Pre-Processing to Leverage Temporal Information with Transfer Learning on Convolutional Neural Networks for End-to-End DeepFake Video Detection

Sakib Ahamed

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh
2021

Abstract

Many state-of-the-art DeepFake video detectors use Long Short-Term Memory-based networks (LSTM) as they can easily capture temporal information and take video as direct input. We opt not to use LSTM-based approaches and instead utilise Convolutional Neural Networks which cannot infer temporal information and instead take images as input. We consider a series of pre-processing methods. Frame differencing (FD) is where we take the difference of frames over subsections of video in hopes to highlight idiosyncratic movements often present in DeepFake videos. We also consider averaging over subsections of video (FA) and compare against a baseline dataset of randomly selecting single frames over the same subsections (RF). We do this in hopes to leverage temporal information in an image: FA uses averaging and blurs to indicate movement, FD uses highlighted colours to do the same, and RF does not capture any temporal data. The aim is to make our model time-aware in some rudimentary sense. We hypothesise that these pre-processing methods, FA and FD, which transform video into a series of images encapsulating movement, will increase performance compared to still, randomly selected frames, RF. We use Transfer Learning on both our MesoInception4 baselines and custom Xception for DeepFake detection. We find that this hypothesis is correct, with the frame averaging dataset yielding consistently higher performance compared to randomly selected frames. However, we did not see this trend with our novel frame differencing approach.

Acknowledgements

I would like to acknowledge my supervisor, James Garforth, for all the insightful conversations, which helped me tremendously with understanding as well as potential directions to take the project. I would also like to acknowledge my sister, Sumia Miah, and my flatmates, for helping me proofread this project during my illness during the semester, as well as pushing me to work hard despite the many unforeseen circumstances present at the time of writing.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

DeepFake is a method in which Deep Learning is employed to synthesise videos in which individuals' facial expressions or faces are transplanted onto other individuals, although there are many different variants. With time, this technology has significantly improved to the point that DeepFakes can be indistinguishable from real videos. The term "DeepFake" is a portmanteau of the words "Deep Learning" (predominant method to generate these videos [29]) and "Fake". Hyper-realistic DeepFake videos are generally created using Generative Adversarial Networks [27].

These videos can be particularly damaging to the target of the DeepFake; prominent celebrities, political figures, and everyday people have had their faces manipulated [42]. In the past, it had been the case that one would need thousands of images to create fake videos. However, as technology has improved, the amount of data required (i.e. images of source and target individuals) to create these fake videos has significantly decreased [41]. In extreme scenarios, we can foresee situations in which influential figures are DeepFaked to say outrageous statements that insight violence, political instability, and even war.

We use Transfer Learning to the problem of DeepFake detection as it allows algorithms to be trained with significantly fewer data and time, as well as being more generalisable compared to training from scratch [30]. We use a "fight AI with AI approach" whereby, when humans cannot discriminate real videos from fake ones, we use a DeepFake detector. This project concerns itself with such algorithms. There have been strides to address this issue and create DeepFake detectors with notable competitions held by Facebook on Kaggle [10].

Deep Learning with high-quality videos requires prohibitive amounts of computation and time to train. Transfer Learning with Convolutional Neural Networks (CNNs) has been shown to yield state-of-the-art results in many domains outside of computer vision by representing data as images [16] [28] [11]. This is discussed in detail in Part 2.4 Image Classifiers for Non-Image Domains. Thus we propose a novel preprocessing method to convert videos into images while retaining relevant information. We encode the contents of a DeepFake video to a single image, to take advantage of temporal information. We try three different methods to achieve this, frame averaging (FA), a randomly selected frame (RF), and frame differencing (FD).

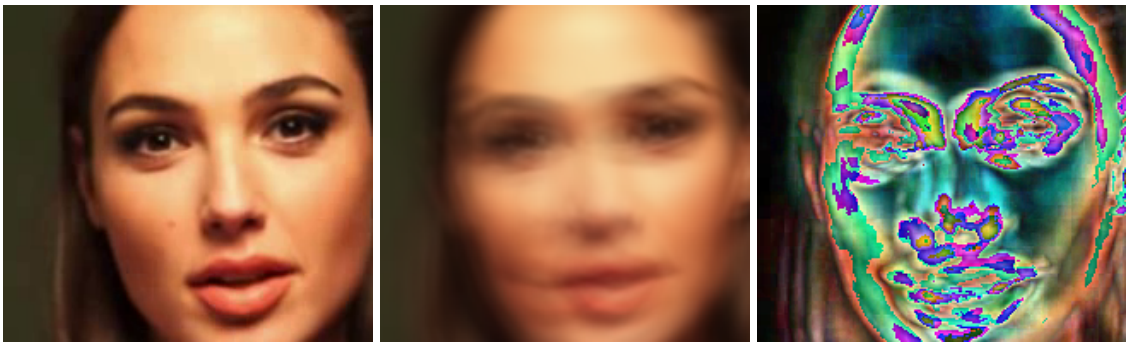

Figure 1.1: **An example of data point from three pre-processed datasets**: Random Frame (RF) *[Left]*, Frame Averaging (FA) *[Middle]*, and Frame Differencing (FD) *[Right]*

FA consists of averaging frames over subsections of video. RF is randomly selecting a frame(s), FD consists of taking the difference of frames every interval and pasting onto a blank image to create an additive image that showcases movement over multiple time intervals. Examples can be seen in Figure 4.1. These images are passed into a pre-trained model and Transfer Learning is applied to determine the legitimacy of the input video. RF is different from the other two datasets as it is not encoding temporal information, whereas the other two are.

1.2 Research Objective

The aim of this project is to create an end-to-end DeepFake detection system, with raw video as input and a prediction of real or fake as output. We hypothesise that training an FD and FA dataset will yield better performance compared to training on an RF dataset. We believe that training on FD may yield better results compared to FA, as it can be considered a blurring effect, which may lose temporal information.

State-of-the-art DeepFake detectors often use Convolutional Long Short-Term Memory Networks (ConvLSTM) [27] which leverages spatio-temporal information to make their predictions. This is also how humans tend to determine whether a DeepFake video is real or fake, e.g. by paying close attention to flickering/artifacts in facial movements.

We opt for a Convolutional Neural Network (CNN) for detection, We do this as it allows us to utilise Transfer Learning which tends to use pre-trained CNNs. This network architecture offers world-class performance on computer vision tasks [23], such as image classification, by taking advantage of spatial information through convolution operations. In conjunction with Transfer Learning, CNNs accelerate the training process while yielding high performance compared to training from scratch. CNNs already use spatial information and we hypothesise that the addition of these pre-processing methods mentioned prior will allow us to capture and utilise temporal aspects of video and potentially increase performance.

The main research objective is to determine whether pre-processing videos to datasets of FA and FD provide tangible benefits in performance compared to RF, as well as explore mechanisms behind them.

Chapter 2

Background

2.1 Deep Learning

Deep Learning (DL) is a subset of Machine Learning. It leverages the power of Artificial Neural Networks (ANN), which can be summarised as several layers performing vector operations followed by non-linear differentiable activation functions. By the Universal Approximation Theorem for ANNs, any continuous function can be approximated to an arbitrary degree of accuracy by an ANN with only a single (sufficiently large, potentially infinite) hidden layer [9].

