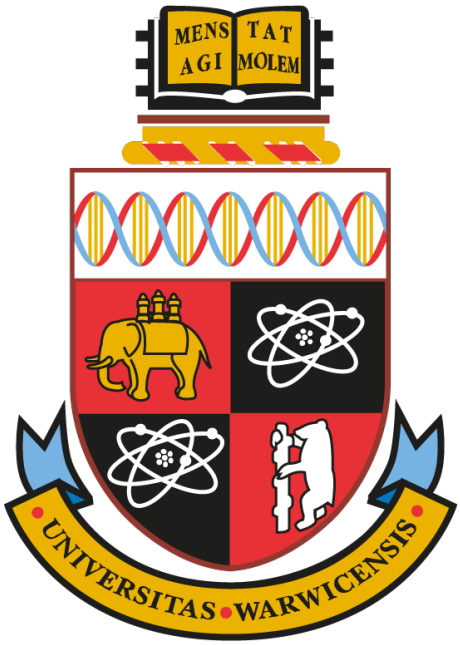
Evaluating the Resiliency of Blink-Based DeepFake Detection Against Adversarial Noise Attacks



Joel Coulon
Department of Computer Science
University of Warwick

Supervised by Long Tran-Thanh
Year of Study: 3rd (2024-2025)

Abstract

DeepFakes are pieces of media that have been generated or manipulated by Artificial Intelligence. They can produce realistic videos of humans that are nearly impossible for humans to classify as fake. Hence, they can be used to easily spread misinformation. Convolutional Neural Networks have traditionally been used to reliably detect DeepFake videos. However, these have been shown to be vulnerable to adversarial noise; targeted noise, which is imperceptible to humans, that can be added to an image which causes a previously reliable detector to classify a fake image as real.

Blink-based detectors involve tracking a human's psychological eye blinking patterns to determine if a video has been DeepFaked. DeepFakes will often blink at a rate that is unnatural and thus can be detected reliably.

This project introduces a novel method of blink detection using univariate time-series analysis on an individual's eyes blinking pattern over the course of a video. It goes on to show that such a method of detection is resilient against adversarial noise attacks on a variety of DeepFake benchmarks.

Keywords: DeepFakes, Blinking, Eye Aspect Ratio, Time Series Analysis, Convolutional Neural Networks, Machine Learning

Acknowledgements

First of all, I would like to thank my supervisor, Long Tran-Thanh, for his help in supervising me throughout the course of this project. It is thanks to him that the project is the success it is.

Secondly, thank you to Alia Meek for helping me keep sane throughout the course of the project and my third year as a whole.

Finally, I would like to thank my friends at Warwick. In particular, I would like to thank Serene Alrawi, Verity Anscombe, Oliver Cordeaux, Ethan Farrell, and many others for their help in keeping me motivated and allowing me to inflict my presentation on them multiple times.

Contents

List of Figures **v**

List of Algorithms **vi**

List of Tables **vii**

1 Introduction **1**

1.1 Project Overview . . . 1

1.2 Aim . . . 2

1.3 Objectives . . . 2

1.4 Novel Contributions . . . 3

2 Literature Review **4**

2.1 A History of DeepFakes . . . 4

2.1.1 Photo Manipulation . . . 4

2.1.2 Artificial Intelligence for Manipulating Media . . . 5

2.1.3 DeepFakes for Malicious Purposes . . . 10

2.2 DeepFake Detection . . . 11

2.2.1 Traditional Neural Network Detection Methods . . . 12

2.2.2 Blink-Based Detection Methods . . . 13

2.3 Adversarial Noise . . . 16

2.3.1 Fast Gradient Sign Method . . . 17

2.3.2 Carlini-Wagner L2-Norm Attack . . . 17

2.3.3 FakeRetouch . . . 18

2.4 Transferability of DeepFake Detectors . . . 19

2.5 Datasets . . . 20

3 Design and Implementation **22**

3.1 Language and Machine Learning Framework . . . 22

3.1.1 Python . . . 22

3.1.2 PyTorch versus TensorFlow . . . 22

3.2 Proof of Concept . . . 23

3.3 Main DeepFake Detection Algorithm . . . 24

3.3.1 Face cropping . . . 25

3.3.2 Eye Landmark Detection . . . 26

3.3.3 Eye Aspect Ratio Analysis . . . 32

3.4 Adversarial Noise . . . 38

3.5 Traditional Detectors . . . 38

3.6 Final Code . . . 39

3.7 Transferability . . . 41

3.8 Datasets . . . 41

4 Results and Evaluation **42**

4.1 Proof of Concept . . . 42

4.2 Main Code Results . . . 43

4.2.1 Unperturbed Blink Detection . . . 43

4.2.2 Unperturbed Traditional Detectors . . . 44

4.2.3 Perturbed Blink Detection . . . 44

4.2.4 Perturbed Traditional Detection . . . 45

4.3 Transferability . . . 46

4.4 Evaluating Blink-Based DeepFake Detection . . . 46

5 Project Management **48**

5.1 Project Plan . . . 48

5.1.1 Gantt Charts . . . 48

5.1.2 Dealing With Setbacks . . . 49

5.1.3 Development Methodology . . . 49

5.2 Resources Used . . . 50

5.3 Legal, Social, Ethical, and Professional Issues . . . 50

6 Conclusions **52**

6.1 Author's Assessment of Project . . . 52

6.2 Future Research Suggestions . . . 53

6.3 Open Source Work . . . 53

Bibliography **54**

A Time Series Classification Architectures **63**

B Traditional Classification Architectures **65**

C Deep Fake Detection Challenge Website **67**

D Raw results **68**

D.1 Proof of Concept 68

D.2 Main Code 68

D.2.1 FaceForensics 68

D.2.2 Celeb-DF 70

D.3 Transferability 72

E Open Source Contributions **73**

List of Figures

1.1 A DeepFake generated with `https://thispersondoesnotexist.com` . . . 1

2.1 Examples of historical images that were edited . . . 4
2.2 Examples of digital image manipulation . . . 5
2.3 A structure of a neural network and the individual neurons within it . . . 6
2.4 A diagram showing the action of a 3×3 filter[58] . . . 7
2.5 A sample CNN[72] . . . 7
2.6 A sample plot of the value of the loss function over varying values of a parameter x . . . 8
2.7 Effect of different learning rates on loss function optimisation . . . 9
2.8 A diagram of the network structure of a GAN . . . 10
2.9 An image generated by GPT-4o[88] using the prompt "a chocolate labrador with glasses programming on a computer" . . . 11
2.10 Output from attention layers of convolutional neural networks on DeepFake detection[12] 13
2.11 The points and resulting graph for the Eye Aspect Ratio . . . 14
2.12 DeepVision's architecture and database queries . . . 15
2.13 A network diagram of Ictu Oculi[77] . . . 16
2.14 The process of adversarial noise being added to an image to cause a misclassification[43] . 17
2.15 A collection of images and their Fourier Transforms . . . 18
2.16 A graph showing the Pd@1%FAR accuracy of a variety of detection models[102] . . . 20

3.1 Network diagram of WSELD[55] . . . 27
3.2 A generic network diagram for a HRNet[120] . . . 28
3.3 The heatmap outputs of HRNet . . . 29
3.4 Network Architecture of PFLD (Generated using visualkeras[44]) . . . 30
3.5 A comparison of HRNet's (red) and PFLD's (blue) predictions . . . 31
3.6 The 68 facial landmarks common in facial landmarking datasets[109] . . . 32
3.7 Faces that have been partially occluded . . . 32
3.8 Plots of sample EAR graphs . . . 33

3.9 Representations of various optimisation to the DTW algorithm[38] 34

3.10 A representation of the different resolutions computed for FastDTW and how they exclude regions of the cost matrix in finer resolution[113] 35

3.11 An example of the sequences and features extracted during the time series forest algorithm 36

4.1 A comparison of landmarks on original and perturbed images 44

5.1 The original Gantt chart planned for the software development section of the project . . . 48

5.2 The actual Gantt chart for the software development section of the project 49

A.1 Architecture for LSTM-FCN[64] 63

A.2 Architecture of Time-CNN 63

A.3 Architecture of ResNet 64

A.4 Architecture of Multi-Layer Perceptron 64

A.5 Architecture of Fully Convolutional Neural Network 64

B.1 Architecture of VGG19 65

B.2 Architecture of ResNet 66

B.3 Architecture of Xception 66

B.4 Architecture of EfficientNetB4 66

C.1 A screenshot of the security warning found when trying to access `https://dfdc.ai/` . . . 67

E.1 A pull request made to the FoolBox[99][100] library to replace a deprecated TensorFlow function 73

E.2 A comment made to an ROI pooling implementation to allow for variable sizing in images 74

List of Algorithms

3.1 Overall architecture of a blink-based DeepFake detector 23
3.2 The method for cropping faces from a video 26
3.3 DTW algorithm 34
3.4 Final Code 40

List of Tables

D.1 Results of the proof of concept . . . 68
D.2 FaceForensics results unperturbed . . . 69
D.3 FaceForensics results perturbed (VGG, $\epsilon = 0.05$) . . . 69
D.4 FaceForensics results perturbed (VGG, $\epsilon = 0.1$) . . . 69
D.5 FaceForensics results perturbed (ResNet, $\epsilon = 0.05$) . . . 69
D.6 FaceForensics results perturbed (ResNet, $\epsilon = 0.1$) . . . 69
D.7 FaceForensics results perturbed (Xception, $\epsilon = 0.05$) . . . 69
D.8 FaceForensics results perturbed (Xception, $\epsilon = 0.1$) . . . 70
D.9 FaceForensics results perturbed (EfficientNet, $\epsilon = 0.05$) . . . 70
D.10 FaceForensics results perturbed (EfficientNet, $\epsilon = 0.1$) . . . 70
D.11 Celeb-DF results unperturbed . . . 70
D.12 Celeb-DF results perturbed (VGG, $\epsilon = 0.05$) . . . 70
D.13 Celeb-DF results perturbed (VGG, $\epsilon = 0.1$) . . . 71
D.14 Celeb-DF results perturbed (ResNet, $\epsilon = 0.05$) . . . 71
D.15 Celeb-DF results perturbed (ResNet, $\epsilon = 0.1$) . . . 71
D.16 Celeb-DF results perturbed (Xception, $\epsilon = 0.05$) . . . 71
D.17 Celeb-DF results perturbed (Xception, $\epsilon = 0.1$) . . . 71
D.18 Celeb-DF results perturbed (EfficientNet, $\epsilon = 0.05$) . . . 71
D.19 Celeb-DF results perturbed (EfficientNet, $\epsilon = 0.1$) . . . 72
D.20 The accuracy of models trained on FaceForensics and tested on FakeAVCeleb . . . 72
D.21 The accuracy of models trained on Celeb-DF and tested on FakeAVCeleb . . . 72

Chapter 1

Introduction

1.1 Project Overview

DeepFakes are defined as "manipulated or synthetic media whose realness is not easily recognisable by the human eye"[4]. In simpler terms, a DeepFake is a video, image, or audio clip that has been digitally altered or created, often using Artificial Intelligence (AI). Concerningly, it is extremely difficult for humans to determine whether or not a particular video has been DeepFaked. In a 2024 study, 86,155 volunteers attempted to detect DeepFakes, the final accuracy for all DeepFake types was 55.54%[30], showing humans are only slightly better than guessing. Furthermore, a Europol report revealed that 72% of the UK population were unaware of DeepFakes and their potential impact[36].

Figure 1.1: A DeepFake generated with `https://thispersondoesnotexist.com`

Because they are so realistic, DeepFakes have garnered wide usage for various nefarious activities. Scams, misinformation, and fraud are all possible uses of DeepFakes and all pose a real threat to society[19]. Deloitte estimates that in 2027, DeepFakes will enable more $40 billion of fraud in the United States[73].

It is obvious that a mechanism to reliably detect DeepFakes would be incredibly useful.

Convolutional Neural Networks (CNNs) are powerful neural network models that learn kernels (sometimes also called filters) in a feed-forward manner[76]. They consist of sequences of layers that gradually

take a more and more general view of an input to produce a final classification. They have a wide variety of applications in various fields of deep learning, such as computer vision and recommendation algorithms. Crucially, they can also be used to create accurate DeepFake detectors. When trained on a specific dataset, they can become extremely accurate, achieving consistent accuracies of over 90% when trained correctly[91].

However, recent studies have shown that CNNs are vulnerable to adversarial noise attacks. Noise, imperceptible to humans, is added to images which can cause a misclassification[43]. This becomes dangerous when noise is added to fake images, causing them to be classified as real. A 90% accurate classifier is going to be implicitly trusted to be correct and a simple and invisible method to fool classifiers is worrying.

A promising new method of DeepFake detection focuses on blinking. DeepFakes and other AI-generated content suffer from various temporal inconsistencies. When a video is real, certain aspects can be predicted between frames, however, AI struggles to replicate these aspects. Blink-based DeepFake detection exploits blinking inconsistencies. Human blinking is a subconscious action and hence periodic: the average period between blinks is 2.8 seconds and a blink takes around 0.1-0.4 seconds[114]. Other eye-based temporal inconsistencies exist, such as the shape of an eye and how far open the eye can be. It is possible to leverage these for DeepFake detection, by analysing how far open the eye is over time.

This project aims to determine whether blink-based DeepFake detection is resistant to adversarial noise. It is believed to be resilient as blink-based detection classifies a video over time. Current methods of adversarial noise can only perturb a single frame at once and so, in theory, they would not be able to coordinate to cause a misclassification overtime. However, adversarial attacks may cause landmarks to be mislocated resulting in the noise causing misclassifications.

This project also aims to evaluate whether blink-based detection can be used as a general DeepFake classifier or not. The majority of CNN-based detectors attain high accuracies by focussing on model-specific artifacts (Figure 2.15). On the other hand, the majority of DeepFakes will miss common factors in blinking such as frequency and period, and these patterns should be consistent across DeepFakes methodologies as similar kinds of mistakes will be made. Hence, it is believed that blink-based DeepFake detectors will be generalised.

1.2 Aim

This paper aims to implement blink-based DeepFake detection and show that such a method is resistant to adversarial noise attacks.

1.3 Objectives

To achieve the aim, as laid out in Section 1.2, the project can be decomposed into the following objectives:

1. Produce a method to reliably track how “open” an eye is over time
2. Analyse the resulting data to determine whether a video is fake or not, accurately
3. Evaluate the performance of the model on standardised benchmarks
4. Implement state-of-the-art traditional models
5. Implement a variety of methods of adversarial noise to target those models

6. Evaluate how responsive blink-based detection is to adversarial noise

The above objectives must be completed for the project to be deemed a success. If they have all been successfully achieved, then the following objectives could be implemented given sufficient time:

1. Evaluate the transferability of blink-based detection to datasets it has not been trained on
2. Review possible noise reduction methods for DeepFake detection

1.4 Novel Contributions

The project has the following novel contributions to the field of DeepFakes:

1. First use of complex time series analysis for blink-based DeepFake detection
2. First evaluation of blink-based DeepFake detection on common datasets
3. First evaluation of transferability of blink-based DeepFake detection

Chapter 2

Literature Review

2.1 A History of DeepFakes

2.1.1 Photo Manipulation

Humans have been manipulating photographs since the 1860s. The first known example of photographic manipulation was when a photo of United States President Abraham Lincoln was composited onto fellow politician John Calhoun's body[117], shown in Figure 2.1a. Photo tampering was then used throughout history as a way to shape the opinions and beliefs of individuals by altering supposed "evidence". As photography was still an entirely analogue process, editing a photo was significantly harder than it is today; nevertheless, it was still done when deemed necessary. Another famous example is the removal of Nikolai Yezhov in 1940 from a 1937 photo with Joseph Stalin after the "Great Purge" in the Union of Soviet Socialist Republics (Figure 2.1b).

(a) A spliced portrait of Abraham Lincoln (left) with the original (right)[117]

(b) A photo of Stalin and Yezhov (left) which was subsequently edited to remove Yezhov (right)

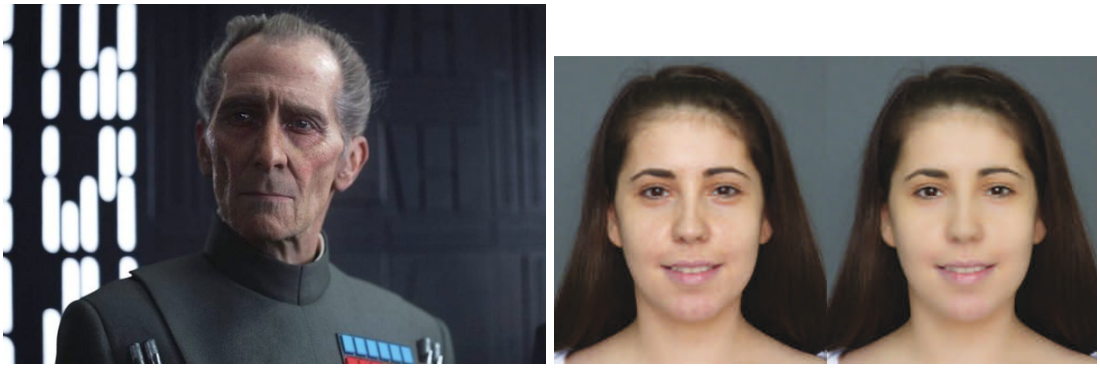
Figure 2.1: Examples of historical images that were edited

With the advent of digital photography and computers, photo manipulation became available to the general public. Suddenly, images were not physical items but data stored on a computer, which could be easily manipulated. Changes could be viewed instantly, reversed, and shared easily. Programs to edit images were released, such as Adobe Photoshop[1] and GIMP[2]. Although these tools made photo

[1] `https://www.adobe.com/uk/products/photoshop.html`
[2] `https://www.gimp.org/`

editing cheap and accessible, they still required time, effort, and skill from humans in order to produce a convincing forgery. An original image is also required for manipulation.

Computer Generated Imagery (CGI) is a technique primarily used in movies to artificially augment a frame, often with completely novel additions. CGI was first used in the 1958 film "Vertigo" and became widespread in the 1990s[89]. Presently, CGI is used in almost every major film to generate realistic images. In "Rogue One: A Star Wars Story" the actor Peter Cushing digitally recreated after his death in 1994 to play Grand Moff Tarkin posthumously(Figure 2.2a). While CGI requires large amounts of computational resources, consumer-grade image generation which runs on much worse hardware has equally progressed. Most modern social media apps contain a variety of beauty filters[24]. For example, a blur can be applied to reduce skin blemishes as shown by Figure 2.2b. These are often less computationally expensive and can run on smartphones and other edge devices, offering the ability to manipulate media to the masses, not just the technically able and skilled.



(a) A digital recreation of Peter Cushing for the film "Rogue One: A Star Wars Story"[34]

(b) An original image (left) blurred to remove skin blemishes (right)[24]

Figure 2.2: Examples of digital image manipulation

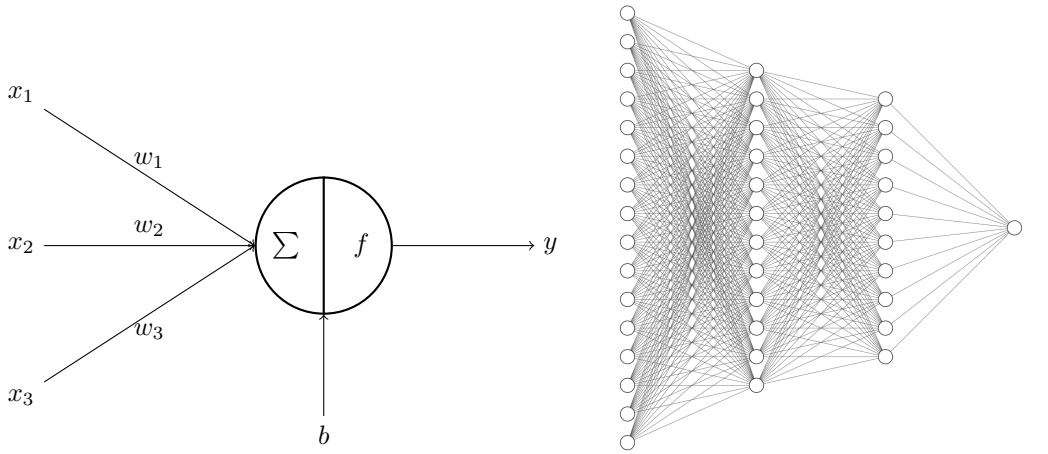
2.1.2 Artificial Intelligence for Manipulating Media

Neural Networks

DeepFakes are a subset of neural networks. Neural networks are a subsection of AI which imitate the structure of a brain to perform computations[59]. Traditional computation manipulates binary data, whereas neural networks manipulate the connections between fixed binary elements. Work on simulating the brain using mathematical functions began in 1920[16], 37 years before the academic formalisation of AI in 1957 at the Dartmouth Workshop[26]. The field would remain relatively stagnant until the proposal of the Multi-Layer Perceptron (MLP) in 1958[106] and how it could "learn" in 1967[61].

Neural networks contain a collection of neurons organised into layers, each layer receives inputs from the neurons earlier in the network (Figure 2.3b). Each of the n inputs to a nueron has an associated weight (w_i) and value (x_i). Once the combined weights and an additional bias weight (b) exceed a determined threshold (usually 0), then the neuron fires with an output (y) determined by an activation function ($f(x)$). This is shown in Equation 2.1 and Figure 2.3. The network shown in Figure 2.3 has a one-dimensional input, however, MLPs can expand to an infinite number of dimensions.

$$W = b + \sum_{i=1}^{n} w_i x_i \qquad y = \begin{cases} f(W) & \text{if } W \geq 0 \\ 0/-1 & \text{otherwise} \end{cases} \tag{2.1}$$



(a) Structure of a single artificial neuron (b) A network of neurons split into layers

Figure 2.3: A structure of a neural network and the individual neurons within it

Neural networks are able to learn. When given a set of labelled data, it is possible for a neural network to use specialised algorithms to adjust the weights and biases in the network so that the final output is similar to the labels provided. A detailed explenation of learning is provided in Section 2.1.2

Convolutional Neural Networks

Although MLPs are exceptionally versatile, they can suffer from overfitting. Overfitting is where a machine learning model learns the training data but not a general solution, becoming exceptionally accurate in the training data, but underperforming on test data. With every neuron in an MLP layer being connected to all other neurons in the layers ahead and behind it, the network is prone to overfitting[87]. A variety of methods can be employed to reduce overfitting, but the most common way is to remove connections from the network by reformatting the layers, this also has the attractive benefit of quicker computations as there are fewer weights to sum resulting in faster training and inference times. Several methods to reduce connections have been proposed, but the most prominent are Convolutional Neural Networks (CNNs).

A CNN is a sequence of three layers: convolution layers, pooling layers, and fully connected layers[58]. Each set of layers reduces the dimensions of the input, allowing each subsequent layer to take into account a larger portion of the image.

- **Convolutional Layer**
  A Convolutional Layer is what a CNN lends its name to. It has two stages. The first stage is a filtering stage, a kernel (sometimes called a filter) is passed over the input. The kernel is placed over a section of the input and the dot product between the values in the kernel and the input is calculated. This then becomes the input to the next layer. The kernel is then shifted over by a fixed stride value to analyse another subset of the input. This process is shown in Figure 2.4. The values (or weights) of each filter remain constant as it moves across the image; there can also be many filters, increasing the dimension of an input. The weights are what are learnt by this layer. Reducing the values to be trained to the size of the filter significantly speeds up the training process. Often some activation layer will be included to process the output matrix. More often than not, this uses the Rectified Linear Unit (ReLU) function, to limit outputs to strictly positive in order to introduce linearity and prevent the vanishing gradient problem (which can hinder learning)[58].

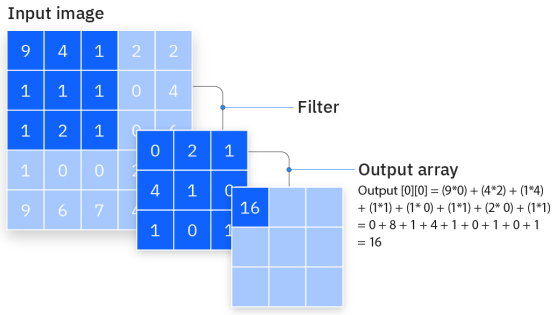


Figure 2.4: A diagram showing the action of a 3×3 filter[58]

- **Pooling Layer**
  The Pooling Layer is similar to the convolution layer but with no learnable parameters. Instead of a weighted kernel, an aggregation function pools multiple values into a single value[87]. Any function could be used, but the most common ones are: max pooling where the maximum value under the kernel is selected; or average pooling where the average of all the values is chosen. Pooling layers further reduce the number of learning parameters: reducing overfitting, model complexity, and computation time[58].

- **Dense Layer**
  To produce an output that is usable, a final dense layer is needed to convert the output of the previous layers into a desired shape. This is a fully connected MLP network. Often these will be referred to as dense or fully connected layers as every neuron is connected to every other neuron.

