Thomas Dunkley

The Feasibility Of Deepfake Detection On A Large Scale

Computer Science Tripos – Part II

Sidney Sussex College

2023

Declaration of originality

I, Thomas Dunkley of Sidney Sussex College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. The project required the use of AI-assisted platform Chat-GPT in Section 3.2, and such use is acknowledged in the text. I am content for my dissertation to be made available to the students and staff of the University.

Signed Thomas Dunkley

Date May 11, 2023

Proforma

Candidate Number:	**2334D**
Project Title:	**The Feasibility Of Deepfake Detection On A Large Scale**
Examination:	**Computer Science Tripos – Part II, 2023**
Word Count:	**9687**[1]
Code Line Count:	**1375**[2]
Project Originator:	**The Candidate**
Supervisor:	**Eric Meissner (UTO: Neil Lawrence)**

Original Aims of the Project

The original aim of the project was to investigate whether deepfakes could be detected in a fast, accurate, and practical way by attempting to create a Chrome extension which does exactly that.

Work Completed

I started by comparing methods of deepfake detection on paper. I then found implementations of those methods. I outlined a Chrome extension which takes a screenshot of the user's tab and evaluates it with one of the classifiers. I created a simulation which can be used to evaluate the speed and accuracy of the extension with different classifiers. I implemented my own version of the optical flow method for deepfake detection. I then created my own dataset for testing. I then tested various implementations with the simulation.

Special Difficulties

None.

[1] This word count was computed by texcount.
[2] Calculated by summing line numbers at the end of files.

Contents

1 Introduction **9**
1.1 Work Completed . . . 9
1.2 Background . . . 9
1.3 Motivation . . . 9
1.4 Related Work . . . 10
1.4.1 Deepfake Detection . . . 10
1.4.2 Existing Chrome Extensions . . . 10
1.4.3 Kaggle Deepfake Detection Challenge . . . 10

2 Preparation **11**
2.1 Requirements . . . 11
2.1.1 List of Core Requirements . . . 11
2.1.2 List of Extension Tasks . . . 11
2.2 Starting Point . . . 12
2.3 Tools Used . . . 12
2.3.1 Language and Libraries . . . 12
2.3.2 Google Colab . . . 12
2.3.3 Dataset . . . 12
2.3.4 Use of Chat-GPT . . . 13
2.4 Software Engineering Techniques . . . 13
2.4.1 Testing . . . 13
2.4.2 User Privacy . . . 14
2.4.3 Use of GitHub . . . 14
2.5 Terminology . . . 14
2.5.1 Speed, Accuracy, and Practicality . . . 14
2.5.2 Deepfakes . . . 14
2.5.3 Deepfake Detection . . . 14
2.6 Technical Background . . . 15
2.6.1 Chrome Extensions . . . 15
2.6.2 Neural Networks . . . 15
2.6.3 Optical Flow . . . 19
2.6.4 ROC and AUC . . . 19

3 Implementation **22**
- 3.1 Outline . . . 23
- 3.2 Chrome Extension . . . 24
- 3.3 Classifier Options . . . 27
  - 3.3.1 MesoNet . . . 27
  - 3.3.2 Ensemble of CNNs . . . 27
  - 3.3.3 Recurrent Neural Networks . . . 27
  - 3.3.4 Optical Flow . . . 28
  - 3.3.5 Pre-Existing Implementations . . . 28
  - 3.3.6 Face Detection . . . 28
- 3.4 Optical Flow Based Detection . . . 30
  - 3.4.1 Managing The Dataset . . . 30
  - 3.4.2 Training . . . 31
  - 3.4.3 One Trainable VGG-16 Layer . . . 31
- 3.5 A More Appropriate Dataset . . . 33
- 3.6 Comparison . . . 34
  - 3.6.1 Results . . . 35
  - 3.6.2 Significance Testing . . . 40
- 3.7 Repository Overview . . . 40

4 Evaluation **41**
- 4.1 The Chrome Extension . . . 41
  - 4.1.1 Speed . . . 41
  - 4.1.2 Accuracy . . . 42
  - 4.1.3 Practicality . . . 42
- 4.2 A Review of Methodology . . . 42
  - 4.2.1 Face Detection . . . 42
  - 4.2.2 Comparisons . . . 42
- 4.3 Development . . . 43
  - 4.3.1 Timings . . . 43
  - 4.3.2 Summary . . . 45
- 4.4 Success Criteria . . . 46
  - 4.4.1 List of Core Requirements . . . 46
  - 4.4.2 List of Extension Tasks . . . 46

5 Conclusion **47**
- 5.1 The Feasibility Of Deepfake Detection On A Large Scale . . . 47
- 5.2 Future Work . . . 48
- 5.3 Lessons Learned . . . 48

Bibliography **49**

A Chrome Extension Simulation Pseudocode **51**

B Output of Tests In More Detail **53**

C Chat-GPT Conversations **58**

D Project Proposal **60**

List of Figures

2.1 An example of face-swap that I generated for my talk on deepfake detection. 15
2.2 A method for deepfake generation that uses auto-encoders. 16
2.3 An example of a GAN. 17
2.4 An example of a convolutional neural network that classifies dogs and cats. 18
2.5 An example of a LSTM cell with a forget gate from [1]. 19
2.6 An example of an optical flow calculation, where two frames gathered by the Chrome extension are converted to an RGB image that represents the optical flow. 20
2.7 An example ROC curve. 20
2.8 The ROC curve for a perfect classifier. 20

3.1 Chrome extension being used in a browser. 24
3.2 The REST API. 24
3.3 The operation of the extension through time. 25
3.4 The original optical flow detection pipeline. 30
3.5 The loss (green) and the accuracy (blue) on the validation data through training flow net. 31
3.6 The loss (green) and the accuracy (blue) on the validation data with a single layer of vgg-16 marked as trainable. 32
3.7 The loss (green) and the accuracy (blue) on the validation data with a single layer of vgg-16 marked as trainable and only 1/49 of the training set used. 32
3.8 An example frame from dfdc. 33
3.9 An example frame from the short videos in mediaset. 33
3.10 Ensemble of CNNs on dfdc. 36
3.11 Ensemble of CNNs on mediaset. 36
3.12 MesoNet on dfdc. 37
3.13 MesoNet on mediaset. 37
3.14 Optical Flow (No VGG-16 Layers Trained - 40 epochs) on dfdc. 38
3.15 Optical Flow (No VGG-16 Layers Trained - 40 epochs) on mediaset. 38
3.16 Optical Flow (One VGG-16 Layer Trained - 40 epochs) on dfdc. 39
3.17 Optical Flow (One VGG-16 Layer Trained - 40 epochs) on mediaset. 39

4.1 The speed of each method for varying internet speeds, assuming 0 latency. 41

B.1 (a) Ensemble of CNNs on dfdc. 54

B.2 (b) Ensemble of CNNs on mediaset. . . . 54
B.3 (c) MesoNet on dfdc. . . . 55
B.4 (d) MesoNet on mediaset. . . . 55
B.5 (e) Optical Flow (No VGG-16 Layers Trained - 40 epochs) on dfdc. . . . 56
B.6 (f) Optical Flow (No VGG-16 Layers Trained - 40 epochs) on mediaset. . . 56
B.7 (g) Optical Flow (One VGG-16 Layer Trained - 40 epochs) on dfdc. . . . 57
B.8 (h) Optical Flow (One VGG-16 Layer Trained - 40 epochs) on mediaset. . 57

Acknowledgements

Created with the help of Eric Meissner, Neil Lawrence, and Matthew Ireland.

Chapter 1

Introduction

1.1 Work Completed

For my project, I have created a Chrome extension that attempts to classify videos in the tab as either real or deepfaked. I have created my own classifier based on [2] that uses optical flows to classify. I have also created a simulation that compares the suitability of different implementations of deepfake detectors for use in a Chrome extension, and used the simulation to conclude that no fast, accurate, practical detection method exists that is suitable for the extension.

1.2 Background

When I laid out my project proposal in October, I described how “deepfakes have become more and more prevalant.” In the months since then, AI has only become more mainstream, with the popularity of “Chat-GPT” propelling other AI models, such as “DALL-E” and “Midjourney” into widespread use. With this comes a rise in a new form of misinformation, where fake images and videos are spread around social media, often leading to users thinking completely false narritives to be true. A recent example is a faked image of Pope Francis, which went viral on Reddit and Twitter[3], with the majority of users thinking it to be real[4].

In a bid to quell misinformation, many social media companies have banned deepfakes from their platforms[5, 6, 7]. However, as of yet, automated deepfake detection is not commonplace. For this project, I wanted to find out why this was.

1.3 Motivation

The motivation for this project was to find out whether there exist methods that are both fast and accurate enough to be implemented on these sites when videos are uploaded. As an extension to this goal, I also planned to employ these methods to create a detection program that works on the user’s side, in the form of a Chrome extension.

1.4 Related Work

1.4.1 Deepfake Detection

An extensive survey of the state of deepfake detection is given by Nguyen et al. in [8], and this was used as a base for comparison of techniques. It gives a good description of a variety of deepfake generation and detection methods.

1.4.2 Existing Chrome Extensions

I wanted to implement a Chrome extension both as a proof-of-concept of both found and created detection methods, and to have something that can be used in the real-world created by my project. For the second point, it is worth taking a brief look at those Chrome extensions that already exist.

There are seemingly three extensions in the Chrome web store that detect deepfakes:

- **Fake Profile Detector (Deepfake, GAN)** [9]
  - Specifically targets profile pictures generated by GANs.
  - Doesn't work on videos.
- **DeepfakeProof** [10]
  - Works in the background as a user browses the internet, classifying every image that the user sees and flagging any deepfakes.
  - Doesn't work on videos.
  - Specifically mentions face swaps.
  - I have used this while researching deepfakes for this project over two months and it is yet to alert me of one.
- **Deepfake Detection** [11]
  - Works on videos.
  - Requires the user to have a "Modzy" account.
  - Only works in YouTube.
  - The user clicks to start a recording and clicks again to stop.

1.4.3 Kaggle Deepfake Detection Challenge

In 2020, Meta (then Facebook) challenged over 2000 applicants to create technologies for detecting deepfakes. With this, they created one of the largest publically available deepfake datasets. The competition can be found in [12].

Chapter 2

Preparation

2.1 Requirements

2.1.1 List of Core Requirements

1. Have compared multiple pre-existing methods in speed, accuracy and practicality in theory.[1] ***ACHIEVED***
2. Have compared implementations of some/all of these methods in reality on those three topics. ***ACHIEVED***
3. Create a Chrome extension which can send data from a video to a server. ***ACHIEVED***
4. Create a program that takes that data and classifies the video from which it originates as either a deepfake or not a deepfake. ***ACHIEVED***

2.1.2 List of Extension Tasks

1. Have created a Chrome extension which sends data to a server and presents to the user the result of the server's classification. ***ACHIEVED***
2. Have the extension do this in a fast, accurate, practical way. ***PARTIALLY ACHIEVED***

[1]See Section 2.5.1

2.2 Starting Point

Prior to the project, I had given a talk on deepfake detection as one of the Churchill Compsci Talks, which involved some background reading on the topic. As such I have some base-level background knowledge on deepfake detection. I had never trained or implemented a neural network, my only familiarity with them came from that background reading. I had no knowledge of the workings of Chrome extensions.

2.3 Tools Used

2.3.1 Language and Libraries

Besides the HTML and JavaScript in the Chrome extension, the project is written for Python 3.10 in Visual Studio Code. The following Python libraries were used (built-in libraries excluded for brevity):

- tensorflow
- keras
- matplotlib
- sklearn
- pandas
- numpy
- pillow
- cv2
- glob
- torch
- nbformat
- nbconvert
- flask
- tqdm

In addition, I have used the BlazeFace implementation found in [13] as described in Section 3.3.6.

2.3.2 Google Colab

Google Colab was used for training and for comparing the implementations. This is because it is free to use and has a GPU with CUDA, which is often a requirement for deep learning models.

2.3.3 Dataset

The dataset I used for training was the Kaggle Deepfake Detection Challenge dataset[12]. It is one of the most extensive deepfake datasets publicly available.

There are many problems with this dataset, though. Meta themselves note that models trained on it don't generalise well[14] and the images included are very different from the screenshots sent from the extension (although the faces themselves are a fair representation).

2.3.4 Use of Chat-GPT

Open AI's Chat-GPT was used to quickly generate the structure of the Chrome extension. It was first asked to create a Chrome extension that takes a screenshot and then sends it to a server. After some debugging, it was asked to extend the extension to take two screenshots. It was then asked to leave a 100ms gap between taking the screenshots. Finally, it was asked to change the original extension to instead take a 500ms video.

Here are some examples of problems with the output of Chat-GPT that had to be ironed out:

- It failed to include the "activeTab" permission required by captureVisibleTab.
- It wanted to use manifest version 3 when 2 was required.

The full list of relevant prompts and responses can be found in Appendix C. All use of Chat-GPT has been redacted from the source code. The Python server and the manifest have been completely rewritten, and so have been included.

2.4 Software Engineering Techniques

2.4.1 Testing

While writing my code, I incrementally made sure that each part was doing what it was supposed to by testing them. For example:

- When developing the Chrome extension, I first made sure that I could successfully send a single frame to the server. I checked this by saving the image locally within the server and manually viewing it.
- I checked that I could send and receive messages from the extension to the server by making a version of the server that just sends a 1 (rather than the output of a classifier) as a response to a POST request and displaying this to the user.
- I checked that the optical flow was correctly generated by testing it on images with lots of moving parts and saving the RGB images locally. I made sure that it consistently coloured parts that move in the same direction the same colour.

This incremental approach to developing and testing was used in all of the code I wrote. It allowed me to be sure that all components worked and, where needed, work together.

2.4.2 User Privacy

Taking screenshots or videos of tabs could lead to serious privacy breaches if the data is not appropriately dealt with. The final extension never saves this data to disk, it just passes through the classifier.

2.4.3 Use of GitHub

I made use of GitHub as a backup and version control.

2.5 Terminology

2.5.1 Speed, Accuracy, and Practicality

When making comparisons, I will refer to speed, accuracy, and practicality.

- Speed should include both how quickly a method works on a single system (e.g a twitter data server) and when data from a video has to be sent from a user to a central server, and then processed (e.g in a Chrome extension).
- Accuracy is how well a method detects deepfakes. How likely are false negatives and false positives? How does accuracy change with compute power?
- Practicality relates specifically to how well a method can be incorporated in a Chrome extension. How much storage would the method take up on both user side and server side? Can data sent from the user to the server be reduced to a single frame of video or even less data than a single frame? This second question is particularly important - some methods will work on the assumption that they have the full video. Do these methods actually need the full video, or can we reverse engineer them to work with some data points that are less than the full video?

2.5.2 Deepfakes

As outlined in [8], deepfake has a very broad definition. For the purposes of this project, when I refer to a deepfake, I am referring to a video which has been digitally altered such that the face of a person who was not in the original video appears. The most common of these methods is a method called face-swap, in which a computer-generated face is pasted onto each frame of a video featuring a different subject. An example of a face swap, taken from my deepfake detection talk, is shown in Figure 2.1.