In practice, this network would be extremely prone to overfitting; This is when the error on the training set is low, however, the error for the testing set is large in comparison. The network memorises training examples themselves, rather than the key features that distinguish them. Thus, the model is no longer generalisable to new unseen examples.

For this reason, we opt for deeper networks, which learn slower and are not as prone to overfitting. "Deep" in "Deep Learning" refers to the depth of the network. DL removes the need for hand-made features. The network approximates the underlying function described by the training data in such a way that the network is able to decompose the derivative at any point in a computationally feasible manner, allowing for gradient-based optimisation methods e.g. Stochastic Gradient Descent.

DL has proven to be effective in many areas, including computer vision tasks such as image classification using Convolutional Neural Networks [23].

2.2 Convolutional Neural Network

Convolutional Neural Networks (CNN) is a special type of neural network design that rose to prominence with AlexNet in 2012 [19], which achieved top-five accuracy of 84.7% test accuracy on ImageNet challenge, a 1000-class image classification problem. This was more than 10.8 percentage points higher than the next runner-up.

CNNs are generally divided into three parts: Convolution layers, Pooling layers, and Fully Connected/Dense layers. Convolutions and Pooling layers are used to extract spatial correlations in the image and reduce the size of input that is fed into the Dense layers. AlexNet built on top of this by introducing a deep structure (more layers) [20],

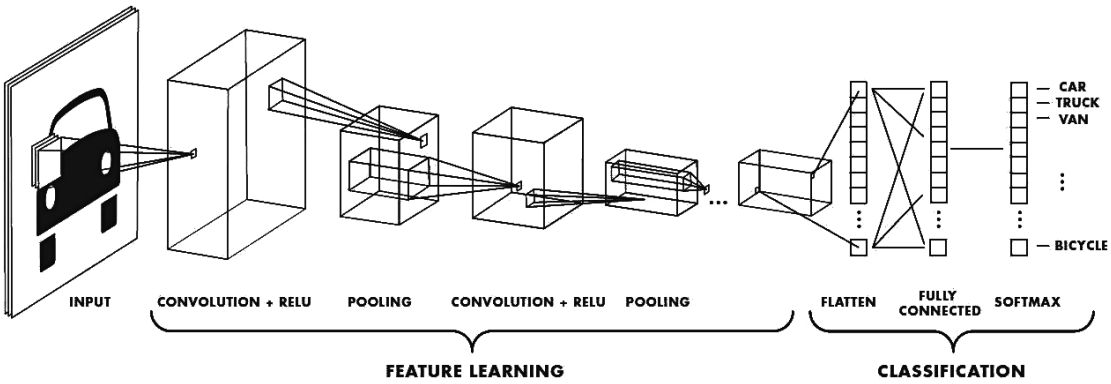

Figure 2.1: **General structure of Convolutional Neural Netowork (CNN)** for image classification, last few layers for classification are dense; with final layer corresponding to number of classes (in this case Softmax activation, i.e. Sigmoid for multi-class)

allowing for more parameters to fit the complexities of input images. This makes them well suited to computer-vision-related tasks [23].

Convolution layers can be thought of as the network breaking down subsections via convolution filters, summarising the image. The Pooling layer is then applied to rejoin, via averaging or taking the maximum value from each convolutional subsection. We apply further convolutions to these summary components images, removing redundant information to increase efficiency. The final Dense layer takes the feature vector produced by preceding layers and performs final abstractions for output e.g. classifying.

2.3 Transfer Learning

Transfer Learning is the technique of using a pre-trained model for a target domain different from its originally trained source domain. We are starting with a mature model and re-purposing it to a new domain, instead of starting completely from scratch. This means we transfer knowledge learned from the source to the target domain, reusing useful information common to both domains. Transfer Learning has been shown to be effective as it requires orders of magnitude fewer data and significantly less time to train.

The main mechanism for Transfer Learning is Fine-tuning, whereby we update pre-trained model parameters by training additional epochs for new data. There are many different ways to do this, including using the pre-trained model as a starting point and updating weights of every parameter in the network; this is analogous to a lucky parameter initialisation. A commonly used method is to freeze all layers from input to a given layer, L', and only update parameters from L' onwards during training. Freezing sets parameters of the network in place, so they can not be updated during Backpropagation.

Previously learned information is stored as the model's parameters i.e. approximation of the underlying function described by data. We freeze parameters to utilise as

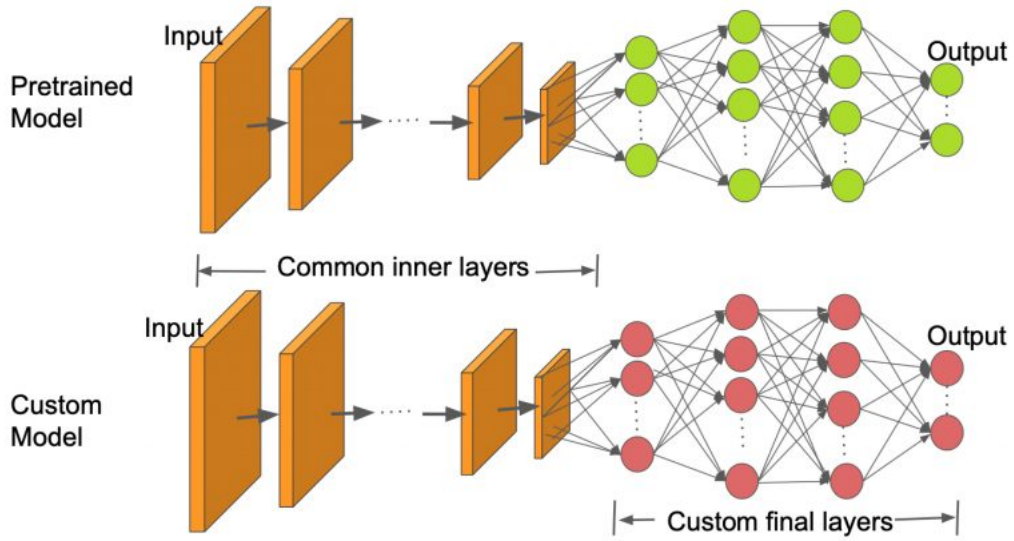

Figure 2.2: **Transfer Learning** approach where we use off-the-shelf pre-trained models as Feature Extractor, i.e. we "chop" of last classifying layer (or more in this case) and attach a new custom classifying network, CCN for a new target-domain

much information learned by the original pre-trained model as possible.

Another option is to freeze all layers until L' and add extra layers for a specific target domain e.g. classification. This can be thought of as freezing and chopping off the network at L' and attaching a trainable sub-network that takes the output from L' as input. Special case being when L' is the last layer of the pre-trained model, where we are adding a sub-network at the end of the existing one (Figure 2.2) - This is using the pre-trained model as a feature extractor, and the model we use in this project due to its simplicity and decreased computational demands [30].

In general, CNN pre-trained models are trained on natural images which means the first few layers of the network tend to be the same regardless of domain. We can visualise parameters of AlexNet (Figure 2.3 [45]). Parameters in layer one learn to distinguish general features; lines, edges, and gradients. Layer two combines features learned in the previous layer to learn more complex features; curves, corners, and simple shapes. Later layers perform further abstractions, learning domain-specific features. Fine-tuning allows us to adapt what later layers learn to focus on new domains, in our case from 1000-class (ImageNet) to binary classification of DeepFakes.