A sequence of these layers are combined to produce a network. The exact sequence and order is defined by the network's architect and depends on the quality of output they want. More layers mean a higher probable accuracy, but incurs increased computation costs and tendency to overfit. Further techniques exist to enhance CNNs such as skip connections, which results in the output of layers to "skip" forward layers and act as the input for layers further down the network. An example of a CNN for classifying images is shown in Figure 2.5:

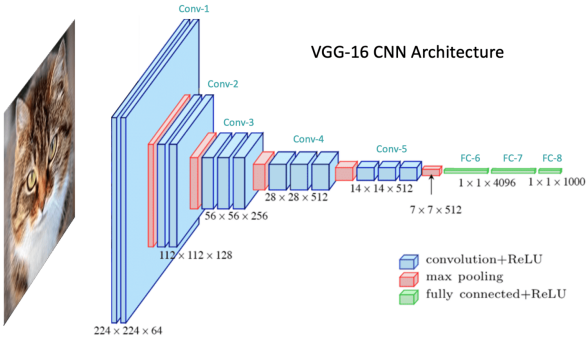


Figure 2.5: A sample CNN[72]

Learning

The process of learning is similar across all kinds of neural networks. Every network will have trainable parameters (the kernel in a convolutional layer or the weights and biases in an MLP) which are initialised randomly. All learnable parameters are then trained in CNNs via backward propagation of error (commonly shortened to backpropagation)[104].

This method of training requires a train set: a set of inputs and their known correct outputs (sometimes called labels). The backpropagation algorithm proceeds as follows:

1. Each input is passed through the network in a forward pass. This produces one or more outputs.
2. The predicted outputs are compared to the known truth outputs. A user-defined loss function evaluates how "correct" the network's predictions are, producing a single value. Common loss functions are categorical cross-entropy for classification tasks and Mean Squared Error (MSE) for location problems[11]
3. A backwards pass is then performed to compute the gradients of the loss function with respect to each learnable parameter. Every parameter x in the network is connected to the output through a composite function $f(x)$, incorporating all relevant activation functions and weights. By applying the chain rule and computing the partial derivative ($\frac{\partial L}{\partial x}$ where L is the loss function), it can be ascertained how much each parameter influences the overall error.

For every parameter, how the loss function varies when x is varied can be expressed as a graph (Figure 2.6). By finding the value of x that minimises the loss function for each parameter in the network, the network's accuracy can be improved. Note that the graph will often be more complex, compromising of a variety of hills, valleys, and plateaus.

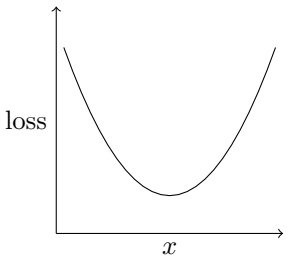
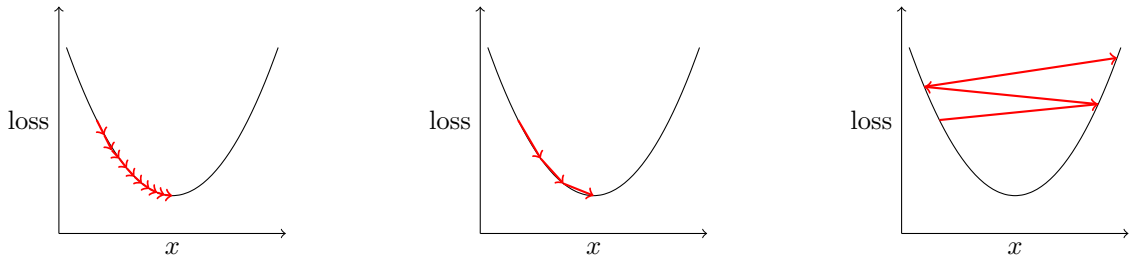


Figure 2.6: A sample plot of the value of the loss function over varying values of a parameter x

A variety of methods exist to find the minimum value for x in the least possible steps. The most widely used is Stochastic Gradient Descent (SGD)[103], first applied to neural networks in 1967[5]. SGD follows the following formula:

$$x_n = x_{n-1} - \alpha l(x) \tag{2.2}$$

x_n is the new value of parameter x which had a previous value x_{n-1}. The value is altered by $l(x)$, the gradient produced by backpropagation for x. α is the learning rate, a hyperparameter. Often a number close to 0, this limits how much the value can vary within the graph. Too low of a learning rate results in slow learning and the potential to get stuck in local minima or plateaus. A high learning rate leads to the possibility of divergence and the minimum never being found.



(a) Too low learning rate (b) Optimal learning rate (c) Too high learning rate

Figure 2.7: Effect of different learning rates on loss function optimisation

Traditionally, finding an optimal learning rate has been a hard problem and thus a variety of methods have been introduced to give a larger safety net when choosing parameters. The most popular of these is the Adaptive Moment Estimator (commonly shortened to Adam)[68].

Adam combines the advantages of two popular optimisation techniques: adaptive learning rates and momentum. Inspired by AdaGrad [32] and RMSProp, Adam maintains a per-parameter learning rate that adapts based on the first and second moments of the gradients. The adaptive learning rate allows the optimiser to apply larger updates for infrequently updated parameters and smaller updates for frequently updated ones. Momentum, on the other hand, helps accelerate the optimisation process by keeping track of the previous gradients, thus smoothing the updates and enabling the optimiser to handle complex loss functions easier.

When training, the above process is done for every learnable parameter for every item in the train set. Often, the train set will be too small to enable meaningful learning. To combat this, the set will be iterated over multiple times. One iteration over the entire dataset is referred to as an epoch[11]. Often training will be done for a set number of epochs or until the loss function stabilises.

Generative Artificial Intelligence

Whilst editing an image was relatively easy with modern photo editing tools, adding original content to an image was more difficult, requiring specialist expertise. The difficulty is further increased when trying to edit a video as more images need to be edited and stitched seamlessly together in a consistent manner. Generative AI (GenAI) is a form of neural network that can generate novel content. In images, they can augment images to fit a specific goal or, in more recent works, convert written prompts into completely original images and videos.

The first notable GenAI was Generative Adversarial Networks (GANs)[46]. GANs are two neural networks that work in opposition to each other, a generator and a discriminator. The training process is a minimax two-player game: when one model loses by a certain amount, the other model gains an equal amount. In the context of image generation, the generator creates novel images based on a dataset; the discriminator is fed a combination of the dataset and the generator's images, attempting to determine their origin. A diagram of a GAN network is shown in Figure 2.8. The discriminator is a CNN, whereas the generator is a deconvolutional neural network[135]. Deconvolutional neural networks are CNNs but in reverse, where the deconvolution layers convert a single value into a $n \times n$ square of new values. To produce an image, the generator is fed random noise, which it converts into a novel image. Over time, the generator becomes better and better at generating convincing images from noise that fool the discriminator, eventually resulting in images that appear identical to the images from the dataset.

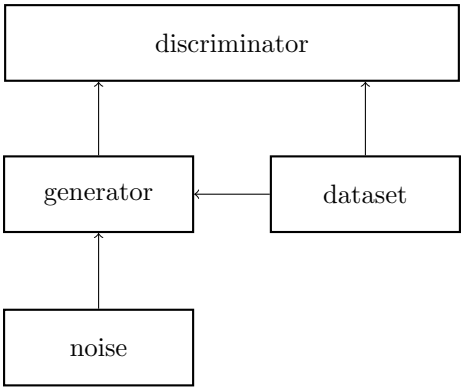


Figure 2.8: A diagram of the network structure of a GAN

Even in the original paper, GANs were being used to generate images of humans[46]. In 2017, the technology would expand beyond the realm of academia and into the public eye. A user named `u/deepfakes` and others on the internet forum site Reddit[3] were discovered to be sharing pornographic material, altered to contain popular celebrities using GANs[23]. Users were taking "conventional" pornographic videos and using GANs to swap the faces in the original video to famous individuals without their consent. By only generating the face, these networks required less computational power and thus videos could be generated quickly. Following legal pressure, Reddit deleted the forum two months later[23], but the damage was done: AI had been shown to generate realistic videos of humans semi-autonomously. The media soon caught on to the story using the subreddit's name to coin the new types of videos as "DeepFakes", a portmanteau of "Deep Leaning" and "fake".

Other methods of generating DeepFakes have been developed since GANs. Diffusion[105] is a process of removing noise from images based on transformers[128]. When given random noise, these models can generate completely novel images. When trained on images with text descriptions, images could be generated from natural language prompts. Multiple different companies began making image generation tools such as OpenAI[97], Stable Diffusion[119], and Midjourney[82]. OpenAI would eventually create Sora[15] a model that could generate realistic videos from a text prompt. The most recent architecture for image generators is Visual AutoRegressive modelling (VAR)[126]. Based on the transformer, it allowed for image quality to scale with the amount of computational power available, enabling OpenAI to release 4o Image Generation[88]. Realistic images could now be generated from a simple text prompt requiring no specialist knowledge, only imagination. An example of such an image is shown in Figure 2.9.

2.1.3 DeepFakes for Malicious Purposes

DeepFakes can generate realistic images that are undetectable by humans. When presented with a combination of real and fake videos, individuals were asked to classify them as either real or fake; the overall accuracy was 55.54%[30], only slightly better than guessing. DeepFakes were initially used for entertainment purposes; for example, individuals would use them to add the actor Nicholas Cage to movies he did not originally star in[49]. However, DeepFakes soon began to be used for various malicious purposes.

DeepFakes have been used by state actors and opponents to misrepresent candidates in elections across the globe. In the 2016 United States election, Russian-generated fake videos were published on social media to "undermine public faith in the U.S. democratic process, denigrate Secretary [Hilary] Clinton, and harm her electability and potential presidency"[49]. A similar misinformation campaign was attempted

[3] `https://reddit.com`

Figure 2.9: An image generated by GPT-4o[88] using the prompt "a chocolate labrador with glasses programming on a computer"

by Russia during the 2017 French elections where documents were stolen from Emmanuel Macron's campaign team and edited in an attempt to hinder his chances of being elected[20]. They were unsuccessful in this campaign due to Macron's team's swift action in countering the documents published.

On an individual scale, DeepFakes are used by scammers to defraud vulnerable people out of their money. A number of celebrities' likenesses have been used to promote products without their consent. Quantum AI is one such DeepFake-based scam[19]. DeepFakes videos of celebrities local to the area (such as Rishi Sunak in England or Justin Trudeau in Canada) were DeepFaked and posted to social media, endorsing the tool which guaranteed "$3,000 as early as day one". The second phase of the scam involved sockpuppets (non-existent people entirely AI-generated) posing as local media reporting on the fake endorsements, further promoting the scam. Victims are prompted to invest a small amount of money to start with, often receiving genuine returns; soon, however, victims would be locked out of their account with their money stolen.

Another disturbing use for DeepFakes has been around since their inception: pornography. Approximately 96% of all DeepFakes generated are pornography[3]. Long video sequences can be created from a handful of images scraped from an individual's public profiles[20]. The videos generated are humiliating for the victims they depict, making them feel extremely vulnerable. Often, these videos are used to further blackmail victims, threatening to release the videos publicly if the victim does not comply.

2.2 DeepFake Detection

With all the harm DeepFakes can do, it is essential to be able to detect them reliably and accurately. Early DeepFakes were easy to spot, often with warping around facial features and other temporal irregularities[30]. Unfortunately, DeepFakes have become increasingly realistic and this trend is only set to continue. Taking inspiration from the GAN model (Figure 2.8), tools were developed to detect DeepFakes using AI. Whilst in the GAN architecture the generator is meant to win, discriminators are being generated to accurately detect DeepFakes. Significant investment and research are being made into DeepFake detection; the US government invested $22,000,000 into DeepFake detection research with the

Media and Semantic Forensics program[49].

2.2.1 Traditional Neural Network Detection Methods

CNNs have been used to detect DeepFakes since the original GAN paper[46]. Although CNNs are infinitely customisable with the different choices of layers and connections, often CNN architectures will consist of a backbone and a head[35]. Backbones are large convolutional neural networks that have been pre-trained on a large dataset that can extract features from an input. A custom head is then added to the network to convert the features into the desired output.

In the context of DeepFake detection, the backbone is trained on ImageNet[29] an image dataset with 3.2 million images. Backbones are initially trained to classify an image in the dataset to an overall class (for example, "cat"). The backbone is then released with the weights from the training. For DeepFake detection, the head ends with a fully connected layer with a binary softmax (Equation 2.3) to classify an image as real or fake. Softmax separates an N large input vector z into separate probabilities, representing the probability that an input is in class y such that $\sum_{i=1}^{N} y_i = 1$.

$$y_i = \frac{e^{z_i}}{e^{z_1} + e^{z_2}}, i \in 1, 2 \tag{2.3}$$

Using the final weights from a previous problem as the initial weights for a similar new problem is a technique called transfer learning[13] and can vastly improve a network's learning time and accuracy. The header's learnable parameters are initialised randomly. To extrapolate image classification to video classification, each frame of the video is independently classified and then the classifications are aggregated. Various aggregation methods exist, but the simplest one is to set a threshold of frames that can be flagged as DeepFaked before the overall video is classified as fake.

Several backbones have been applied to DeepFake detection with varying success[125]. The first backbones to be applied to DeepFake detection were VGG16/19[116] and ResNet[50]. VGG is a typical CNN as outlined in Section 2.1.2, ResNet introduced the concept of skip connections, allowing for the interaction of high and low level features.

State-of-the-art backbones have also been applied to DeepFakes, the best performing CNNs are currently based on the EfficientNet[121] and Xception[22] backbones. These achieve impressive accuracy on standard benchmarks (Section 2.5), achieving over 90% accuracy[12].

EfficientNet is a model based on the MobileNet[54] architecture, which aims to reduce the size of neural networks. EfficientNet is built upon depthwise separable convolutions. A traditional convolutional layer will analyse every channel of an input (such as the red, green, and blue channels in an image) at once, and then combine them in a single step for every pixel in an image. A depthwise separable convolution layer instead analyses each input channel independently and then combines them afterwards. This vastly reduces the computation required as learnable parameters are separated from each other in different stages, as such when combined over larger images the parameters combine linearly rather than exponentially, reducing computation costs. Bottlenecks further speed up EfficientNet by reducing the dimensions of feature vectors. As data flows through a traditional CNN it can reach a large number of dimensions, bottlenecks reduce the dimensions of a neural network which exponentially increases the network's speed. Bottlenecks reduce the dimension by having fewer neurons than the layer before it, forcing data to be aggregated and hence reducing complexity. Whilst MobileNet was intended to reduce the size of CNNs, it can make larger ones more efficient on the same number of parameters. EfficientNet uses MobileNet to allow it to contain more layers and parameters, increasing overall accuracy.

Xception has an architecture similar to ResNet with skip connections. However, rather than direct skips, residual layers pass through a convolution layer to enhance predictions. Furthermore, the same residuals are used in different layers of the network to keep higher-level features in context throughout the network.

CNN detection works by pixel-level analysis of images. Due to the nature of CNNs, it is difficult to tell exactly what features they are focussing on for the final classification. However, thanks to attention layers, it is possible to view the regions of interest (shown in Figure 2.10), implying that the particular areas of attention are the front facial features. Often these will be the areas that DeepFakes will have the hardest time accurately replicating and will often leave artifacts, such as inconsistent shaping[129]. CNNs are then picking up on these artefacts as markers to classify a DeepFake. This leaves them vulnerable to possible pixel-based attacks which can disrupt the initial feature-extraction[43]. Furthermore, reliance on specific artefacts causes CNNs to only be particularly effective on the DeepFake methodology they were trained and hence suffer reduced accuracy when viewing novel DeepFake methods[125].

Figure 2.10: Output from attention layers of convolutional neural networks on DeepFake detection[12]

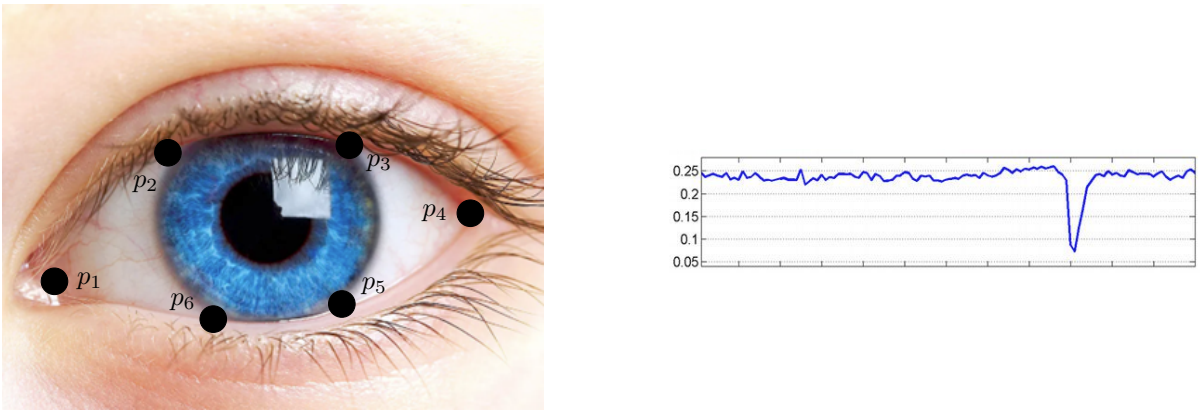
2.2.2 Blink-Based Detection Methods

DeepFakes have been shown to suffer from temporal inconsistencies[62]. Temporal inconsistencies are where videos suffer from unrealistic deviations between frames. For example, an object in the background may be temporarily obscured by the subject but when they move away from the object, it will have disappeared.

A more subtle but ever-present temporal consistency is blinking. Humans blink predictably, with the average person blinking for 0.1-0.4 seconds and a 2.8 second blink period[114]. These timings can vary according to several factors like age, gender, time of day, and activity[63]; nevertheless, blinking remains a consistent physiological act that can be predicted, which DeepFakes struggle to reproduce. Blinking patterns can therefore be leveraged for DeepFake classification.

Eye Aspect Ratio

To determine when an individual is blinking or not, the most common approach is the Eye Aspect Ratio (EAR)[118]. The EAR is a ratio of the distance between six eye landmarks (Figure 2.11a). Two landmarks are labelled as the caruncle and lateral canthus (p_1 and p_4, respectively) and the others are equidistant around the eyelid, labelled clockwise. The EAR is the ratio between the height and width of the eye (Equation 2.4) and is relatively constant when the eye is open but verges close to zero when closed. It is partially consistent between people and variations in poses. The modal person has two eyes and so the final EAR is often taken as the average of the two eyes. When plotted as an EAR-time graph (Figure 2.11b), a blink is evident as a steep drop in the graph.



(a) An image of a human eye with the 6 points necessary for the EAR labelled

(b) A graph of EAR over time with a blink in the final third [118]

Figure 2.11: The points and resulting graph for the Eye Aspect Ratio

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2||p_1 - p_4||} \tag{2.4}$$

DeepVision

DeepVision[63] is a detection model that utilises the EAR and other environmental factors to determine whether a video is DeepFaked or not by comparing its blink patterns to a database of known correct blinking patterns.

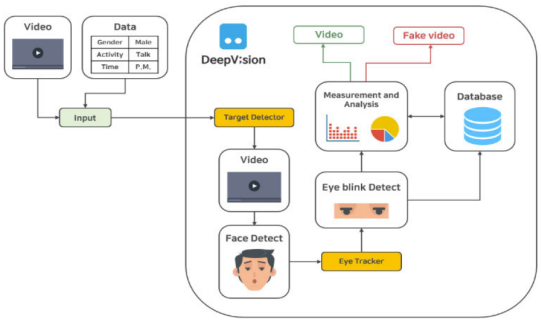
DeepVision's input includes not only the video to be classified but also several environmental factors. A user must manually estimate the: gender, age, time, and activity (whether the subject is moving or stationary) of the main subject of the video.

Two algorithms are used to detect a blink. The first algorithm is a target detector that uses Fast-HyperFace[98] which is a pre-trained CNN that can identify subjects in a video and produce basic landmarks. If the detection confidence is over 70%, the subject is identified as the target and a basic crop of their face is calculated from the landmarks and forwarded to the next algorithm.

A second algorithm takes the face crops and locates the landmarks (p_{1-6}) for the EAR. Whilst the exact method is not defined, it is assumed to be the neural network used in the original EAR paper, Intraface[132]. From these six points, the EAR is calculated for each eye and then averaged across the two. A blink is defined as when the EAR drops below a certain threshold for multiple consecutive frames. DeepVision sets this threshold as two standard deviations below the mean. Data such as when the blink occurred in time, the period of blink, and frequency are then used as features for the next step.

DeepVision compares these features with known good features from a database. The database was created using data from the Eye Blinking Prediction Dataset[1]. The database is queried for individuals in the same environment as the subject and the corresponding features of blinking across the video are returned. The database's features are compared to the video's and if they are "within an allowable" range the video is declared as real, otherwise fake. DeepVision achieves an overall accuracy of 87.5%.

DeepVision's primary advantage is that it is easy to compute and implement. Fast-HyperFace is a pre-trained model and therefore is a simple drag-and-drop implementation. Once p_{1-6} are located in a frame, it is a simple calculation to determine the EAR. The subsequent statistical analysis to determine whether blinking is consistent with real videos is also relatively simple and not computationally intensive



(a) A flow diagram for DeepVision[63]

(b) A sample database query for DeepVision[63]

Figure 2.12: DeepVision's architecture and database queries

as it is a database query. Therefore, any methodology utilising DeepVision would be fast to implement and relatively easy to debug. It also benefits from more granularity in eye state data which could enable more complex analysis.

The primary downside to DeepVision is the potential occlusion of the eye(s). The EAR requires the entire eye to be visible in the frame of a video to identify all the points. A partial EAR can be calculated with a single eye, but the accuracy of this is less than if two eyes were visible. The second disadvantage is the database required to determine whether blinking is consistent with normal blinking. The database requires many labelled examples which would be time-consuming to produce. Furthermore, the exact methodology for determining whether a video's data is similar enough to the dataset's is never explained, nor is the "allowable" range, resulting in DeepVision's accuracy being unlikely to be replicable.

Ictu Oculi

Another method for blink-based DeepFake detection is Ictu Oculi[77]. It uses a more complex method of blink detection which allows a simple method for blink verification.

An initial pre-processing step is done to identify the faces of the image using dlib's[67] face detector and distort and crop them so that the face is: in the centre of the image; rotated such that the eyes form a horizontal line; and scaled to a similar size across the duration of the video. The face tracked in the video is the one which the network predicts with the highest confidence. This face is further cropped to just the eyes which is then passed to the main model.

To determine the current state of the eye (open or closed), Ictu Oculi uses a custom Recurrent Neural Network (RNN). RNNs are similar to CNNs but contain recurrent connections that can loop backwards through the network, allowing for a form of memory. This makes RNNs very effective for analysing time-based data as each position in time can be predicted using the context of the previous prediction; unlike CNNs, which would analyse each position independently.

The specific form of RNN used by Ictu Oculi is Long Short-Term Memory (LSTM) modules[52]. LSTM modules dictate how many of the previous states to remember and how much influence the previous states have on the current prediction. Ictu Oculi consists of three main stages. The first stage is feature extraction using a VGG16-based CNN, discriminative features are extracted from the cropped eyes. The features and the previous LSTM's output are fed into an LSTM in the sequence learning stage. This takes temporal features into account to enhance the blink detection; for example, if the previous five frames have all shown the eye in the process of closing, it is likely the current frame will also show the eye closing. The final stage is the state prediction, which uses a dense layer to determine whether the eye is open or closed. The complete diagram is shown in Figure 2.13.

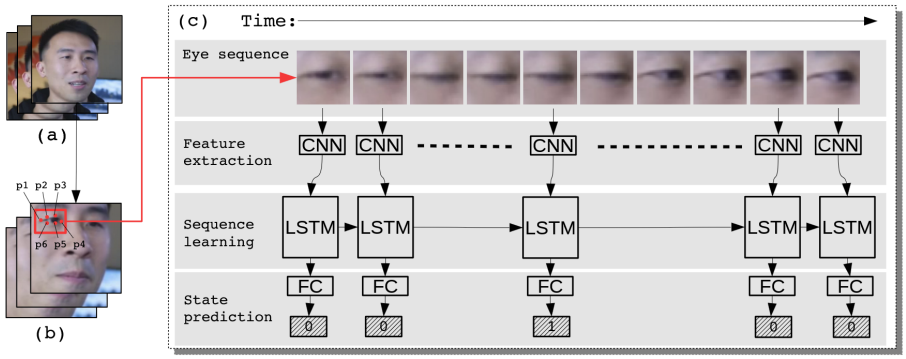


Figure 2.13: A network diagram of Ictu Oculi[77]

As the accuracy of data is reduced from the amount an eye is open to a binary open or close, the complexity of the validity algorithm needs to be greatly reduced. The algorithm implemented by Ictu Oculi detects the number of blinks in a set period (60 seconds), the average person blinks 34.1 times a minute. A video is deemed fake or real based on how closely the blinking in the video matches this number. This leads to an accuracy of 99% when tested on a custom dataset of 32 videos.

