 'jaw':

            break

        b_rect = cv.boundingRect(np.array([shape[j:k]]))

        whole_face_rect = face_utils.rect_to_bb(faces[0])

        for j, p in enumerate(fp):

            if rect_contains(whole_face_rect, p.pt):

                des_vector = des[j, :]

                add_to_row(metric, 7, des_vector)

                add_to_row(counts_column, 7, [1])

                w = b_rect[2]

                h = b_rect[3]
                if rect_contains((b_rect[0] - w/10, b_rect[1] - h/10, 1.1 * w, 1.1 * h), p.pt):
                    add_to_row(metric, l, des_vector)
                    add_to_row(counts_column, l, [1])

    if avg:
        take_avg(metric, counts_column)
    if extra_column:
        metric = np.concatenate((metric, counts_column), axis=1)
    data.append(np.take(metric, rows, 0).flatten())
return data
```

testing_models.py:

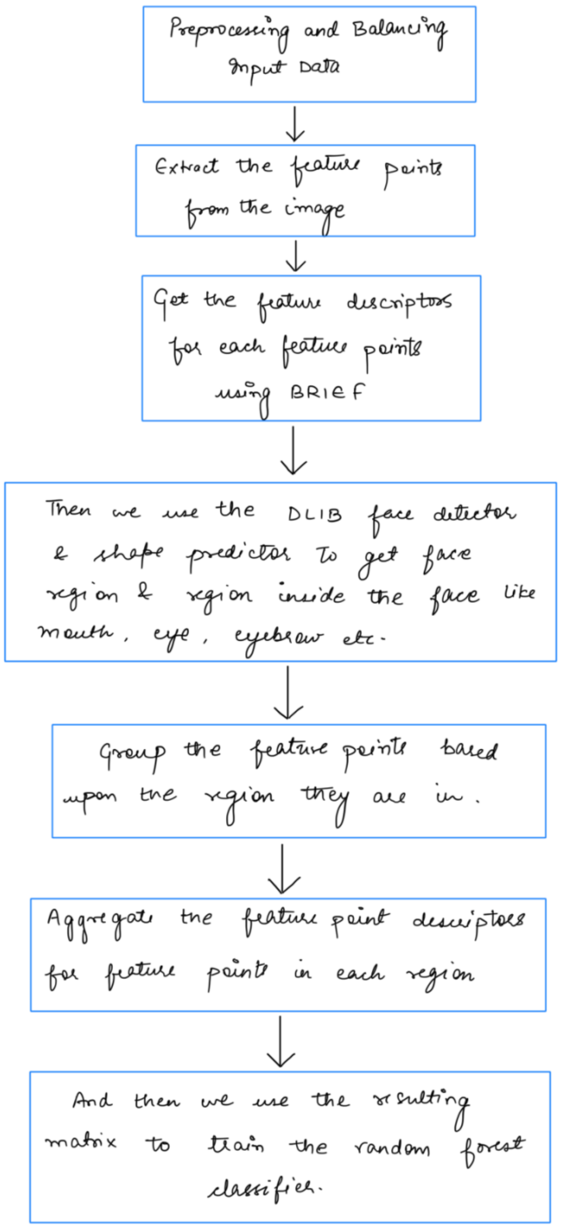
```
import numpy as np
from compute_metric import create_data, labels, detector, detect_face, img_paths
import sklearn
from sklearn.ensemble import RandomForestClassifier
import joblib
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import random
import cv2 as cv
import json
import dlib

import os

import tqdm

import math

from sklearn.linear_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from sklearn import svm

from sklearn.naive_bayes import GaussianNB

from sklearn.ensemble import StackingClassifier


random.seed('1')


#remove images with undetectable faces

print("removing images with undetectable faces")

if not os.path.exists('pruned.json'):

    kept_image_indices = [i for i, ip in tqdm.tqdm(enumerate(img_paths)) if

len(detect_face(cv.imread(ip, 0))[0]) != 0]

    with open('pruned.json', 'x') as outfile:

        json.dump(kept_image_indices, outfile)


kept_set = None

with open('pruned.json') as infile:
    kept_set = set(json.load(infile))