2.4 Image Classifiers for Non-Image Domains

CNNs are exceptional at image classification but fall short when the domain is not visual. However, data can be transformed to create images.

For example, we can convert sounds to a spectrogram which is a visual representation of frequencies and amplitudes over time. This can be seen in the top right of Figure 2.4. We can see that each sound has a distinct spectral signature associated with it. Mushtaq et al. showed state-of-the-art performance of 99.49% for UrbanSound8K

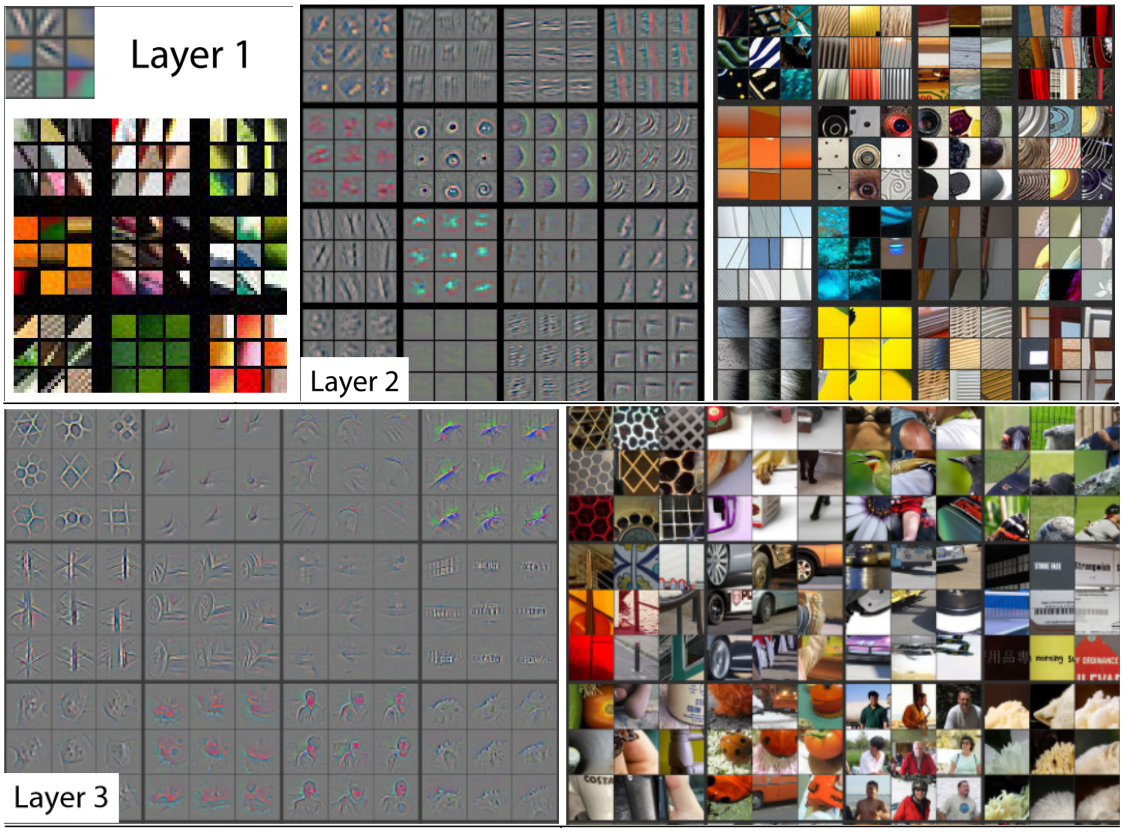

Figure 2.3: **Visualisation of learned features** for ImageNet via weights in AlexNet [45]

dataset (10-class sound classification [33]) using spectrograms and Transfer Learning [28].

Similarly, Mahmoud Kalash et al. achieved state-of-the-art results in malware classification [16] by transforming binary machine code to grayscale images (bottom of Figure 2.4). Another example of this is Gleb Esman's anti-fraud system at Splunk [11]. Using user's mouse movements and clicks to draw images (top left of Figure 2.4), Esman's network can recognise a real customer from a fake customer with over 80% accuracy given user activity.

The ground-breaking performance achieved by these methods was the main motivation for this project, transforming data into a meaningful image that captures the essence of the original video.

2.5 Types of DeepFake Videos

There are generally three levels of DeepFake. Fully Synthesised is when no component of the image is real, such as produced by Nvidia's StyleGAN [17]. Examples of this can be view on popular websites such as This Person Does Not Exist. The next level of facial manipulation is face-swapping i.e. transforming a target face onto another source face, where both individuals are real [6]. The lowest level of DeepFake is switching facial expressions or some other attributes such as hair colour [43], e.g. using Trump's

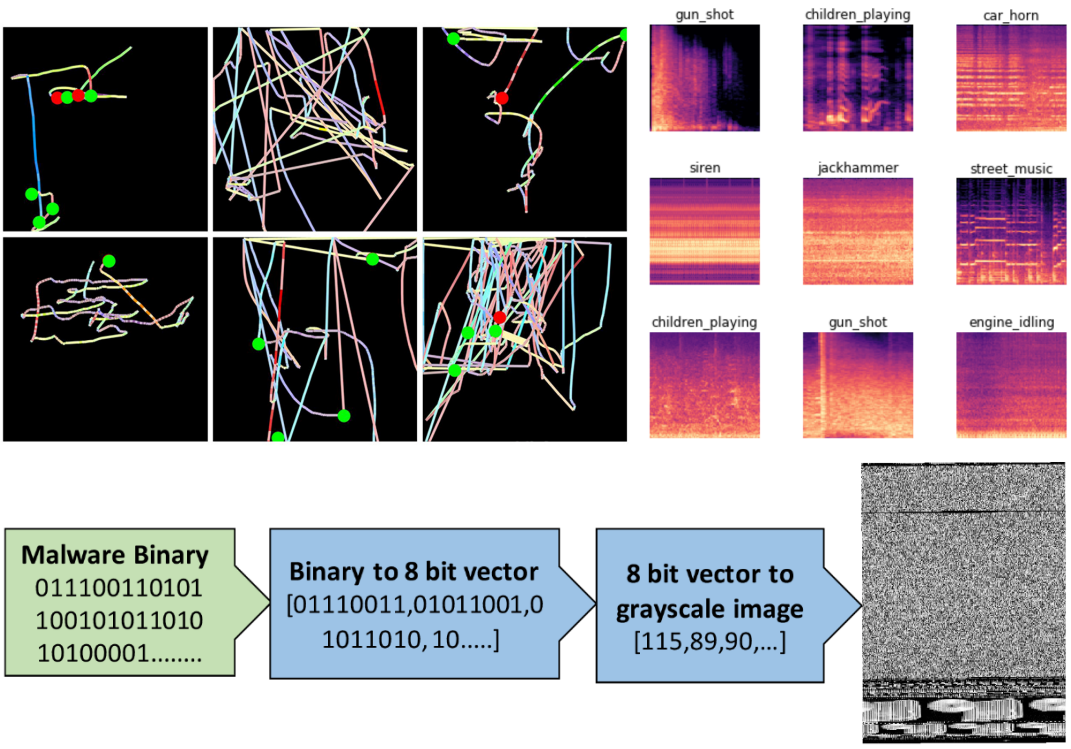

Figure 2.4: **Three examples of transforming data to an image for use with CNNs**; top left transforms mouse movements and clicks to line drawing for fraud detection, top right transforms sounds to images for sound classification, and bottom image shows malware software being transformed into binary images for malware detection.

face to make him say something novel, this can be achieved via Thies et al.'s Face2Face [40].