Ictu Oculi has a much more complex blink detection system, requiring two separate machine learning models trained to identify if a blink is happening or not. This is more complex and requires pre-processing resulting in longer development and training times. On the other hand, it results in a much more resilient blink detection mechanism as the eye's state can be accurately determined even when partially obscured.

The drawback is that the state of the eye is binary open or close, any information relating to the speed of the blink is lost resulting in a very rudimentary system to determine whether a sequence of blinks is a DeepFake or not. Yet, this has little effect on the accuracy of detection.

2.3 Adversarial Noise

When operating on images, CNNs heavily rely on the individual pixels in the image. When these pixels are corrupted by noise, it can seriously degrade the performance of CNNs[134]. For DeepFakes, this noise can cause a misclassification: a fake video could be classed as real, and vice versa. Adversarial noise is deliberate noise added to an image to cause a desirable misclassification. Perturbed images are meant to be imperceptible to humans to avoid manual detection.

Adversarial noise becomes especially dangerous if a CNN is trusted to be accurate. Hypothetically, a CNN is produced that, on unperturbed videos, is 100% accurate and hence its classifications are trusted absolutely. A malicious actor could add noise to their DeepFake, causing the CNN to declare it real, resulting in the DeepFaked video being trusted as real. Such a scenario is extremely dangerous and so it is important to develop noise-resistant detection models.

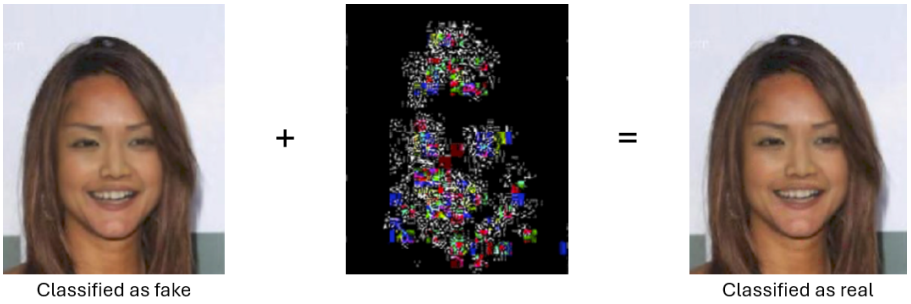


Figure 2.14: The process of adversarial noise being added to an image to cause a misclassification[43]

There are a large number of methods that have been proven to cause CNNS to misclassify. A smaller subset of these have been proven to be effective in DeepFake classification. It is important to note that adversarial noise is only tested and applied to fake images, as when used in the real world, only the fake videos will be perturbed to make them seem real.

2.3.1 Fast Gradient Sign Method

Fast Gradient Sign Method (FGSM)[45] is the simplest method to add adversarial noise to an image. An image is attacked as follows:

$$\mathbf{x}_{adv} = \mathbf{x} + \varepsilon \text{sign}(\nabla_x J(\mathbf{x}, \mathbf{y}, \theta)) \tag{2.5}$$

$\mathbf{x}_{adv}$ and $\mathbf{x}$ are the perturbed and original frames, respectively. Note that the images are normalised so that $\mathbf{x}_{adv}, \mathbf{x} \in [0, 1]^3$. $J(\mathbf{x}, \mathbf{y}, \theta)$ is the model's (θ) training loss function on image $\mathbf{x}$ and target class $\mathbf{y}$. If the model's loss function is not known, it must be guessed; however, most classification models use some form of cross-entropy. Finally, ϵ is a hyperparameter to control the magnitude of noise. Higher values of ϵ add more noise to the image, increasing the chance of a misclassification from a CNN, but a similarly higher chance of being detected by humans.

The attack works by estimating the gradient of the loss function for the image $\mathbf{x}$. By adding noise in the direction of that loss function, the model becomes less accurate, increasing the risk of misclassification. FGSM is simple and quick to compute as it only requires one call to the CNN being attacked. When tested against DeepFakes, FGSM reduced a ResNet-based detector from 95.4% accuracy on fake images to 7.5% accuracy[43].

2.3.2 Carlini-Wagner L2-Norm Attack

A secondary adversarial noise shown to be effective on DeepFakes is the Carlini-Wagner L2-Norm (CW-L2) attack[18]. It is more complex than FGSM and aims to minimise two objectives: reduce the L2-norm of the noise (Equation 2.7) while still adding enough noise to cause a misclassification (Equation 2.8).

$$\mathbf{x}_{adv} = \frac{1}{2}\left(\tanh(\omega^*) + 1\right) \tag{2.6}$$

$$\omega^* = \arg\min_{\omega} \left(\|\mathbf{x}' - \mathbf{x}\|_2^2 + cf(\mathbf{x}')\right) \tag{2.7}$$

$$f(\mathbf{x}') = \max\left(\max_{i \neq y}\left(\mathbf{Z}(\mathbf{x}')_y - \mathbf{Z}(\mathbf{x}')_i\right), -\kappa\right) \tag{2.8}$$

Equation 2.8 aims to cause the model to misclassify an image. $\mathbf{Z}(\mathbf{x})$ is the pre-softmax vector output of the neural network, otherwise known as the logits of a model. This allows the CW-L2 attack to analyse the features that a model is using for a prediction rather than the final prediction. **x'** is a sample perturbed image, y is the index of the target class, and i is the current classification. κ acts as a hyperparameter threshold defining the maximum amount the logits can be altered. By minimising $f(\mathbf{x'})$, the difference between the logits of the correct class and the target class is maximised, raising the chances of a misclassification.

By minimising the L2-norm of the difference between the original and noisy image in Equation 2.7, the final noise chosen is the one with the least amount of noise, reducing the chance that humans will detect the noise. c is another hyperparameter that controls the relative strength of each objective. Finally, the noise is normalised to $[0, 1]$ in Equation 2.6 to produce the final image.

Often, c is found by a binary search during run time to further minimise the amount of noise on a frame-by-frame basis. To find the argument minimum (Equation 2.7), SGD[103] is used, often with one thousand iteration steps. Both of these result in the CW-L2 attack taking a long time to run in comparison to other methods.

When tested on the same ResNet detector as FGSM (Section 2.3.1), the CW-L2 attack reduced the accuracy on fake videos from 95.4% to 0% accuracy. Furthermore, it is more resistant to various noise-reducing filters that may be employed as defence[43].

2.3.3 FakeRetouch

FakeRetouch[56] is a novel approach to adversarial noise that uses offline-trained CNNs. Instead of targeting a specific model, FakeRetouch aims to reduce a GAN's fingerprints that can show up in the Fourier Transform of an image. As shown in Figure 2.15, GANs have a much more nosier Fourier transform which some CNNs can leverage as a potential DeepFake identifier. FakeRetouch attempts to minimise the effect a GAN can have on the Fourier transform.

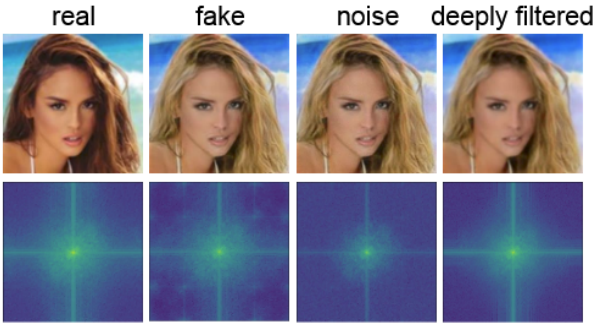


Figure 2.15: A collection of images and their Fourier Transforms

FakeRetouch is defined by the following equation:

$$\mathbf{x}_{adv} = \mathbf{K} \circledast (\mathbf{x} + \mathbf{A} \odot \mathbf{N}_\sigma) \tag{2.9}$$

$$\text{Where } \mathbf{A} = \arg\max_{\mathbf{A}} (J(\mathbf{x} + \mathbf{A}, y, \theta) + ||\mathbf{A}||_1) \tag{2.10}$$

$\mathbf{K}$ is a Kernel Prediction Network (KPN)[83]. This produces a kernel, similar to the filters in a CNN, that averages out a certain area of pixels to reduce the noise in an image. This can be sufficient for

conventional noise reduction but is unfortunately not successful in causing misclassification[56] so further guided noise is required.

FakeRetouch uses Gaussian noise ($\mathbf{N}_\sigma$) to further blur the image, to reduce the perceptibility of the noise, it is passed through a binary map $\mathbf{A}$, which only allows noise in certain regions of the image. It is desirable that $\mathbf{A}$ is as sparse as possible to avoid human detection of the noise, hence $\mathbf{A}$ is regulated by Equation 2.10. J is the same loss function defined the same as Equation 2.5 but is set as the categorical cross-entropy loss function. y is the true class label of the image $\mathbf{x}$. Equation 2.10 regulates the amount of Gaussian noise added to a frame by minimising the L1-norm of the noise.

When tested on DeepFakes, FakeRetouch has limited success. It does not target the neural network directly and so relies on both the neural network analysing the Fourier transform of an image and the DeepFake generator producing excessive fingerprints in the Fourier space. For scenarios where both requirements hold, FakeRetouch can reduce accuracy by 67%; on the other hand, in scenarios where either one or none of the conditions hold, FakeRetouch either decreases detection by an insignificant amount or can even increase the detection accuracy.

2.4 Transferability of DeepFake Detectors

Another potential flaw with CNN-based DeepFake detectors is their lack of transferability[102] and thus generality. CNNs will often score highly, near 100% accuracy, on standardised datasets. However, this is often because they have been trained on the same DeepFake methodologies which they are then being tested on. As such, they may have overfitted to recognise the methods on which they have been trained, rather than becoming a generic classifier able to accurately classify unknown DeepFakes.

This problem was demonstrated by Ricker et al.[102]. Three state-of-the-art CNN-based DeepFake detectors were trained on ten datasets each generated using either a GAN or diffusion model. The results are shown in Figure 2.16. Accuracy is measured using the Probability of detection at a Fixed Alarm Rate (Pd@FAR). As expected, when a model is trained and tested on the same dataset, it achieves close to 100% accuracy. However, when tested on unknown datasets, a model's accuracy would drop significantly. This effect is more pronounced when models trained on GANs are evaluated on diffusion models, however less so when models trained on diffusion DeepFakes are evaluated against other diffusion DeepFakes. Note in the figure that the GAN, DM, and All columns denote models trained on all GAN-based, Diffusion-based, and all models, respectively.

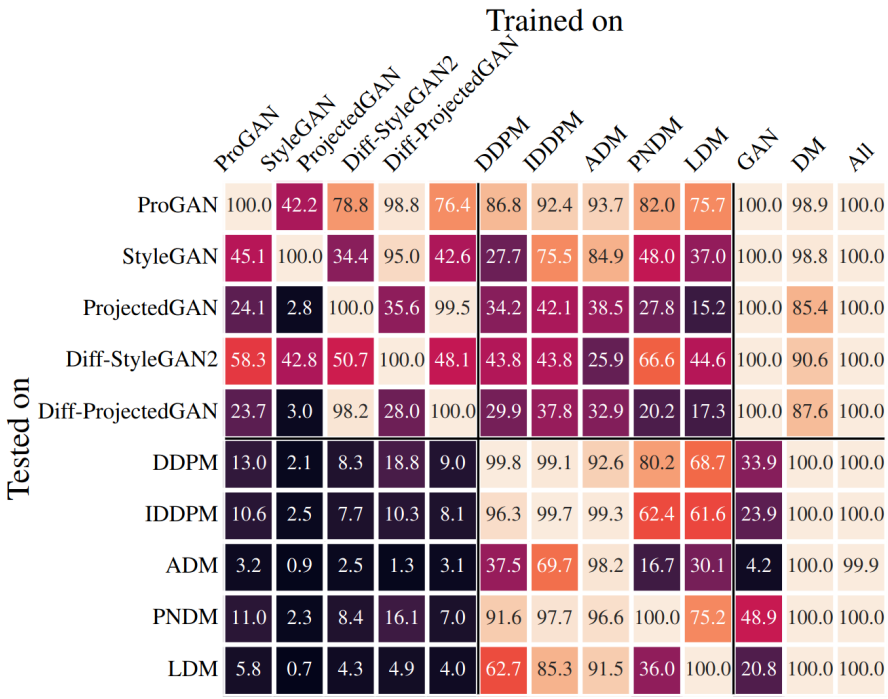


Figure 2.16: A graph showing the Pd@1%FAR accuracy of a variety of detection models[102]

2.5 Datasets

To compare the performance of multiple proposed models, common datasets of DeepFakes have been made. Datasets are large collections of data relevant to a problem that have been collated and labelled by a third party. For DeepFakes, these datasets take the form of videos or images that have been scraped from the internet. These are then passed into at least one DeepFake generator which produces a set of known fake media. This project is only concerned with videos.

FaceForensics++[107][108][33] is one of the most popular datasets for DeepFake detection on videos. 1,000 images were taken from YouTube and then were DeepFaked with four different methods: Fac2Face[124], FaceSwap[70], DeepFakes[37], and NeuralTextures[123]. This results in a total of 4,000 videos to train and test classifiers on.

As part of a 2020 Kaggle competition, Meta released the DeepFake Detection Challenge (DFDC) dataset[31]. Eight facial modification algorithms were used to create a unique dataset of approximately 124,000 videos. The DFDC is by far the most extensive dataset available. A further testest is also available for use that was not available to participants of the competition although this is not used when benchmarking.

Celebrities are often used in DeepFakes due to their popularity. A number of datasets based on celebrities have been created, these are easier to create as videos involving celebrities are easier to license and larger in number than the typical individual. Celeb-DF[78] uses videos of 59 celebrities taken from YouTube. 590 real videos are altered using a custom algorithm to make 5,639 DeepFaked videos. Celebrities were specifically selected to cover a wide range of age, gender, race, and ethnicity. Another dataset that uses videos of celebrities is FakeAVCeleb[66]. As the name suggests, it is an audio-visual model, able to fake both the sound and visual components of a video. Similar to CelebDF, videos were deliberately chosen to promote diversity within the dataset. Five hundred videos were altered using four algorithms to create 19,500 videos.

These datasets were chosen for their comprehensive coverage of video-based DeepFake detections. A large number of DeepFake methodologies produce DeepFakes over a wide range of scenarious. Promoting doversity in datasets allows for a more general classifier which will perform better in real-life.

Chapter 3

Design and Implementation

3.1 Language and Machine Learning Framework

3.1.1 Python

When choosing a language to code this project in Python[1] was a clear choice. Whilst other languages offer better speed (C++[2], Rust[3]), Python's advantage comes from its libraries.

Libraries are modular pieces of code written by other developers that can be integrated into a new piece of software to make common tasks easier. In Python, this is accomplished using the `import` and `from` syntax. Libraries vastly simplify coding as they can vastly reduce complex problems to simple functions. This is especially useful when dealing with complex machine learning functions, advanced mathematical algorithms such as Adam can be reduced to an import. Many Python libraries are written in lower-level languages like C or C++, allowing Python code to benefit from high performance while maintaining ease of use.

The vast majority of popular packages have a Python implementation that can be easily installed by the Python Package Index[4]. Especially for machine learning, Python has some of the largest collections of relevant libraries of any language and is therefore the language of choice for this project.

3.1.2 PyTorch versus TensorFlow

There are many machine learning frameworks in Python, however, the primary two are PyTorch[93] and TensorFlow[2]. They both offer nearly identical feature sets and performance so choosing between them is not a simple matter.

PyTorch was created by Meta in 2019. It is a lot newer and viewed as a more "pythonic" framework[21]. To be more "pythonic" is to be more developer-friendlier by providing a simple interface to the library that developers already familiar with Python should easily pick up on. PyTorch supports a dynamic computation graph which enables dynamic changes of model architectures.

TensorFlow is a much more mature framework by Google released in 2015. It offers similar abstractions as PyTorch but is widely viewed to be slightly harder to develop in[21]. TensorFlow uses an eager

[1] `https://www.python.org`
[2] `https://cplusplus.com/`
[3] `https://www.rust-lang.org/`
[4] `https://pypi.org/`

computation graph meaning no changes to model architectures can be made once a model is defined. Whilst this reduces flexibility during runtime, it allows for more optimisations to be made to the model architecture meaning greater accuracy, smaller model architectures, and sometimes quicker computations. TensorFlow also scales effectively running as efficiently as possible on desktop computers to large GPU clusters.

Unfortunately, TensorFlow and PyTorch are mutually exclusive. Both rely on separate versions of the CUDA backend to allow for GPU acceleration on NVIDIA architectures. Whilst this can be accomplished using virtually environments and other work arounds, it is generally recommended to use one or the other.

Due to their similarities, there is no clear choice between PyTorch and TensorFlow. Some trends have emerged, however, with PyTorch being used for research and development work where iterative improvements are desired whereas TensorFlow is employed for production environments.

TensorFlow was chosen for this project for a number of reasons. Firstly, the flexibility that PyTorch offers is less important to this project as all models being used have had their architectures pre-defined. On the other hand, TensorFlow's ability to scale and optimise for the compute resources will be valuable for this project as it will be developed on smaller desktops but the final evaluation loops will be run on large GPU clusters. Furthermore, after trying out both TensorFlow and PyTorch, the author preferred the overall syntax of TensorFlow.

3.2 Proof of Concept

To validate whether blink-based DeepFake detection is resistant to adversarial noise, it was decided to construct a proof of concept during the Christmas holiday. As it is intended to be a small test-bed, small flaws in the design are acceptable. Therefore, it makes sense to use pre-existing models, solutions, and implementations where possible in order to speed up development.

The overall architecture of blink-based DeepFake detection is relatively simple. First, the landmarks around the face are computed to determine for every frame to generate an EAR-time graph. The graph can then be analysed to determine whether the video is real or not.

Algorithm 3.1 Overall architecture of a blink-based DeepFake detector

```
Require: video
  ears ← {}
  for frame ∈ video do
      landmarks ← GetLandmarks(frame)
      ear ← CalculateEAR(landmarks)          ▷ If no landmarks, no value appended
      Append(ears, ear)
  end for
  classification ← EARAnalysis(ears)
```

It was decided to use an EAR-based approach over a custom blink model because it allows for complex analysis. EAR data is more precise than the binary open or close of a custom model, allowing for analysis of not just the frequency of blinks but more subtle details like how the blink starts and ends, the velocity of the eye, and whether the eye consistently returns to the same position when open. The flaws in EAR noted by Ictu Oculi[77] are valid but they rely on the eye landmarks being unreliably marked, with sufficiently advanced models trained with occlusions this can be overcome.

To calculate the EAR, the six landmarks around the eye need to be located. Google's MediaPipe[79] was used for this purpose. It is designed to run in real-time on mobile devices and other edge computing

resources and hence is very lightweight and fast. MediaPipe produces 478 landmarks in three dimensions that cover the entire face, a subset of twelve (six landmarks per eye) is then used to be the six landmarks for EAR. From the six landmarks, the EAR can be calculated using Equation 2.4 and the average taken for the final EAR-time graph. If no landmarks can be detected for the frame, no EAR value is appended to the list.

EAR analysis is performed using the hybrid of DeepVision[63] and Ictu Oculi[77] by counting the number of blinks in the EAR graph and comparing that to the human average. DeepVision's database was viewed as too time-consuming to produce and the simplicity of Ictu Oculi's approach was appealing for its speed advantage. In experiments, DeepVision's threshold of two standard deviations below the mean proved to be too inconsistent so a revised threshold of half a standard deviation above the minimum EAR value was chosen. The average human blinks 14 times per minute[114], DeepFakes will blink less than this and so if the subject of the video blinks less than 14 times a minute, the video is deemed as a fake. For an accurate classification, a minimum of 30 frames need to be in the final EAR list. If there are fewer values in the final list, the video classification of the video is unknown.

For traditional detectors, both ResNet and VGG architectures have been proven vulnerable to adversarial noise[43]. A ResNet-based DeepFake detector was implemented using an implementation from Tiwari[127], a VGG-detector was implemented using a header from Krishna et al[71] but the backbone was upgraded to VGG19 following research from Yadav et al[133]. To extend frame-by-frame detection to entire videos, each frame in a video is analysed independently. If the frame is classified as fake then a counter is incremented by one. If the counter exceeds a given threshold (set as one hundred frames), the video is classed as fake.

Adversarial noise was added to images using the Foolbox library[99][100]. FGSM was used as the method for noise due to its known effectiveness and speed[43]. ϵ was set to 0.1. To replicate real-life scenarios, noise was only added to faked videos and all models were solely trained on original videos, not noisy ones. This is because models would not be able to be trained on videos that have been manipulated by attacks specifically designed to disrupt the very model that is training. Only the VGG model is attacked during the proof of concept, with the ResNet model acting as a control model to show the effects of indirect noise on a model.

The models were trained on a subset of fifty real and fifty fake videos from the FaceForensics++ dataset[33]. Testing was run over a further fifty real and fifty fake videos. The results of the tests can be found in Section 4.1. FaceForensics++ was chosen as it was the most robust dataset available at the time. More have been tested in the main code. Hyperparameters for the VGG and ResNet models were left the same as the original implementations. To reduce overfitting, only one frame per second was extracted and used for training. To ensure a realistic test, no model is ever trained on images that have been injected with adversarial noise.

3.3 Main DeepFake Detection Algorithm

The main DeepFake detection algorithm is similar to the one discussed in the proof of concept (Algorithm 3.1) in terms of structure. The code in the main algorithm is an alteration of the subroutine calls, making each subroutine return more accurate results. Other improvements were made such as general improving speed and modularity of the code. To further improve testing and evaluation, a number of different options for each component were developed as each section of the algorithm is interchangeable.

To improve resiliency against adversarial noise attacks, the entire algorithm is fail negative. If any section of the algorithm were to fail then the video would be deemed a fake. For example, if no facial landmarks are detected in a frame, then the video is declared fake.

Whilst all of the separate components, such as facial landmarking models, are pre-existing neural networks, certain novel contributions have been made. The first is the use of univariate time series analysis of the EAR graph. This allows for more sophisticated and complex analysis to be done, which will hopefully allow for consistent and accurate classification. Furthermore, some of the facial landmarking models used had only been implemented in PyTorch, this project marks their first implementation using TensorFlow.

3.3.1 Face cropping

One common feature across all DeepFake detection and facial landmarking models is that they perform best when the face being analysed fills the entire frame, because background information is often irrelevant and can disrupt the model. As such, it is useful to have a preprocessing step which crops a frame to just the faces in the frame, using a face detector model.

One of the most efficient models available is YuNet[131]. YuNet is designed to run in real-time (approximately 1 millisecond per frame) on a CPU, whilst retaining a high degree of accuracy. It only contains 75,856 parameters, with many models requiring fifty times more parameters to reach similar levels of accuracy. More parameters mean longer computation times, which was deemed unacceptable for a model designed to run on edge devices. With neural networks, quicker computation times often come at the expense of reduced accuracy; whilst this is still the case with YuNet, it still achieves "similar accuracy to other small models" on standardised benchmarks. `OpenCV` contains an implementation of YuNet for use in Python[85].

A more accurate CNN for detecting faces is Multitask Cascaded Convolutional Networks (MTCNN)[136]. MTCNN is more accurate, achieving 0.851 mean average precision on benchmarks, compared to YuNet's 0.836. The accuracy comes with a significant speed decrease. Where YuNet takes one millisecond per frame, MTCNN takes ten milliseconds. The `mtcnn` library[94] is a popular Python implementation of MTCNN written in TensorFlow.

To attain the optimal combination of speed and accuracy both YuNet and MTCNN are used to identify faces in a video. An initial pass on all frames is done via YuNet, if no faces are found in a frame then MTCNN is used to identify the faces. To improve the performance of MTCNN, faces are processed in batches of 8. It was noted during testing that when exposed to adversarial noise, YuNet was prone to miss faces, as such the confidence YuNet required to mark a face was reduced from 0.9 to 0.7. Only the primary face in each frame is used, for the first frame this is taken as the face with the highest confidence from the model, for subsequent frames it is the face with the smallest Euclidean distance between the vertices of the bounding box. The process for cropping the face from a video is shown in Algorithm 3.2.

Algorithm 3.2 The method for cropping faces from a video

```
Require: video
  faces_per_frame ← {}
  for frame ∈ video do
      faces_coords ← YUNET(frame)
      if faces_coords ≠ {} then
          APPEND(faces_per_frame, face_coords)
      else
          faces_coords ← MTCNN(frame)
          APPEND(faces_per_frame, face_coords)        ▷ Will deliberately append empty set if no faces
      end if
  end for
  faces ← {}
  previous_face ← {}
  for frame_faces∈ faces_per_frame do
      best_face ← {}
      if previous_face = {} then
          best_face ← frame_faces[0]
      else
          best_face ← MINIMUMDISTANCE(frame_faces,previous_face)
      end if
      previous_face ← best_face
      APPEND(faces, best_face)
  end for
```

3.3.2 Eye Landmark Detection

To determine the EAR in a specific frame, the six landmarks around each eye need to be located. There are a large number of Python packages which come with facial landmarking features, unfortunately most of these are unsuitable for use in a state-of-the-art detector as they are insufficiently accurate. The models included in packages are either compressed for ease of distribution or optimised to run on devices with reduced computational capabilities. Hence, a custom implementation of a pre-existing method is required.