#balance dataset


real_indices = [index for index, label in enumerate(labels) if label == 0 and index in kept_set]

print(len(real_indices))

fake_indices = [index for index, label in enumerate(labels) if label == 1 and index in kept_set]

print(len(fake_indices))


print("balancing dataset")

fake_sample = random.sample(fake_indices, len(real_indices))


balanced_data = real_indices + fake_sample


random.shuffle(balanced_data)


print("pruned dataset size", len(balanced_data))


all_data = balanced_data


#all_data = list(kept_set)
```

```
random.shuffle(all_data)


def create_model(num_data, avg=False, custom=False, rows=range(7)):
    print("creating training data")
    data = all_data[0:num_data]
    X = create_data(data, avg=avg, extra_column=custom, rows=rows)
    print("dataset of our metric created")

    print("standardizing dataset")
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    print("creating labels")
    print("finished creating labels")

    y = [labels[i] for i in data]

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=1
    )

    depths = [20]
    y_preds = []
```

```
y_train_preds = []

clf = None

print("fitting classifiers")

for d in depths:

    clf_rf = RandomForestClassifier(n_estimators=100, max_depth=d, class_weight='balanced')

    clf_lr = LogisticRegression(max_iter=1000)

    clf_svm = svm.SVC()

    clf_nb = GaussianNB()

    clfs = [('rf', clf_rf), ('lr', clf_lr), ('svm', clf_svm), ('nb', clf_nb)]

    clfs = [clfs[0], clfs[2]]

    clf = StackingClassifier(

        estimators=clfs, final_estimator=LogisticRegression())


    clf = clf_rf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    y_train_pred = clf.predict(X_train)


    y_preds.append(y_pred)

    y_train_preds.append(y_train_pred)


if not avg:

    joblib.dump(clf, 'FFR_FD_no_ave_model_' + str(num_data) + '.pkl')
```

```
        print('exported model file as ', 'FFR_FD_no_ave_model_' + str(num_data) + '.pkl')
    elif not custom:
        joblib.dump(clf, 'FFR_FD_ave_model_' + str(num_data) + '.pkl')
        print('exported model file as ', 'FFR_FD_ave_model_' + str(num_data) + '.pkl')
    else:
        joblib.dump(clf, 'custom_model_' + str(num_data) + '.pkl')
        print('exported model file as ', 'custom_model_' + str(num_data) + '.pkl')

    return y_preds, y_train_preds, X_train, X_test, y_train, y_test, depths



num_data = len(balanced_data) - 1
avg = True
custom = True
rows = [0, 3, 4, 6, 7]

def train():
    y_preds, y_train_preds, X_train, X_test, y_train, y_test, depths = create_model(num_data,
avg=avg, custom=custom, rows=rows)

    for i in range(len(depths)):
        print(depths[i], f"Test set accuracy is {accuracy_score(y_preds[i], y_test) * 100:.2f} %")
        print(depths[i], f"Train set accuracy is {accuracy_score(y_train_preds[i], y_train) * 100:.2f}
%")

def test():
    X_test = create_data(all_data[100:400], avg=avg, extra_column=custom, rows=rows)
    scaler = StandardScaler()
    X_test = scaler.fit_transform(X_test)
    y_test = [labels[i] for i in all_data[100:400]]
    print("loading model from file")
    clf = joblib.load('custom_model_' + str(num_data) + '.pkl')

    y_pred = clf.predict(X_test)

    print(list(y_pred))
    print(y_test)
    print(clf.final_estimator_.coef_)
    #print(np.sum(np.vectorize(abs)(clf.coef_.reshape(8, 33)), axis=1))
    #print(np.sum(np.vectorize(math.log)(clf.feature_importances_.reshape((8, 33))), axis=0))

    #print(np.sum(np.vectorize(math.log)(clf.feature_importances_.reshape((8, 33))), axis=1))
    print(f"Accuracy is {accuracy_score(y_pred, y_test)*100:.2f} %")


train()

#test()
```

6.2: DESIGN DOCUMENT AND FLOWCHART


Preprocessing and Balancing Input Data
Extract the feature points from the image
Get the feature descriptors for each feature points using BRIEF
Then we use the DLIB face detector & shape predictor To get face region & region inside the face like mouth, eye, eyebrow etc.
Group the feature points based upon the region they are in.
Aggregate the feature point descriptors for feature points in each region
And then we use the resulting matrix to train the random forest classifier.

CHAPTER-7: DATA ANALYSIS AND DISCUSSION

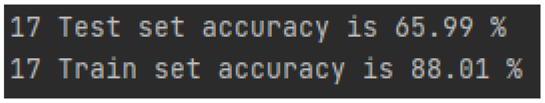
7.1: OUTPUT GENERATION

- Extract the feature points from the images in training dataset using FAST and get the feature point descriptors using BRIEF.
- Then using DLIB face detector to detect face region and regions inside the face.
- Group the feature points based on the region that they are falling in.
- The resulting feature point descriptors are aggregated to train the random forest classifier.
- Use this random forest classifier for testing the deepfakes and output generation.

7.2: OUTPUT ANALYSIS

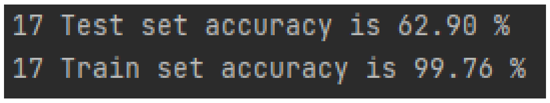
Our algorithm ended up outperforming FFR_FD on the test set, while reaching a lower accuracy on the training set. This makes sense, as we took many steps to reduce over fitting. However, the final accuracy we reach is nevertheless not entirely ideal. We speculate that the unique challenges presented by this dataset make a feature point/descriptor classification approach less successful. For example, the fact that the images are lower resolution may make the feature points and descriptors less distinguishable. From our analysis, the feature points in real and fake images do not seem to differ by as much as in the datasets used by the FFR_FD paper in their analysis.

Results after all changes