This project concerns itself with the face-swapping level of manipulation as we believe it has the most potential for causing harm to society currently. One could argue that lower facial attributes and expression manipulation could be worse for society as one could manipulate political figures into saying abhorrent statements, but this can also be achieved with face-swapping level with carefully chosen target actors.

A malicious user of this technology intending to do harm, would not use facial attribute and expression manipulation as it requires videos that are real, meaning that it could be traced back and proven fake; this is not the case for face-swapping.

The majority of DeepFakes online are used for pornography in some shape or form, with some estimates claiming over 96% of fake videos online are pornographic in nature [39]. Pornographic DeepFakes are usually created using celebrity faces as source face as there are large amounts of facial data available for these individuals.

However, state-of-the-art DeepFake generators are able to create fake videos using significantly less data [35], meaning prevalence of revenge porn using faces of everyday people, via pictures on social media, has become more prominent.

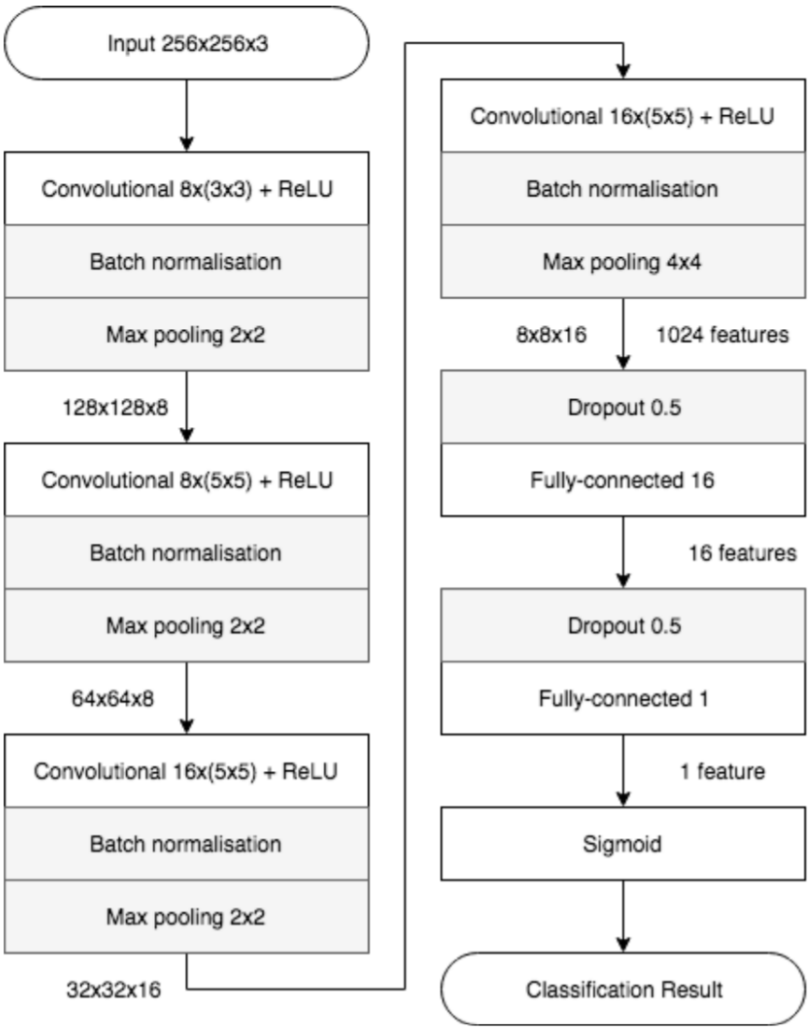

Figure 2.5: **Structure of Meso4 DeepFake detection CNN**; MesoInception4 replaces first two convolution layers with Inception layers, otherwise identical to Meso4 [3]

2.6 DeepFake Video Detection

Older methods of DeepFake detection utilise hand-made features to detect fake videos, more recent work use DL to inspect for a number of anomalies present in DeepFake videos. This ranges from idiosyncrasies in frames (e.g. twitching), visual artifacts (e.g. face warping), to more sophisticated methods e.g. measure an individual's heartbeat through the video to detect DeepFakes [37] [13].

The majority of state-of-the-art DeepFake methods are time-aware, using ConvLSTMs or their derivatives. We do not use ConvLSTM-based methods as our baseline, as it is not a fair comparison to our CNN-based solution. Instead, we opt to use a state-of-the-art CNN-based detector, MesoNet [3], first introduced by Afchar et al. in 2018 which demonstrated a detection rate of more than 95% for Face2Face generation method. MesoNet created two networks Meso4 and MesoInception4 which utilise mesoscopic features, i.e. scale between microscopic (very small; pixel-level features)

and macroscopic (large; visible to the naked eye).

Structure of Meso4 can be seen in Figure 2.5. MesoInception4 marginally improves upon Meso4 by replacing first two convolution layers with Inception layers. Inception layers were first seen in InceptionNet [38],

It can be challenging to figure out optimal convolution kernel sizes, Inception remedies this by trying all of them. Inception layers combine convolution layers with kernel sizes of 1×1, 3×3, and 5×5 with their output filters concatenated and reshaped to a single output vector for the next stage. Due to limits in time and computation, we have opted to only consider MesoInception4 as it yields slightly better performance compared to Meso4 [3]. In addition, our final CNN-based model uses Xception [8] with ImageNet weights, as a base network, which also uses Inception layers and is the latest iteration of Inception architecture.

Chapter 3

Dataset

There are a number of promising open-source DeepFake datasets that can be used for the purpose of detection. We considered FaceForensics++ (FF), DeepFake Detection Challenge (DFDC), and CelebDFv2 (CDFv2); all datasets proposed are at the middle, face-swapping level of manipulation.

3.1 FaceForensics++

The FF dataset [32] contains $1,000$ video sequences manipulated with four methods which include, Face2Face, FaceSwap, NeuralTextures, and the classical Deep-Fake. FF released 01-2019, which contains $1,000$ real and fake videos, gathered from genuine YouTube videos at a resolution of 256×256, totaling $509,900$ frames per class. This dataset mostly contains frontal faces with minimal obstruction, allowing for best-case video forgery which may aid in generalisability. FF also provides binary face masks, which allow for more complex tasks such as segmentation.

The main reason we chose not to use this dataset was due to ethical issues around the lack of actor consent. The videos within the dataset were all sourced from YouTube. We deem it unethical to conduct research and benefit from this, on the basis of unconsenting (non-celebrity) individuals, even if it is closer to a real-world use case. FF is also used in MesoNet's original paper, but this dataset is outdated compared to the other options.

3.2 Deepfake Detection Challenge

This was the dataset created by Facebook for the world-famous DeepFake detection challenge on Kaggle [10]. Although it was a very promising candidate, We chose not to use this dataset due to the sheer size of it, with 119146 .mp4 files totaling 471.84GB compressed. DFDC released 10-2019 contains 1,131 real and 4,113 fake videos. Created with the consent of 66 actors of various genders, ages, and ethnicities, totaling 488,400 real frames and 1,783,300 fake frames.