Weakly Supervised Eye Landmarks Detection

The first method researched for eye landmark detection was Weakly Supervised Eye Landmarks Detection (WSELD)[55]. In facial landmarking benchmarks (Section 3.3.2), WSELD is the most accurate with respect to eye landmarking when compared to current state-of-the-art methods on mean squared error loss.

A Regional-CNN (R-CNN)[101] outputs initial landmarks. R-CNNs are a special type of CNN that uses Region of Interest (ROI) to locate objects within an input so that later layers can specifically focus on those regions. R-CNN consists of three primary components, a traditional CNN backbone for feature extraction, a Region Proposal Network (RPN) which selects hypothetical areas of interest, a final ROI pooling layer that combines the features from the backbone with the regions to produce regions that can be used for future layers of the network. WSELD uses R-CNN to produce both eye-bounding box regions (the regions of interest) and initial predictions of eye landmarks.

To augment the initial eye landmarks to final, precise landmarks, WSELD employs a custom RNN. The primary component of the RNN is an LSTM module which uses the previous predictions to refine the current prediction. Dense layers surround the LSTM unit to adapt the input and output data into usable formats.

An overall network diagram is shown in Figure 3.1.

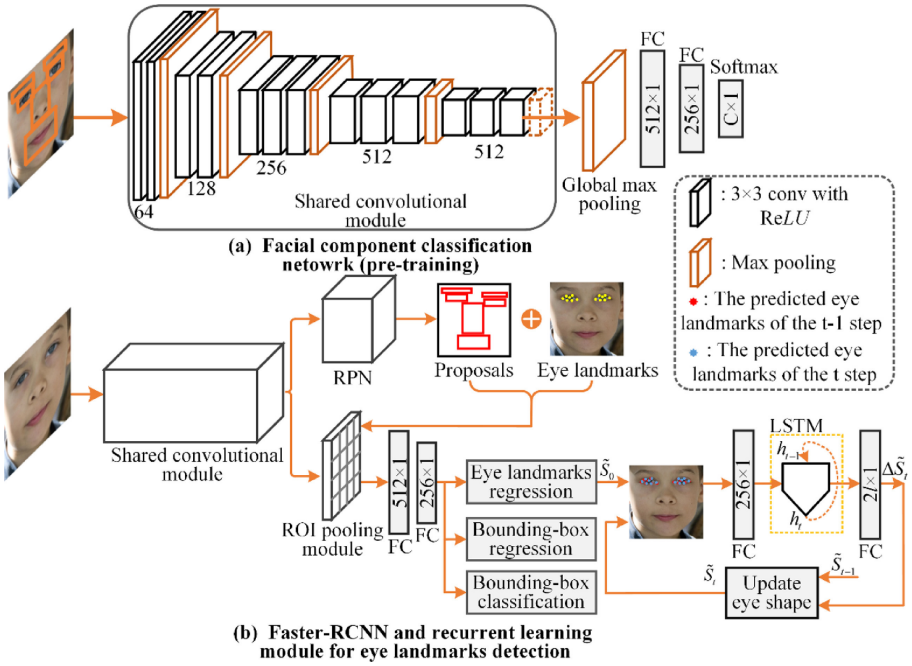


Figure 3.1: Network diagram of WSELD[55]

Whilst the original paper gave clear implementations, their findings were difficult to replicate. No code was supplied with the paper so all code had to be created from scratch. The most confusing aspect of the model was generating landmarks from a RPN. As the name suggests, RPNs are only able to generate regions, as such it would seem to be impossible to generate landmarks. A similar paper was able to produce results on multiple facial regions[122] but, again, no code was provided. In this implementation, it was assumed a dense layer with the number of outputs equal to the number of landmarks was incorporated into the output of the RPN. Furthermore, this was the author's first time coding a substantial learning framework in TensorFlow. There is no pre-built RPN module so one needs to be created from scratch, a number of open source implementations exist[57][65] but these do not generate landmarks. It was attempted to create a custom R-CNN but there was a seeming constant stream of bugs. The bugs, along with there being no way to confirm whether the method of adding landmarks to RPN was successful, resulted in the development of WSELD being halted in favour of a new facial landmarking network.

Pre-implemented Models

To avoid a repetition of the WSELD implementation, it was decided to switch to a reuse-oriented methodology for facial landmarking. For a model to be considered, it must have an open-source implementation available. PapersWithCode[5] is an aggregation site which ranks models on popular machine-learning problems and links to known codebases where possible.

Facial landmark detection is a common problem in machine learning. The vast majority of available training datasets (Section 3.3.2) contain the six points required for EAR calculation and as such any

[5] `https://paperswithcode.com`

model that can be trained on these datasets can be used for eye landmarking. In theory, it is possible to optimise the networks for solely eye landmark detection, however, this will not be done due to the potential that has to disrupt a network that is known to be working.

High-Resolution Network

The most implemented facial landmark detector on PapersWithCode is High-Resolution Network (HRNet)[120]. HRNet is a proposal for a novel architecture of CNNs that has applications in many fields and one of those fields is facial landmarking.

HRNet's novel contribution is CNNs working in parallel. Typical CNNs have one layer following another, whilst HRNet contains multiple CNNs working at a different resolution over the image. Similar to a CNN, the network gradually introduces lower and lower resolution layers to focus on more generic features, but, at all times, keeps the higher resolution layers in context which can refine their feature predictions based on the input from lower-level feature predictions. Lower-resolution layers are connected to higher-resolution layers through fusion layers which either upsample or downsample the feature predictions using either a CNN for downsampling or a bilinear for upsampling. An overview of the architecture for an HRNet is shown in Figure 3.2.

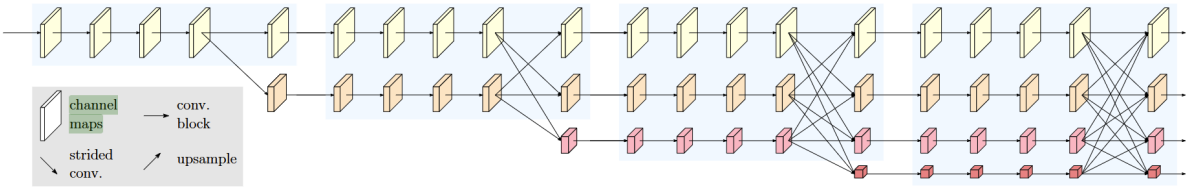
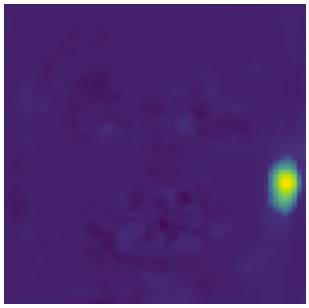


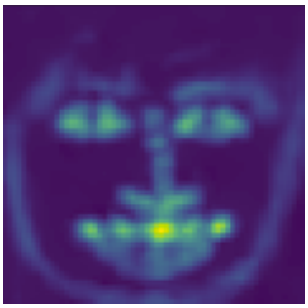
Figure 3.2: A generic network diagram for a HRNet[120]

Whilst HRNet is a generic architecture, it has been applied with great effect to facial landmarking. Such an implementation has been made open source in HRNet's official GitHub repository[137] which details the specific architecture of HRNet uses for facial landmarking. Similar to Figure 3.2, the network consists of four distinct modules, with each module containing four convolutional layers. Each module introduces one additional parallel network. Each layer consists of 18, 36, 72, and 144 filters respectively.

Instead of a set of coordinates for landmarks, the output of HRNet is a heatmap of the likely locations of a landmark with the peak of the heatmap being the most probable location (Figure 3.3). As such, the location of the maximal value of the heatmap can be taken as the final location of a landmark. Whilst a simple version would be to simply take the index of the maximal value in the array, this neglects the possibility of the landmark being in the subpixel domain. The output heatmap is relatively low resolution (64×64), so it is highly likely that the peak is within a subpixel. The problem has been research by Fisher et al.[42]. The most accurate algorithm proposed is Centre of Mass 7 (CoM7) which assumes the heatmap follows a Gaussian distribution and can therefore be estimated using a weighted average as shown through Equation 3.1. Firstly, the coordinates of the maximal value are computed and stored as x, y. Six values on either side of the maximal value are retrieved ($f(x)$). The original equation assumes one dimension, but it can be expanded to two by performing the calculation twice, one for x and one for y.



(a) The heatmap for a single landmark



(b) The heatmap for all facial landmarks

Figure 3.3: The heatmap outputs of HRNet

$$\hat{x} = \frac{3f(x+3) + 2f(x+2) + f(x+1) - f(x-1) - 2f(x-2) - 3f(x-3)}{f(x+3) + f(x+2) + f(x+1) + f(x) + f(x-1) + f(x-2) + f(x-3)} \tag{3.1}$$

HRNet achieves state-of-the-art performance on facial landmarking datasets. It is currently ranked in the top 25 on a variety of datasets by the PapersWithCode[92]. As multiple CNNs are working in parallel, HRNet is often slower than a CNN on equivalent tasks. When trained, HRNet operates at approximately 30 frames per second when run on an RTX3060ti. As such, HRNet is not viable for real-time computation on edge devices but can be used for detection on powerful machines.

When training, the ground truth heatmap was generated by centring Gaussian noise with a $\sigma = 1.5$ on the true landmark, all other values in the heatmap were set to 0. Following the HRNet code[137], each heatmap was 64×64. To align with the inputs for all the other models used in this project, the input was scaled up from 112×112 to 256×256. The model was trained for 60 epochs with mean squared error as the loss and the Adam[68] optimiser.

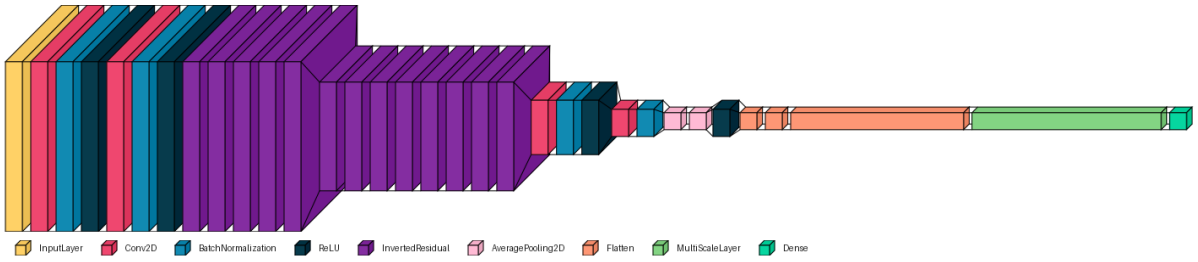
Practical Facial Landmark Detector

A Practical Facial Landmark Detector (PFLD)[48] is an accurate landmarker that is primarily designed for speed. Developed to run on edge devices such as phones, the model can achieve speeds of 150 frames per second on some phones with a minimal reduction in accuracy.

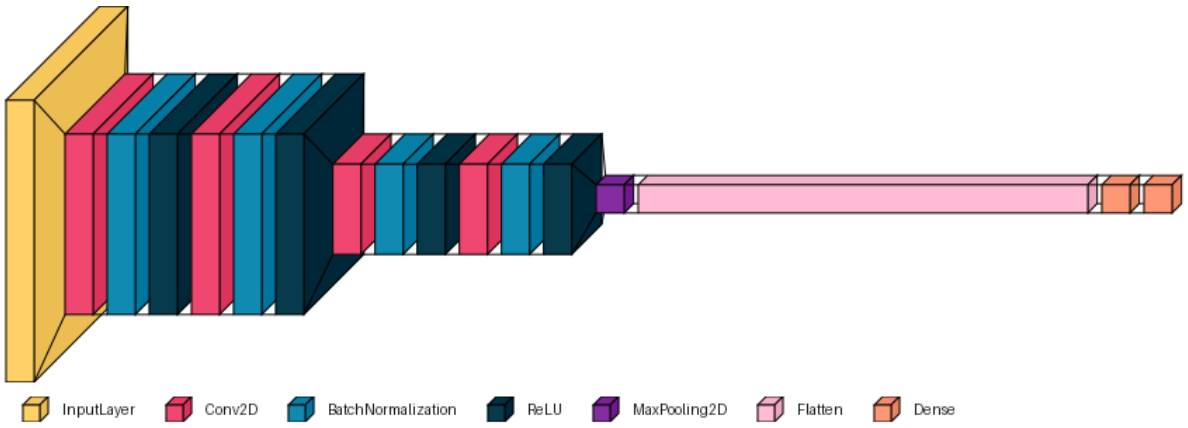
The architecture of PFLD is more similar to a traditional CNN: a backbone that leads into a number of dense layers which produce the final classification. PFLD uses the MobileNet backbone[54] for feature extraction. As discussed in Section 2.2.1, MobileNet reduced the number of parameters in a network whilst retaining similar accuracy. While EfficientNet then scales up its network in response, PFLD keeps the reduced model size to allow it to run with reduced compute times. MobileNet also comes with a width hyperparameter that can further narrow down a network, increasing speed at the expense of accuracy. For this project, accuracy is more important than speed so only the most accurate MobileNet model is used, as such the width parameter is set to $1\times$.

PFLD also includes an auxiliary network that is only used for training. To improve accuracy in a small network, PFLD uses a custom loss function which requires the pitch, yaw, and roll of a face. Whilst these could be calculated from the landmarks produced by the PFLD network, during the early stages, this is very inaccurate and can cause the model learning to stall. The auxiliary network picks up from an intermediary layer midway through the main backbone and outputs three values which are the predicted pitch, yaw, roll of the face.

The overall main backbone is taken from the PyTorch implementation[138] but with a header from a TensorFlow fork[95] to align with the number of facial landmarks required.



(a) The network diagram for PFLD



(b) The network diagram for PFLD's auxiliary network

Figure 3.4: Network Architecture of PFLD (Generated using visualkeras[44])

One of the key contributions to PFLD's accuracy is a custom loss function. A typical facial landmarking model will use some form of distance metric for the loss function. These can only be done in two dimensions, neglecting geometric and structural information. For example, the more you turn your head relative to a camera, the closer your eyes will appear to the camera despite them remaining the same distance apart in three-dimensional space. Whilst large-scale models can get away with using conventional loss functions and brute force their way the inconsistencies, smaller models cannot. To take into account geometrical information, PFLD uses the pitch, yaw, and roll of a face from the auxiliary network to compute the following loss function:

$$L = \sum_{n=1}^{N} \left(\sum_{c=1}^{C} \omega_n^c \sum_{k=1}^{K} (1 - \cos \theta_n^k) \right) ||d_n||_2^2 \tag{3.2}$$

n represents the n^{th} of N landmarks. C and ω are weighting parameters to equalise datasets. The dataset is divided into C classes. A number of classes are proposed, but these require augmenting the dataset with hand labelled, as this is an individual project and some facial landmark datasets contain over 10,000 images (Section 3.3.2) these weights are omitted to save time. The primary weighting parameters are $\theta^1, \theta^2, \theta^3$ which represent the difference the difference between the estimated and truth pitch, yaw, and roll respectively. As the delta goes up, so does the penalisation allowing the model to compensate for large deviations in head poses. Finally, L2 loss ($||d_n||_2^2$) is used as the primary loss component that is then weighted by the previous factors.

To find the ground truth angles of the face, the Perspective-n-Point algorithm[81][53] maps sample points in three dimensions to two dimensions on the face, from which the facial angles can be calculated.

The model was trained for 500 epochs following the PyTorch implementation[136] using the Adam[68] optimiser.

Choice of Model

Both models offer their own advantages and disadvantages. PFLD offers a much faster model, at the expense of accuracy, whereas HRNet offers accuracy at the expense of speed. The difference in accuracy is only expressed in the literature comparing them on the AFLW[69], where PFLD has a normalised mean error (NME) of 1.88 versus HRNet with 1.57. These are exceptionally close as most state-of-the-art models achieve an NME of 1.5-2[6]. As such, both models will be tested over the course of this project. If preference is required, due to time constraints, HRNet will be used as accuracy over speed is the priority for this project. Figure 3.5 shows HRNet and PFLD landmarking the frame. HRNet is more accurate, landmarking the coordinates precisely. PFLD is less accurate often landmarking sections of the lower eyelid rather than the eye itself. However, these landmarks are still accurate in comparison to each other, the entire eye is simply translated down. As such, the EAR will not be affected and so PFLD can still be used effectively.

Furthermore, it is believed that slight inaccuracies in landmark detection will be consistent across the entire EAR graph, as such the various EAR analysis models should be able to learn the idiocracies of each model and compensate for them.

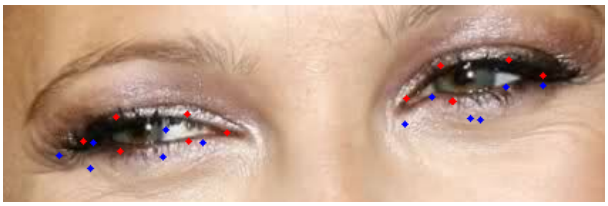


Figure 3.5: A comparison of HRNet's (red) and PFLD's (blue) predictions

Facial Landmark Datasets

Similar to DeepFake detection (Section 2.5), large numbers of faces have been pulled from the internet and hand-labelled to enable the training and comparison of accurate facial landmarking models. The precise number of landmarks varies but the standard is 68 with some datasets labelling more landmarks, acting as a superset. 68 landmarks were used for this project as only the six eye landmarks for calculating the EAR are useful and the 68 landmarks are the smallest standardised size that covers all of them. Reducing the number of outputs significantly speeds up the model as fewer nodes are required to be trained. Where a model used more than 68 landmarks, a mapping was made from the model's landmarks to the main 68 and the rest of the landmarks were discarded.

A variety of datasets were chosen with the primary aim to include a wide variety of facial poses, environmental factors, and occlusions. Increasing the variety of data allows for the model to learn in more challenging scenarios and hopefully become more robust.

The following datasets were used: 300W[109][110][111], Labelled Face Parts in the Wild (LFPW)[10][84], Caltech Occluded Faces in the Wild (COFW)[17], Annotated Faces in the Wild[139], HELEN[75], and Wider Facial Landmarks in the Wild (WFLW)[130].

In total, these contain 11,495 labelled images. A common method of augmenting the dataset is to flip the images horizontally, which increases the dataset to 22,990 images. Data is scraped from a variety of

[6] `https://paperswithcode.com/sota/facial-landmark-detection-on-aflw-full`

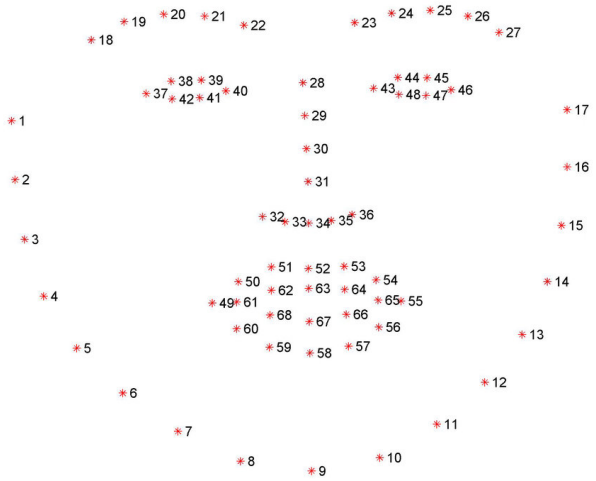


Figure 3.6: The 68 facial landmarks common in facial landmarking datasets[109]

imaging sites such as Flickr[7], Google[8], and Yahoo![9]. The dataset is very comprehensive covering a wide variety of individual characteristics like age, gender, and race. Environmental factors are also covered such as lighting, whether the photo was taken outside or inside.

Two datasets (COFW and WFLW) were specifically chosen for their coverage of facial occlusions. Occlusions are where certain landmarks are obstructed from the view of the camera; for example, sunglasses blocking eyes and hair obscuring one side of the face. Some examples of occlusions are shown in Figure 3.7.

(a) A face occluded by shadow

(b) A face occluded by a mask

Figure 3.7: Faces that have been partially occluded

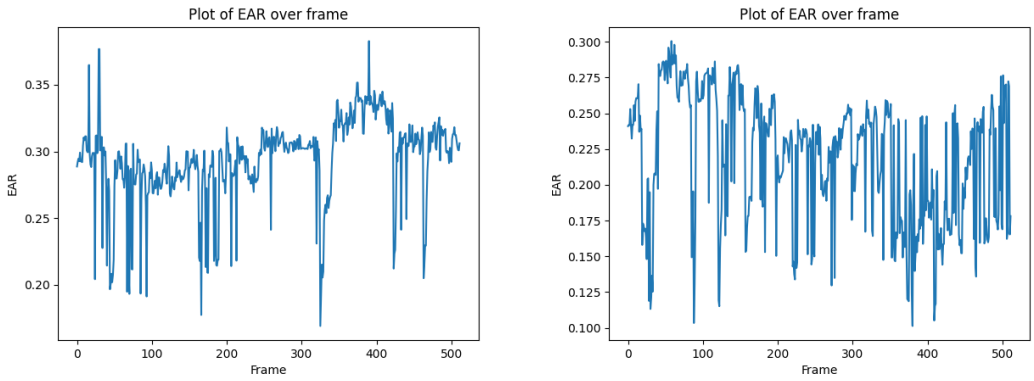
As explained in Section 3.3.1, landmarking models benefit from faces filling the majority of images. To account for this, when training, the images are cropped to the facial region. This region is defined as the area bounded by the ground truth eye landmarks with 2% padding on each side.

3.3.3 Eye Aspect Ratio Analysis

With the eye landmarks located and the EAR calculated for each frame, the result is an EAR-time graph (Figure 3.8). Existing blink-based DeepFake detectors will extract features from the graph such

[7] `https://www.flickr.com/`
[8] `https://images.google.co.uk/`
[9] `https://images.search.yahoo.com/`

as frequency and period. A novel approach from this project is to use the entire EAR graph for analysis by abstracting the problem into univariate time series classification.



(a) The plot of a real EAR graph

(b) The plot of a fake EAR graph

Figure 3.8: Plots of sample EAR graphs

A univariate time series is a single point of data that varies over time. A large amount of research has been done into time series due to their frequent appearance in statistical models. Traditionally machine learning models struggle with temporal structures seen in time series but over time reliable methods have been developed. A comprehensive overview of sate-of-the-art time series algorithms for time series classification has been provided by Johann Faouzi[38].

Figure 3.8 shows how much a fake EAR graph differs compared to a fake graph. The real EAR plot has clear distinct blinks and a relatively consistent EAR when the eye is open. On the other hand, the fake EAR graph has a lot more noise an variation. The graph is almost constantly varying with no clear distinction between blinks and the eye being open and there is a lot more overall noise in the graph as a whole. As there are clear differences between a real EAR graph and a fake one, time series analysis should be able to successfully classify the two.

K-Nearest Neighbours with Dynamic Time Warping

K-Nearest Neighbours (k-NN) is a common algorithm for classification problems. Known training data is plotted in n-dimensional space where n is the number of points of data. A new piece of data is classified by plotting it in the same space and determining which points it is "nearest". The k nearest neighbours' classification is checked and the majority classification is set as the classification for the new point. The metric for how "near" one point is to another is a hyperparameter along with the value of k.

Euclidean distance is the most common algorithm for k-NN classifiers[38] but it cannot be readily applied to time series. Each point in the time series would be compared independently by Euclidean distance which would throw away valuable temporal data. For example, with speech recognition where one sentence is spoken slower than another, the two time series would represent the same data but be far apart via Euclidean distance due to the time discrepancy.

The common solution is Dynamic Time Warping[112]. DTW is programmatically defined in Algorithm 3.3. A cost matrix is computed which has the dimensions of the two time series being compared. For every value in the matrix, the cost is computed as the squared delta of the two values. A warped path is then constructed through the matrix starting at 0,0 and ending in the opposite corner. This path is constructed as the path that has minimum cost across the matrix whilst only moving towards the goal.

The total cost of the path is the sum of all the cells it crosses through. The problem can be optimised with dynamic programming as shown in Algorithm 3.3.

Algorithm 3.3 DTW algorithm

Require: x, y ▷ time series to be compared
  n ← LENGTH(x)
  m ← LENGTH(y)
  DTW ← n × m matrix initialised to ∞
  DTW[0,0] ← 0
  for i ← 1 ... n **do**
    for j ← 1 ... m **do**
      cost ← $(x[i]+y[j])^2$
      DTW[i,j] ← MINIMUM(DTW[i-1,j], DTW[i,j-1], DTW[i-1,j-1])
    end for
  end for
  RETURN DTW[n,m]

Although better than Euclidean distance, DTW has some flaws. By comparing every value to every other value, DTW is computationally expensive running in $O(nm)$ time. Furthermore, it allows for very large time warps as every time is compared to every other time which can result in overfitting. To constrain both problems several algorithms have been proposed which introduce a width parameter which causes points in time to only be compared to other points within the width. How the width parameter varies is dependent on the algorithm with some setting it to be a fixed constant throughout the entire algorithm (Sakoe-Chiba band[112]) and others having it vary throughout the matrix, starting small at the start, growing to its maximum in the middle, and reducing back again for the end (Itakaru parallelogram[60]). Representations of how these optimisations can exclude regions of the cost matrix are shown in Figure 3.9.