2.5.3 Deepfake Detection

Deepfake detection refers to the binary classification problem of deciding whether a video includes a deepfake or not.

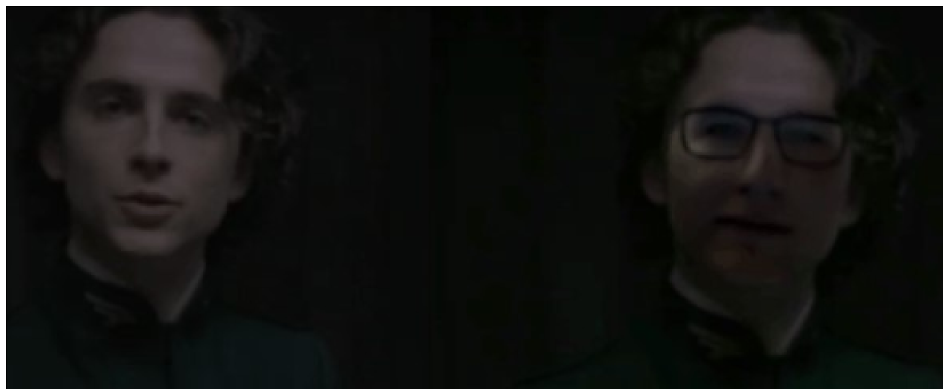


Figure 2.1: An example of face-swap that I generated for my talk on deepfake detection.

2.6 Technical Background

2.6.1 Chrome Extensions

A Chrome extension is a program that can be installed on the Chrome browser to add additional functionality. They are built in HTML, which means that they can be connected to javascript programs and css in the same way as a website.

Chrome Extensions are usually very small, and also should access minimal memory. [15]

2.6.2 Neural Networks

Neural networks come in a variety of shapes and sizes, with different categories being more or less useful for different purposes. This section is a whistle-stop tour of just a few of these categories.

Feedforward Neural Networks

In a feedforward neural network, nodes do not form any cycles. A data point can be fed into the neural network, and the value of each node will be calculated in each layer in turn, ultimately resulting in the calculation of the output layer.

RNNs

In a recurrent neural network (RNN), cycles do exist, but the same basic concept of connected neurons holds. This means that in training, we have to be careful to have both a forward propagation phase and a backpropagation phase, as the output of a future node can affect the value of an earlier node.

Auto-Encoders

Auto-encoders are a widely used method for dimensionality reduction. They learn to embed high-dimension information in lower dimensions and simultaneously how to decode that-dimension information to get back the original high-dimension information.

The idea behind the auto-encoder architecture is to simultaneously train two components (typically two neural networks): an encoder, which compresses a high-dimension image into a smaller space, and a decoder, which converts the simplified format back to the input format.

Auto-encoders were the first way in which many deepfake generators were trained[8]. Figure 2.2 shows how this was implemented.

The encoder is a general network, shared across all target faces, whereas the decoder is specific to a single person. Some deepfake generators come with pre-trained encoders so that the training phase only needs data from a single person.

To use the auto-encoder, we feed a datapoint from person **A** to the encoder and use **B**'s decoder. This will result in a datapoint that has the latent features of the input but with the specific features of **A**. For example, in Figure 2.2, we pass a smiling face from person **A** into the decoder and use **B**'s decoder to generate a new smiling picture of person **B**.

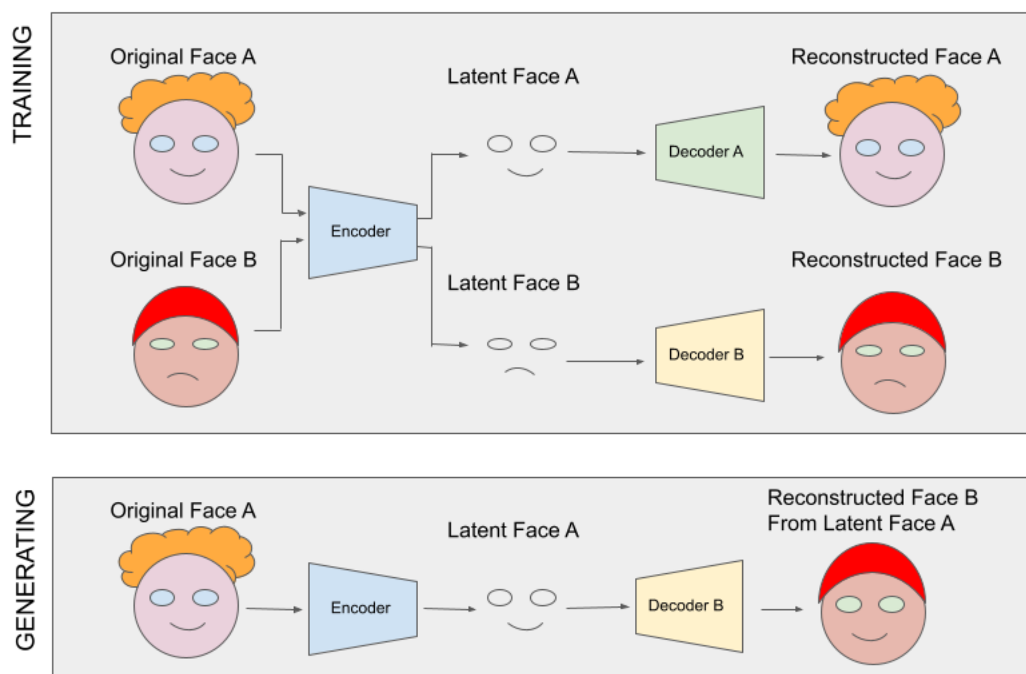


Figure 2.2: A method for deepfake generation that uses auto-encoders.

GANs

An alternative way to generate deepfakes is to use a generative adversarial network (GAN). Similarly to an auto-encoder, GANs also train two neural networks at the same time: a generator and a discriminator. The goal of the generator is to convert noise into fake images that are similar to a real dataset. The goal of the discriminator is to tell the difference between the fake images and the real images. In this way, the neural networks battle each other in a zero-sum game, both improving in the process. This is shown in Figure 2.3.

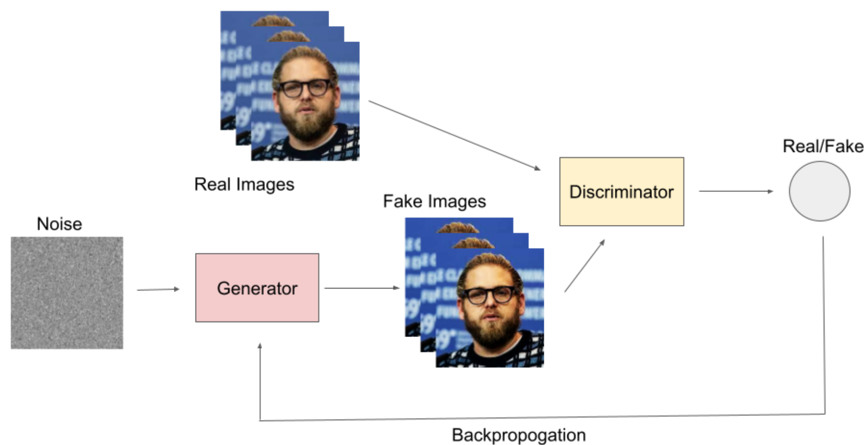


Figure 2.3: An example of a GAN.

Note that the standard GAN architecture doesn't create "deepfakes" by the definition that we are using in this paper (a video of a person which is manipulated to have a new face) but it is often used as a building block in a larger deepfake generating pipeline. In [16], the GAN architecture is extended to use latent faces to create deepfakes that more closely match our definition with a structure called Style-GAN.

CNNs

A convolutional neural network (CNN) is a type of feedforward neural network that was designed for use on images, to compare local data points in a (usually) two-dimensional data structure. To be a CNN, a neural network must contain at least one convolutional layer. A convolution is achieved by "sliding" a $n \times n$ filter (kernel) across a vector, moving a distance called the stride after each calculation, to create a vector that is n-stride rows and n-stride columns smaller.

CNNs usually also have pooling layers, in which the 2D vector is split into $k \times l$ smaller rectangles and a calculation is performed on each rectangle to give a single value, resulting in a $k \times l$ vector. The most commonly used pooling method is max pooling, where the maximum value in the rectangle is taken.

Another layer that may be included is a flatten layer, in which a 2D vector is turned into a long 1D vector. A flatten layer has no weights, it is just a reshaping of the data.

In classifiers, this is usually fed into one or more fully connected layers where every input is connected to every output.

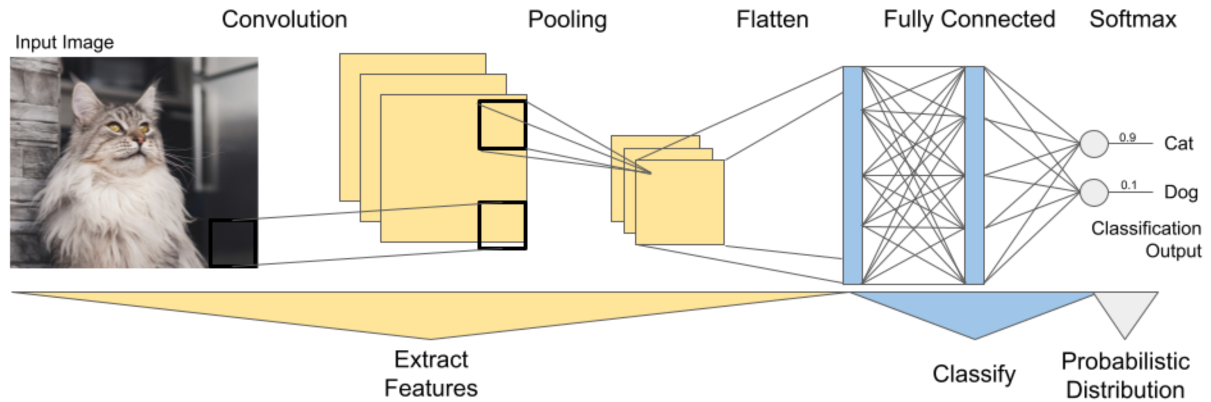


Figure 2.4: An example of a convolutional neural network that classifies dogs and cats.

LSTM

Long short-term memory (LSTM) is a type of recurrent neural network, meaning that it has feedback connections allowing it to process not just frames but videos by "remembering" parts of earlier frames. The particular advancement of LSTM over standard RNNs is the use of gate which has been taught which features to remember and which to forget.

Figure 2.5 shows how a particular LSTM cell takes as input the previous output h, the state of the memory c, and the input to the network x, and emits the output at time t while feeding into the cell at the next timestep that output and also the updated memory state.

The specifics of LSTM's inner workings are not required for this project, but more detail can be found in [17].

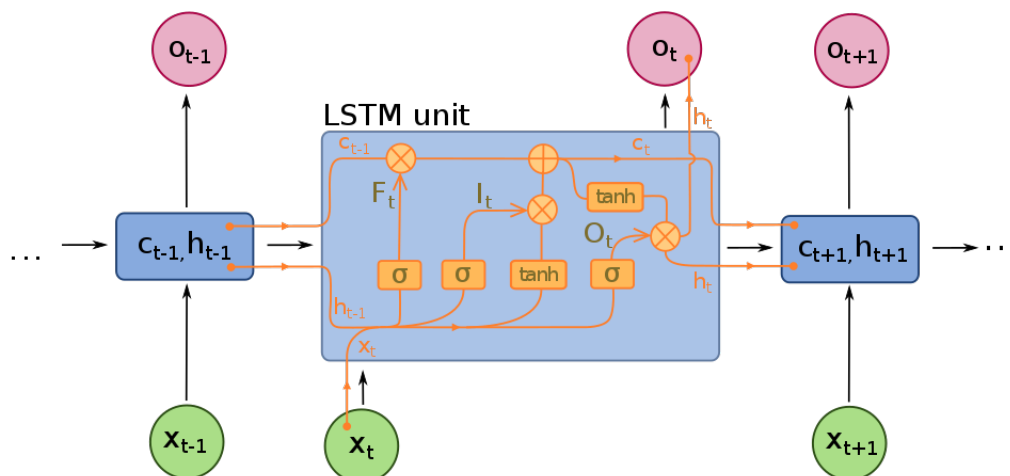


Figure 2.5: An example of a LSTM cell with a forget gate from [1].

2.6.3 Optical Flow

Optical flow is a vector representation of the movement of a particular pixel. It is calculated by comparing two consecutive frames in a video. A visualisation of optical flow is shown in Figure 2.6.

2.6.4 ROC and AUC

A receiver operating characteristic (ROC) curve is a way of visualising how effectively a binary classifier can make predictions. The curve is set against two axes: the false positive rate (also known as the sensitivity, the ratio of actual positives that are correctly predicted as positives) and the true positive rate (specificity, the ratio of actual negatives that are incorrectly predicted to be positive).

To understand this, it helps to think about the problems with using accuracy as a test statistic for a classifier. A binary classifier doesn't have a binary output: instead, it outputs a probability of the input data having a particular property. In our case, it outputs the probability of a face being a deepfake. As such, when we generate accuracy, we have to choose which outputs represent a positive and which outputs represent a negative. It is natural to set the boundary at 0.5 (as we do later on when we calculate the accuracy test statistic), but this doesn't discriminate between a classifier that puts all fakes at 0.75 and all reals at 0.25, and a discriminator that puts all fakes at 1.0 and all reals at 0.0, even though the latter is a better classifier.

This is where we can use a ROC curve. Consider the example in Figure 2.7. As we move from the bottom left to the top right, we are moving the boundary for what we consider to be a positive output and what we consider a negative output. At the extremes, at the bottom left we consider every output to be a negative. As such, we have a false positive rate of 0 and a true positive rate of 0 (we never predict positive). At the top right, we

Figure 2.6: An example of an optical flow calculation, where two frames gathered by the Chrome extension are converted to an RGB image that represents the optical flow.

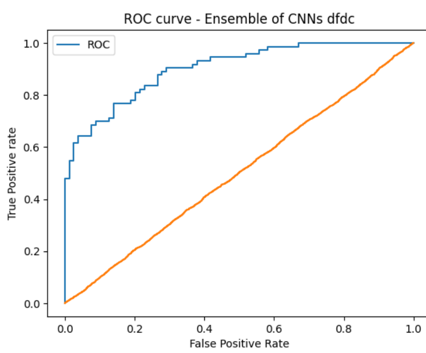


Figure 2.7: An example ROC curve.