This dataset features eight facial modification algorithms which would help us create a generalisable solution. We considered using a subset of DFCF, although the other datasets were more compelling due to actor diversity. DFCF only contains explicitly

consenting actors which helps alleviate the ethical issues. However, a major drawback with this dataset is that there is no ground truth for every DeepFake. Some actors only appear in a single class e.g. fake, and therefore there is a chance that the model will learn which individuals appear in which classes instead of if they're fake or not. This is not the case with CDFv2.

3.3 CelebDFv2

CDFv2 [24] released 11-2019 contains 590 and 5,639 real and fake videos, respectively. Created from YouTube interview clips of 62 celebrities DeepFaked among each other based on gender. These actors are of differing ages and ethnicities, with balanced genders. Totaling 225,400 real frames and 2,116,800 fake frames. We opt to use the CDFv2 dataset for a number of reasons. The main one being that every DeepFake has an associated ground truth i.e. every real video has at least one corresponding fake video; 9.6 fake videos for every real video. Ground truths are not included in the other datasets mentioned in this section, making it impossible to isolate actors for a fair test.

The majority of videos within CDFv2 are of an interview-style format which is helpful for pre-processing as backgrounds will not change over time and faces do not move as much as compared to the other datasets. This dataset consists of celebrities, whose faces are already in the public domain. We see them on the movie screen and although these actors have not consented to have DeepFakes created of them, we deem it more ethically allowable than "every day" strangers having their faces DeepFaked which is the case for FF.

Comparing DeepFake datasets is non-trivial and while there is no one agreed-upon metric for such comparisons, the Structural Similarity Index Measure (SSIM) [26] is often used. It provides a score between the DeepFake in question and the original real face. Using mask-SSIM, where "mask" refers to the segmented facial area. CDFv2 shows the highest mask-SSIM score with a 0.04 improvement on its predecessor [24], meaning the DeepFakes provided by CDFv2 are the most realistic to date.

Ground truths present in CDFv2 allow us to create actor-isolated test sets which may aid in determining model generalisability. We take care to split the train-validation and test set in such a way that actors who appear in the train-validation set will never appear in the test set. We use identical train-validation and test sets for training and evaluation of all models, this includes the random seeds and the order in which images are shown to each model, ensuring a fair test.

Chapter 4

Data Pre-Processing

Pre-processing is the main novel contribution for the project, we compress the contents of a DeepFake video to a single image every one-second interval. At the start of the project, we opted for a single image over the entire video, but due to severe data imbalance, we later opted for an image every one second ($k = 30$ or 30 frames). The reasons for doing so are covered in detail within 4.6 Class Imbalance.

These videos are on average 13 seconds in length at 30 frames per second and high definition, resulting in large file sizes. This compression allows us to reduce the file size and take advantage of temporal information through our pre-processing methods. We try three different methods to achieve this, frame averaging (FA), frame differencing (FD), and using randomly selected frames (RF) as a baseline. This means we create 13 pre-processed images per video. Reducing the original 8.8GB CDFv2 dataset to 5.7GB, a 35.2% decrease in the magnitude of data we are working with.

Order of the sections in this chapter roughly corresponds to the order of operations for our pre-processing pipeline. A high-level diagram of the pipeline can be seen in Figure 4.1. This shows an example of pre-processing over intervals of three frames ($k = 3$), i.e. we use three frames to compute a new pre-processed video, with the frame-to-face-cropping, and edge-case checks omitted for brevity.

4.1 Train-Test Actor Isolation

Each test set has pre-processing applied which corresponds to the train set. i.e. if a model is trained on FA, we evaluate it on the test set for FA. Each set contains images created from the same videos. To keep the test as fair as possible, we isolate an equal number of male and female actors for test purposes only. There are 62 actors in this dataset, we pick 18 actors exclusively for testing purposes; 9 male and 9 female actors, this means that we are using 29.9% of the total dataset for testing. This ensures that there is no chance for the model to learn actors' faces and utilise them for its predictions. This means 18,038 fake and 1,600 real images per pre-processed test set.

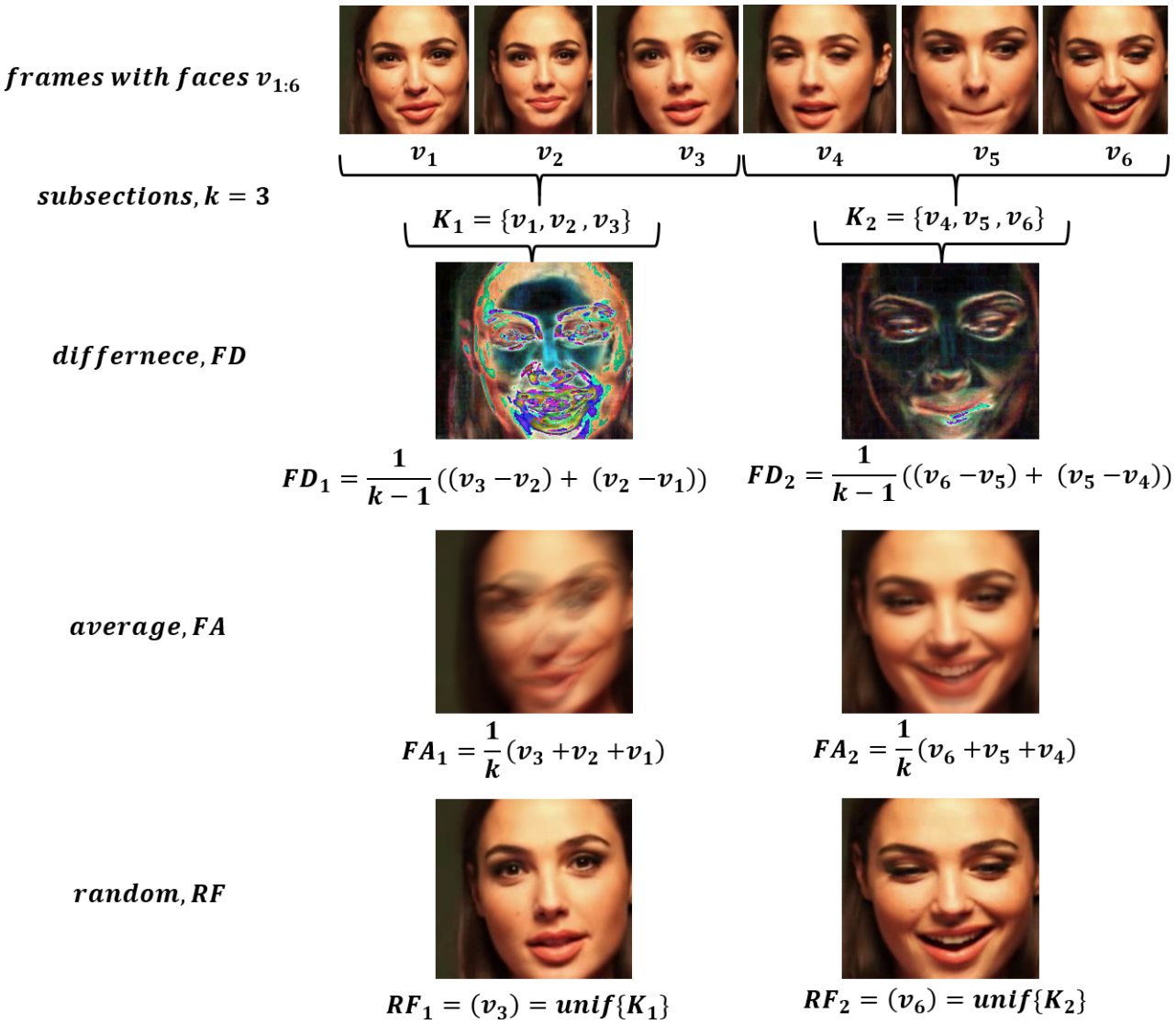

Figure 4.1: **Toy example of pre-processing pipeline** with six frames for FD, FA, and RF datasets with cropping (ExtractFaceLocations($\{v_i\}$)) omitted for clarity.