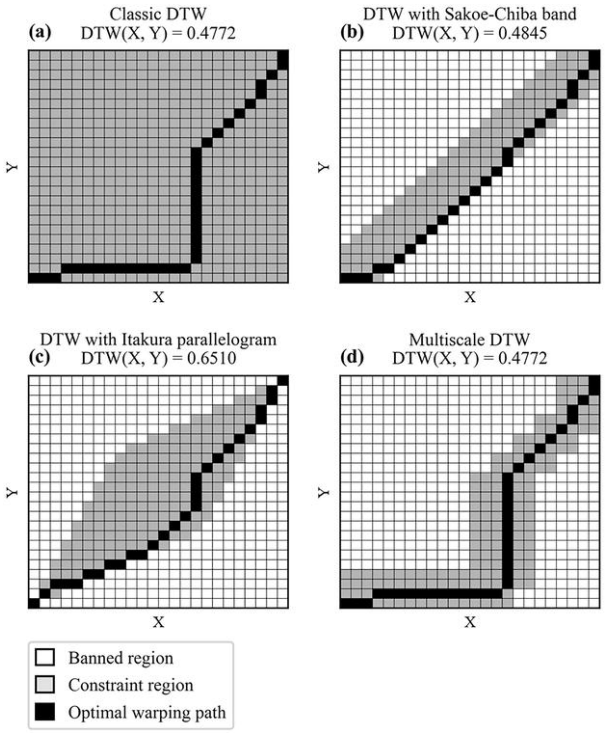


Figure 3.9: Representations of various optimisation to the DTW algorithm[38]

FastDTW looks to accelerate DTW by reducing the resolution of the matrix and gradually increasing it to give an accurate estimation[113]. The time series are reduced in size by combining values, which leads to a smaller matrix and easier computing. The warped path from this matrix can then be used to exclude sections of the matrix in higher resolutions. Eventually, the full warped path through the full-resolution matrix is computed but with a significant proportion of the matrix removed. This is illustrated by Figure 3.10.

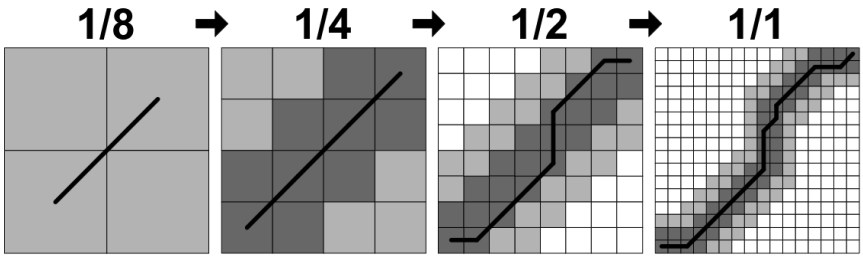


Figure 3.10: A representation of the different resolutions computed for FastDTW and how they exclude regions of the cost matrix in finer resolution[113]

.

Whilst k-NN can be done for any value k, it is common to use 1-NN when dealing with time series classification.

DTW is useful for blink-based DeepFake detection as humans do not blink in unison and hence blinks will appear at seemingly random intervals in the time series. This would cause a traditional distance-based k-NN but would be correctly handled by DTW, resulting in more accurate DeepFake detection.

Learning Shapelets

Shapelets are defined as "a subsequence of consecutive observations from a time series"[38]. These can be used to get a sense of the building blocks of a time series and enable classification based on fundamental features. Learning shapelets[47] is a modification of conventional shapelet classification by optimising shapelets during training.

First k shapelets of length l are generated randomly. Let $S = (s_1, \ldots, s_l)$ be the shapelet and $T = (t_1, \ldots, t_l$ be a time series. The "distance" between the time series and the shapelet is defined in Equation3.3, note the use of Log Sum Exponent function (Equation 3.4) instead of a minimum function to make Equation 3.3 differentiable.

$$d(S,T) = \min_{j \in \{0,\ldots,n-l\}} \sum_{i=1}^{l} (s_i - x_{j+i})^2 \tag{3.3}$$

$$\min_j a_j \approx -\frac{1}{\beta} \log \sum_j e^{-\beta a_j} \tag{3.4}$$

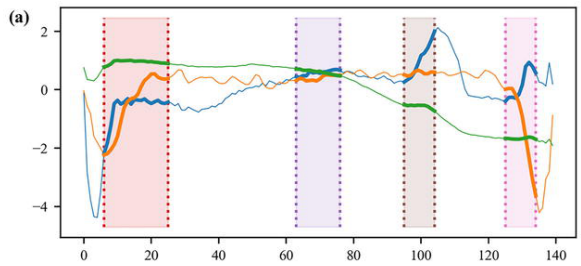
A time series can then go under a change of basis to represent them as a feature vector of the distance between it and each of the k shapelets. Hence, T can be represented as $(d(S_1,T), \ldots, d(S_k,T))$. The new feature vector can now be used in a conventional classifier, such as logistic regression[25].

Whilst the initial shapelets are random and are therefore likely to be ineffective, they can be learned. Both the distance function and logistic regression are differentiable and so can be learned via SGD[103]. This can form shapelets into effective dataset discriminators.

In this case, one shapelet might be the EAR over the course of a normal blink. DeepFakes cannot simulate a blink well and might close the eye too fast or too slow, which learning shapelets may be able to pick up on to classify videos.

Time Series Forest

Many algorithms based on decision trees have been proposed for time series classification. Time series forest[28] is one such algorithm. Random intervals of a minimum length are extracted from the same locations in each time series (Figure 3.11a). From these intervals, the mean, standard deviation, and gradient (slope) are extracted as feature vectors (Figure 3.11b).



(b)

	Interval 1			Interval 2			Interval 3			Interval 4		
	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope
Time series 1	-0.61	0.504	0.052	0.58	0.086	0.02	1.004	0.572	0.197	0.189	0.504	0.156
Time series 2	-0.467	0.977	0.158	0.403	0.084	0.017	0.568	0.061	0.011	-1.204	1.305	-0.431
Time series 3	0.944	0.061	0.002	0.604	0.075	-0.018	-0.57	0.065	-0.017	-1.671	0.024	0.003

Figure 3.11: An example of the sequences and features extracted during the time series forest algorithm

These are then fed into a forest of decision trees[14]. Decision trees are binary trees where at each node a field is compared to a value, if it is greater than a value it goes to the left child, otherwise to the right child. This continues until a leaf node, whose value acts as the final classification. A forest of decision trees processes similar data through a number of different decision trees with the final classification being the majority vote. In this case, each tree corresponds to a different interval of the time series.

Decision trees are trained from the root down. For each feature and every value in the test dataset, the dataset is split using the feature and threshold. Each split is then evaluated via a split metric function (for classification this is usually entropy). The best feature and corresponding threshold are then set for the node and the data is split. The process repeats recursively for each node until some stopping factor is reached. Stopping factors act to limit the size of tree, preventing overfitting. Common stopping factors are the depth of the tree or the minimum samples a node can contain. If a leaf node contains multiple classes from the test dataset, the class of the node is decided by a majority vote.

Time Series Bag-of-Features

Similar to a time series forest, time series bag-of-features[8] extracts random intervals of a minimum length from a time series. However, each interval is further split into 3 equal intervals, the mean, gradient, and standard deviation of these sub-intervals plus the interval as a whole are calculated.

Two forests are trained in this algorithm. The first is a regression tree which rather than outputting the class of an item, outputs the probability of it being in a class, taken by the number of members of each class in a leaf node. Each tree in a forest is trained on a separate subsequence. To make the probabilities more reliable a subsequence is then classified by one it was not trained on and the entire interval has its probabilities averaged across its subsequences.

The result is for each time series, there is a set of features containing the average probability for each class and the actual probabilities of each class. These features are used as the training dataset for a second random forest which produces the final classification.

Neural Networks

Given the versatility of neural networks, it is unsurprising that several architectures have been proposed for time series classification.

Intuitively, RNNs offer ideal performance as they are designed to work on temporal structures. Most RNN architectures proposed focus on LSTM modules[64]. The architecture used for this project is a hybrid LSTM-FCN approach proposed by Karim et al[64]. One branch of the network is a single LSTM module which can either be a standard LSTM module or an attention-LSTM module. The other branch is a standard one-dimension CNN. One-dimension CNNs are similar to their two-dimension counterparts but rather than the filter being a two-dimensional matrix, it is a one-dimensional sliding window. The results from the LSTM and CNN branches are then concatenated and fed into a single dense softmax classifier.

A number of conventional CNNs have been proposed for time series classification. A comprehensive overview is provided by Fawaz[40]. They propose several separate architectures, whilst the majority are for multivariate time series, some can be narrowed down to work on univariate time series.

The first model is an MLP to act as a baseline. The architecture contains three fully connected hidden layers. The only addition is dropout layers which regulate overfitting by removing neurons randomly from a layer (typically between 10% and 15%). Secondly, Fawaz proposes a CNN with three convolution layers, each performing a linear transformation on the time series. A final global average pooling is used before the final dense classification to allow for time series of varying lengths to use the same architecture, however, this is not used in this project as outlined in Section 3.3.3. Time-CNN is a typical CNN for classification apart from it using mean squared error as the loss function during training instead of categorical cross entropy which is the usual loss function for classification tasks. Time series class boundaries are able to be not as distinct as other boundaries so mean squared error allows for "smoother" classifications during training, potentially improving accuracy. The final model proposed for univariate time series is a model based on ResNet, including residual layers to improve classification.

Architectures for the above models are shown in Appendix A.

Implementation

The `pyts` library[39] is an open-source library which implements a variety of classical time series classification methods, including all the methods mentioned in this project. The CNNs proposed in Section 3.3.3 by Fawaz have had their architectures and training parameters open-sourced in a companion GitHub repository[41]. The LSTM architecture was implemented using the description from the corresponding paper. The number of epochs for LSTM was reduced from 2000 to 500 to reduce overfitting. Apart from this, all hyperparameters were left as the default values from their respective implementations.

Some of the neural network architectures require a fixed input length. To reduce computation time it was therefore decided to crop or pad all time series to 512 values. Assuming a video runs at 30 frames per second, 17 seconds of footage can be analysed, giving an ample number of blinks to be analysed. If a video has less than 512 frames, the EAR is padded with -1s at the end.

To train the models, every video in the train set is analysed by one of the facial landmark models (Section 3.3.2) to produce the EAR graph for that video. It is assumed that the train set being given to this

section is balanced between real and fake videos. The graphs are then further split into a train and test set with 80% being used for training and 20% for testing. Models are then trained on the train set and then compared to each other on the test set. The model with the greatest accuracy is then selected as the best EAR analyser for the specific model on the specific dataset.

3.4 Adversarial Noise

There are two popular libraries for adding adversarial noise: `foolbox`[99][100] and `cleverhans`[90]. Both offer similar feature sets with native support for TensorFlow, a wide variety of documentation and open-source code bases. Hence, any choice between is based on preference rather than an objective view. `foolbox` was chosen as the library for adversarial noise as it was the most up-to-date library. Furthermore, a bug in `cleverhans`' source code meant the auto-generated documentation appeared blank and so had to be inferred by looking at the source code. On the other hand, `foolbox` had a much richer and working documentation site, allowing for easier bug fixing if necessary.

Whilst `foolbox` does have an implementation of FGSM, it is an untargeted attack in their implementation. As a result, the attack is less reliable than the targeted attack proposed by Gandhi et al.[43]. To target the FGSM attack using `foolbox`, a Projected Gradient Descent attack with 1 step in its search can be used[80]. This is an equivalent attack and as such can be introduced without loss of generality.

The CW-L2 attack proved to be too slow for videos. To attack a video, FGSM took 10 seconds, whereas the CW-L2 attack to 3 hours on the same hardware and video. The majority of video-based DeepFake detection datasets contain thousands of videos. As such, the CW-L2 attack was not tested in this project due to time constraints.

Finally, FakeRetouch comes with no open-source code implementation. Following the delay caused by the failed implementation of WSELD due to its lack of implementation, there was little time available for further development work. Therefore, no evaluation was done on FakeRetouch. The reasons for this choice are explained in greater detail in Chapter 5.

3.5 Traditional Detectors

The VGG19[71][133] and ResNet[127] models from the proof of concept are used in the main algorithm loop. Additional state-of-the-art DeepFake detectors have been created to evaluate their performance against adversarial noise. State-of-the-detectors were taken as the best performing DeepFake detection models uploaded to the PapersWithCode leaderboard[10][11] that had open-source repositories. The models chosen were based on EfficinetNetB4[12] and Xception[108].

EfficientNetB4 was chosen from the EfficinetNet family of models by Bonettini et al. due to its balance between the number of parameters, run time, and classification performance. Two models based on EfficientNet are proposed in the paper, one using the standard EfficientNet and another with an attention module added. These were shown to have very similar results so, to keep the network architecture simple, just EfficientNetB4 is used as the backbone with no attention model added.

Xception was chosen by Rössler et al for classification from a pool of four other CNNs. When compared to the other CNNs, Xception proved to be the most accurate at 99.26% accuracy.

Custom headers were added to these backbones to allow for accurate classification. Firstly, the networks are stripped of their initial classification heads to allow fine tuning on DeepFakes. Secondly, global

[10] `https://paperswithcode.com/sota/deepfake-detection-on-faceforensics`
[11] `https://paperswithcode.com/sota/deepfake-detection-on-faceforensics-1`

pooling reduces the high dimension features outputted from the backbone into a single one dimensional vector. The resulting vector is then normalised to allow for quicker and easier training. To reduce overfitting, a dropout layer is added to reduce the size of the final vector by 50%. Finally, a dense layer produces the final classification.

Model diagrams of the CNNs used for classifications can be found in Appendix B.

The training dataset for these CNNs is created by splitting the overall training set into 80% for model training, and 20% for model validation. For every video in the model training set, 32 frames are extracted and cropped to the facial region, following the method to train EfficientNet[12]. To prevent overfitting, it is important for frames in the same video to not be used in both the training set and validation set, a phenomenon referred to as data leakage. Similarly to the proof of concept, no videos that have adversarial noise added to them are used in the training data as this would not reflect the scenarios DeepFake detectors would see in real life.

When training, models are trained using categorical cross-entropy. The Adam[68] optimiser is used for optimising gradients with hyperparameters from the original paper's implementations. Models were trained for 20 epochs, however early stopping was implemented to reduce the time to learn and to reduce overfitting. If a model's validation loss stops improving for 5 consecutive epochs, training is halted. To further reduce overfitting, the learning rate of the Adam optimiser is reduced by 90% if the validation loss does not improve for 3 consecutive epochs. To gain the benefits of transfer learning, all models were initialised with their weights from the ImageNet[29] dataset.

When classifying a video, each frame is independently classified by the model. If more than 50% of the frames in a video are reported fake, then the video is deemed a fake. This threshold was chosen over a static threshold to account for varying length videos. 50% was chosen as the threshold as it means the majority of frames needed to be classified fake for a video to be fake. One potential concern is uncertain predictions which could cause the model to predict randomly, therefore causing false classifications. However, when testing on images, the model was often confident in its output, often outputting probabilities close to 1 for the chosen classification.

3.6 Final Code

To make final testing and evaluation easier, it was decided to centralise the code for this project in one main loop. One piece of code, upon given a dataset and facial landmarking model, would train all other models required, and evaluate them. Facial landmarking models are trained separately as these require a different dataset for training and are invariant across datasets and therefore training them in the main program would be inefficient.

The program structure is shown in Algorithm 3.4.

Algorithm 3.4 Final Code

```
Require: dataset, FaceLandmarks                ▷ dataset and facial landmarker function call
  train_set, test_set ← SplitDataset(dataset)
  CustomAnalyser ← GetBestAnalyser(train_set, FaceLandmarks)
  TraiditionalModels ← GetModels(train_set)
  for video, label ∈ test_set do
      landmarks ← {}
      model_predictions ← {}
      noisy_landmarks ← {}
      noisy_model_predictions ← {}
      for frame ∈ video do
          Append(landmarks, FaceLandmarks(frame))
          Append(model_predictions, TraditionalModels(frame))
          if label = FAKE then
              noisy_frames ← AttackFrame(frame, TraditionalModels)
              Append(noisy_landmarks, FaceLandmarks(noisy_frames))
              Append(noisy_model_predictions, TraditionalModels(noisy_frames))
          else
              Append(noisy_landmarks, FaceLandmarks(frame))
              Append(noisy_model_predictions, TraditionalModels(frame))
          end if
      end for
      traditional_classifications ← ClassifyVideosClassical(model_predictions)
      noisy_traditional_classifications ← ClassifyVideosClassical(noisy_model_predictions)
      custom_classifications ← CustomAnalyser(landmarks)
      noisy_custom_classifications ← CustomAnalyser(noisy_landmarks)
  end for
```

Often with DeepFake datasets, multiple DeepFake videos are generated from a single real video. As such, there are often more fake videos than real ones. This can cause issues when training as a model could achieve a high accuracy by simply predicting fake every time. To ensure an even split, the dataset is split according to the number of real videos in the dataset. 80% of the real videos are taken for training, the same number of fake videos are then used in the train set for an even split. The remaining videos are used for the testing set. Note that the train set is further split into a training set and validation set in the GetBestAnalyser and GetModels functions as outlined in Sections 3.3.3 and 3.5.

Frames are only attacked if they are fake frames as this reflects reality where an adversary would only add noise to a video if the video was faked. This also has the advantage of significantly reducing compute times as not only is attacking the frame skipped, but classifying each attacked frame with every model is avoided as well. `noisy_frames` is a tuple of 4 frames, each frame attacking a single one of the traditional models.

In line with good scientific computing practices, the main code has checkpoints. All models trained are saved so that they can be retrieved should the program ever terminate. The path to the best model selected by GetBestAnalyser is also saved so that all the models do not have to be re-evaluated. Every 25 videos, the ongoing results of the main video loop are saved so in the event the main loop crashes or is timed out, the loop can pick up from where it last saved.

3.7 Transferability

To evaluate the transferability of blink-based DeepFake detection, the FakeAVCeleb dataset from Section 2.5 will be reserved. As such, no model shall be trained on FakeAVCeleb. The main program will be run across all datasets which will produce models that have been trained on each of the other datasets. All of the traditional models and each of the custom models will then be tested on the FakeAVCeleb dataset. To save time and compute resources, only the eye landmarked and associated best EAR analysis model for the dataset it was trained on will be used to evaluate transferability.

The testing algorithm follows the same rough structure as Algorithm 3.4, with three main differences. The dataset is not split into a train and a test set as the entire dataset is used for testing. Secondly, all the models are loaded as they have already been trained. In the main classification loop, no adversarial noise is added to the frames, hence the `noisy_*` result stores are omitted along with the calls to AttackFrame. Similar to the main code loop, the results are saved every 25 videos to be able to recover from crashes and other associated errors.

3.8 Datasets

An attempt was made to access all of the datasets listed in Section 2.5. For the majority of them, the application involved filling in a form and agreeing to each dataset's respective terms and conditions. However, gaining access to the DFDC dataset[31] proved difficult. The original Kaggle competition the dataset was posted under[12] has closed and hence it is impossible to download the dataset from that source. From the DFDC's website, the dataset should be available at `https://dfdc.ai/`, however, the site known longer has a valid SSL certificate (Appendix C) and can therefore not be accessed securely. The website requires a username, password, and Amazon Web Service account ID. Some of this is sensitive information and is therefore unadvisable to upload over an insecure connection. As a result, testing on the DFDC is not completed in this project.

All datasets were downloaded from their original source. FaceForensics has the option to be downloaded at multiple compression levels, to balance disk space and quality, the middle compression level (c23) was chosen.

[12] `https://www.kaggle.com/competitions/deepfake-detection-challenge`

Chapter 4

Results and Evaluation

This section will show a summary of of results, the raw results can be found in Appendix D.

Results are classified into True Positives (TP, declared real when real), True Negatives (TN, declared fake when fake), False Positives (FP, declared real when fake), and False Negatives (FN, declared fake when real). Overall accuracy is defined in Equation 4.1 and accuracy on fake videos is defined by Equation 4.2.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

$$\text{Fake Accuracy} = \frac{TN}{TN + FP} \tag{4.2}$$

Traditional DeepFake detection uses the Area Under Curve (AUC) metric for model evaluation, which involves comparing the true positive rate against the false positive rate over all possible classification thresholds. AUC was not chosen for this project due to its tendency to over-inflate model performance by averaging results over a range of thresholds, most of which would be unsuitable for real-world implementations[102]. Furthermore, in real-world applications, DeepFake detectors would operate at predetermined fixed thresholds which adversarial attacks would be fine-tuned to work against, hence an accuracy metric which varies such a threshold would be masking the true accuracy (or lack thereof) for a model.

4.1 Proof of Concept

The initial results for proof of concept are promising. Firstly, when analysing unperturbed videos, blink detection has an overall accuracy of 80% and an accuracy on fakes of 72%. In comparison to VGG and ResNet models, which had an accuracy of 98% and 91% across all videos respectively, blink-based detection performs at a slightly lower accuracy but still achieves an overall impressive accuracy, clearly identifying the originality of videos. It was found that the majority of videos classed as unknown were DeepFaked and as such if a video was classed as unknown it is deemed to be a fake.

VGG19, the model which was directly attacked by adversarial noise, reduced from 100% accuracy on faked videos to 0%. Every faked video was declared real by the model, showing that the adversarial noise attack was effective. On the other hand, neither the blink detection nor ResNet models were affected to a similar degree by the noise. The custom model suffered a slight degradation in performance, falling to 64% accuracy. Unexpectedly, the accuracy of ResNet increased to 100% accuracy on fake videos.

This shows that adversarial noise is very effective on the model that is attacked; however, other models which are not the target of the attack are not affected to the same extent, showing that adversarial noise cannot be used as a general attack and is only effective on the model it is specifically designed to attack. Whilst initially disheartening at first, the results are not conclusive as only one attack was performed against a limited selection of models, hence a more diverse set of models and data is required for conclusive proof. It may be the case that an FGSM attack against a VGG19 model on this subset of the FaceForensics dataset me be a highly specialised model, but all other attacks are general attacks.

It is worth noting that during initial experimentation the strength of the noise (ϵ) was varied. Although the results are not recorded, it was noted that the blink detection accuracy would stay similar, whereas the ResNet detector's accuracy would vary. Hence, blink-based DeepFake detectors are provisionally shown to be more resilient against adversarial noise attacks.

4.2 Main Code Results

4.2.1 Unperturbed Blink Detection

Overall, the results from testing show that conventional detection methods are vulnerable to adversarial noise attacks, even ones that are not specifically targeted to the model being tested. Blink-based DeepFake detection, on the other hand, is initially less accurate but proves resilient to adversarial noise attacks.

On unperturbed videos, HRNet achieves an average overall accuracy of 52.5% and PFLD 63.3% on FaceForensics and 61.2% and 59.4% on Celeb-DF. This is lower than expected as Ictu Oculi and DeepVision report achieving accuracies of 99%[77] and 87.5%[63], respectively. The discrepancy could be due to a number of factors, Firstly, both Ictu Oculi and DeepVision were tested on small, custom datasets, whereas the models proposed in this project are tested on much larger, general benchmarks. Such benchmarks offer a much wider variety of situations and DeepFake methods and as such offer a more challenging scenario for DeepFake detection, hence a possible discrepancy. The other possibility is the use of a more complex analysis. Previous methods used relatively simple abstractions of the EAR graph, such as counting the number of blinks over the course of a video or extracting a couple of key features, the proposed method solely uses the raw EAR graph which allows for more information to be processed but may confuse a model with too much background data.

However, the model custom models retain their accuracy across datasets, showing that the model is not just guessing but is making consistent predictions. As a result, although less accurate than other solutions, blink-based DeepFake detectors can be used as a DeepFake detector. It may be possible that in future works, the accuracy of detection will be increased to enable consistent and accurate classification. Whilst this specific iteration cannot, it still proves a suitable baseline for comparison when adversarial noise is applied.

One major surprise is that PFLD produces a similar or more accurate DeepFake detector than HRNet. It was assumed that due to HRNet being more accurate, the corresponding DeepFake detector would be more accurate. This is not the case with PFLD consistently outscoring HRNet in all accuracy metrics. There are no obvious reasons for this but a couple of theories are proposed. Firstly, is that PFLD may be more sensitive to irregularities, and the slight randomness indirectly introduced to the landmarks by PFLD may be being amplified by DeepFakes, causing an easier classification of the EAR graph. Secondly, an implicit randomness in landmark locations may act as a form of dataset augmentation. A common technique in machine learning is to augment a dataset to reduce overfitting by randomly modifying certain aspects of the data, the slight irregularities present in PFLD landmarks may have acted to augment the EAR dataset, enabling a more robust classifier to be trained.