```
17 Test set accuracy is 65.99 %
17 Train set accuracy is 88.01 %
```

Base FFR_FD



```
17 Test set accuracy is 62.90 %
17 Train set accuracy is 99.76 %
```

7.3: COMPARE OUTPUT AGAINST HYPOTHESIS

In our hypothesis, we believed that we could outperform FFR_FD by averaging and adding a column for the counts of feature points in regions, along with stacking classifiers. This proved to be the case. However, we also believed that we would be able to achieve state of the art performances, akin to deep learning based methods of deepfake detection. For this dataset, this goal was unfortunately not achieved.

7.4: ABNORMAL CASE EXPLANATION

In some of the images, there simply is no person or face. These data points we simply drop from our training and testing process. A harder case is images which show the face from the side. Since side face and facial region detection is much more difficult, we initially simply aggregated all feature point descriptors for the whole image and used that as the metric for the side profile faces. However, this was detrimental to performance. We ended up restricting the scope of our project to detecting frontal face deepfakes.

Our dataset also exhibited a hugely disproportionate amount of images for each class. We had many times more fake images than real images. The result is that our classification algorithm would not learn to predict real images; it would simply get high accuracies from predicting fake almost exclusively. To combat this, we equalize the number of real and fake images somewhat, and also weight the real samples more when we train the random forest.

7.5: DISCUSSION

Deepfakes have begun to erode trust of people in media contents as seeing them is no longer commensurate with believing in them. They could cause distress and negative effects to those targeted, heighten disinformation and hate speech, and even could stimulate political tension, inflame the public, violence or war. This is especially critical nowadays as the technologies for creating deepfakes are increasingly approachable and social media platforms can spread those fake contents quickly. Sometimes deepfakes do not need to be spread to massive audience to cause detrimental effects. People who create deepfakes with malicious purpose only need to deliver them to target audiences as part of their sabotage strategy without using social media. For example, this approach can be utilized by intelligence services trying to influence decisions made by important people such as politicians, leading to national and international security threats. Catching the deepfake alarming problem, research community has focused on developing deepfake detection algorithms and numerous results have been reported. This paper has reviewed the state-of-the-art methods and a summary of typical approaches is provided in Table II. It is noticeable that a battle between those who use advanced machine learning to create deepfakes with those who make effort to detect deepfakes is growing.

Deepfakes' quality has been increasing and the performance of detection methods needs to be improved accordingly. The inspiration is that what AI has broken can be fixed by AI as well.

Detection methods are still in their early stage and various methods have been proposed and evaluated but using fragmented datasets. An approach to improve performance of detection methods is to create a growing updated benchmark dataset of deepfakes to validate the ongoing development of detection methods. This will facilitate the training process of detection models, especially those based on deep learning, which requires a large training set. On the other hand, current detection methods mostly focus on drawbacks of the deepfake generation pipelines, i.e. finding weakness of the competitors to attack them. This kind of information and knowledge is not always available in adversarial environments where attackers commonly attempt not to reveal such deepfake creation technologies. Recent works on adversarial perturbation attacks to fool DNN-based detectors make the deepfake detection task more difficult. These are real challenges for detection method development and a future research needs to focus on introducing more robust, scalable and generalizable methods.

CHAPTER-8: CONCLUSIONS AND RECOMMENDATIONS

8.1: SUMMARY AND CONCLUSIONS

We presented FFR FD, a vector representation for DeepFake detection, which can be constructed from different facial regions in combination with various feature descriptors. Inspired by local feature detection description algorithms to extract fine-grained features, we explored the feature points in DeepFakes. Through FAST&BRIEF the experimental results indicate current DeepFake faces lack a sufficient number of feature points. Without the need for powerful GPUs, we trained the random forest classifier with FFR FD. Experimental results showed that our approach can achieve state-of-the-art detection performance while considering efficiency and generalization. FFR FD relies heavily on feature point detector descriptors, but current algorithms are not specifically designed for DeepFake detection tasks, given that they must compromise between distinguishability and invariance. In future work, we would like to design a discriminative feature descriptor for face forensics.