4.2 Face Recognition

The pipeline starts with extracting all faces as images from a video. We do this by extracting all frames from the video, scanning each frame for a face, discarding all frames which do not contain a face, pre-processing and cropping any faces found with the smallest bounding box which encapsulates the frames over the interval being considered, there are minor modifications to this technique for each of the pre-processing methods but this will be covered in more detail in the respective sections.

Consider an .mp4 video, $\mathcal{V}$ of class Real or Fake. W, H corresponds to the width and height of a single frame in $\mathcal{V}$. The 3 refers to the number of channels (RGB), and N the total number of frames.

Figure 4.1 captures the essence of the pre-processing being carried out with a toy example of six frames and an interval, $k = 3$. This means we will group k frames

together (if they contain faces). We use this group, $\mathcal{K}_i$, to create the $\lfloor \frac{N}{k} \rfloor$ images from each group, $\mathcal{K}_i$.

$$\mathcal{V} = W \times H \times 3 \times N \in \{\text{Real}, \text{Fake}\}$$

Note, in this and all other examples, we assume that every frame contains a face, in the code, we perform many checks to mitigate edge cases - even so, we may not always end up $\lfloor \frac{N}{k} \rfloor$ images from N frames. But as we are assuming all N frames have detectable faces for purposes of explainability, we will have no issues with the numbers.

We extract all frames in $\mathcal{V}$ resulting in a list of frames, $\{v_1, v_2, ..., v_N\} \in \mathcal{V}$, where v_i corresponds to the i-th frame of the video which have faces.

$$\text{ExtractFrames}(\mathcal{V}) = \{v_1, v_2, ..., v_N\}$$

We then partition the video into groups or subsections with k elements in each (if they contain a face). Since CDFv2 is at 30 frames per second, we use the arbitrary choice of $k = 30$, we experimented with different intervals (we tried $k = N$; not enough data, $k = 2$; too much data and infeasible compute) but due to time constraints, we could not determine optimal intervals for each pre-processed dataset.

$$\text{Group}(k, \mathcal{V}) = \{\forall \mathcal{K}_i \subseteq \mathcal{V} | \{\mathcal{K}_1, \mathcal{K}_2, ..., \mathcal{K}_{\lfloor \frac{N}{k} \rfloor}\}$$

Where grouped subsection of frames, $\mathcal{K}_i$ for any i, we have at-most k frames in $\mathcal{K}_i$. The special cases being when $k = N$ where we average/difference/randomly select over the entire video and $k = 2$ where we just average/difference/randomly select over pairs of consecutive frames, provided they contain faces ($k = 1$ does not make sense for FD and FA).

For any set of frames, $\{v_i\}$, we scan for faces (using the face_recognition python library [12]), this returns the face location as a four-tuple corresponding to the smallest bounding box which contains the face(s) found in $\{v_i\}$. Four-tuple because this is the four lines needed to create this box, which can be seen in Figure 4.2. We assume that there will only be one face per video, any extra faces detected are discarded.

$$\text{ExtractFaceLocations}(\{v_i\}) = \forall i(t_{max(i)}, b_{min(i)}, l_{max(i)}, r_{min(i)}) = c_{\{v_i\}}$$

ExtractFaceLocations$(\{.\})$ calculates the smallest bounding box which encapsulates all faces within any set, $\{v_i\}$, and returns it as a four-tuple face location. This face four-tuple location, c_i can be used to "crop" v_i, in Python this is done via list splicing e.g. `face = frame[top:bottom, left:right]` but here we use $\mathcal{F}_i = v_i((t^{(i)}, b^{(i)}, l^{(i)}, r^{(i)}))$ to denote the cropped face frame, $\mathcal{F}_i$ (seen in bottom right of Figure 4.2), This cropping only effects W and H. More concisely, $\mathcal{F}_i = v_i(c_{\{v_i\}})$. Now that we have the face frame, $\mathcal{F}_i$, and the locations associated with each frame, c_i, this is where the pipeline splits into three.

4.3 Random Frame

RF is randomly selecting a frame(s). We pick a random frame from every grouped subsection, K_i for all i with uniform distribution. This can be seen at the bottom of

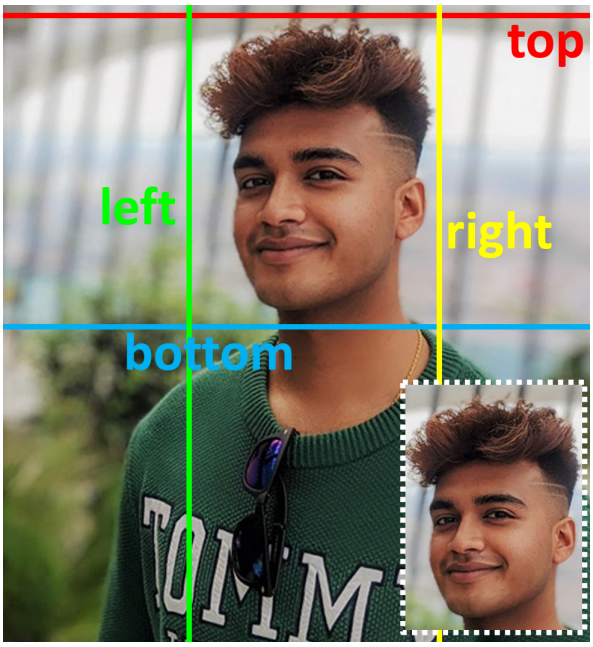

Figure 4.2: **Example of four-tuple output** returned by ExtractFaceLocations$(\{v_i\})$, where t, b, l, r correspond to *top, bottom, left* and *right*, respectively

Figure 4.1, "Random, RF". $\forall$ random frames, r_i,

$$r_i = \text{unif}\{\mathcal{K}_i\} = \text{unif}\{\{v_1^{(i)}, v_2^{(i)}, ...v_k^{(i)}\}\}$$

r_i is our single, randomly selected frame (in one of our k grouped subsections), now we must crop it. To obtain the final image, since there's only one frame to deal with, ExtractFaceLocations$(\{.\})$ just returns the four-tuple location of the face, which is inherently the smallest bounding boxing containing the face:

$$\mathcal{RF}_i = r_i(\text{ExtractFaceLocations}(\{r_i\})) = r_i(f_{r_i}))$$

4.4 Frame Averaging

FA consists of averaging the frames over the grouped subsections, $\mathcal{K}_i$, of the video. Before averaging, we find the largest bounding box which contains all faces being averaged, this is again done using ExtractFaceLocations$(\{.\})$.

We care about this as when it comes to averaging and differencing, it does not make sense to look at individual faces, we need a set reference point so that there's no overlapping - this concept is similar to matrix operations which rely on consistent shape.