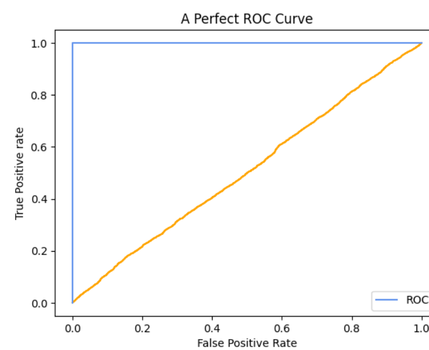


Figure 2.8: The ROC curve for a perfect classifier.

are doing the opposite, predicting every output to be a positive, so both the true positive rate and the false positive rate are 1.

The orange line in Figure 2.7 represents a classifier randomly (uniformly) outputting values between 0 and 1. It is always equally likely for a true positive to occur and a false positive to occur. What we want from a classifier is for true positives to be more likely than false positives. As such, the ROC curve being above the orange line is good. If the curve dips below the orange line, it means that we are classifying worse than random chance. If we are significantly below the line then it is likely that we have made a mistake somewhere.

If the classifier were to always output 1.0 for a real positive and always output 0.0 for a real negative, we would get a graph that looks like Figure 2.8. In reality, classifiers will fall between the two lines in Figure 2.8 in almost all cases.

If we want to compress these curves into a single statistic, we can use AUC (area under curve). This refers to the area found under the ROC curve. Intuitively, we can think of this as being similar to the integral of the accuracy with respect to a boundary from 0 to 1. More correctly, though, it is the integral of the true positive rate with respect to the false positive rate.

An AUC of 0.5 represents random chance and an AUC of 1.0 represents a perfect classifier.

More detail on ROC and AUC can be found in [18].

Chapter 3

Implementation

3.1 Outline

In the rest of this chapter, I describe how I attempted to implement a Chrome extension that classifies deepfakes. Table 3.1 is a summary of each section.

Section	Summary
3.2 Chrome Extension	**Goal: Create a Chrome extension that classifies deepfakes** At the highest level of abstraction, I wanted to create a Chrome extension that tells a user whether there is a deepfake. In Section 3.2, I describe how the extension gives the user a button they can press to get a number representing the chance of a deepfake being present.
3.3 Classifier Options	**Goal: Find a classifier for deepfake detection** I needed to choose a classifier for the extension, so in Section 3.3 my aim was to compare, at least on paper, possible methods that could be used to detect deepfakes. This led me to four options, only two of which I had access to implementations of, neither of which used temporal features. To allow a more complete comparison of methods, I wanted to implement a method that did use temporal features.
3.4 Optical Flow Based Detection	**Goal: Implement a classifier that uses optical flow to detect deepfakes** The method I chose to implement was Optical Flow, and Section 3.4 outlines how I did this.
3.5 A More Appropriate Dataset	**Goal: Create a dataset that better suits the extension** I wanted to compare how well the implementations would work in the extension but the dfdc dataset wasn't designed to replicate videos uploaded to social media, let alone the data captured by the extension. Hence, in Section 3,5, I created a dataset based on samples taken directly from the extension. A classifier that performs better on this dataset would be likely to produce better results for the user when used in the extension.
3.6 Comparison	**Goal: Compare how well the classifiers would do if used in the extension in reality** I now had a set of implementations I could compare and a dataset to compare them on, so I could commence testing them in a simulation. Section 3.6 describes the creation of the simulation and the results of the simulation when run on the implementations.

Table 3.1: An overview of this chapter.

3.2 Chrome Extension

In this section I outline the structure of the Chrome extension. The goal is to have the user press a button and receive a number representing the likelihood that the image on the page is a deepfake. This is shown in Figure 3.1.

Due to the typically small size of Chrome extensions (usually no more than 50MB), compared to the large size of deepfake classifiers (more like 500MB), the amount of memory used by classifiers (many hundreds of MB), and the massive speed boost given by access to a GPU (which users may not necessarily have), it is appropriate to host the classifier on an external server. For this, I used Python's Flask, which uses the REST API (see Figure 3.2).

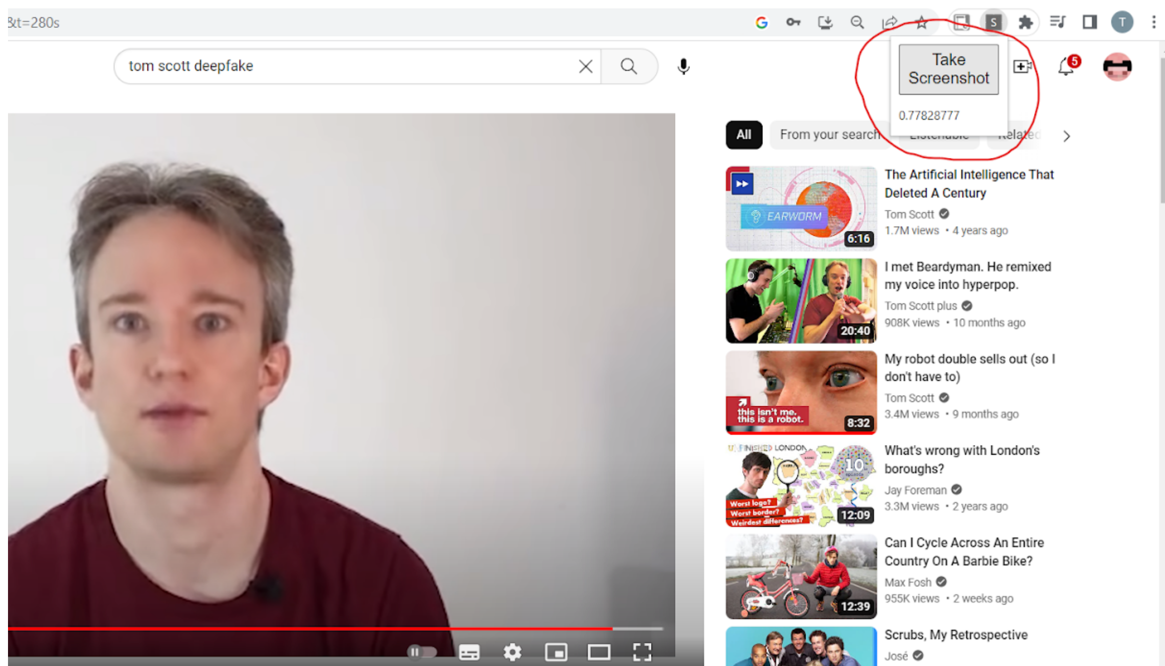


Figure 3.1: Chrome extension being used in a browser.

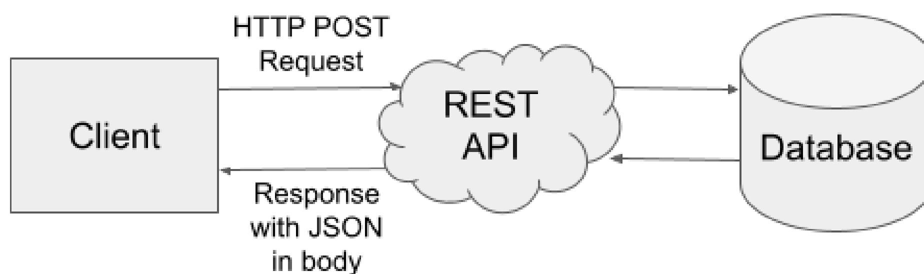


Figure 3.2: The REST API.

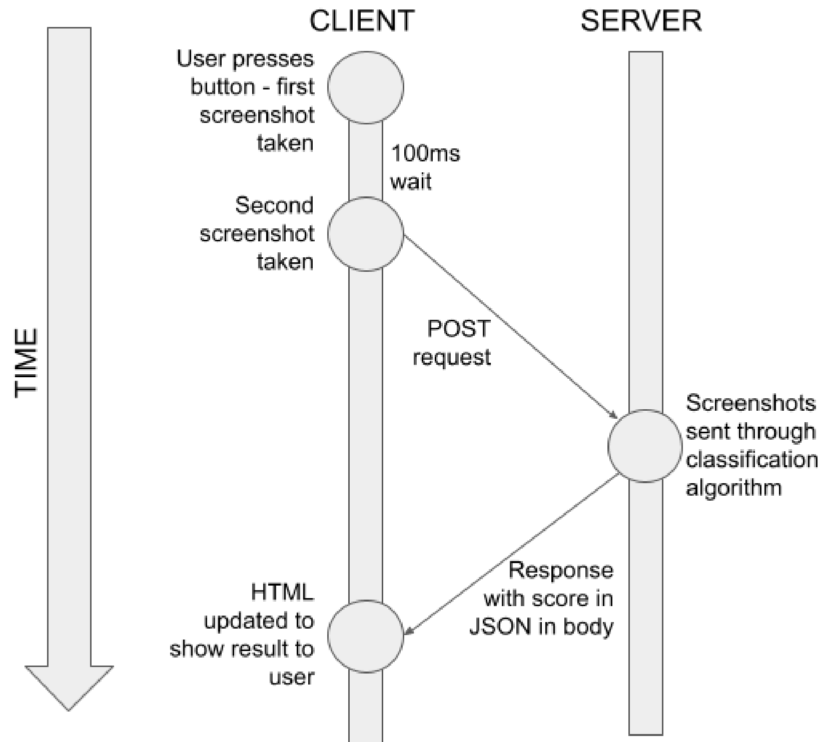


Figure 3.3: The operation of the extension through time.

I will now go through the extension first from the user's perspecrtive, then from the perspective of the extension, and finally from the perspective of the server.

User

Use of thee deepfake is shown in Figure 3.1. The user presses the Take Screenshot button, the data is collected, and, after a brief wait, the user is presented with a number that represents how likely it is that the image on the tab is a deepfake.

Extension: User Side

This button has an Event Listener that triggers the "takeScreenshots" function, which begins the rest of the process. Upon receiving the response, the HTML is updated to show the result.

The function captureVisibleTab is used to take a screenshot. This function is restricted to two applications per second, limiting how often the extension can be used.

We take two screenshots 100ms apart, with the assumption that the frame rate will be at least 10 per second, Any less than this and there is no guarantee that the extension captures two separate frames. If the frame rate is faster than 20 per second, there is also a chance that the frames captured won't be consecutive. This matters less, though, as there will still be an optical flow calculation.

captureVisibleTab takes screenshots of the entire tab, so if there is more than one face or more than one image in the tab, we won't necessarily get the correct face. From the user's perspective, this is implicit. When the user presses the button, they aren't specifying a particular part of the screen. Currently, the face detection algorithm chooses the part of the screen most likely to be a face - an alternative implementation could use the largest face, the most central face, or could ask the user to specify a face (by, for example, clicking a particular part of the screen to identify the face they wish to be classified).

One advantage of captureVisibleTab is its ability to capture "sensitive sites", such as the pages of other extensions [19]. This is unlikely to be used in practice, though, as the list of such sites is relatively small.

There is also an extension that records a 500ms video using MediaRecorder [20]. The advantage of this is that we can guarantee two consecutive frames are fed to the classifier. However, it means approximately 10 times the data being sent than a single image, and so isn't really practical. For that reason, I have stuck with methods of detection that use one or two frames, but it is worth noting that the videos in mediaset, which is outlined in Section 3.5, were recorded with the MediaRecorder API.

Extension: Communication With Server

The data, in the form of a FormData object that is the body of a HTTP POST request, are sent as the body of the request, as per the REST API. The server sends back a json with a single element with the score that we want to be displayed to the user as the response. There is more about this stage later in this chapter.

Server

The server uses whatever method we are using to classify deepfakes, resulting in a number from zero to one, where zero represents real and one represents fake. In the next section we look at a variety of ways we can get this number.

It is worth noting that this number is not a probability, rather the output of a sigmoid function. That is, while the outputs do not follow a probability density function, a value closer to one does mean that the input was more likely to be a fake.

Use of Chat-GPT

As mentioned in Section 2.3.4, I used Chat-GPT to generate an example Chrome extension that sent (a) a single frame to a server, (b) two frames to a server, and (c) a video to a server. It generated javascript, HTML, a manifest, and a Python server in all three cases. The server has been completely replaced by two I've created, one that classifies deepfakes and another that was used to generate mediaset (see Section 3.5). The manifest

has also been rewritten in both cases. The javascript and HTML, however, have gone almost unchanged, and so have been redacted in the source code submitted alongside this dissertation.

3.3 Classifier Options

In this section, I look at a selection of methods for deepfake detection, and compare how well they could be used on the server to classify deepfakes.

3.3.1 MesoNet

MesoNet [21] is a relatively early attempt to use a CNN to detect deepfakes by Afchar et al. It claims a 90% detection rate on single frames simply by chaining convolutions together. Since this is an image-based method, it is both fast and practical - it can work in a Chrome extension that takes a single screenshot.

3.3.2 Ensemble of CNNs

In [22], a later attempt to detect deepfakes with a CNN, a small tweak is made to the pre-existing EfficientNetB4 network to include an attention mechanism that focuses on particular features of the input. This allowed the authors to find which parts of an image are particularly useful when detecting deepfakes - the eyes, nose, and mouth tended to be selected by the attention network.

In the paper, the method was able to achieve a detection rate of around 88% on the dfdc dataset described in Section 2.3.3.

3.3.3 Recurrent Neural Networks

Various methods for detecting deepfakes are described in [8], along with the split between classifying an image and classifying a video. Videos are typically more compressed than images - particularly when detecting on social media, which is the motivation for this project - and so methods specifically designed for images are less likely to be able to detect deepfakes successfully in videos. Furthermore, videos may include inter-frame characteristics, which methods designed for images would not be able to make use of.

In [17], temporal features are exploited by making use of LSTM in their detection architecture. The video frames are each fed into a CNN like those in the two previous methods. These new vectors are then fed through a convolutional LSTM, giving a vector that describes the entire sequence. This is fed through a fully connected layer and a softmax activation function to classify. With this they achieved an accuracy of 97.1%.

3.3.4 Optical Flow

The main issue with RNN based methods is that they require an entire video to work effectively. The method proposed in [2] is to calculate optical flow and feed that into a CNN. This allows us to analyse temporal features while only doubling the amount of data that needs to be fed into the network. The reported accuracy of this method is 81.61%.

3.3.5 Pre-Existing Implementations

Of the methods described in this section, only MesoNet and Ensemble of CNNs have publically available implementations. Since I wanted to compare to a method which included some temporal analysis, I chose to implement one of the other methods myself, even though the success criteria and extensions only dictate that I use pre-existing implementations. I went for optical flow as the RNN-based method was tested on at least 20 frames of video in the paper, which is impractical for the Chrome extension as it would take too long to collect and then send this length of video over the internet. My implementation of optical flow based detection is outlined in the next section.

3.3.6 Face Detection

All of the above methods start by cropping the image to a face. To do this, we need a face detection algorithm. The Ensemble of CNNs implementation includes a face detector based on BlazeFace [23], which uses a lightweight CNN to find faces extremely quickly.

I have used this to process frames as input to all of the implementations. In doing this, I had to make some specific decisions about how I was going to use the output of BlazeFace.

Which Face?

Firstly, I had to decide which face to use if the detector found more than one. The function outputs a (possibly empty) list of faces in order of the probability that they are actually a face. I decided to use the face that the detector decided was most likely to be a face, but this isn't necessarily the face that the user intended to classify.