Consistently, the time series classifier chosen for EAR analysis was learning shapelets. It was chosen in 3/4 scenarios, only being beaten by a time series forest classifier for HRNet operating over the FaceForensics dataset. Due to the way neural networks, it is difficult to determine the exact reason as to why the learning shapelets classifier is consistently the best. One possible reason is that it is one of the few trainable networks employed that keep temporal data into account. The majority of classifiers used either can refine their model over several epochs or can keep the temporal nature of the EAR graph into account, few manage to do both and it so happens that learning shapelets is the best of both worlds.

4.2.2 Unperturbed Traditional Detectors

All traditional CNN-based DeepFake detectors perform at similar levels to the known literature, consistently achieving accuracies 90% and above. The worst-performing model was the Xception model on the FaceForensics dataset, achieving an accuracy of 85.2%, which is still impressive. Clearly, CNNs are better at conventional DeepFake detection than the proposed blink methods.

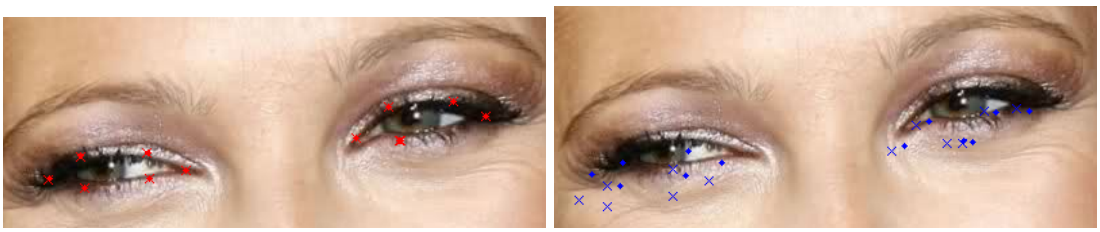
Whilst blink-based detection maintained a fairly consistent accuracy across datasets, CNN detectors achieved much better accuracies on Celeb-DF. This is notable as Celeb-DF is intended to be a "harder" dataset than FaceForensics[78] and therefore models should not be achieving higher accuracies. One possible reason is that Celeb-DF has higher-quality images than FaceForensics. Videos from the FaceForensics dataset are compressed using JPEG compression[107] and hence contain artefacts that may throw off CNNs causing a decrease in accuracy.

4.2.3 Perturbed Blink Detection

The tables turn significantly when adversarial noise is introduced to attack the traditional detectors.

As expected, blink-based detectors are unaffected by adversarial noise, as all models remain within $\pm 0.2\%$ of the unperturbed baseline. This proves the initial aim of this project: blink-based DeepFake detection is resilient against adversarial noise attacks. The largest reduction in accuracy was on the Celeb-DF dataset. HRNet initially classified 3021 unperturbed videos correctly, when exposed to adversarial noise 3019 videos were correctly classified as fake. Such a small drop (2 videos) shows that blink-based detection is resistant to noise-based attacks.

More often than not, the addition of noise would increase the reliability of blink-based DeepFake detectors. Furthermore. it was found that if more noise (a higher ϵ) was used to target a model, the accuracy of blink detection would increase. For example, when targeting the VGG model on the FaceForensics dataset, the HRNet accuracy rose from 52.6% to 54.1%. This is likely due to adversarial noise partially affecting the exact coordinates of landmarks (Figure 4.1). These slight adjustments would add noise to the EAR graph, causing the time series analyser to be more likely to declare the video as fake.



(a) HRNet's landmarks on an original image (dots) and perturbed image (crosses)

(b) PFLD's landmarks on an original image (dots) and perturbed image (crosses)

Figure 4.1: A comparison of landmarks on original and perturbed images

Both PFLD and HRNet were shown to be resistant to attacks. As such, it is shown that the overall concept of blink-based DeepFake detection is resistant to adversarial noise, rather than just a specific facial landmarking model. This allows for more accurate landmark detectors to be introduced which will produce more accurate EAR graphs which can further improve the accuracy of blink-based detection.

Overall, blink-based is resistant to adversarial noise attacks because it deals with the abstraction of a video over time. Whilst CNNs classify a video on a per-frame basis, blink detection takes every frame into account. Whilst adversarial noise does affect the exact coordinates of landmarks in some models, as shown in Figure 4.1, the added variation in landmarks only makes the resulting EAR graph more varied, highlighting the originating video as a likely fake.

As a result, to produce a successful attack against a blink-based detection model, the noise would need to be consistent across frames to cause a constant mislocation of eye landmarks, to produce an EAR graph that would be classified as real. Such a challenge would require an attack that it is temporally aware. As discussed in Section 2.2.2, neural networks struggle with any problem of a temporal nature, as such noise required to attack a blink-based DeepFake detector would be computationally challenging to produce.

4.2.4 Perturbed Traditional Detection

On the other hand, traditional CNN-based detectors fall significantly in accuracy when exposed to adversarial noise attacks, even when the noise is indirectly attacking the model. As expected, when attacked by adversarial noise, a traditional classifier's accuracy would fall significantly, often to 0% when attacked with the strongest amount of noise. On average, when opposed to the weak noise ($\epsilon = 0.05$), accuracy decreased to an average of 0.025%. Similarly, when exposed to higher levels of noise ($\epsilon = 0.1$), accuracy on the targeted model would reduce to 0%; apart from the ResNet model on the Celeb-DF dataset which attained an accuracy of 68.1%. Whilst still a far cry from the unperturbed accuracy of 99.4%, it is nonetheless surprising. The experiment was rerun, but the result stayed the same. The results show that lower-strength noise is sufficient to produce a successful attack. FGSM is a black box attack, only needing the input and output of a model, so can be applied to any DeepFake detector that uses a conventional CNN as the backbone.

Furthermore, an FGSM attack is shown to be a general attack. Whilst not as effective as the model being targeted, other models experienced a significant drop in accuracy. VGG was the most vulnerable of the models, dropping to accuracies below 20% in every scenario and in most cases dropping to below 5%. On the other hand, Xception proved relatively resilient to indirect adversarial attacks, repeatedly scoring higher than all other indirectly attacked models.

There was a significant variety in accuracy between datasets. EfficientNet would attain high accuracies of 30-60% on FaceForensics but would drop to under 10% for Celeb-DF. On the contrary, Xception obtained around 30% on FaceForensics but would score over 70% on Celeb-DF when indirectly attacked. This is due to the differences between the two datasets. As stated in Section 4.2.2, Celeb-DF is a more difficult dataset than FaceForensics, hence it contains different DeepFake methods. Therefore, the attacks on each model will be different across datasets as each model will be focusing on different aspects of the image. The resulting noisy images will be different and thus cause different levels of errors on models.

Overall, every traditional model experienced a loss in accuracy when attacked by adversarial noise, either directly or indirectly. This shows that blink-based DeepFake is indeed resilient against adversarial noise as it was not affected by indirect attacks.

4.3 Transferability

Appendix D.3 shows the raw results of transferability.

Unfortunately, the results for transferability are inconclusive. There is no correlation between any of the blink-based DeepFake detectors. Whilst they are slightly more accurate than the traditional detectors, none achieve sufficient accuracy across both and real fake videos to be deemed suitable for a general DeepFake detector. Whilst the HRNet model trained on Celeb-DF has a high accuracy score on faked videos (99.7%), it consistently declares any video as fake, resulting in it having an accuracy of 0.6% on real videos from the FakeAVCeleb dataset. The accuracies of the other blink detection models bear little similarity with the models they were trained on, meaning they cannot be used as general detectors.

One potential reason for the discrepancy is that whilst real blinking is consistent across datasets, fake blinking is not. One DeepFake may blink too frequently, another may blink for too long, etc. Similar to how a CNN picks up on the irregularities in the data it was trained, so does the time series classifier. The time series classifiers employed in this project have all overfitted on the methods they were trained on and thus cannot be used as general detectors.

As has been shown in literature[102], CNNs are only effective on the data they are trained on. The same phenomenon is shown in this project. One discovery that is novel to Ricker et al's research is that the models trained for this project seem to consistently classify unknown videos as real. One potential reason is the set threshold for classification. This project uses the most probable class as the final classification, resulting in a threshold of 50% for classification. If a CNN is unsure of classification, the probability for each class will be close to 50. The CNNs trained in this project have a slight leniency towards a real classification with the average confidence that a frame was real being 52%, hence a tendency to classify unknown videos as real.

4.4 Evaluating Blink-Based DeepFake Detection

As shown, the primary benefit of blink-based DeepFake is the native resiliency to adversarial noise attacks. Adversarial Noise is a relatively computationally cheap and reliable way of causing an otherwise accurate DeepFake detector to classify fake videos. Blink-based detectors are resistant to such attacks and therefore can be used reliably when there is suspicion that an image or video has been manipulated. Whilst the accuracy of the detectors produced in this project are relatively inaccurate, they maintain a consistent accuracy above 50%. As such, they are not merely guessing but making consistent classifications. Therefore, the resiliency to adversarial noise shown is native to the network architecture. If the overall accuracy was to increase to a valid rate (for example, 90%), then the accuracy on perturbed videos would equally rise. Hence, this project can be used as a proof-of-concept that blink-based DeepFake detection is resistant to adversarial noise and future works can improve on the overall accuracy.

The primary downside to blink-based detection as an architecture is that it requires a video where the subject's eyes are clearly visible throughout the majroity of the runtime. Whilst it is possible to detect landmarks through occlusions, it is significantly less accurate than if the eyes were clearly visible, potentially leading to a misclassification. Furthermore, the eyes need to be faked to be classified. If just the mouth is faked, then blink detection will be unable to classify the video correctly.

Hence, it is recommended that a blink-based DeepFake detector should not be used on its own as a classifier. It should instead be used in two scenarios as a supplement to other detection methods. The first scenario for its use is if a video is suspected to have been attacked with adversarial noise. If a traditional detector is consistently classifying a video as real but something feels "off" about the video then it is recommended to pass the video through a blink detector for some clarification.

Another potential use for these detectors is as part of an ensemble of classifiers. Ensemble classifiers are group of classifiers all running on the same video, with the final classification being the majority vote of the classifiers. A blink based detector has a place among the ensemble as a form of resiliency against adversarial noise.

Chapter 5

Project Management

5.1 Project Plan

5.1.1 Gantt Charts

Good time management is vital to planning a successful project.. A Gantt chart allows for a detailed view of how long each section is planned to take. This further enables accurate tracking of how well the project is progressing, such as whether it is behind or ahead of schedule.

The originally proposed Gantt chart for the project is shown below in Figure 5.1. Note that the first term of the Gantt was originally drafted before the full scope of the project was defined. It was assumed that this project would be developing a proof of concept DeepFake detection method and analysing its effectiveness rather than evaluating an existing detector.

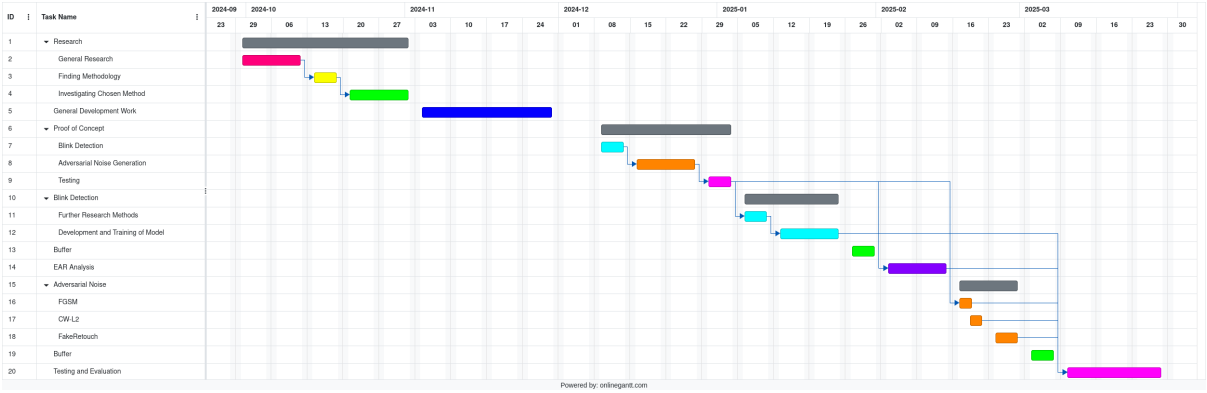


Figure 5.1: The original Gantt chart planned for the software development section of the project

Most notable is the introduction of buffer weeks. The vast majority of projects will have unforeseeable events that will take longer to accomplish than originally planned: a piece of code might be particularly hard to implement, or research in a particular area might be sparse resulting in source aggregation taking time. To account for these, buffer weeks are introduced at the end of major software development milestones to allow for potential delays.

The following Gantt chart is the actual timeline of the project.

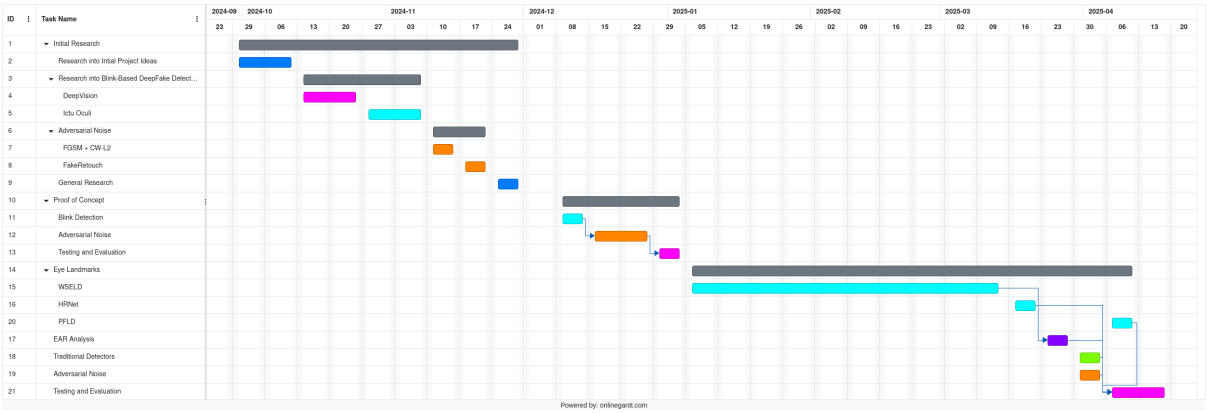


Figure 5.2: The actual Gantt chart for the software development section of the project

5.1.2 Dealing With Setbacks

There are two main differences between the proposed Gantt chart and the actual one. Firstly is the length of time taken for research. This can be further decomposed into two more factors: workload from other coursework and research required.

It was originally believed that with effective time management and load balancing, coursework would not have any significant impact on the progress of the project. However, some coursework, most notably the coursework associated with CS325 Compiler Design, required more work than anticipated. As a result, work on the project was often delayed in order to free up time to focus on coursework. This led to the project being delayed by roughly two weeks. To remedy this, Wednesday afternoons were set aside as dedicated time to work on the project. Other times were also used if they were free.

The initial project specification was written assuming that the project would only implement an existing DeepFake detection methodology that only existed as a theoretical proof of concept. The chosen methodology, blink detection, involves two models: one to determine the state of the eyes, and the other to determine whether the sequence produced by the previous model is feasible. Additionally, methods to add adversarial noise to images needed to be researched. Thus, the research phase took three weeks rather than the intended two, further setting the project behind.

The other major setback was the failed implementation of WSELD. Due to the various difficulties in implementing WSELD discussed in Section 3.3.2, it was decided to switch to a different landmarking model. At that time approximately one month of work had gone into the failed development of the model. Whilst some of this work, such as normalising eye landmarking datasets and experience with TensorFlow, was transferable, the majority of work was not. Thus, the project was delayed by approximately three weeks.

This was primarily mitigated using the buffer weeks built into the Gantt chart. Hence, the overall delay was reduced from three weeks to one, which was much more manageable. To speed up the design of its replacements, it was decided to switch to a reuse-oriented methodology. As such, models were only considered if they had an existing open-source implementation that could replicate their findings.

5.1.3 Development Methodology

The overall software for this software is the agile methodology[9]. This methodology promotes focussing on rapid development as specification, design and implementation are interleaved[7]. This is ideal for any

flexible project, such as a research project, as requirements can shift rapidly to account for no research and aims.

A common implementation of agile methodology is the scrum. A scrum consists of a sprint cycle where a set of requirements is outlined to be completed within that cycle. Normally, such a cycle is time-based, however, for this project, it is requirements-based with each sprint consisting of a single block of the original Gantt chart (Figure 5.1). Sprints will therefore last approximately one week. As this is an individual project, there is no need for daily stand-ups or a scrum master as is normal for a scrum-based project.

The flexibility allowed by this methodology was shown in the switch to reuse-oriented design for WSELD. If the project used a plan-based methodology then work would not have been able to resume as soon as it did. An agile methodology also allowed for continual usage of reuse-oriented design throughout the project, such as its use for EAR time series analysis (Section 3.3.3 and 3.4). The `pyts` library sped up development of the EAR analysis, reducing a predicted 2-week long job to a single week.

5.2 Resources Used

Primary development of the project was done on the Department of Computer Science's (DCS) systems. These are accessible as physical systems in the department building, or remotely via a Secure Socket Shell (SSH) connection. These provide the advantages of being constantly accessible, backed up nightly, and the physical computers have access to local GPUs to test models.

The main loop of the program is a long-running program and can scale to the available GPU. As such, access to High Performance Compute (HPC) clusters, was needed to gather the final results in a reasonable time. The primary cluster used was DCS' kudu cluster, specifically the Gecko and Falcon partition. The other HPC cluster used was from Warwick University's Scientific Computing Research Technology Platform's (SCRTP) GPU cluster. SCRTP was used as sparingly as possible due to there being department-level limits on its use. The main loop was only run over the Easter break so that other individuals using the clusters would be less impacted.

Code was synchronised across platforms using GitHub[1], an online version of `git`. GitHub primarily acts as a remote backup of the code for this project, completing the 3-2-1 methodology for data backup[115]. Its secondary purpose is as a version control system, allowing for ease of traceability for bugs. Finally, it allowed for code synchronisation access devices. For example, partial development was done on the SCRTP cluster which was then instantly synchronised to the code on the DCS machines

Finally, the report is written in LaTeXusing Overleaf[2]. Overleaf is an online LaTeXeditor, allowing for the report to be written on any computer with an internet connection. Overleaf also comes with GitHub integration, meaning the report can also be tracked using `git`.

5.3 Legal, Social, Ethical, and Professional Issues

There are no legal issues associated this project. One potential cause for concern is copyright issues with the collection of data for datasets. All datasets used are freely available for academic and non-commercial use, which this project conforms to. All collection methods used by the datasets were within the terms of service of the websites the data was scraped from. Therefore, there are no substantial legal concerns for this project.

[1] `https://github.com/Mole1424/3rd-year-project`
[2] `https://www.overleaf.com/`

A major ethical and social issue this project has is its potential to help the advancement of DeepFakes. Similar to their creation in GANs, modern DeepFake detection is full of leap-frogging: a method will be proposed for reliable DeepFake detection, DeepFakes will advance so that the method is no longer viable, and the cycle repeats. This project has shown that not only is blink detection a viable method for detection but it is resilient to simple attack methods. Hence, future DeepFake development may focus on improving DeepFake blinking to the extent that EAR analysis is no longer reliable.

Obviously, improvements in DeepFake technologies are a social and ethical issue, allowing for the spread of harmful content as discussed in Section 2.1.3. However, by this logic, there is no worth in ever developing DeepFake detection methods. It goes without saying that this is a worse ethical and social issue, allowing DeepFakes to remain undetected. Hence, any method of detecting DeepFakes, no matter how short-lived, is of overall benefit.

There are no professional issues associated with this project.

Chapter 6

Conclusions

In conclusion, this project has shown that blink-based DeepFake detectors are resilient to adversarial noise attacks.

The project constructs a novel method for DeepFake detection through univariate time series analysis on the EAR graph. To accomplish this, 2 separate facial landmarking models and 13 time series classifiers were tested to produce the best combination on individual datasets. All models were implemented and trained locally to be optimised for DeepFake detection. The resulting classifiers consistently achieved accuracies of 50-60%. Whilst not as accurate as current state-of-the-art classifiers, the project still proves that the overall architecture is resilient to adversarial noise attacks. As such, it paves the way for future research to improve the overall accuracy to a level that is usable.

4 state-of-the-art CNN-based DeepFake detectors were implemented and trained. These all achieved accuracies on unperturbed videos in line with expectations, correctly classifying videos around 90% of the time.

When exposed to adversarial noise attacks, traditional models would be heavily impacted, misclassifying the majority of DeepFaked videos as real, even when not the direct target of the attack. Importantly, however, all blink-based detectors were unaffected by noise to any reasonable degree. The valuation was done across 2 standardised datasets, a first of its kind for blink-based detection. Therefore, blink-based DeepFake detectors are resilient against adversarial noise attacks.

Finally, all models were tested for their ability to classify unseen DeepFakes from a different DeepFake generator. This showed that whilst a slight improvement on existing CNNs blink-based DeepFake detectors cannot be used as a general classifier.

6.1 Author's Assessment of Project

Overall, I am pleased and proud of the outcomes of this project. The initial aim of the project (Section 1.2) was to investigate whether or not blink-based DeepFake detection is resilient to adversarial noise attacks and I believe this was completed successfully. An accurate, resilient, and novel DeepFake detector was constructed and tested on a wide variety of datasets with varying strengths of noise. A multitude of state-of-the-art DeepFake CNN-based DeepFake detectors were also implemented and evaluated. The primary outcome of the project was as suspected with blink-based DeepFake detection being resilient to current adversarial noise attacks. As a result all of the core objectives in Section 1.3 have been completed successfully, producing a significant amount of novel research in the field of DeepFake detection (Section 1.4)

Furthermore, one of the stretch goals outlined in Section 1.3 was achieved, although it did not produce the desired outcomes. Nevertheless, it still adds to the novel research in this project.

Whilst the majority of this project was a success, there were a few low points. The failed implementation of WSELD (Section 3.3.2) comes to mind as a particularly low point, to lose approximately a month of work would be disheartening to anyone. However, such a challenge was overcome and the delay to the project was much less pronounced than it could have been. Unfortunately, there was still a delay in the project, resulting in a significant amount of adversarial noise methodologies not being evaluated to save time. Obviously, such a loss slightly degraded the final conclusions of this paper, but it still stands.

Were I to do the project differently, I would have implemented the open-source and verifiable facial landmarking algorithms before I began work on unverified, potentially more accurate, facial landmarking models. This would have allowed me to get final results quicker and allowed for more extensive testing of my final models.

6.2 Future Research Suggestions

The first potential area for research is improving the accuracy of blink-based Detection. DeepVision and Ictu Oculi could both be tested against adversarial noise, along with a wider range of attacks. Whilst several time series analysis methods were proposed and tested, it is a wide and diverse field, hence there may be a time series classifier more accurate than learning shapelets that could raise the base accuracy of blink detection to the same as CNNs. Improving the accuracy of blink-based detection to a suitable level (over 90%) would allow for its use as a classifer on its own.

A few papers have shown that diffusion models can be used as an initial parse to reduce adversarial noise in images[86][27][6]. Whilst this project has shown that blink-based DeepFake detection has a natural resiliency to adversarial noise, an extra preprocessing step of a diffusion model may be added to further reduce noise and improve accuracy.

As discussed in Section 4.2.3, it is theoretically difficult to produce adversarial noise that can reliably target a blink-based DeepFake detector. However, it may still be possible. As such, further research should be completed to show whether such noise is possible to consistently create or not. If it is impossible, it shows that EAR analysis is a viable form of noise-resistant detection. If adversarial noise is possible to generate, then releasing the findings academically will allow other noise-resilient detection techniques to be researched and implemented.

Other methods of DeepFake detection have been proposed that focus on physiological factors in humans. Both heartbeat[96][51] and breathing[74] are viable signs for DeepFake detection, such methods rely on more subtle human movements and may therefore be even harder to attack with adversarial noise.

6.3 Open Source Work

Due to the help provided by the open source community in this project, most notably the PapersWithCode[1] website, it felt right to give back to the community.

Firstly, the code for this project is open source and available on GitHub at `https://github.com/Mole1424/3rd-year-project`. This allows for anyone to replicate the research done in this project, and too potentially enhance it.

[1] `https://paperswithcode.com/`

A number of bugs and depreciations were found in the open-source projects used for this project. These were fixed where possible and communicated to the authors of the code in the appropriate ways. Screenshots of these contributions can be found in Appendix E.

Bibliography

[1] A Turing Machine. *Eye Blinking Prediction*. Hosted on kaggle, Last accessed 08/04/2025. 2018. URL: https://kaggle.com/competitions/compomicssummer2018.

[2] Abadi, Martín et al. "Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". *arXiv preprint arXiv:1603.04467* (2016).

[3] Ajder, Henry et al. *The State of Deepfakes: Landscape, Threats, and Impact*. Tech. rep. Deeptrace, 2019.

[4] Altuncu, Enes, Franqueira, Virginia NL and Li, Shujun. "DeepFake: Definitions, Performance Metrics and Standards, Datasets, and a Meta-Review". *Frontiers in Big Data* 7 (2024), p. 1400024.

[5] Amari, Shunichi. "A Theory of Adaptive Pattern Classifiers". *IEEE Transactions on Electronic Computers* EC-16.3 (1967).