We achieve this by averaging over all face frames and taking the largest bounding box which contains all faces being averaged. $\forall$ averaged frames, a_i,

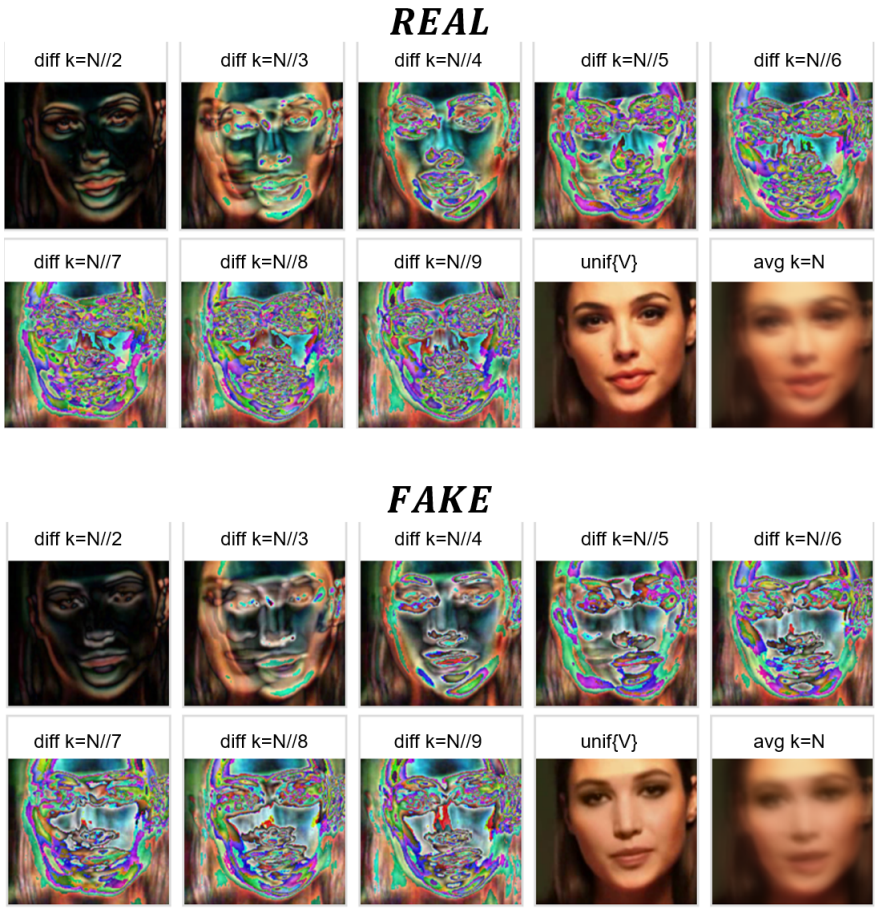

Figure 4.3: *Top*: real frame, with RF, FA and FD pre-processing applied with varying intervals, k. *Bottom*: fake frame, with same pre-processing showcase differences among real and fake

$$a_i = \frac{1}{k} \sum_{\mathcal{K}_i} v_i^{(i)}$$

In this case, although a_i is again a single image corresponding to the averaging over k frames in $\mathcal{K}_i$, we need to crop and get the smallest bounding box which contains all faces in $\mathcal{K}_i$.

$$\mathcal{FA}_i = a_i(\text{ExtractFaceLocations}(\mathcal{K}_i)) = a_i(c_{\mathcal{K}_i})$$

4.5 Frame Differencing

FD is very similar to FA, however, instead of summing and dividing by k, we subtract consecutive frames, in an overlapping fashion, and sum the differences up, then normalise.

This process is clearly illustrated at the top of Figure 4.1 whereby the first $k = 3$ frames are grouped to create $\mathcal{K}_1 = \{v_1, v_2, v_3\}$ and then $\mathcal{K}_1$ is used to subtract consecutive frames and sum, similar to zipping and subtracting in Python. This results in the difference final frame in the example being equal to $\mathcal{FD}_1 = \frac{1}{k-1}((v_3 - v_2) + (v_2 - v_1))$. More generally this can be stated as, $\forall$ differenced frames, d_i:

$$d_i = \frac{1}{k-1} \sum_{\mathcal{K}_i} (v^{(i)}_{j+1} - v^{(i)}_j)$$

$$\mathcal{FD}_i = d_i(\text{ExtractFaceLocations}(\mathcal{K}_i)) = d_i(c_{\mathcal{K}_i})$$

We also considered performing no overlapping subtracts, i.e. $v_1, v_2, v_3, v_4 \rightarrow (v_4 - v_3) + (v_2 - v_1)$, but this did not make sense as we would be losing the information between these sub-groups. We normalise due to the fact that as we keep summing, the RGB values start increasing past 255, normalisation must be applied to keep the image from being completely white as differences accumulate. Our first implementation of these pre-processing techniques did not yield k distinct images (per subsection), instead, we combined the k images into one image, this can be seen in Figure 4.3. This was later changed due to severe levels of class imbalance and poor performance on the validation set.

Modern DNNs, including Xception, are more than capable of handling class imbalance provided enough data. We tried many different techniques for easing the class imbalance (9.6 to 1), but one of the simplest ways to ease class imbalance is to use more data [15]. Thus, we omit the combining of k images to one image and instead output the k images themselves. The reason we chose to use FD was that we believed that the network could pick up on some meaningful details within the FD frames.

Prior to doing this change, we heavily experimented with values of k, to see if there was any tangible difference detectable (via the human eye) from a real video to a fake one, the results of which can be seen in Figure 4.3.

We do indeed see some artifacts only present in the fake videos using FD. If we consider the real image at *"diff $k = N//5$"* (i.e. the video was split into 5 equal parts and differenced, then combined; in our final dataset we do not combine), we can see the FD frame, with highly saturated parts of the image indicating lots of movement and black parts of the image showing little movement (frame to frame over the five subsections). If we then consider the same *"diff $k = N//5$"* image in the fake image, we see that the image is very similar but colors are muted. This indicates that fewer changes are occurring from frame to frame. This is just one example, this trend holds true for most videos tested.

This is odd, as a key giveaway for DeepFakes is temporal flickering, which would show up as a more colorful image with many saturated spots compared to the real counterpart. Instead, we see the opposite trend, real images have more movement from frame to frame, we hypothesise this is due to the micro facial expressions performed by real humans which many DeepFakes fail to emulate, leading to unrealistic images. In DeepFakes which look very real, this muted color phenomenon is less dramatic.

4.6 Class Imbalance

As mentioned prior, we made a lot of changes due to the severe class imbalance present in CDFv2, there are approx. 9.6 fake videos for every real video. We tried a variety of techniques to address this issue including: *Sigmoid Focal Loss* (niche loss function used in image segmentation problems where the class imbalance is extreme as most of the data is not the class being segmented [25]), *data augmentation* (creating new data from existing data via transformations), *synthetic minority oversampling technique* (SMOTE, discarding extra fake examples), *thresholding* (changing the threshold for prediction, i.e. at which point do we class as real/fake), *ensembling* (creating 10 models and training on all reals and a tenth of the fake images per model, then casting a weighted vote for predictions), and using *more data*. With preliminary experiments, we found that Sigmoid Focal Loss had zero effect on the final AUC, thus we did not include it in our final model.