This is particularly a problem if there are many faces on the screen, as there would be on social media platforms. The intuition behind the decision I made was that the user most likely meant the clearest face, and that was likely to be the one that the detector was most certain was a face.

Alternative approaches could have seen me use the biggest face, test all faces and return either the probability of any of them being a fake or a list of probabilities with one for each face, or ask the user to identify which face they meant.

Cropping or Scaling?

The classifiers all require a specific shape of vector as input. For example, MesoNet requires a 256×256 pixel image. There are two ways to get this: either draw a box of the correct size around the center of the box returned by the detector or transform by re-scaling the image to the correct size.

The advantage of the latter is that we know that the face will fill the box in its entirety, minimising interference from objects other than the target face, like other nearby faces or objects in the background of the video. It also means that parts of the face like the eyes, nose, and mouth are more consistently in the same part of the image.

The main disadvantage is that, if the box is far from square, there could be significant distortion of the shape of the features, which some classifiers may misinterpret as an artefact of the deepfake process.

Further, for small kernels in a convolutional layer (3×3 or close to that), the distortion means that adjacent pixels weren't necessarily adjacent in the original image - for small faces they could even be the same pixel repeated 9 times. This would certainly affect the validity of the output of the convolution layer.

The former method of adjusting the crop has the inverse of these problems - there is no distortion but we no longer know where the face features are and we cannot guarantee that the face will be framed well. This is most noticeable when faces are very large (we may not have the entire face) or very small (we may have many faces within the cropped image).

In the end I chose cropping as I was concerned about how scaling would affect optical flow in particular. The most important thing was to make the input to the classifier match as closely as possible the training data, so I made sure to use the same method (cropping) for both. I also used this method when evaluating MesoNet and Ensemble of CNNs in Section 3.6.

3.4 Optical Flow Based Detection

I have implemented my own version of the optical flow based detection method outlined in Section 2.6.3. The thinking was that such a method would lend itself to use in a Chrome extension, as it benefits from many of the advantages of video-based methods, while only doubling the amount of data being sent to the server (two screenshots rather than one). It is also relatively easy to implement, as opencv has an optical flow calculator (in particular we used the Farneback method) and the Ensemble of CNNs implementation has a face detector, so it is simply a matter of assembling these components.

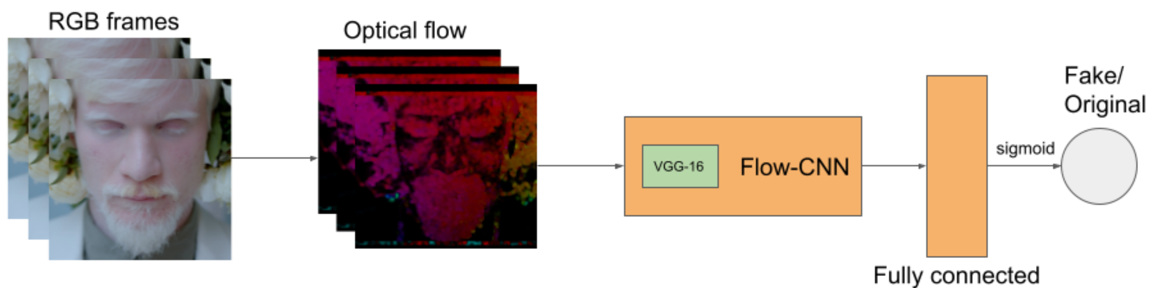


Figure 3.4: The original optical flow detection pipeline.

Shown in Figure 3.4 is the pipeline described by [2]. VGG-16, a pre-trained model used in image classification, is used to get a 1000×1 feature vector, which is then used in a soft-max (sigmoid) layer to classify the flows. This means that there are only 1001 parameters to be trained. In the paper, Flow-CNN is described as a "semi-trainable" network with VGG-16 as its base, but it is not specified precisely which layers are to be trained. To start with, I assumed that no layers in Flow-CNN could be trained.

3.4.1 Managing The Dataset

I decided to train on the dfdc dataset outlined in 2.4.2 for its massive size. The basic idea was to first generate a cropped RGB flow image for every 60th pair of frames in the dataset, and then use tensorflow to train based on these flows. This can be seen in generate_optical_flows_dataset_rgb.py.

The dfdc dataset is extremely imbalanced, with around 10 times as many fake videos as real. As such, I randomly skipped a portion of the fake videos when generating the flows. I also skipped pairs where a face couldn't be found. The resulting dataset has a total of 185662 flow images.

3.4.2 Training

The model, as in the original paper, was trained using keras with Adam optimisation. The results on the validation data are shown in Figure 3.5. The accuracy on the validation data after 40 epochs was 58%.

Note the decrease in validation accuracy while validation loss decreases. This shouldn't happen, and means that there is something wrong with the methodology. This is likely to be, at least in part, due to not allowing any part of VGG-16 to be trained.

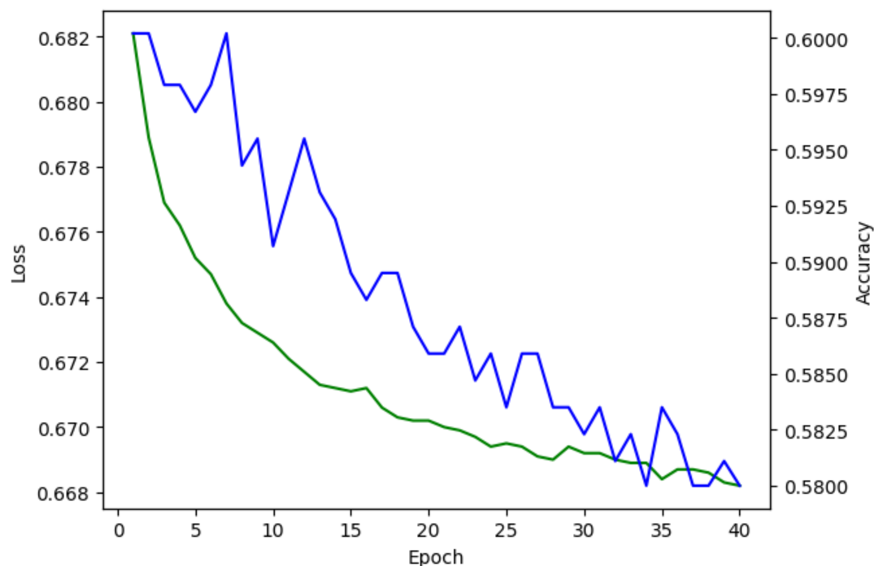


Figure 3.5: The loss (green) and the accuracy (blue) on the validation data through training flow net.

3.4.3 One Trainable VGG-16 Layer

We now look at what happens when we allow ourselves to train the final layer of VGG-16 (and thus making it semi-trainable as in [2]. The loss and accuracy on the validation data can be shown in Figure 3.6. I made the decision to decrease the learning rate by a factor of ten from 0.001 to 0.0001 as a possible way to counteract over-training. Looking at Figure 3.6, though, the accuracy on the validation set still has a downward trend (although a less clear one).

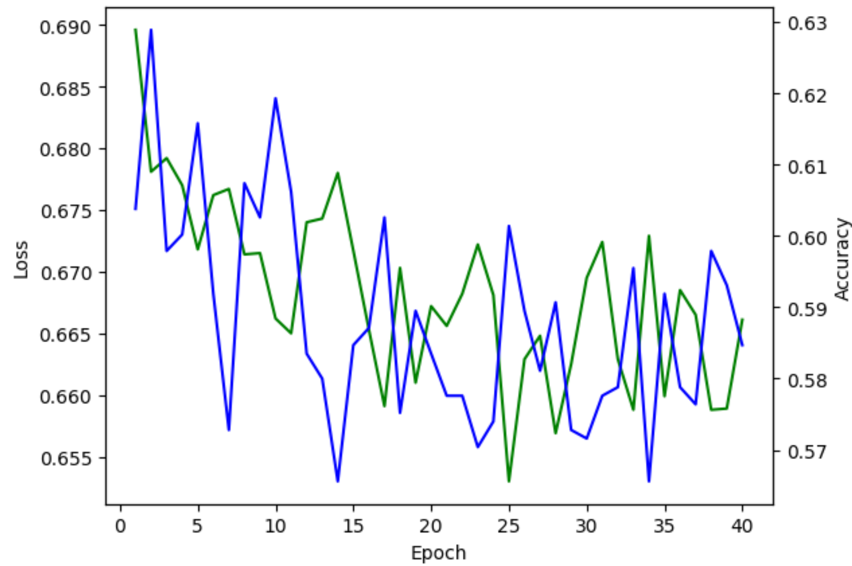


Figure 3.6: The loss (green) and the accuracy (blue) on the validation data with a single layer of vgg-16 marked as trainable.

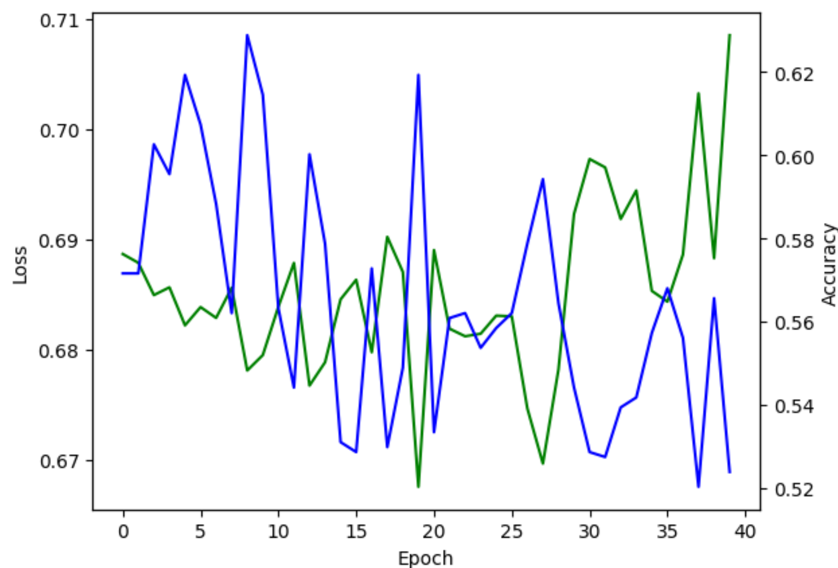


Figure 3.7: The loss (green) and the accuracy (blue) on the validation data with a single layer of vgg-16 marked as trainable and only 1/49 of the training set used.

I also tried training on a much smaller training set - just using flows from dfdc_train_part_1 and so creating a training set approximately 1/49 of the size. The loss and accuracy on the validation set are shown in Figure 3.7 - the accuracy still tends slightly downward and the loss arguably tends slightly upwards, although both of these trends are small enough that I cannot be confident in them.

3.5 A More Appropriate Dataset

I already touched on the known issues of the dfdc dataset in Section 2.3.3. Beyond this, it is also not a very representitive dataset of the sort of information sent to the server in the Chrome extension. Screenshots taken by the user will be of lower quality (due to compression on social media), have a variety of aspect ratios, and be very busy - with the entire website being sent to the server rather than just the video.

For these reasons, I decided to create my own dataset, with short (500ms) videos taken in the same way that the Chrome extension will record data. I've called this dataset ***mediaset***. It contains 50 faked videos and 50 real videos from a variety of social media platforms. An example is shown in Figure 3.9.

Each video was manually classified according to its context within the page. mediaset is based on videos uploaded publicly to social media. It is not used in the training of any model and so neither the dataset nor any model trained on it will be made commercially or even publicly available. It is merely used as a representative sample of the type of videos likely to be fed to the extension.

Figure 3.8: An example frame from dfdc.

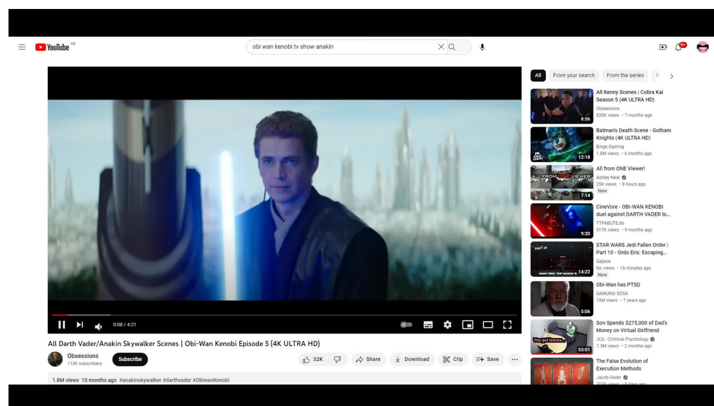


Figure 3.9: An example frame from the short videos in mediaset.

3.6 Comparison

Based on the pipeline of the Chrome extension, I also built a program that returns both average response time and accuracy of a particular method, based on the type of data sent to the server and time for the method itself to run.

The simulation splits the journey from a screenshot being taken to a score being shown into its component parts, with four different functions being passed to the simulator function:

- A dataprocessing function, that allows us to process the data from the format that is passed to the simulation to the format collected by the extension. For example, when testing MesoNet with mediaset, we used the paths to the video files as the data, and converted those to frames from the videos in dataprocessing. It doesn't count towards the time taken.

- A preprocessing function that represents what happens to the data between the user pressing the button and the data being sent to the server. We used the identity function for all of our tests, but one could consider a method that, for example, converts to optical flow on the client side.

- A server_func function that represents what happens to the data on the server side to produce a datapoint that is to be sent back to the user. In all of our uses, this was the classifier function.

- A postprocessing function that represents anything that has to happen to the data on the client side before a result of either real, fake, or inconclusive is shown to the user.[1]

We calculate the response time by summing the time taken by the last three functions, a user-inputted latency, and the time taken to transfer data to the server and back (calculated according to a user-inputted internet speed).

A pseudocode representation of the simulation can be found in Appendix A.

[1]This doesn't precisely match the Chrome extension as outlined in Section 3.2, which happily shows the user a decimal between 0 and 1 rather than an outright prediction, but this function allows us to count the success of the classification by matching the format of the test labels.

3.6.1 Results

	Test Accuracy[2]		AUC[3]		Mean RTT[5]	Average Size of Data Transfer
	dfdc[4]	mediaset	dfdc	mediaset		
MesoNet (Single Frame)	49.7%	49.0%	0.49	0.52	0.12s	6220944B (6.22MB)
CNN Ensemble	68.4%	47.0%	0.90	0.57	0.069s	6220944B (6.22MB)
Optical Flow No Trained VGG-16 Layers 1 epoch	53.7%	42.0%	0.54	0.47	0.15s	12441888B (12.44MB)
Optical Flow No Trained VGG-16 Layers 20 epochs	51.4%	48.0%	0.55	0.48	0.15s	12441888B (12.44MB)
Optical Flow No Trained VGG-16 Layers 40 epochs	51.4%	51.0%	0.54	0.49	0.15s	12441888B (12.44MB)
Optical Flow 1 VGG-16 Layer Trained 5 epochs	57.1%	45.0%	0.55	0.46	0.15s	12441888B (12.44MB)
Optical Flow 1 VGG-16 Layer Trained 20 epochs	50.1%	47.0%	0.57	0.47	0.15s	12441888B (12.44MB)
Optical Flow 1 VGG-16 Layer Trained 40 epochs	57.1%	50.0%	0.61	0.49	0.15s	12441888B (12.44MB)

Table 3.2: The output of the comparison.