8.2: RECOMMENDATIONS FOR FUTURE STUDIES

Another research direction is to integrate detection methods into distribution platforms such as social media to increase its effectiveness in dealing with the widespread impact of deepfakes. The screening or filtering mechanism using effective detection methods can be implemented on these platforms to ease the deepfakes detection. Legal requirements can be made for tech companies who own these platforms to remove deepfakes quickly to reduce its impacts. In addition, watermarking tools can also be integrated into devices that people use to make digital contents to create immutable metadata for storing originality details such as time and location of multimedia contents as well as their untampered attestment. This integration is difficult to implement but a solution for this could be the use of the disruptive blockchain technology. The blockchain has been used effectively in many areas and there are very few studies so far addressing the deepfake detection problems based on this technology. As it can create a chain of unique unchangeable blocks of metadata, it is a great tool for digital provenance solution. The integration of blockchain technologies to this problem has demonstrated certain results but this research direction is far from mature. Using detection methods to spot deepfakes is crucial, but understanding the real intent of people publishing deepfakes is even more important. This requires the judgment of users based on social context in which deepfake is discovered, e.g. who distributed it and what they said about it. This is critical as deepfakes are getting more and more photorealistic and it is highly anticipated that detection software will be lagging behind deepfake creation technology. A study on social context requires careful documentation for each step of the forensics process and how the results are reached. Machine learning and AI algorithms can be used to support the determination of the authenticity of digital media and have obtained accurate and reliable results, but most of these algorithms are unexplainable. This creates a huge hurdle for the applications of AI in forensics problems because not only the forensics experts oftentimes do not have expertise in computer algorithms, but the computer professionals also cannot explain the results properly as most of these algorithms are black box models. This is more critical as the most recent models with the most accurate results are based on deep learning methods consisting of many neural network parameters. Explainable AI in computer vision therefore is a research direction that is needed to promote and utilize the advances and advantages of AI and machine learning in digital media forensics.

BIBLIOGRAPHY

[1] Thanh Thi Nguyen, Quoc Viet Hung Nguyen, Cuong M. Nguyen, Dung Nguyen, Duc Thanh Nguyen, Saeid Nahavandi, Fellow, IEEE, 2021. “Deep Learning for Deepfakes Creation and Detection: A Survey”. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[2] Gaojian Wang, Qian Jiang, Xin Jin, Xiaohui Cui, “FFR FD: Effective and Fast Detection of DeepFakes Based on Feature Point Defects”, 2020.

[3] X. Yang, Y. Li, S. Lyu, Exposing deep fakes using inconsistent head poses, in: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 8261–8265.

[4] Y. Li, M.-C. Chang, S. Lyu, In ictu oculi: Exposing ai created fake videos by detecting eye blinking, in: 2018 IEEE International Workshop on Information Forensics and Security (WIFS), IEEE, 2018, pp. 1–7.

[5] S. Agarwal, H. Farid, Y. Gu, M. He, K. Nagano, H. Li, Protecting world leaders against deep fakes., in: CVPR Workshops, 2019, pp. 38–45.

[6] U. A. Ciftci, I. Demir, L. Yin, Fakecatcher: Detection of synthetic portrait videos using biological signals, IEEE Transactions on Pattern Analysis and Machine Intelligence (2020).

[7] H. Qi, Q. Guo, F. Juefei-Xu, X. Xie, L. Ma, W. Feng, Y. Liu, J. Zhao, Deeprhythm: exposing deepfakes with attentional visual heartbeat rhythms, in: Proceedings of the 28th ACM International Conference on Multimedia, 2020, pp. 4318–4327

[8] T. Mittal, U. Bhattacharya, R. Chandra, A. Bera, D. Manocha, Emotions don’t lie: An audio-visual deepfake detection method using affective cues, in: Proceedings of the 28th ACM International Conference on Multimedia, 2020, pp. 2823–28

Datasets to be used:

http://cs.binghamton.edu/~ncilsal2/DeepFakesDataset/

https://www.kaggle.com/unkownhihi/deepfake?select=DeepFake06

https://www.cs.albany.edu/~lsw/celeb-deepfakeforensics.html

https://ai.facebook.com/datasets/dfdc/