[6] Ankile, Lars Lien, Midgley, Anna and Weisshaar, Sebastian. "Denoising Diffusion Probabilistic Models as a Defense Against Adversarial Attacks". *arXiv preprint arXiv:2301.06871* (2023).

[7] Archbold, James. *Topic 2: Software Development Methodologies 2, 2 Software 2 Methodology*. 2023.

[8] Baydogan, Mustafa Gokce, Runger, George and Tuv, Eugene. "A Bag-of-Features Framework to Classify Time Series". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11 (2013), pp. 2796–2802.

[9] Beck, Kent et al. *Manifesto for Agile Software Development*. Last accessed 20/04/2025. 2001. URL: https://agilemanifesto.org/.

[10] Belhumeur, Peter N et al. "Localizing Parts of Faces using a Consensus of Exemplars". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (2013), pp. 2930–2940.

[11] Bergmann, Dave and Stryker, Cole. *What is Backpropagation?* Last accessed 04/04/2025. URL: https://www.ibm.com/think/topics/backpropagation.

[12] Bonettini, Nicolo et al. "Video Face Manipulation Detection Through Ensemble of CNNs". *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 5012–5019.

[13] Bozinovski, Stevo and Fulgosi, Ante. "The Influence of Pattern Similarity and Transfer Learning on the Base Perceptron Training". *Proceedings of Symposium Informatica 3-121-5*. 1976.

[14] Breiman, Leo. "Random Forests". *Machine Learning* 45 (2001), pp. 5–32.

[15] Brooks, Tim et al. "Video Generation Models as World Simulators". *OpenAI Blog* 1 (2024), p. 8.

[16] Brush, Stephen G. "History of the Lenz-Ising model". *Reviews of Modern Physics* 39.4 (1967), p. 883.

[17] Burgos-Artizzu, Xavier, Perona, Pietro and Dollar, Piotr. *Caltech Occluded Faces in the Wild (COFW)*. 2022.

[18] Carlini, Nicholas and Wagner, David. "Towards Evaluating the Robustness of Neural Networks". *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 39–57.

[19] Cavalli, Francesco. *The State of DeepFakes 2024*. Tech. rep. sensity.ai, 2024.

[20] Chesney, Bobby and Citron, Danielle. "Deep Fakes: A Looming Challenge for Privacy, Democracy, and National Security". *California Law Review* 107 (2019), p. 1753.

[21] Chirodea, Mihai Cristian et al. "Comparison of TensorFlow and PyTorch in Convolutional Neural Network - based Applications". *2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE. 2021, pp. 1–6.

[22] Chollet, François. "Xception: Deep Learning with Depthwise Separable Convolutions". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1251–1258.

[23] Cole, Samantha. *Reddit Just Shut Down the Deepfakes Subreddit*. Last accessed 05/04/2025. 2018. URL: `https://www.vice.com/en/article/reddit-shuts-down-deepfakes/`.

[24] Corcoran, Peter et al. "Digital Beauty: The good, the bad, and the (not-so) ugly". *IEEE Consumer Electronics Magazine* 3.4 (2014), pp. 55–62.

[25] Cramer, Jan Salomon. *The Origins of Logistic Regression*. Tech. rep. Tinbergen Institute Discussion Paper, 2002.

[26] Crevier, Daniel. *AI: The Tumultuous History of the Search for Artificial Intelligence*. BasicBooks, 1993.

[27] Croitoru, Florinel-Alin et al. "Diffusion Models in Vision: A Survey". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.9 (2023), pp. 10850–10869.

[28] Deng, Houtao et al. "A Time Series Forest for Classification and Feature Extraction". *Information Sciences* 239 (2013), pp. 142–153.

[29] Deng, Jia et al. "ImageNet: A Large-Scale Hierarchical Image Database". *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee. 2009, pp. 248–255.

[30] Diel, Alexander et al. "Human Performance in Detecting DeepFakes: A Systematic Review and Meta-Analysis of 56 papers". *Computers in Human Behavior Reports* 16 (2024), p. 100538.

[31] Dolhansky, Brian et al. "The DeepFake Detection Challenge (DFDC) Dataset". *arXiv preprint arXiv:2006.07397* (2020).

[32] Duchi, John, Hazan, Elad and Singer, Yoram. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". *Journal of Machine Learning Research* 12.7 (2011).

[33] Dufour, Nicholas et al. *DeepFakes Detection Dataset by Google & JigSaw*. 2019.

[34] Edwards, Gareth et al. *Rogue One: A Star Wars Story*. Lucasfilm Ltd. 2016.

[35] Elharrouss, Omar et al. "Backbones-Review: Feature Extraction Networks for Deep Learning and Deep Reinforcement Learning Approache". *Computer Science Review* 53 (2024), p. 100645.

[36] Europol. "Facing Reality? Law Enforcement and the Challenge of Deepfakes" (2022).

[37] *Faceswap: Deepfakes Software for All*. Last accessed 13/04/2025. 2016. URL: `https://github.com/deepfakes/faceswap`.

[38] Faouzi, Johann. "Time Series Classification: A Review of Algorithms and Implementations". *Time Series Analysis*. IntechOpen, 2024. Chap. 2.

[39] Faouzi, Johann and Janati, Hicham. "pyts: A Python Package for Time Series Classification". *Journal of Machine Learning Research* 21.46 (2020), pp. 1–6.

[40] Fawaz, Hassan Ismail. "Deep Learning for Time Series Classification". *arXiv preprint arXiv:2010.00567* (2020).

[41] Fawaz, Hassan Ismail. *Deep Neural Network Ensembles for Time Series Classification*. Last accessed 18/04/2025. 2019. URL: https://github.com/hfawaz/ijcnn19ensemble.

[42] Fisher, Robert and Naidu, D. "A Comparison of Algorithms for Subpixel Peak Detection". *Image Technology: Advances in Image Processing, Multimedia and Machine Vision*. Springer, 1996, pp. 385–404.

[43] Gandhi, Apurva and Jain, Shomik. "Adversarial Perturbations Fool Deepfake Detectors". *2020 International joint conference on neural networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[44] Gavrikov, Paul. *visualkeras*. Last accessed 28/04/2025. 2020. URL: https://github.com/paulgavrikov/visualkeras.

[45] Goodfellow, Ian J, Shlens, Jonathon and Szegedy, Christian. "Explaining and Garnessing Adversarial Examples". *arXiv preprint arXiv:1412.6572* (2014).

[46] Goodfellow, Ian J et al. "Generative Adversarial Nets". *Advances in nNeural Information Processing Systems* 27 (2014).

[47] Grabocka, Josif et al. "Learning Time-Series Shapelets". *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2014, pp. 392–401.

[48] Guo, Xiaojie et al. "PFLD: A Practical Facial Landmark Detector". *arXiv preprint arXiv:1902.10859* (2019).

[49] Harris, Laurie and Sayler, Kelley. *Deep Fakes and National Security*. Tech. rep. Congressional Research Survey, 2023.

[50] He, Kaiming et al. "Deep Residual Learning for Image Recognition". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[51] Hernandez-Ortega, Javier et al. "DeepFakesON-Phys: DeepFakes Detection based on Heart Rate Estimation". *arXiv preprint arXiv:2010.00400* (2020).

[52] Hochreiter, Sepp and Schmidhuber, Jürgen. "Long Short-Term ¡emory". *Neural Computation* 9.8 (1997), pp. 1735–1780.

[53] Hou, Jerry. *Face Yaw Roll Pitch from Pose Estimation using OpenCV*. Last accessed 16/04/2025. 2018. URL: https://github.com/jerryhouuu/Face-Yaw-Roll-Pitch-from-Pose-Estimation-using-OpenCV.

[54] Howard, Andrew G et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". *arXiv preprint arXiv:1704.04861* (2017).

[55] Huang, Bin et al. "Eye Landmarks Detection via Weakly Supervised Learning". *Pattern Recognition* 98 (2020), p. 107076.

[56] Huang, Yihao et al. "FakeRetouch: Evading DeepFakes Detection via the Guidance of Deliberate Noise". *arXiv preprint arXiv:2009.09213* 1.2 (2020).

[57] hxuaj. *A TensorFlow2 Implementation of Faster R-CNN*. Last accessed 14/04/2025. 2021. URL: https://github.com/hxuaj/tf2-faster-rcnn.

[58] IBM. *What are Convolutional Neural Networks?* Last accessed 03/04/2025. URL: https://www.ibm.com/think/topics/convolutional-neural-networks.

[59] Islam, Mohaiminul, Chen, Guorong and Jin, Shangzhu. "An Overview of Neural Networks". *American Journal of Neural Networks and Applications* 5.1 (2019), pp. 7–11.

[60] Itakura, Fumitada. "Minimum Prediction Residual Principle Applied to Speech Recognition". *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1 (2003), pp. 67–72.

[61] Ivakhnenko, A.G. and Lapa, V.G. *Cybernetics and Forecasting Techniques*. American Elsevier Publishing Company, 1967.

[62] Juefei-Xu, Felix et al. "Countering Malicious DeepFakes: Survey, Battleground, and Horizon". *International Journal of Computer Vision* 130.7 (2022), pp. 1678–1734.

[63] Jung, Tackhyun, Kim, Sangwon and Kim, Keecheon. "DeepVision: Deepfakes Detection Using Human Eye Blinking Pattern". *IEEE Access* 8 (2020), pp. 83144–83154.

[64] Karim, Fazle et al. "LSTM Fully Convolutional Networks for Time Series Classification". *IEEE Access* 6 (2017), pp. 1662–1669.

[65] Kewar, Akash. *Region Proposal Network from Scratch in Keras*. Last accessed 14/04/2025. 2021. URL: `https://martian1231-py.medium.com/region-proposal-network-rpn-in-faster-rcnn-from-scratch-in-keras-1311c67c13cf`.

[66] Khalid, Hasam et al. "FakeAVCeleb: A Novel Audio-Video Multimodal Deepfake Dataset". *arXiv preprint arXiv:2108.05080* (2021).

[67] King, Davis E. "Dlib-ml: A Machine Learning Toolkit". *Journal of Machine Learning Research* 10 (2009), pp. 1755–1758.

[68] Kingma, Diederik P and Ba, Jimmy. "Adam: A Method for Stochastic Optimization". *arXiv preprint arXiv:1412.6980* (2014).

[69] Koestinger, Martin et al. "Annotated Facial Landmarks in the Wild: A Large-Scale, Real-World Database for Facial Landmark Localization". *2011 IEEE International Conference on Computer Vision Workshops (ICCV workshops)*. IEEE. 2011, pp. 2144–2151.

[70] Kowalski, Marek. *FaceSwap*. Last accessed 13/03/2025. 2016. URL: `https://github.com/MarekKowalski/FaceSwap`.

[71] Krishna, Navneeth et al. *DeepFake Detection - VGG16*. Last accessed 11/04/2025. 2022. URL: `https://www.kaggle.com/code/navneethkrishna23/deepfake-detection-vgg16`.

[72] Kromydas, Bill. *Convolutional Neural Network (CNN): A Complete Guide*. Last accessed 03/04/2025. 2023. URL: `https://learnopencv.com/understanding-convolutional-neural-networks-cnn/`.

[73] Lalchand, Satish et al. *Generative AI is Expected to Magnify the Risk of DeepFakes and other Fraud in Banking*. Last accessed in 27/04/2025. 2024. URL: `https://www2.deloitte.com/us/en/insights/industry/financial-services/financial-services-industry-predictions/2024/deepfake-banking-fraud-risk-on-the-rise.html`.

[74] Layton, Seth et al. "Every Breath You Don't Take: Deepfake Speech Detection Using Breath". *arXiv preprint arXiv:2404.15143* (2024).

[75] Le, Vuong et al. "Interactive Facial Feature Localization". *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part III 12*. Springer. 2012, pp. 679–692.

[76] LeCun, Yann, Bengio, Yoshua and Hinton, Geoffrey. "Deep Learning". *Nature* 521.7553 (2015), pp. 436–444.

[77] Li, Yuezun, Chang, Ming-Ching and Lyu, Siwei. "In Ictu Oculi: Exposing AI Created Fake Videos by Detecting Eye Blinking". *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE. 2018, pp. 1–7.

[78] Li, Yuezun et al. "Celeb-DF: A Large-scale Challenging Dataset for DeepFake Forensics". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 3207–3216.

[79] Lugaresi, Camillo et al. "Mediapipe: A Framework for Building Perception Pipelines". *arXiv preprint arXiv:1906.08172* (2019).

[80] Madry, Aleksander et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". *arXiv preprint arXiv:1706.06083* (2017).

[81] Mallick, Satya. *Head Pose Estimation using OpenCV and Dlib*. Last accessed 16/04/2025. 2016. URL: https://learnopencv.com/head-pose-estimation-using-opencv-and-dlib/.

[82] Midjourney. Last accessed 06/04/2025. 2022. URL: https://x.com/midjourney/status/1547108864788553729.

[83] Mildenhall, Ben et al. "Burst Denoising with Kernel Prediction Networks". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2502–2510.

[84] Mondal, Amit. *Labelled Face Parts in the Wild Dataset*. Last accessed 16/04/2025. 2021. URL: https://www.kaggle.com/datasets/amitmondal98/lfpw-labelled-face-parts-in-the-wild.

[85] Nelson, Phil. *OpenCV Face Detection: Cascade Classifier vs. YuNet*. Last accessed 14/04/2025. 2022. URL: https://opencv.org/blog/opencv-face-detection-cascade-classifier-vs-yunet/.

[86] Nie, Weili et al. "Diffusion Models for Adversarial Purification". *arXiv preprint arXiv:2205.07460* (2022).

[87] O'Shea, Keiron and Nash, Ryan. "An Introduction to Convolutional Neural Networks". *arXiv preprint arXiv:1511.08458* (2015).

[88] OpenAI. *Introducing 4o Image Generation*. Last accessed 06/04/2025. 2025. URL: https://openai.com/index/introducing-4o-image-generation/.

[89] Ozturk, Selen. "Vicious Circle: John Whitney and the Military Origins of Early CGI". *Bright Lights Film Journal* (2023).

[90] Papernot, Nicolas et al. "Technical Report on the CleverHans v2.1.0 Adversarial Examples Library". *arXiv preprint arXiv:1610.00768* (2018).

[91] PapersWithCode. *DeepFake Detection*. Last accessed 31/03/2025. URL: https://paperswithcode.com/task/deepfake-detection.

[92] PapersWithCode. *HRNet leaderboard*. Last accessed 15/04/2025. URL: https://paperswithcode.com/paper/high-resolution-representations-for-labeling.

[93] Paszke, Adam et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". *CoRR* abs/1912.01703 (2019).

[94] Paz Centeno, Iván de. *MTCNN - Multitask Cascaded Convolutional Networks for Face Detection and Alignment*. Last accessed 14/04/2025. 2024. URL: https://github.com/ipazc/mtcnn.

[95] Qi, Guoqiang. *PFLD Implementation with TensorFlow*. Last accessed 16/04/2025. 2019. URL: https://github.com/guoqiangqi/PFLD.

[96] Qi, Hua et al. "DeepRhythm: Exposing DeepFakes with Attentional Visual Heartbeat Rhythms". *Proceedings of the 28th ACM International Conference on MultiMedia*. 2020, pp. 4318–4327.

[97] Ramesh, Aditya et al. "Hierarchical Text-Conditional Image Generation with CLIP Latents". *arXiv preprint arXiv:2204.06125* 1.2 (2022), p. 3.

[98] Ranjan, Rajeev, Patel, Vishal M and Chellappa, Rama. "HyperFace: A Deep Multi-Task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.1 (2017), pp. 121–135.

[99] Rauber, Jonas, Brendel, Wieland and Bethge, Matthias. "Foolbox: A Python Toolbox to Benchmark the Robustness of Machine Learning Models". *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*. 2017.

[100] Rauber, Jonas et al. "Foolbox Native: Fast Adversarial Attacks to Benchmark the Robustness of Machine Learning Models in PyTorch, TensorFlow, and JAX". *Journal of Open Source Software* 5.53 (2020), p. 2607.

[101] Ren, Shaoqing et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". *Advances in Neural Information Processing Systems* 28 (2015).

[102] Ricker, Jonas et al. "Towards the Detection of Diffusion Model DeepFakes". *arXiv preprint arXiv:2210.14571* (2022).

[103] Robbins, Herbert and Monro, Sutton. "A stochastic approximation method". *The Annals of Mathematical Statistics* (1951), pp. 400–407.

[104] Rojas, Raúl. *Neural Networks: A Systematic Introduction*. Springer Science & Business Media, 2013.

[105] Rombach, Robin et al. "High-Resolution Image Synthesis with Latent Diffusion Models". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10684–10695.

[106] Rosenblatt, Frank. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". *Psychological Review* 65.6 (1958), p. 386.

[107] Rössler, Andreas et al. "FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces". *arXiv* (2018).

[108] Rössler, Andreas et al. "FaceForensics++: Learning to Detect Manipulated Facial Images". *International Conference on Computer Vision (ICCV)*. 2019.

[109] Sagonas, Christos et al. "300 Faces In-The-Wild Challenge: Database and Results". *Image and Vision Computing* 47 (2016), pp. 3–18.

[110] Sagonas, Christos et al. "300 Faces In-The-Wild Challenge: The First Facial Landmark Localization Challenge". *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2013, pp. 397–403.

[111] Sagonas, Christos et al. "A Semi-Automatic Methodology for Facial Landmark Annotation". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2013, pp. 896–903.

[112] Sakoe, Hiroaki and Chiba, Seibi. "Dynamic Programming Algorithm Optimization for Spoken Word Recognition". *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49.

[113] Salvador, Stan and Chan, Philip. "Toward Accurate Dynamic Time Warping in Linear Time and Space". *Intelligent Data Analysis* 11.5 (2007), pp. 561–580.

[114] Schiffman, Harvey Richard. *Sensation and Perception: An Integrated Approach*. John Wiley & Sons, 1990.

[115] Seagate. *What is a 3-2-1 Backup Strategy?* Last accessed 20/04/2025. URL: `https://www.seagate.com/gb/en/blog/what-is-a-3-2-1-backup-strategy/`.

[116] Simonyan, Karen and Zisserman, Andrew. "Very Deep Convolutional Networks for Large-Scale Image Recognition". *arXiv preprint arXiv:1409.1556* (2014).

[117] Singh, RD. "The Art and Science of Digital Visual Media Forensics". *Journal of Forensic, Legal & Investigative Sciences* 4 (2018), p. 021.

[118] Soukupova, Tereza and Cech, Jan. "Real-Time Eye Blink Detection using Facial Landmarks". *21st Computer Vision Winter Workshop, Rimske Toplice, Slovenia*. Vol. 2. 2016, p. 4.

[119] stability.ai. *Stable Diffusion Launch Announcement*. Last accessed 06/04/2025. 2022. URL: `https://stability.ai/news/stable-diffusion-announcement`.

[120] Sun, Ke et al. "High-Resolution Representations for Labeling Pixels and Regions ". *arXiv preprint arXiv:1904.04514* (2019).

[121] Tan, Mingxing and Le, Quoc. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". *International Conference on Machine Learning*. PMLR. 2019, pp. 6105–6114.

[122] Tang, Xin et al. "Facial Landmark Setection by Semi-Supervised Deep Learning". *Neurocomputing* 297 (2018), pp. 22–32.

[123] Thies, Justus, Zollhöfer, Michael and Nießner, Matthias. "Deferred Neural Rendering: Image Synthesis using Neural Textures". *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–12.

[124] Thies, Justus et al. "Face2Face: Real-Time Face Capture and Reenactment of RGB Videos". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2387–2395.

[125] Thing, Vrizlynn LL. "Deepfake Detection with Deep Learning: Convolutional Neural Networks versus Transformers". *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE. 2023, pp. 246–253.

[126] Tian, Keyu et al. "Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction". *Advances in Neural Information Processing Systems* 37 (2024), pp. 84839–84865.

[127] Tiwari, Manas. *DeepFake Image Detection Using ResNet50*. Last accessed 11/04/2025. 2024. URL: `https://www.kaggle.com/code/lightningblunt/deepfake-image-detection-using-resnet50`.

[128] Vaswani, Ashish et al. "Attention Is All You Need". *Advances in Neural Information Processing Systems* 30 (2017).

[129] Verdoliva, Luisa. "Media Forensics and DeepFakes: An Overview". *IEEE Journal of Selected Topics in Signal Processing* 14.5 (2020), pp. 910–932.

[130] Wu, Wayne et al. "Look at Boundary: A Boundary-Aware Face Alignment Algorithm". *CVPR*. 2018.

[131] Wu, Wei, Peng, Hanyang and Yu, Shiqi. "YuNet: A Tiny Millisecond-level Face Detector". *Machine Intelligence Research* 20.5 (2023), pp. 656–665.

[132] Xiong, Xuehan and De la Torre, Fernando. "Supervised Descent Method and Its Applications to Face Alignment". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 532–539.

[133] Yadav, Pradyumna, Sharma, Priyansh and Verma, Sakshi. *DeepFake Detection Using VGG-19 Model*. Last accessed 11/04/2025. 2024. URL: `https://github.com/rahul9903/Deepfake`.

[134] Yim, Jonghwa and Sohn, Kyung-Ah. "Enhancing the Performance of Convolutional Neural Networks on Quality Degraded Datasets". *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. IEEE. 2017, pp. 1–8.

[135] Zeiler, Matthew D, Taylor, Graham W and Fergus, Rob. "Adaptive Deconvolutional Networks for Mid and High Level Feature Learning". *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2018–2025.

[136] Zhang, Kaipeng et al. "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks". *IEEE Signal Processing Letters* 23.10 (2016), pp. 1499–1503.

[137] Zhao, Yang et al. *High-Resolution Networks (HRNets) for Facial Landmark Detection*. Last accessed 15/04/2025. 2019. URL: `https://github.com/HRNet/HRNet-Facial-Landmark-Detection`.

[138] Zhichao, Zhao, Guo, Harry and Andres. *PFLD-pytorch*. Last accessed 16/04/2025. 2019. URL: `https://github.com/polarisZhao/PFLD-pytorch`.

[139] Zhu, Xiangxin and Ramanan, Deva. "Face Detection, Pose Estimation, and Landmark Localization in the Wild". *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 2879–2886.

Appendix A

Time Series Classification Architectures

Where uncited, model architectures were generated using the `visualkeras` library[44].