The output of the simulation for eight different implementations of four methods can be found in Table 3.2.

In the rest of this section, we can see the outputs of each version of the neural networks on a selection of histograms. The charts can be seen more clearly in Appendix B.

A perfect network would have a single blue tower at 0.0 and a single orange tower at 1.0, representing an output of 0 for every real and 1 for every fake. Any network doing

[2] Assumes > 0.5 means fake and < 0.5 means real. Failure to find a face counts as an inaccuracy.

[3] See Section 2.6.4.

[4] dfdc here refers to the first frame (or first flow) of each video in a balanced version of dfdc_train_part_0, which has in total 177 videos. This is not the same as the validation dataset used in training, which is larger and includes every 60th frame.

[5] Calculated within google colab (each with the same GPU settings) and on dfdc_train_part_0. Does not include data transfer.

better than random chance during training should move towards this, as a marginal increase from 0.51 for fakes and 0.49 for reals to 0.52 and 0.48 will result in a lower loss. Conversely, if detecting randomly, that push towards the edges will serve no advantage, so in training weights will be pushed both out and in equally, and we would expect a normal distribution centered at 0.5.

Ensemble of CNNs

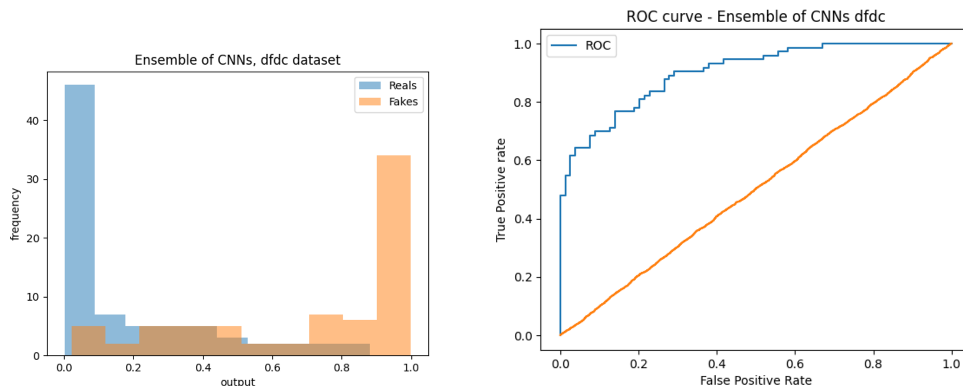


Figure 3.10: Ensemble of CNNs on dfdc.

Figure 3.10, which shows Ensemble of CNNs tested on data taken from dfdc, serves as a good representation of what we are aiming for, with a high volume of outputs close to the correct extremes.

Unfortunately, [22] used the dfdc dataset to train their model, so these results are of little to no value as a representation of the capabilities of Ensemble of CNNs.

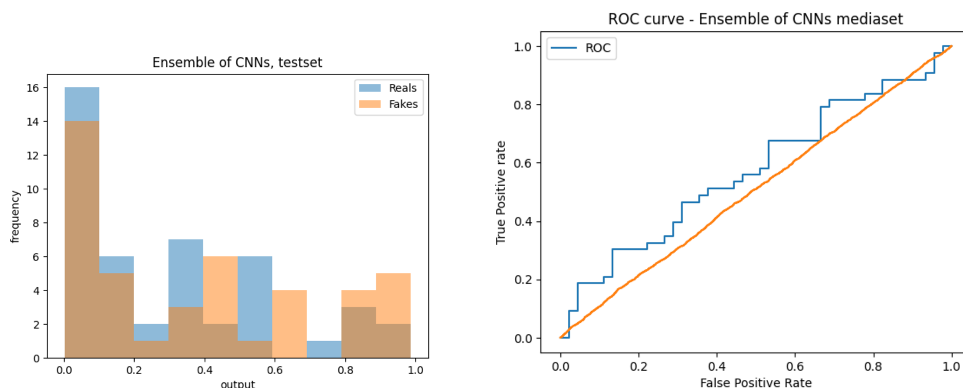


Figure 3.11: Ensemble of CNNs on mediaset.

On mediaset in Figure 3.11, a dataset that wasn't seen in training, the network is less sure about classifications. Note that at the extremes it is still more likely correct than incorrect. Note also from the histogram that Ensemble of CNNs seems to tend towards labelling as real on mediaset. The AUC here was 0.57, making it the only method that did better than random chance on both datasets.

MesoNet

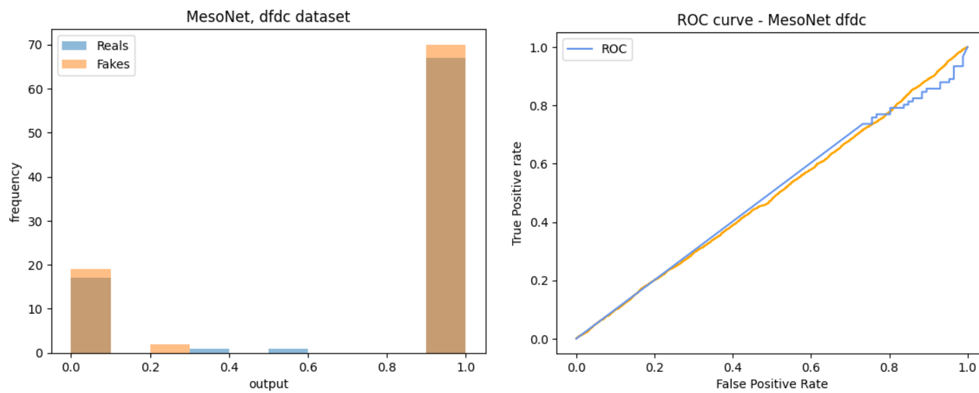


Figure 3.12: MesoNet on dfdc.

The histogram for MesoNet also pushes towards extremes, suggesting that on its training data it was able to classify better than random chance. However, it is unable to distinguish between reals and fakes on the dfdc data. In fact, with an AUC of 0.49, it is the only method to do worse than random chance on dfdc.

It is worth noting that MesoNet was created in 2018, before the dfdc dataset existed and, according to the authors of [21], before any standard deepfake dataset existed.

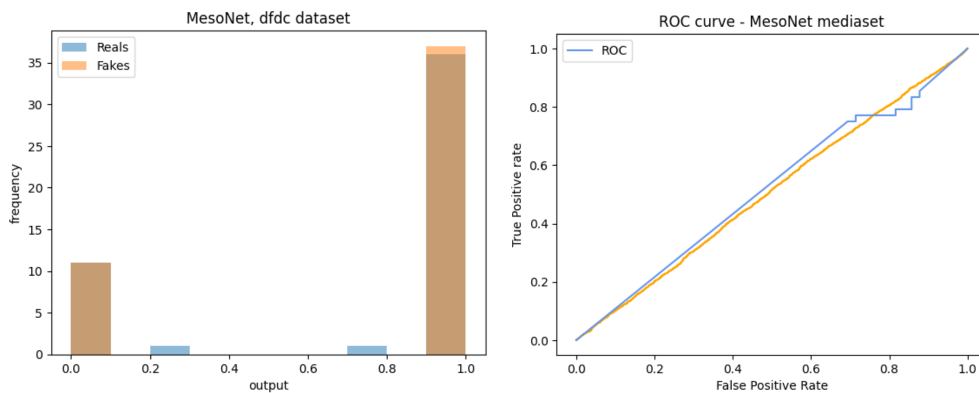


Figure 3.13: MesoNet on mediaset.

Unsurprisingly, MesoNet is also no better than random chance on mediaset, producing similar graphs.

Optical Flow (No VGG-16 Layers Trained)

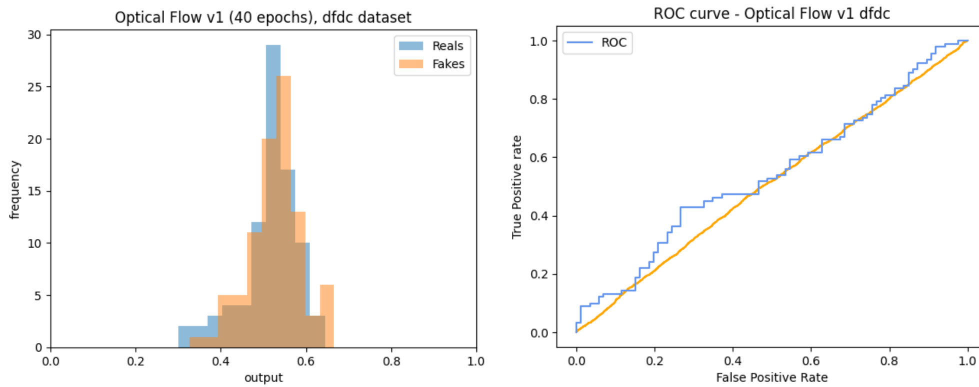


Figure 3.14: Optical Flow (No VGG-16 Layers Trained - 40 epochs) on dfdc.

In Figure 3.14 we see an example of what happens when we have a method that doesn't work for detecting deepfakes. In the histogram, we can see a peak forming in the centre rather than two at the sides. It seems that there is no way to classify on the 1000 features produced by VGG-16 - or, at least, 40 epochs are not enough for us to find a way to classify from those features better than random chance.

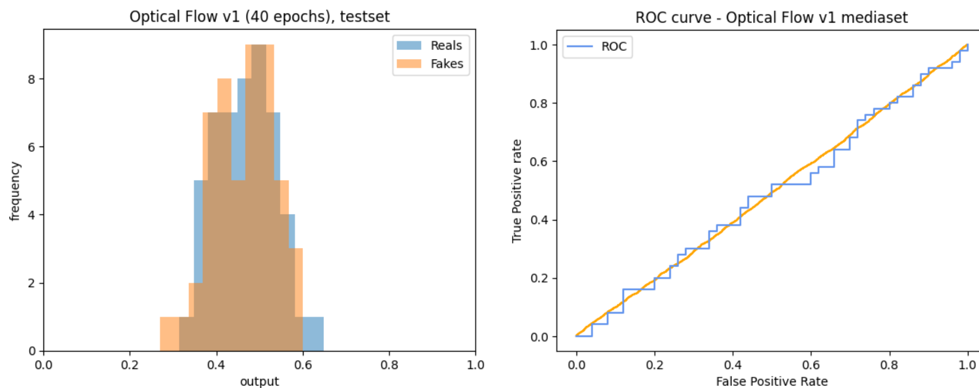


Figure 3.15: Optical Flow (No VGG-16 Layers Trained - 40 epochs) on mediaset.

The charts based on mediaset look similar to those from the dfdc dataset. This is what we would expect if the classifier had no significant difference to random guessing.

Optical Flow (One VGG-16 Layer Trained)

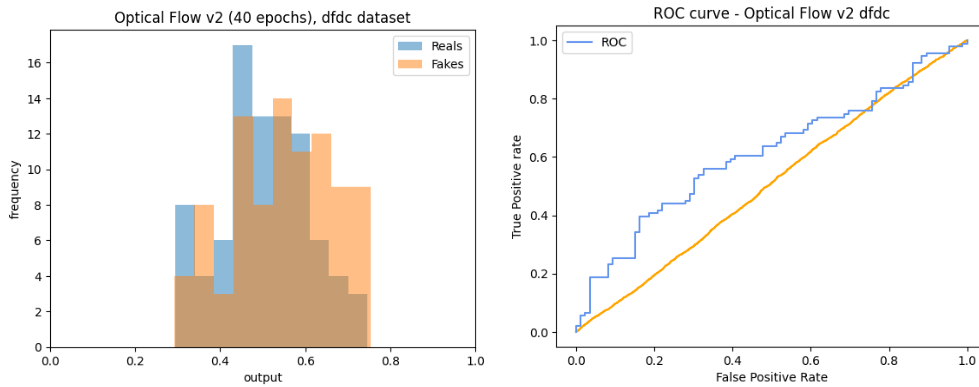


Figure 3.16: Optical Flow (One VGG-16 Layer Trained - 40 epochs) on dfdc.

Now that we've made VGG-16 "semi-trainable", the results are more promising, with a noticeable hump in the ROC curve and an AUC of 0.61.

The histogram also looks better, with a more square distribution than the previous method. This means that classifications are being pushed to the edges rather than held at the centre. Note that, with the reduced learning rate mentioned in Section 3.4.3, 40 epochs hasn't been enough for the distribution to reach the edges of the chart.

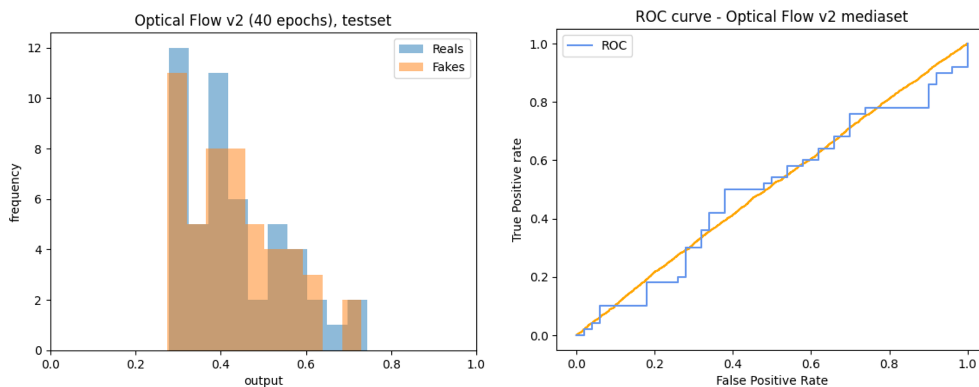


Figure 3.17: Optical Flow (One VGG-16 Layer Trained - 40 epochs) on mediaset.

On mediaset, though, we weren't able to achieve results better than random chance. It is possible that this is due to there not being enough training, or could be due to the worse quality of the videos in mediaset.

As in Ensemble of CNNs, we are more likely to label videos in mediaset as real. The reason for this phenomenon is unclear, but could just as easily be a fault with mediaset as a product of the operation of the classifiers.

3.6.2 Significance Testing

With an alpha of 0.05, we can use the inverse binomial distribution to see that we need greater than 55.9% accuracy on dfdc and 58% on mediaset to be doing better than random chance. Only CNN Ensemble and Optical Flow achieve this, and only on data taken from the dataset that they were trained on.

3.7 Repository Overview

```
root
├─ optical_flow_deepfake
│  └─ __init__.py
├─ screenshot-extension-made-for-two
│  ├─ manifest.json
│  ├─ popup.html
│  └─ script.js
├─ video-extension-for-generating-mediaset
│  ├─ manifest.json
│  ├─ popup.html
│  └─ script.js
├─ training and testing
│  ├─ ensemble_of_cnns_test.py
│  ├─ flow_net_v2_test.py
│  ├─ flow_net_v2_training.py
│  ├─ generate_graphs.py
│  ├─ mesonet_test.py
│  └─ README.txt
├─ generate_optical_flows_dataset_rgb.py
├─ server-for-two.py
└─ server-to-collect-mediaset.py
```

Chapter 4

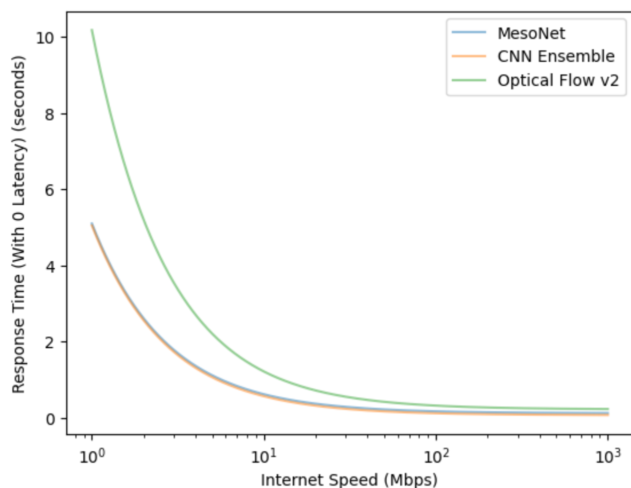
Evaluation

4.1 The Chrome Extension

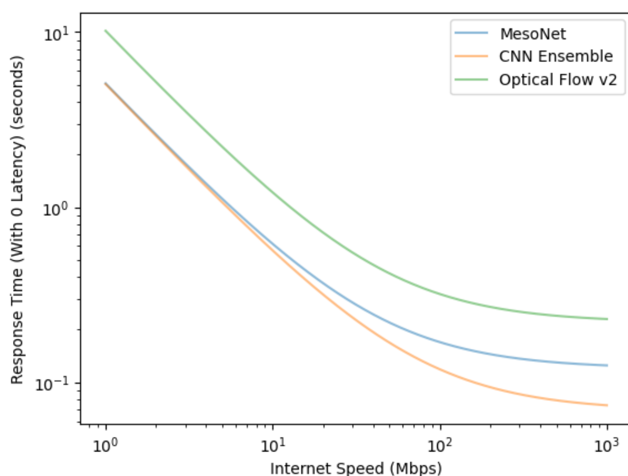
The success of the extension lies in successfully balancing speed, accuracy and practicality.

4.1.1 Speed

The slow part of detection is the transfer of data. This is shown in Figure 4.1. It is assumed that there is zero latency - to include latency, translate in the y-axis by twice the size of the latency in seconds. Note how the time taken for the computation only starts to matter at very high internet speeds.



(a) Speed Comparison Across Methods



(b) A Log-Log Version of (a)

Figure 4.1: The speed of each method for varying internet speeds, assuming 0 latency.

4.1.2 Accuracy

With the methods considered, we are unable to conclude that the Chrome extension is going to be able to accurately classify on data recorded by capturing the entire tab, as we were unable to classify mediaset significantly better than random chance with any method. This was shown in Table 3.2.

The optical flow method showed promise in [2], but when implemented it was only able to obtain results slightly better than random chance on data taken from the same source as the training data, and no better than random chance on mediaset.

4.1.3 Practicality

By keeping as much computation as possibile on the server side, we have kept the extension to just over 2KB in size, which is almost as small as an extension can possibly be. The server falls between 500MB and 1GB depending on the implementation used.

4.2 A Review of Methodology

4.2.1 Face Detection

In Section 3.3.6, I talked about the face detection method I used, and the decisions I had to make when it came to face selection and re-sizing.

To choose a face from multiple on the page, I opted to choose the one that was most likely to be a face. I think if I had more time, I would have tried some more approaches and compared how they felt. It's a really interesting UI question, although a bit of a tangent from the purpose of developing the extension in this project, which was to see whether large-scale deepfake detection was feasible.

I then decided to use a cropping method to get the face to be the right size to input to the classifier. If I had had more time, I would have tested with both cropping and scaling for both training and testing.

4.2.2 Comparisons

Original Papers

It would be a valuable exercise to compare my tests to those in the original papers in order to evaluate how well my tests worked.

The only test that I performed which was a replication of a test in the original papers was Ensemble of CNNs on the dfdc dataset. In [2], the authors achieved an AUC of around 0.88 on dfdc. I got a similar result of 0.90, which suggests that that test statistic is being calculated accurately.

Datsets

At the end of Section 3.6.1, I briefly touched on why I felt that even classifiers that performed well on data taken from dfdc did not get similar results on mediaset. There are three obvious components which this could be a result of:

- **The training set**:
  Models trained on dfdc are known not to generalise well, so it was always likely that Optical Flow and Ensemble of CNNs would do better on dfdc data than they would in general.
- **The test set**:
  mediaset, by design, is extremely impure. That is, it includes videos with many faces as well as many other components as it is designed to look like a user's tab when they are on social media. This means that it was likely that any model trained on more pure data (such as dfdc) would perform less well on mediaset.
- **Face detection**:
  It is likely that the face detection algorithm would perform better on pure data like that in the dfdc dataset than impure data like that in mediaset. mediaset both has more varied face sizes (and so would be susceptible to the problems of cropping and scaling in Section 3.3.6) and also has more faces on the screen (making it more difficult to choose the correct face as outlined in Section 3.3.6). This means that the algorithm is more likely to go wrong in the face detection phase when evaluating on mediaset than on the dfdc dataset.

4.3 Development

4.3.1 Timings

Table 4.1 shows where in the project I was on time and behind.

Table 4.1: The timetable from my proposal revisited.

Package	Intended Content	Date Due	Difficulties/Notes	On Time?	Extensions
One	Pre-reading	26/10	None	Yes	Not completed.

Continued on next page

Table 4.1: The timetable from my proposal revisited. (Continued)

Two	Find implementations	09/11	I found it extremely difficult to find many implementations of the methods found in Package One. I only ended up with two implementations (MesoNet and Ensemble of CNNs) and none that used temporal features.	Yes	Not completed.
Three	Simulation (test) coding	23/11	Due to the Advanced Data Science deadline, I didn't have time to code the test. This was moved into Package Four.	No	Not completed.
Four	Run tests	07/12	I continued to work on my Advanced Data Science coursework so was not able to finish coding the test. I now planned to do it over Christmas.	No	Not completed.
Five	Create a Chrome extension	18/01	Over the Christmas break, I was able to catch up with the creation of the simulation and also create a Chrome extension that classified deepfakes.	Yes	Not completed.
Six	Testing on UCS and progress report	01/02	By testing in Google Colab, I had found that the UCS would not be necessary for testing.	Yes	None proposed.
Seven	Progress report presentation	15/02	The feedback from the overseers was that the project didn't yet have enough substance. It was suggested that I implement one of the methods myself. I also decided to create my own test set.	Yes	None proposed.

Continued on next page

Table 4.1: The timetable from my proposal revisited. (Continued)

Eight	Dissert-ation planning	01/03	As well as making a plan for my dissertation, I researched how to implement the optical flow method.	Yes	Not completed - I didn't start writing my dissertation but I did begin to implement the optical flow method.
Nine	Dissert-ation writing	15/03	Due to needing to revise for my Computer Systems Modelling assessment and falling behind on lectures, I didn't have time to do work in this package.	No	None proposed.
Ten	Dissert-ation writing	26/04	I spent most of my time in Easter trying to train my optical flow implementation, which I did manage to do in the end. I also created my own test set (mediaset) and ran tests on that. I also wrote about 60% of a first draft of my dissertation,	No	None proposed - but the optical flow implementation and mediaset were both extensions.
Bey-ond	Dissert-ation revisions	12/05	I completed any outstanding tests and finished writing my dissertation. I also ran tests to find AUC.	Yes	None proposed.

4.3.2 Summary

In terms of timings, the project went well, with me remaining on time for much of it. The main issues were ends of terms, where I had given myself too much work to do, and after the overseers meeting, when I realised I needed to do a lot more to flesh out the project.

If I were to do a similar project again, I would make sure that I did any training as early as possible to get it out of the way, as it is extremely time consuming.

4.4 Success Criteria

I will now revisit the success criteria I outlined in my proposal and consider the extent to which I have accomplished each one.

4.4.1 List of Core Requirements

1. ***Have compared multiple pre-existing methods in speed, accuracy and practicality in theory.***
   Completed - This was the first thing I did in the project, and was a good way to get a lay of the land. This can be found in section 3.3.
2. ***Have compared implementations of some/all of these methods in reality on those three topics.***
   Completed - This is covered in section 3.6 and touched on in 4.1.
3. ***Create a Chrome extension which can send data from a video to a server.***
   Completed - First with a single image, then two images and finally a short video, I describe how I did this in section 3.2.
4. ***Create a program that takes that data and classifies the video from which it originates as either a deepfake or not a deepfake.***
   Completed - I was both able to incorporate pre-existing implementations (MesoNet and Ensemble of CNNs) to do this, and also to create my own implementation of the Optical Flow method in section 3.4.

4.4.2 List of Extension Tasks

1. ***Have created a Chrome extension which sends data to a server and presents to the user the result of the server's classification.***
   Completed - The extension is described in section 3.2.
2. ***Have the extension do this in a fast, accurate, practical way.***
   Partially Completed - I have been able to balance speed and practicality by having the classifier on a server separate to the extension and specifically choosing methods that minimise data transfer from client to server. However, these methods are not accurate on data taken from social media. I have concluded that there does not yet exist a fast, accurate, practical method that would allow us to detect deepfakes in a Chrome extension.

Chapter 5

Conclusion

5.1 The Feasibility Of Deepfake Detection On A Large Scale

In Section 1.3, I said that the motivation for the project was to see whether there existed deepfake detection that was fast, accurate, and practical, and as such that could be used on social media sites and in a Chrome extension. Having now completed the project, I feel that the answer to that question is no. I have laid out a series of methods that, on paper, seem like they may meet all three requirements, but all of them fall down on at least one of those options.

MesoNet is small in size and only detects on a single image, which makes it very practical. However, it was designed in 2018, before most modern deepfake creation algorithms had been created, so the accuracy reported in the paper no longer holds on even the most widely used detection dataset.

Ensemble of CNNs is larger than MesoNet but uses many of the same ideas. However, it doesn't generalise well to videos uploaded to social media.

RNN based methods seem more likely to be accurate as they make use of temporal features, but they use too much data to classify deepfakes and so are impractical for use in a Chrome extension.

Optical flow based methods also use temporal features, but only require two frames from a video to work, and so are more likely to be practical. However, they are not much better at classifying than random guessing, and also suffer from the same problem as Ensemble of CNNs of not generalising well to videos collected from social media.

Overall, I think that RNNs are currently the best way to get as much accuracy from deepfake detection as possible, and as such a fast, accurate, practical Chrome extension is not a feasible task.

However, social media sites such as YouTube already have time dedicated to processing videos for copyrighted content, and I do think it would be possible for such a system to also include a phase of deepfake detection as a part of this processing. It would be possible to send each video through a RNN without adding too much time to this processing period, and marking videos that are extremely likely to be deepfakes with a warning. If implemented, this could massively reduce misinformation going forward.

5.2 Future Work

I think there are various aspects that this project has skimmed the surface of that could be of interest in future works. I have only looked at video deepfake detection, but since I started the project audio deepfakes have risen in popularity beyond that of videos, and so there could be a complete second project with a similar title but focusing on audio.

There is also more work to be done on detecting deepfakes with optical flow. Combining optical flow and RNNs could be a way to get the most out of both approaches.

Finally, I think that it would be worth considering the feasibility of running convolutional - or even recurrent - neural networks within browser extensions. If a user has the right hardware to do this - enough memory and the correct GPU - then they should be able to bypass the need for a server, which could lead to speed improvements and the ability to use more temporal features (as there would be no time wasted transferring data). To find wide use, such a system would have to be able to adapt to the user's particular hardware, which is a complex problem, and one that would require in-depth study.

5.3 Lessons Learned

I have learned to be wary when considering results presented in papers around machine learning. There are many factors that can affect the validy of reported tests, and neural networks develop so quickly that even if reported results were accurate, they quickly become invalid as we saw with MesoNet.

The main takeaway from this project though has been a crushing pessimism with respect to the detection of deepfakes. It seems that simply decreasing the quality of a video is enough to get past almost any detection algorithm, and with social media being the most common way misinformation spreads and the newfound popularity of deepfakes and artificial intelligence undoubtedly propelling the area of deepfake creation forward, it is unlikely that deepfake detection will ever be able to work well enough to be of any practical use.

Bibliography

[1] Wikipedia Commons. Long short-term memory. https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg.

[2] Irene Amerini, Leonardo Galteri, Roberto Caldelli, and Alberto Del Bimbo. Deepfake video detection through optical flow based cnn. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 1205–1207, 2019.

[3] CBS News. Fake photos of pope francis in a puffer jacket go viral, highlighting the power and peril of ai. https://www.cbsnews.com/news/pope-francis-puffer-jacket-fake-photos-deepfake-power-peril-of-ai/.

[4] Hank Green. Twitter poll. https://twitter.com/hankgreen/status/1639982683878211585.

[5] BBC. Facebook to ban 'deepfakes'. https://www.bbc.co.uk/news/technology-51018758.

[6] Meta. Enforcing against manipulated media. https://about.fb.com/news/2020/01/enforcing-against-manipulated-media/.

[7] Twitter. Synthetic and manipulated media policy. https://help.twitter.com/en/rules-and-policies/manipulated-media.

[8] Thanh Thi Nguyen, Quoc Viet Hung Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, Thien Huynh-The, Saeid Nahavandi, Thanh Tam Nguyen, Quoc-Viet Pham, and Cuong M. Nguyen. Deep learning for deepfakes creation and detection: A survey. *Computer Vision and Image Understanding*, 223:103525, oct 2022.

[9] Fake profile detector (deepfake, gan). https://chrome.google.com/webstore/detail/fake-profile-detector-dee/jbpcgcnnhmjmajjkgdaogpgefbnokpcc.

[10] Deepfakeproof. https://chrome.google.com/webstore/detail/deepfakeproof/ehjldchkbnfkmicpofahcghimhkkpngo.

[11] Deepfake detection. https://chrome.google.com/webstore/detail/deepfake-detection/gbokgdgbgfcdlbnonmlajiapbcpkdgnk/related.

[12] Meta/Kaggle. Deepfake detection challenge. https://www.kaggle.com/competitions/deepfake-detection-challenge/.

[13] icpr2020dfdc/face_extract.py. `https://github.com/polimi-ispl/icpr2020dfdc/blob/master/blazeface/face_extract.py`.