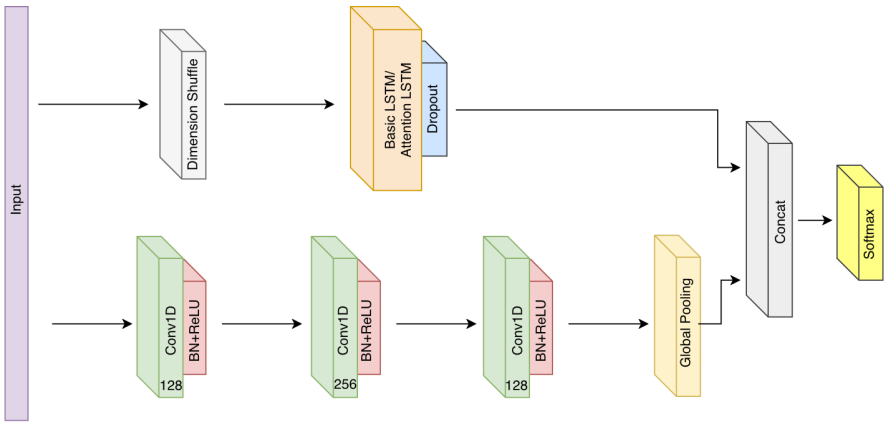


Figure A.1: Architecture for LSTM-FCN[64]

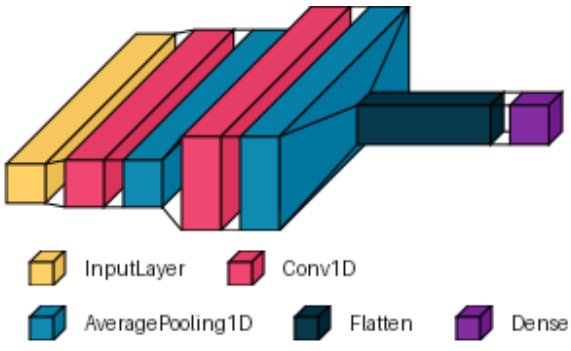


Figure A.2: Architecture of Time-CNN

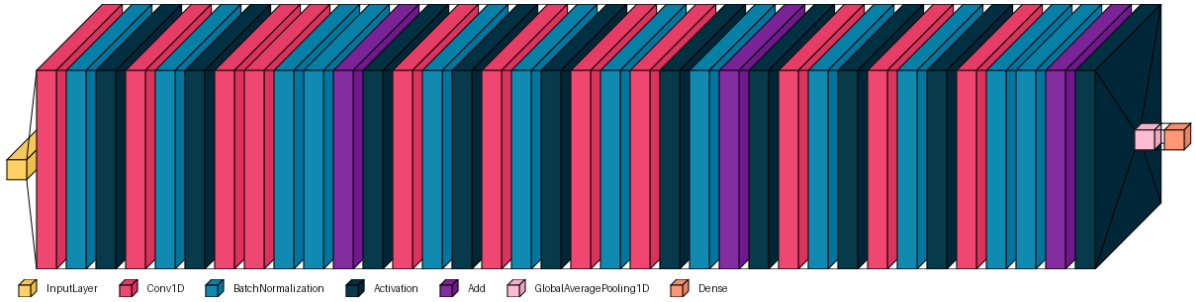


Figure A.3: Architecture of ResNet

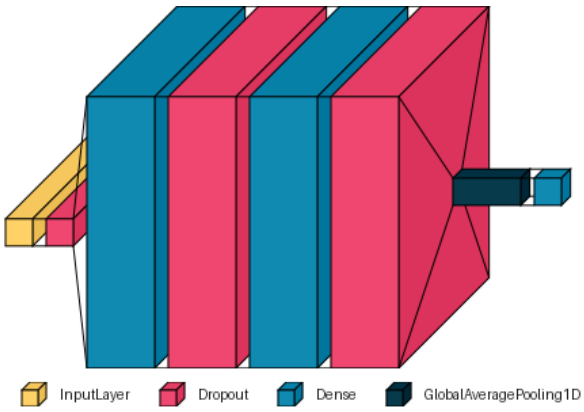


Figure A.4: Architecture of Multi-Layer Perceptron

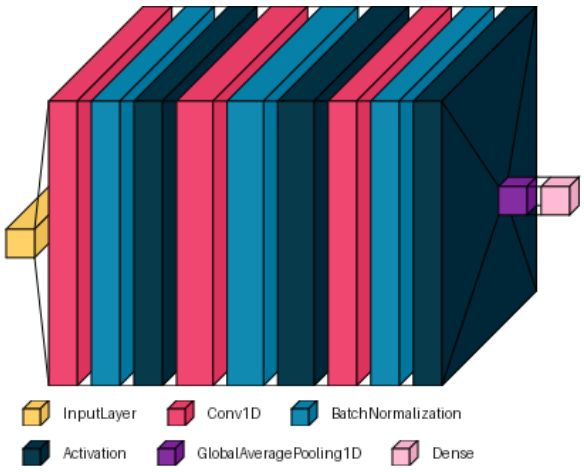


Figure A.5: Architecture of Fully Convolutional Neural Network

Appendix B

Traditional Classification Architectures

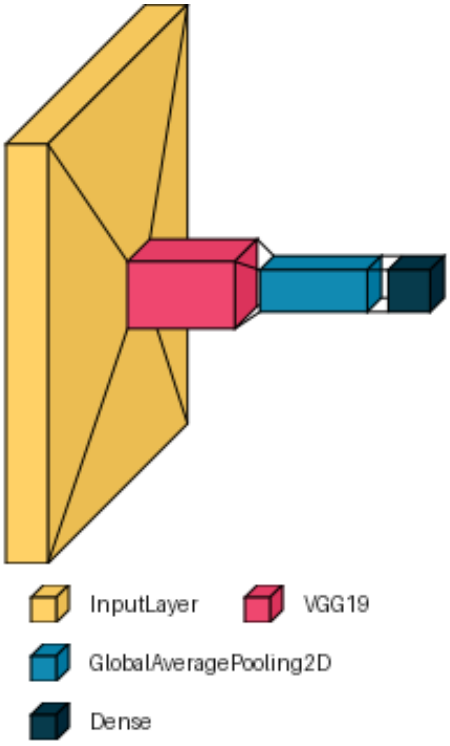


Figure B.1: Architecture of VGG19

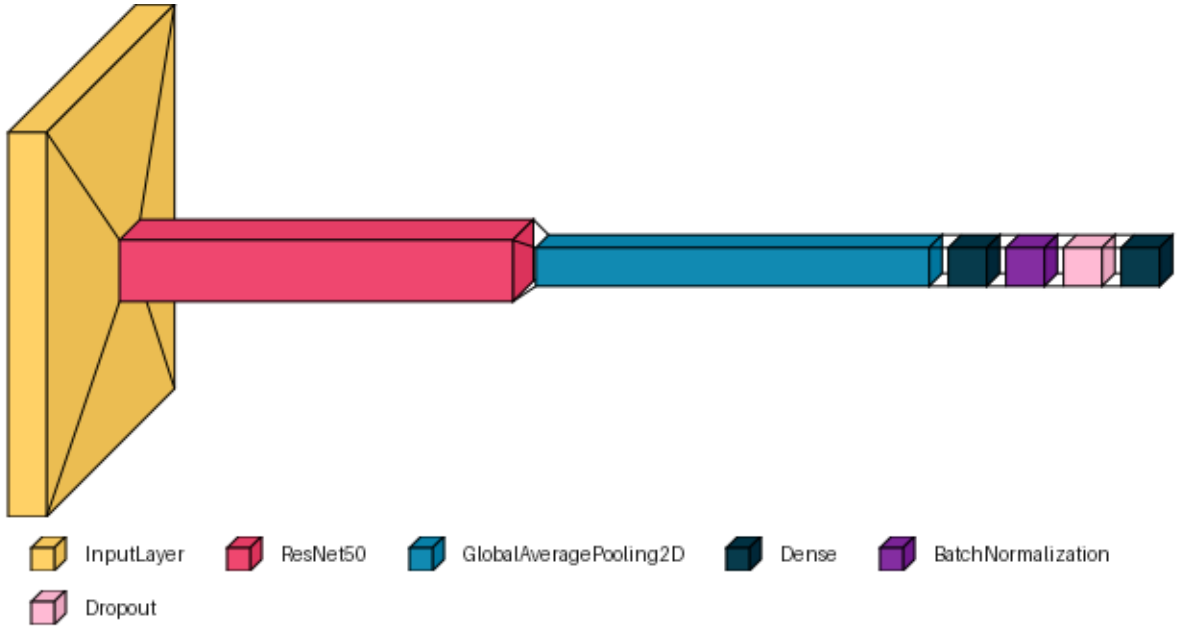


Figure B.2: Architecture of ResNet

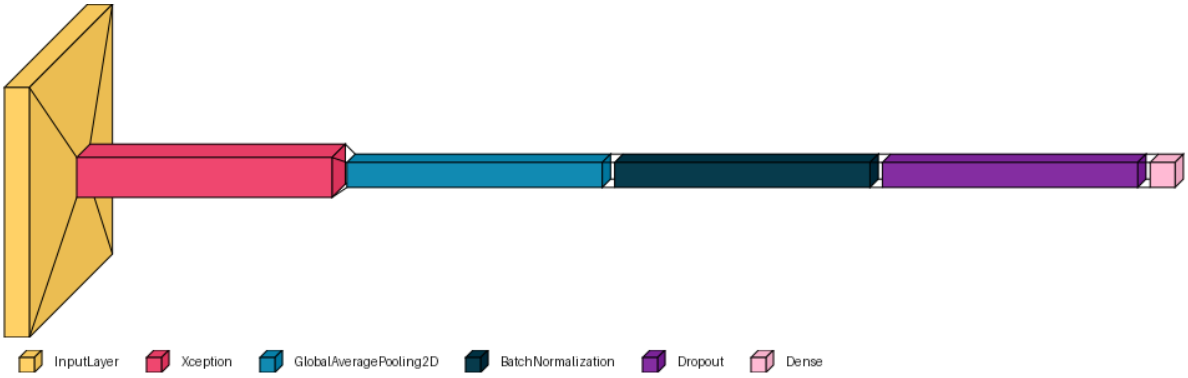


Figure B.3: Architecture of Xception

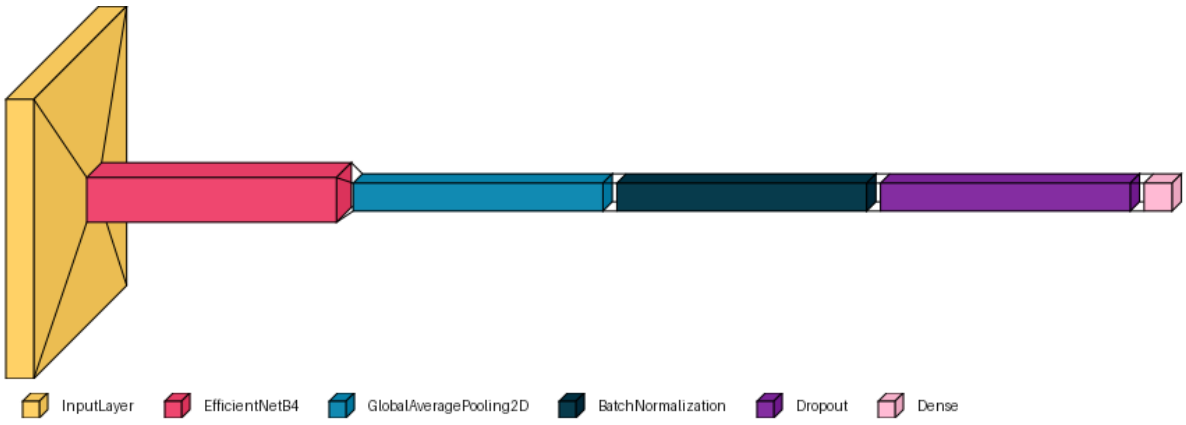


Figure B.4: Architecture of EfficientNetB4

Appendix C

Deep Fake Detection Challenge Website

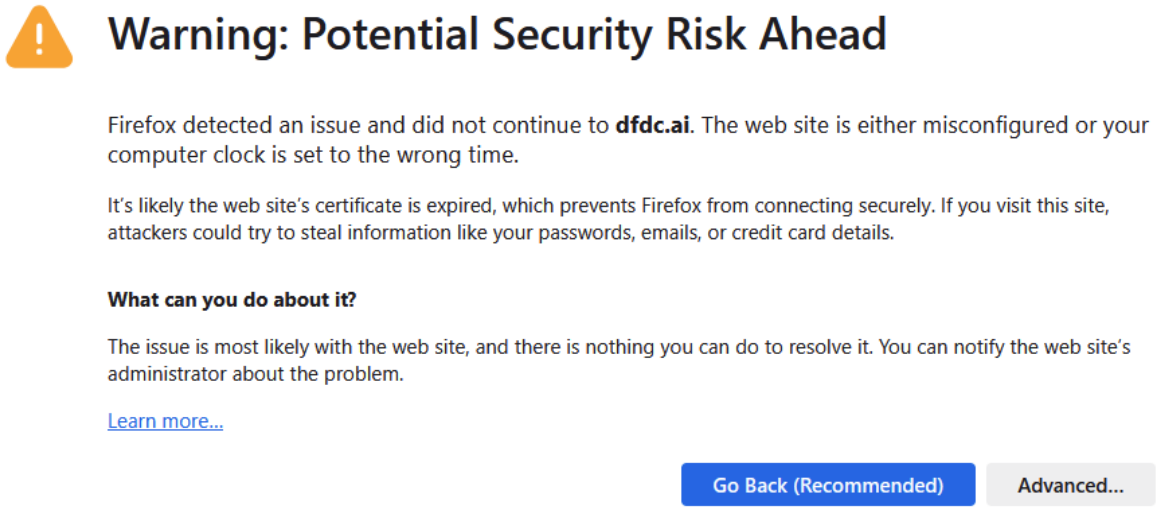


Figure C.1: A screenshot of the security warning found when trying to access `https://dfdc.ai/`

Appendix D

Raw results

True Positives: declared real when real
True Negatives: declared fake when fake
False Positives: declared real when fake
False Negatives: declared fake when real

D.1 Proof of Concept

	Unperturbed			**Perturbed (VGG, $\epsilon = 0.1$)**		
	Blink Detection	**VGG**	**ResNet**	**Blink Detection**	**VGG**	**ResNet**
True Positives	44	48	45	44	48	45
True Negatives	36	50	46	32	0	50
False Positives	14	0	4	18	50	0
False Negatives	6	2	5	6	2	5
Overall Accuracy	80%	98%	91%	76%	45%	95%
Real Accuracy	88%	96%	90%	88%	96%	90%
Fake Accuracy	72%	100%	92%	64%	0%	100%

Table D.1: Results of the proof of concept

For unperturbed videos, 1 real video and 27 fake videos were classed as unknown. For perturbed videos, 1 real and 18 fake videos were classed as unknown.

D.2 Main Code

D.2.1 FaceForensics

The time series analysis paired with HRNet was the time series forest.
The time series analysis paired with PFLD was learning shapelets.

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Positives	113	146	175	182	184	176
True Negatives	1672	2007	2813	3035	2725	2889
False Positives	1528	1193	387	165	475	311
False Negatives	87	54	25	18	16	24
Overall Accuracy	52.5%	63.3%	87.9%	94.6%	85.6%	90.1%
Real Accuracy	56.5%	73%	87.5%	91%	92%	88%
Fake Accuracy	52.3%	62.7%	87.9%	94.8%	85.2%	90.3%

Table D.2: FaceForensics results unperturbed

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	1683	2006	9	486	541	1069
False Positives	1517	1194	3191	2714	2659	2131
Accuracy	52.6%	62.7%	0.2%	15.2%	16.9%	33.4%

Table D.3: FaceForensics results perturbed (VGG, $\epsilon = 0.05$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	1730	2006	0	155	1086	1972
False Positives	1470	1194	3200	3045	2114	1228
Accuracy	54.1%	62.7%	0%	4.8%	33.9%	61.6%

Table D.4: FaceForensics results perturbed (VGG, $\epsilon = 0.1$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	1671	2005	605	0	507	992
False Positives	1529	1195	2595	3200	2693	2208
Accuracy	52.2%	62.7%	18.9%	0%	15.6%	31%

Table D.5: FaceForensics results perturbed (ResNet, $\epsilon = 0.05$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	1716	2007	33	0	1107	1966
False Positives	1484	1993	3167	3200	2093	1234
Accuracy	53.6%	62.7%	1%	0%	34.6%	61.4%

Table D.6: FaceForensics results perturbed (ResNet, $\epsilon = 0.1$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	1681	2007	497	267	2	924
False Positives	1519	1193	2703	2933	3198	2276
Accuracy	52.5%	62.7%	15.5%	8.3%	0%	28.9%

Table D.7: FaceForensics results perturbed (Xception, $\epsilon = 0.05$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	1699	2007	27	178	1	1921
False Positives	1501	1193	3173	3022	3199	1279
Accuracy	53.1%	62.6%	0.1%	5.6%	0%	60%

Table D.8: FaceForensics results perturbed (Xception, $\epsilon = 0.1$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	1673	2006	605	425	537	0
False Positives	1527	1194	2595	2775	2663	3200
Accuracy	52.3%	62.7%	18.9%	13.3%	16.8%	0%

Table D.9: FaceForensics results perturbed (EfficientNet, $\epsilon = 0.05$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	1722	2008	33	134	1085	0
False Positives	1478	1992	3167	3066	2115	3200
Accuracy	53.8%	62.8%	1%	4.2%	33.9%	0%

Table D.10: FaceForensics results perturbed (EfficientNet, $\epsilon = 0.1$)

D.2.2 Celeb-DF

The time series analysis paired with HRNet was learning shapelets.
The time series analysis paired with PFLD was learning shapelets.

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Positives	103	104	165	174	172	171
True Negatives	3021	2930	4838	4896	4833	4877
False Positives	1906	1997	89	31	94	50
False Negatives	75	74	13	4	6	7
Overall Accuracy	61.2%	59.4%	98%	99.3%	98%	98.9%
Real Accuracy	57.9%	58.4%	92.7%	97.8%	96.6%	96.1%
Fake Accuracy	61.3%	59.5%	98.2%	99.4%	98.1%	99%

Table D.11: Celeb-DF results unperturbed

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	3020	2937	0	4040	4647	486
False Positives	1907	1990	4927	887	280	4441
Accuracy	61.3%	59.6%	0%	82%	94.3%	9.9%

Table D.12: Celeb-DF results perturbed (VGG, $\epsilon = 0.05$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	3021	2952	0	4927	3754	4
False Positives	1906	1957	4927	0	1173	4923
Accuracy	61.3%	59.9%	0%	100%	76.2%	0.1%

Table D.13: Celeb-DF results perturbed (VGG, $\epsilon = 0.1$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	3019	2939	215	0	4582	414
False Positives	1908	1988	4712	4927	345	4513
Accuracy	61.3%	59.7%	4.3%	0%	93%	8.4%

Table D.14: Celeb-DF results perturbed (ResNet, $\epsilon = 0.05$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	3021	2952	103	3357	3724	3
False Positives	1906	1975	4824	1570	1203	4924
Accuracy	61.3%	59.9%	2.1%	68.1%	75.6%	0.1%

Table D.15: Celeb-DF results perturbed (ResNet, $\epsilon = 0.1$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	3020	2936	193	3323	2	134
False Positives	1907	1991	4734	1604	4925	4793
Accuracy	61.3%	59.6%	3.9%	67.4%	0%	2.7%

Table D.16: Celeb-DF results perturbed (Xception, $\epsilon = 0.05$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	3021	2951	96	4927	0	3
False Positives	1906	1976	4831	0	4927	4924
Accuracy	61.3%	59.9%	1.9%	100%	0%	0%

Table D.17: Celeb-DF results perturbed (Xception, $\epsilon = 0.1$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	3019	2938	215	3955	4580	0
False Positives	1908	1988	4712	972	347	4927
Accuracy	61.3%	59.6%	4.4%	80.3%	93%	0%

Table D.18: Celeb-DF results perturbed (EfficientNet, $\epsilon = 0.05$)

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Negatives	3021	2952	96	4927	0	3
False Positives	1906	1975	4831	0	4927	4924
Accuracy	61.3%	59.9%	1.9%	100%	0%	0%

Table D.19: Celeb-DF results perturbed (EfficientNet, $\epsilon = 0.1$)

D.3 Transferability

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Positives	345	119	485	492	494	482
True Negatives	2512	6414	4269	1500	2332	2956
False Positives	7197	3295	5440	8209	7377	8209
False Negatives	155	381	15	8	6	18
Overall Accuracy	28%	64%	46.6%	19.5%	27.7%	33.7%
Real Accuracy	69%	23.8%	97%	98.4%	98.8%	96.4%
Fake Accuracy	25.9%	66.1%	44%	15.4%	24%	30%

Table D.20: The accuracy of models trained on FaceForensics and tested on FakeAVCeleb

	HRNet	**PFLD**	**VGG**	**ResNet**	**Xception**	**EfficientNet**
True Positives	3	148	500	500	500	499
True Negatives	9678	5699	344	427	317	657
False Positives	31	4010	9365	9282	9392	9052
False Negatives	397	352	0	0	0	1
Overall Accuracy	94.8%	57.3%	8.3%	9.1%	8%	11.3%
Real Accuracy	0.6%	29.6%	100%	100%	100%	99.8%
Fake Accuracy	99.7%	58.7%	3.5%	4.4%	3.3%	6.8%

Table D.21: The accuracy of models trained on Celeb-DF and tested on FakeAVCeleb

Appendix E

Open Source Contributions

`https://github.com/bethgelab/foolbox/pull/738`

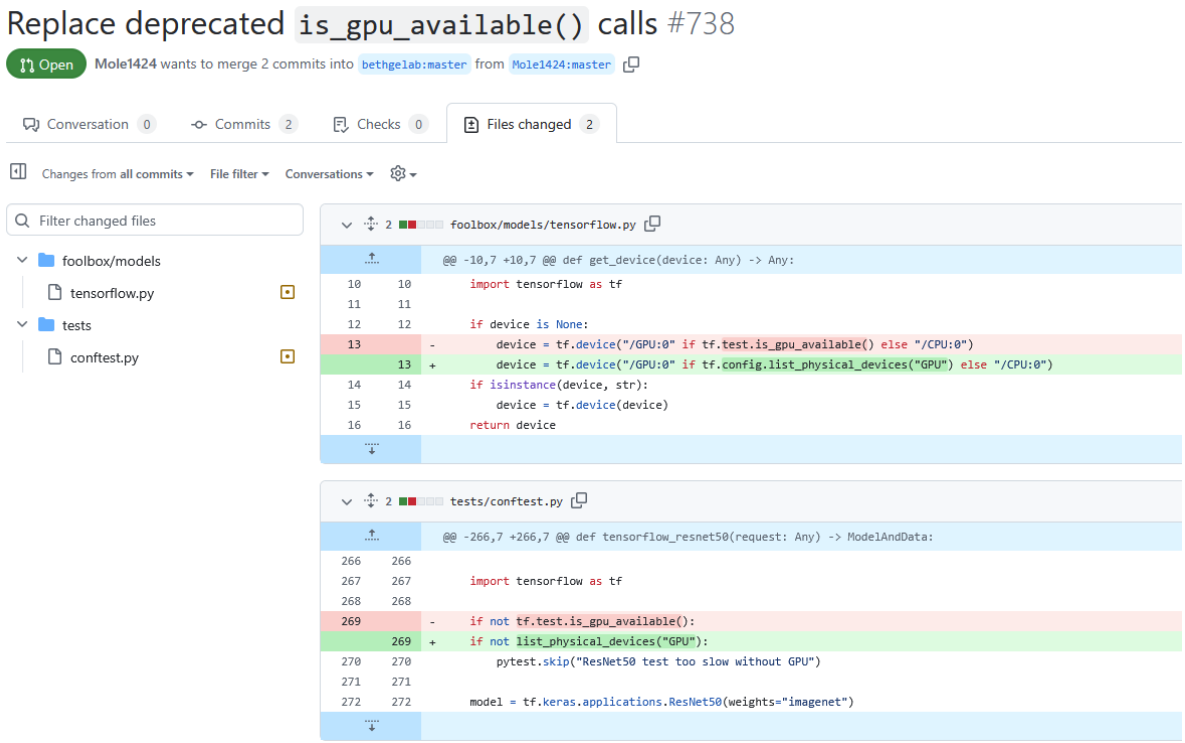
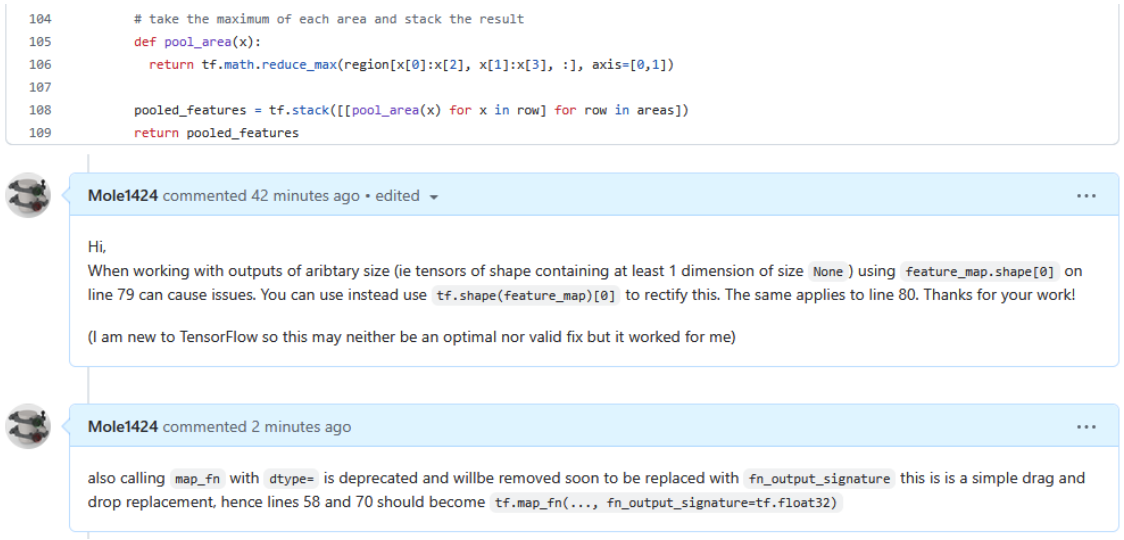


Figure E.1: A pull request made to the FoolBox[99][100] library to replace a deprecated TensorFlow function

`https://gist.github.com/Jsevillamol/0daac5a6001843942f91f2a3daea27a7`



```
104         # take the maximum of each area and stack the result
105         def pool_area(x):
106           return tf.math.reduce_max(region[x[0]:x[2], x[1]:x[3], :], axis=[0,1])
107
108         pooled_features = tf.stack([[pool_area(x) for x in row] for row in areas])
109         return pooled_features
```

Mole1424 commented 42 minutes ago • edited

Hi,
When working with outputs of aribtary size (ie tensors of shape containing at least 1 dimension of size `None`) using `feature_map.shape[0]` on line 79 can cause issues. You can use instead use `tf.shape(feature_map)[0]` to rectify this. The same applies to line 80. Thanks for your work!

(I am new to TensorFlow so this may neither be an optimal nor valid fix but it worked for me)

Mole1424 commented 2 minutes ago

also calling `map_fn` with `dtype=` is deprecated and willbe removed soon to be replaced with `fn_output_signature` this is is a simple drag and drop replacement, hence lines 58 and 70 should become `tf.map_fn(..., fn_output_signature=tf.float32)`

Figure E.2: A comment made to an ROI pooling implementation to allow for variable sizing in images