[14] Meta : Deepfake detection challenge dataset. `https://ai.facebook.com/datasets/dfdc/`.

[15] DebugBear. How do chrome extensions impact browser performance? `https://www.debugbear.com/blog/chrome-extension-performance-2021`.

[16] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

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

[18] Analytics Vidhya. Guide to auc roc curve in machine learning : What is specificity? `https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/`.

[19] Chrome Developer. captureVisibleTab. `https://developer.chrome.com/docs/extensions/reference/tabs/#method-captureVisibleTab`.

[20] Chrome Developer. Record audio and video with MediaRecorder. `https://developer.chrome.com/blog/mediarecorder/`.

[21] Darius Afchar, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. MesoNet: a compact facial video forgery detection network. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, dec 2018.

[22] Nicolò Bonettini, Edoardo Daniele Cannas, Sara Mandelli, Luca Bondi, Paolo Bestagini, and Stefano Tubaro. Video face manipulation detection through ensemble of cnns. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 5012–5019, 2021.

[23] Google Research Perception. Blazeface. `https://sites.google.com/view/perception-cv4arvr/blazeface`.

Appendix A

Chrome Extension Simulation Pseudocode

```
function chrome_extension_simulator(...):
    # a_b means that the prediction was a and the actual was b
    correct_fake = 0
    incorrect_fake = 0
    none_fake = 0

    correct_real = 0
    incorrect_real = 0
    none_real = 0

    sum_time_taken = 0

    total_data_transferred = 0

    results = []


    for index from 0 to number of datapoints:
        total_time = ping_ms / 1000

        d = dataprocessing(test_data[index])

        if d not None:
            t = current_time()
            d = preprocessing(d)
            total_time += t - current_time()

        total_data_transferred += byte_size_of(d)
        total_time += byte_size_of(d)*8/internet_speed_mbps
```

```
        if d not None:
            t = current_time()
            d = server_func(d)
            total_time += t - current_time()

        total_time += byte_size_of(d)*8/internet_speed_mbps

        if d not None:
            t = current_time()
            d = postprocessing(d)
            total_time += t - current_time()

        total_time += byte_size_of(d)*8/internet_speed_mbps

        actual = test_labels[index]

        results.append((d, actual))

        if actual == False:
            if prediction == False:
                correct_real += 1
            elif prediction == True:
                incorrect_real += 1
            else:
                none_real += 1
        else:
            if prediction == True:
                correct_fake += 1
            elif prediction == False:
                incorrect_fake += 1
            else:
                none_fake += 1

        sum_time_taken += total_time

    return results, sum_time_taken, total_data_transferred,
        len(test_data_iter),
        {"real":{"correct":correct_real,
                 "incorrect":incorrect_real,
                 "none":none_real},
         "fake":{"correct":correct_fake,
                 "incorrect":incorrect_fake,
                 "none":none_fake}}
```

Appendix B

Output of Tests In More Detail

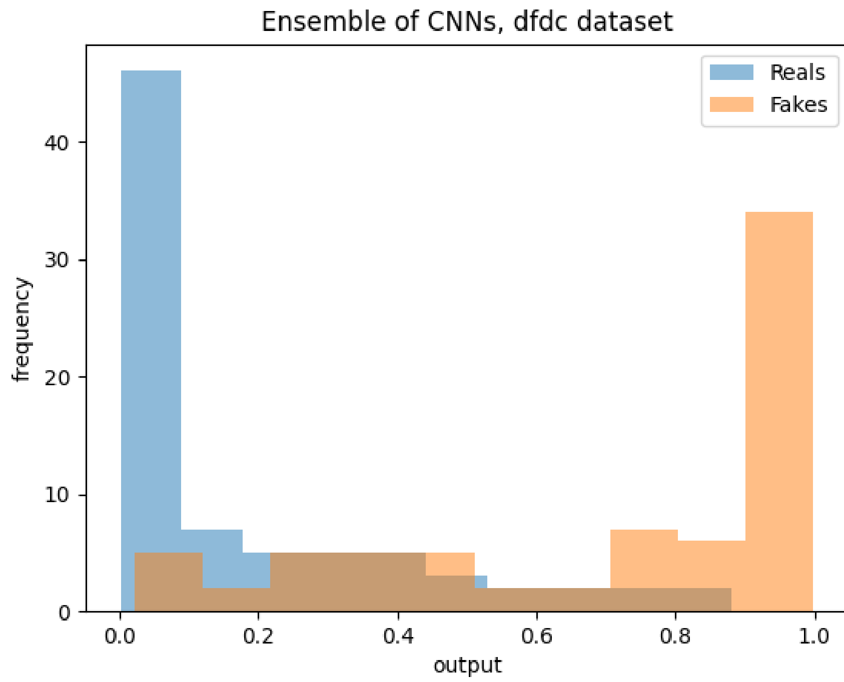


Figure B.1: (a) Ensemble of CNNs on dfdc.

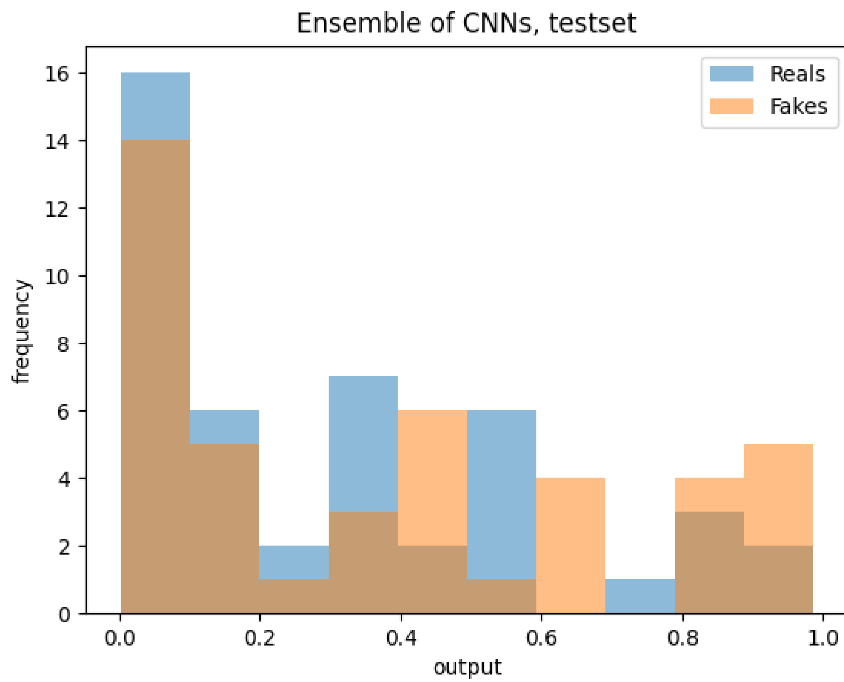


Figure B.2: (b) Ensemble of CNNs on mediaset.

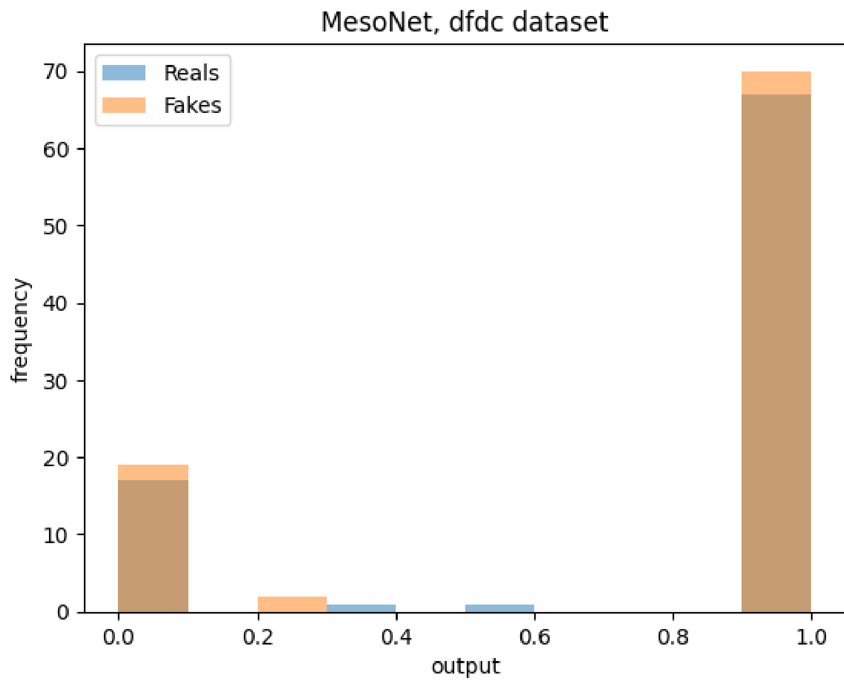


Figure B.3: (c) MesoNet on dfdc.

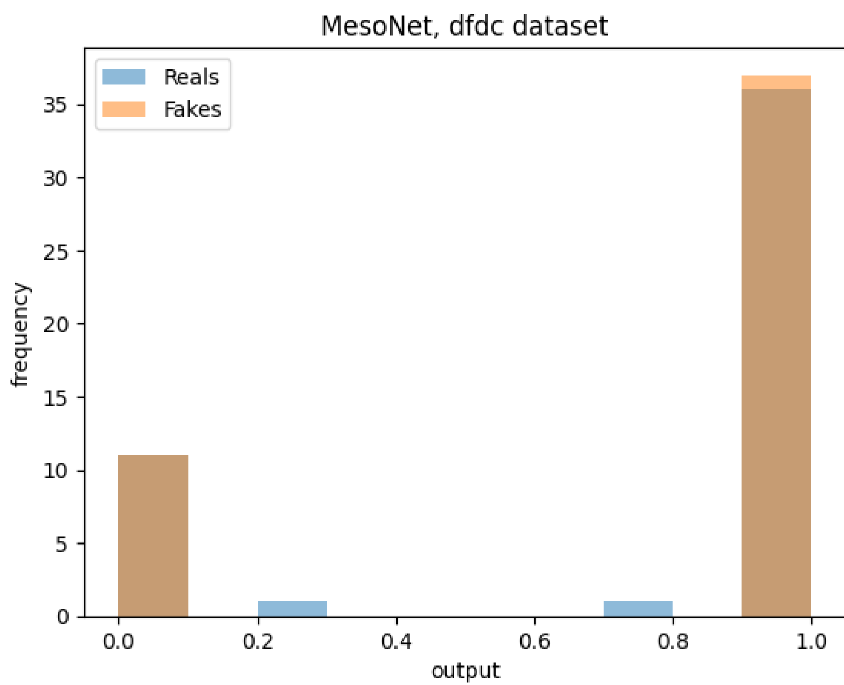


Figure B.4: (d) MesoNet on mediaset.

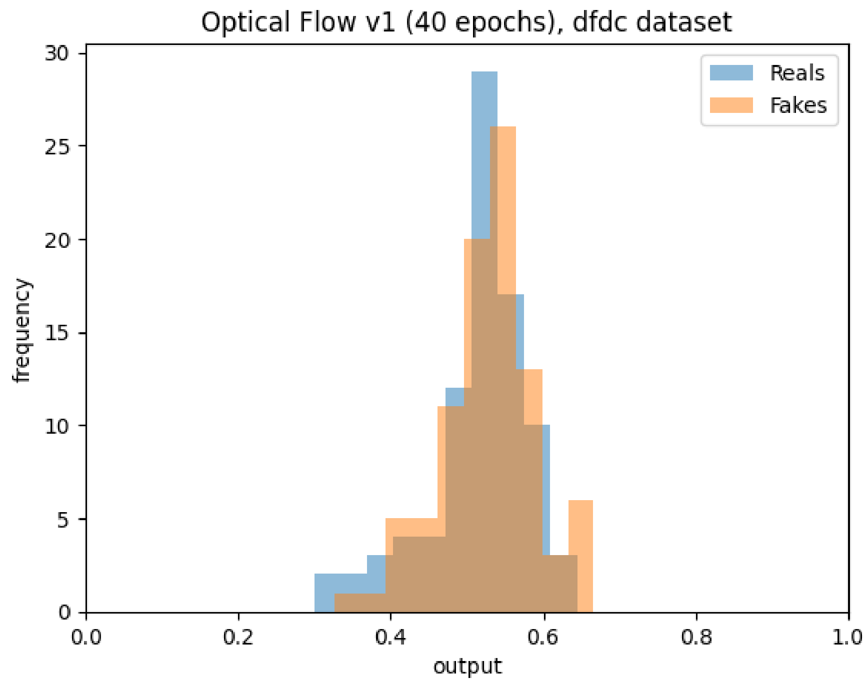


Figure B.5: (e) Optical Flow (No VGG-16 Layers Trained - 40 epochs) on dfdc.

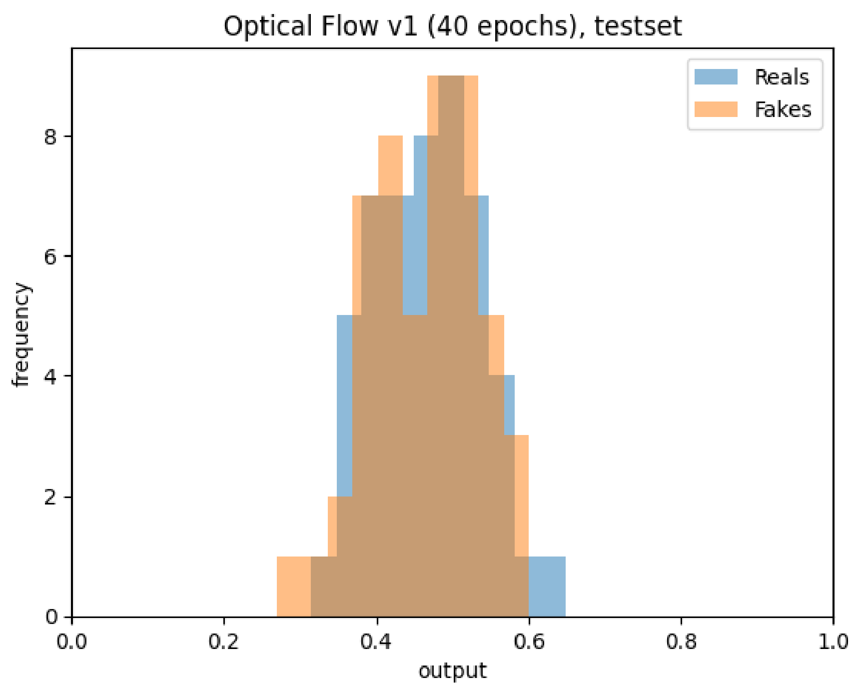


Figure B.6: (f) Optical Flow (No VGG-16 Layers Trained - 40 epochs) on mediaset.

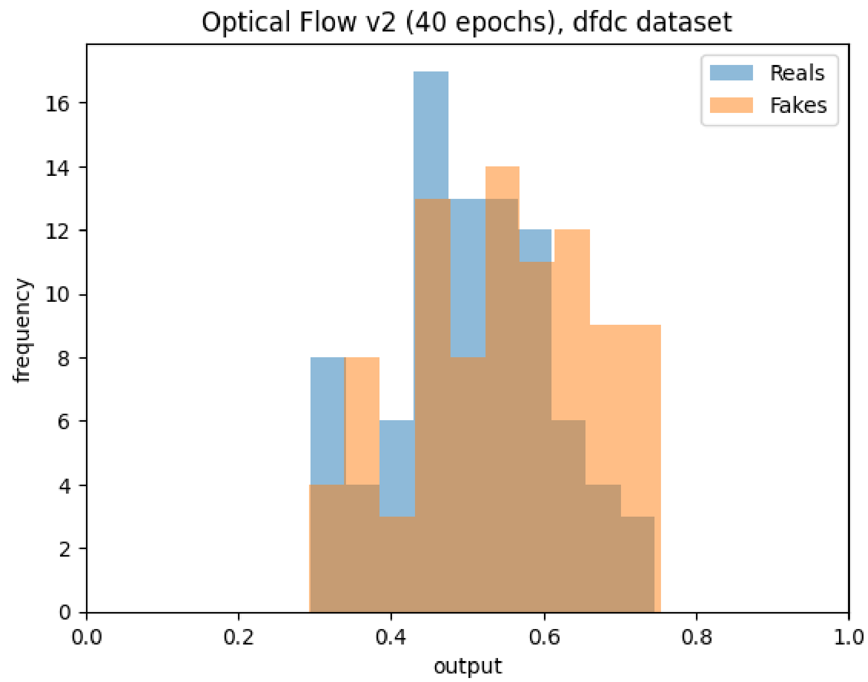


Figure B.7: (g) Optical Flow (One VGG-16 Layer Trained - 40 epochs) on dfdc.

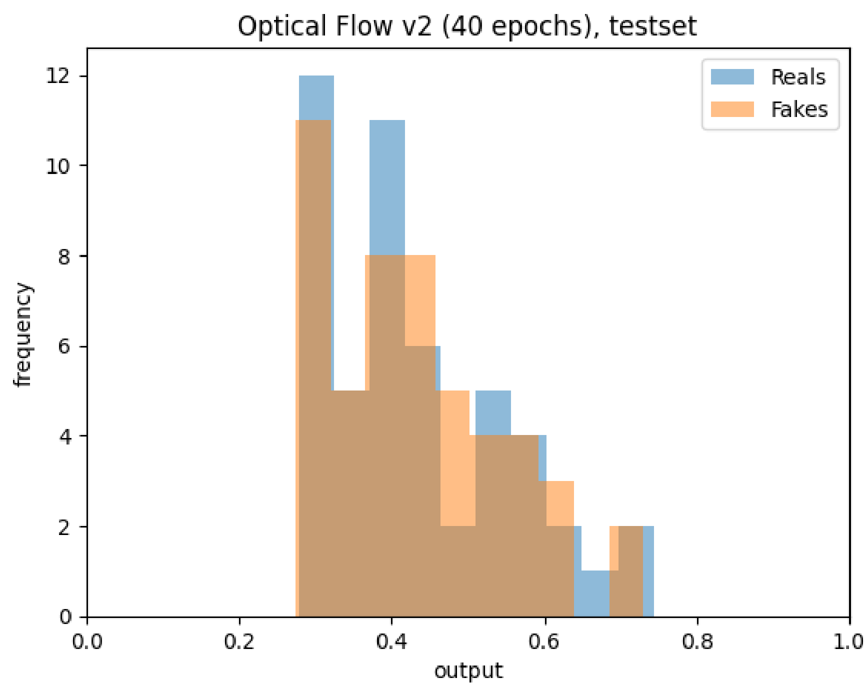


Figure B.8: (h) Optical Flow (One VGG-16 Layer Trained - 40 epochs) on mediaset.

Appendix C

Chat-GPT Conversations

Table C.1: A summary of relevant conversation with Chat-GPT.

Prompt	Response (Oultine)
write a chrome extension that sends a screenshot of a tab to a server and receives a number from the server in response, which it displays to the user. Also write the server in python, which handles the request using the flask library. I should pass the screenshot as a pillow image type to a pre-existing function called "detect_df".	The javascript, an explanation of how it works, a basic python flask application, a suggestion for the function detect_df.
update the server to also locally save the image in a file called image,jpg	A failed attempt to save the image locally (I re-implemented this myself).
what should manifest.json, popup.html, and script.js look like?	An example manifest, html, and script file. I incorporated the script file as the last line of background.js.
the server raises the following error: [redacted for anonymity]	A successful attempt to fix the error.
what should server.py look like with these changes	An example of server.py.
the extension now says "unidentified" when it receives the response	An explanation that the body of the POST request was likely not in the correct format and examples of how to fix this.

Continued on next page

Table C.1: A summary of relevant conversation with Chat-GPT. (Continued)

the server returns invalid image file	Some suggestions to try and find the error.
the extension raises the error: Unchecked runtime.lastError: Either the 'all_urls' or 'activeTab' permission is required.	An updated manifest with the activeTab permission.
the server raises the following error: [redacted for anonymity]	An unsuccessful attempt to fix the error.
background.js currently looks like this: [redacted for AI content] does this help you to debug?	A further unsuccessful attempt to fix the error.
extend this js code to take two screenshots and send both of them: [redacted for AI content]	The javascript for an updated extension that takes two screenshots.
edit the code to have a 0.5s delay between the screenshots	Updated javascript that uses the setTimeout function to add a 500ms delay (later changed to 100ms).
change the following js code to capture a 500ms video instead of taking screenshots: [redacted for AI content]	An example using MediaRecorder to record the tab.

Appendix D

Project Proposal

Part II Project Proposal - [redacted]
The Feasibility Of Deepfake Detection On A Large Scale

INTRODUCTION

In recent years, deepfakes have become more and more prevalent. Accusations of their use in major conflicts[1], financial scams[2], and job interviews[3] have been on the rise[4]. As both the methods and compute power used to generate these videos improves, the threat of widespread misinformation becomes more serious as deepfakes are less likely to be caught by the human eye.

Various outlets have banned deepfakes[5] but as of yet none are using automated deepfake detection methods to prevent these videos from being uploaded to their sites. In my project, I will look at whether there exist methods that could work quickly and accurately enough to verify every video uploaded to such a site.

In the second part of my project I will consider whether users could be able to detect these deepfakes themselves - at the user's end, speed and accuracy must also be balanced with practicality: a user would need a small file size on their computer for the detection software and, if communicating with some central server, would need to send and receive minimal data to that server as internet speed becomes an issue (it would be impractical, for example, to send a whole video to a server for detection). A good benchmark for this balance of practicality is a chrome extension, and so I will aim to create an extension that is able to classify deepfakes and non-deepfakes.

STARTING POINT

I have previously given a talk on deepfake detection as one of the Churchill Compsci Talks[6], which involved some background reading on the topic. As such I have some base-level background knowledge on deepfake detection.

An excellent overview of pre-existing methods is given in this paper:
Deep Learning for Deepfakes Creation and Detection: A Survey (arXiv:1909.11573)
This presents both methods for detecting deepfakes from single images and from whole videos.

I have yet to do any coding for the project itself or any technical reading outside of that which was required for the above talk. I have done very little coding of machine learning systems but have a baseline understanding of them from both part IB courses and background reading for the talk.

---

[1] https://techcrunch.com/2022/03/16/facebook-zelensky-deepfake/
[2] https://www.boston25news.com/news/local/better-business-bureau-warns-about-deepfake-elon-musk-investment-scam/
[3] https://www.marketwatch.com/story/remote-work-has-created-yet-another-problem-colleagues-who-may-be-deepfakes-11659727993
[4] https://trends.google.com/trends/explore?date=today%205-y&geo=GB&q=deepfake
[5] https://www.bbc.co.uk/news/technology-51018758; https://help.twitter.com/en/rules-and-policies/manipulated-media
[6] [redacted]

WORK TO BE DONE

- Stage One: Comparing pre-existing methods of deepfake detection
  - Pre-reading of existing methods
  - Comparing some of those methods on paper
    - A comparison can be drawn based on speed, accuracy, and practicality.
      - Speed should include both how quickly a method works on a single system (e.g a twitter data server) and when data from a video has to be sent from a user to a central server, and then processed (e.g in a chrome extension).
      - Accuracy is how well a method detects deepfakes. How likely are false negatives and false positives? How does accuracy change with compute power?
      - Practicality relates specifically to how well a method can be incorporated in a chrome extension. How much storage would the method take up on both user side and server side? Can data sent from the user to the server be reduced to a single frame of video or even less data than a single frame? This second question is particularly important - some methods will work on the assumption that t hey have the full video. Do these methods actually need the full video, or can we reverse engineer them to work with some data points that are less than the full video?
      - Note that this comparison will be most valuable in practicality - accuracy may be misreported to make the method seem better than it is and will vary by implementation and testing method. Speed will also face these problems and is likely to not be reported at all.
  - Create a program with which we can compare methods in reality
    - Should be able to imitate a chrome extension operating on different video lengths and internet speeds
    - Should take as input two functions, one that imitates the user-side of the extension by getting data from a video, and another that uses that data to classify whether the video is a deepfake
    - In return should output the accuracy and speed of the extension when the user has various internet speeds and when the video is of various lengths
  - Use this structure to run tests on a small number of existing methods that seem on paper like they would be viable
- Stage Two: Creating an example of fast, accurate, practical deepfake detection
  - Create a chrome extension which sends data taken from a video to a server and then receives a response from that server
    - Note that I am deliberately vague about what this data is as that will depend on the implementation. It could be anything from a single pixel (inaccurate) to the whole video (impractical) but would in reality be something in between.

        - Create a program that classifies deepfakes based on one or more of the more viable methods found in stage one, taking as input the same data that the chrome extension sends to a server.
            - This should be done at a minimum by using a pre-existing implementation and adapting it to be usable in a chrome extension.

ROOM FOR EXTENSION

Both stages are open for change due to misjudged workload.

Stage one should involve me comparing at least two methods, but I could compare as many as 10-12 on paper and run tests on as many as 5-6 of those.

Stage two can be much more varied. At the bare minimum I should be able to send and receive data to a server from a chrome extension and also have a program that classifies deepfakes. However, I would like to be able to connect these to get an extension that works fully - where I can have a video in my browser and use the extension to predict whether it is a deepfake. There will also be a miriad of changes I can make to improve speed, accuracy and practicality.

I can also as an extension re-implement a pre-existing method rather than just adapting a pre-existing implementation.

SUCCESS CRITERIA

CORE (The project will be considered a success if I…)

- Have compared multiple pre-existing methods in speed, accuracy and practicality in theory
- Have compared implementations of some/all of these methods in reality on those three topics
- Create a chrome extension which can send data from a video to a server
- Create a program that takes that data and classifies the video from which it originates as either a deepfake or not a deepfake

EXTENSION (If the project has gone very well then I may…)

- Have created a chrome extension which sends data to a server and presents to the user the result of the server's classification
- Have the extension do this in a fast, accurate, practical way
    - The extension should be made more accurate in a way that does not make it too large at the user's end or make it take longer than a few seconds

RESOURCES / PRACTICALITIES

The majority of the project will work in Python, but the user-facing side of the chrome extension will be written in JavaScript. No critical resources are required but the Managed Cluster Service will be used. I will not use human participants. A large dataset of deepfakes and real videos is available at
https://www.kaggle.com/competitions/deepfake-detection-challenge/data

TIMETABLE

PACKAGE ONE: 13/10/2022-26/10/2022 (2 weeks) [Busyness: Low, ~8 lectures/week, No deadlines]
Reading about pre-existing methods Create a report/presentation which compares, on paper, their speed, accuracy and practicality Plan a method for testing speed and accuracy of a method when used in a chrome extension By 26/10/2022: I should be able to communicate with my supervisor in the form of a brief powerpoint presentation about a range of pre-existing methods, comparing their speed, accuracy and practicality. I should also have written a report on these findings which could be used in my dissertation. Finally, I should be able to show my supervisor a clear plan for the implementation of my test. Extension: Compare more pre-existing methods Begin to code my test

PACKAGE TWO: 27/10/2022-09/11/2022 (2 weeks) [Busyness: Very High, ~13 lectures/week, 2 advanced data science deadlines]
Find viable implementations that I can test By 09/11/2022: I should have a list of implementations, what methods they implement, and be able to explain how I will use them in my planned test. Extension: Begin/continue to code my test

PACKAGE THREE: 10/11/2022-23/11/2022 (2 weeks) [Busyness: High, ~12 lectures/week, 1 advanced data science deadline]
Complete coding of my test Check that it will work with all of the implementations I have found By 23/11/2022: I should be able to explain my test to my supervisor and use it to estimate the accuracy and speed of one of the implementations I found Extension: Begin planning and coding of my chrome extension

PACKAGE FOUR: 24/11/2022-07/12/2022 (2 weeks)
[Busyness: Medium, ~4 lectures/week, 2 advanced data science deadlines]

Use the university cluster service to run the test to find the speed and accuracy of various implementations across various video lengths and internet speeds
Present the results of the tests in the form of a report/presentation

By 07/12/2022:
I should be able to present to my supervisor a comparison of the speed and accuracy of various implementations at different video lengths and internet speeds

Extension:
Begin/continue planning and coding my chrome extension

PACKAGE FIVE: 08/12/2022-18/01/2023 (Christmas - 6 weeks)
[Busyness: Medium, Christmas break]

Create a chrome extension which sends data from a video to a server, receives some response from the server and presents this result to the user
Create a program that uses the data sent by the chrome extension to classify deepfakes using a method based on one or more of the pre-existing methods

By 18/01/2023:
I should be able to get data from my chrome extension to a server
I should be able to use that data to classify deepfakes

Extension:
Host the program on a server and connect the extension to the program
Make improvements to the speed, accuracy and practicality of the program and the extension

PACKAGE SIX: 19/01/2023-01/02/2023 (2 weeks)
[Busyness: Medium, ~9 lectures/week]

Test my method on the university cluster service
Complete the progress report

By 01/02/2023:
I should have compared my implementation with other implementations and submitted my progress report

PACKAGE SEVEN: 02/02/2023-15/02/2023 (2 weeks)
[Busyness: Medium, ~9 lectures/week]

Create and give a progress report presentation

PACKAGE EIGHT: 16/02/2023-01/03/2023 (2 weeks) [Busyness: Medium, ~8 lectures/week]
Dissertation writing By 01/03/2023: I should have a plan for my dissertation and have made a start on writing it

PACKAGE NINE: 02/03/2023-15/03/2023 (2 weeks) [Busyness: Low, ~6 lectures/week]
Dissertation writing By 15/03/2023: I should have an extremely early draft of my dissertation

PACKAGE TEN: 16/03/2023-26/04/2023 (Easter - 6 weeks) [Busyness: Medium, Easter break]
Dissertation writing By 26/04/2023: I should have finished my dissertation