Data augmentation helped the model's generalisability, we used default, commonly used settings for data augmentation found on Keras Blogs as these examples yielded positive results. SMOTE and the majority class oversampling counterpart showed little to no impact on our model performance, thus we opted not to use it as we would be throwing away nine-tenths of our dataset or increasing the chance of overfitting. Thresholding yielded positive results improving metrics like F1 (harmonic average of precision and recall) but this effect was negligible, AUC curves inherently vary the threshold for us. We did not manage to create and test ensemble models due to time and computational constraints.

Chapter 5

Methodology

5.1 Overview

This chapter covers methodology, order of experiments, and general motivation. Although there are many different approaches to Transfer Learning, we chose to focus on using a pre-trained model as a feature extractor.

For example, using Xception [8] as a feature extractor consists of utilising the network without its final classification layer. This transforms input images to a 2048-dimensional vector, which extracts valuable features from the source-domain, and we can leverage this knowledge to a new-domain task.

We do this by discarding the final classification layer and freezing parameters of our pre-trained model, so as not to alter the learned features. Then a number of trainable custom final layers are concatenated, a classifying network of our own (seen in Figure 2.2).

However, for purposes of determining optimal hyperparameters, we make our custom classifying network, CCN, a single output neuron until we determine all other optimal hyperparameters; i.e. we take the output from the penultimate layer as the feature vector and feed it to a single dense unit (for binary classification) for all rounds of the experiment which do not consider network structure. After calibration of all other components is complete, we experiment to determine the optimal structure. We then use this optimal CCN and train on all datasets and evaluate.

We consider a single untrained network, MesoInception4, as our baseline. This network comes with two sets of pre-trained weights, which correspond to training on the Deep-Fake (DF) and Face2Face (F2F) datasets. This allows us to circumvent training from scratch and evaluate for each dataset and for each pre-trained weight. UntrainedBaselines $= \{\text{Weights}_{F2F}, \text{Weights}_{DF}\} \times \{\text{RF}, \text{FA}, \text{FD}\} = \{$U-F2F-RF, ..., U-DF-FD$\}$. Due to poor performance with these out-of-the-box baselines, we consider a minimally trained network, using our baseline with the best weights and apply basic Transfer Learning. In the same fashion as our Xception network, we use MesoInception4 loaded with F2F weights (U-F2F). This model is then used as a feature extractor, only training the last layer to obtain a series of minimally trained feature extractor baselines that have been fine-tuned to our datasets. This makes baseline comparisons to our custom Xception model fairer. FeatureExtractorBaselines $= \textit{Best}(\text{UntrainedBaselines}) \times \{\text{RF}, \text{FA}, \text{FD}\} = \{\text{FE-RF}, \text{FE-FA}, \text{FE-FD}\}$.

5.1.1 Automated Machine Learning Approach

We utilise the Hyperband [22] algorithm to find optimal hyperparameters of the model and CCN structure. We use this hyperparameter search algorithm due to its performance in the literature and its ability to find optimal calibrations quickly.

We treat this search algorithm as a black-box, whereby we only give it a subset of the hyperparameters to determine the optimal settings so far. We opt for a tournament-style experimentation method to determine the best parameters and structure, we consider the hyperparameters with the broadest effects first; loss, optimiser, and CCN structure.

5.1.2 Experiment Outline

Hyperband was used to get optimal hyperparameters, λ, and we use $|\lambda| \times 100$ epochs of exploration to ensure the reliability of the result. This number of epochs of exploration is arbitrary but we thought it would be useful to increase epochs for exploration as the number of hyperparameters being considered increased. The order of experiments are as follows:

0.) Evaluate untrained baselines, with both sets of weights on all datasets (U-F2F & U-DF).

1.) Pick best untrained baselines and apply fine-tuning on the last layer to create feature extractor baselines (FE) and re-evaluate.

2.) Given CCN $= \{\text{Dense}(1)\}$, pick best Optimiser $\hat{O} \in O = \{$Binary Cross-Entropy (BCH), Sigmoid Focal Cross-Entropy (FL)$\}$.

3.) Given $\hat{O}, CCN = \{Dense(1)\}$, pick best Loss $\hat{L} \in L = \{$Adam, RMSprop, SGD$\}$.

4.) Given $\hat{O}, \hat{L}$ pick best structure $\hat{\text{CCN}} \in \text{CNN} = \{(\text{Dense(d)}|\text{Dropout(p)}) * \text{Dense(1)}\}$ where d and p corresponds to the number of hidden units and Dropout probability for the given layer, respectively. Note: we use regular expression notation for CCN. This means that CCN is a sequence of any number of Dense or Dropout layers (with any number of hidden units or dropout probability), which must end with a single Dense unit for the final layer.

5.) Evaluate our model (Xception $\rightarrow \hat{\text{CCN}}(\hat{O}, \hat{L})$) and compare against all other models.

The main metric we will consider when evaluating is Area Under the ROC (Receiver Operating Characteristic) Curve, also known as AUC [7]. We chose to use this metric over others, such as accuracy, as it is commonly used in image classification tasks and indifferent to class imbalance. We also consider other metrics such as F1 and the confusion matrix to gain further insights into specific models. We use the identical train and test sets for all models, with the same random seeds, batch sizes, order of evaluation, etc., ensuring that the only thing that changes is the model we are training or evaluating on.

5.2 Pre-Trained Model & Baseline

Table 5.1 [1] shows performance of a number of pre-trained models on the ImageNet challenge. Xception has a number of desirable features, reaching the highest top-1

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	*88 MB*	*0.790*	*0.945*	*22,910,480*	*126*
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	*92 MB*	*0.779*	*0.937*	*23,851,784*	*159*
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201

Table 5.1: **Performance of a variety of pre-trained models on ImageNet**, sorted by top-one accuracy [1]

accuracy of 79%, which is 1.1% higher than InceptionV3 which is the model that MesoInception is based on. Xception also falls into the bottom third for model size at 88MB, this makes it easier to download and work with. For these reasons, we use Xception as our pre-trained network to build on top of. As mentioned in 2.6 DeepFake Video Detection we will use the MesoInception4 (Figure 2.5) as our baseline network, due to its similarity to Xception, and marginal performance gain over Meso4.

Baseline experiments will consist of evaluating the out-of-the-box, untrained MesoInception4 on our three datasets. We expect this to yield low AUC, thus we apply minimal training on MesoInception4, in the same fashion as Xception. We use U-F2F, as it is the best performing untrained baseline, as a feature extractor and only train the last layer, with the identical loss, optimiser, and other parameters outlined in Afchar et al.'s original paper [3]. We hypothesise that the RF dataset will perform best on this baseline, with FA as a close second and we expect FD to perform poorly, but we will test this regardless.

5.3 Hyperband Hyperparameter Tuning

Hyperband performs hyperparameter tuning as a pure-exploration non-stochastic infinite-armed bandit problem [22].

Similar to Bayesian Optimisation (BO), Hyperband is a search optimisation algorithm. We chose not to use BO as the literature suggests that Hyperband is better than Bayesian optimisation provided enough epochs of exploration, yielding up to an order-of-magnitude speedup over other methods [22].

Hyperband utilises successive halving, i.e. dropping the bottom worst half of hyperparameters, and randomly distributes resources to each batch in order to explore different convergence behaviors. This is essentially an augmented random search that

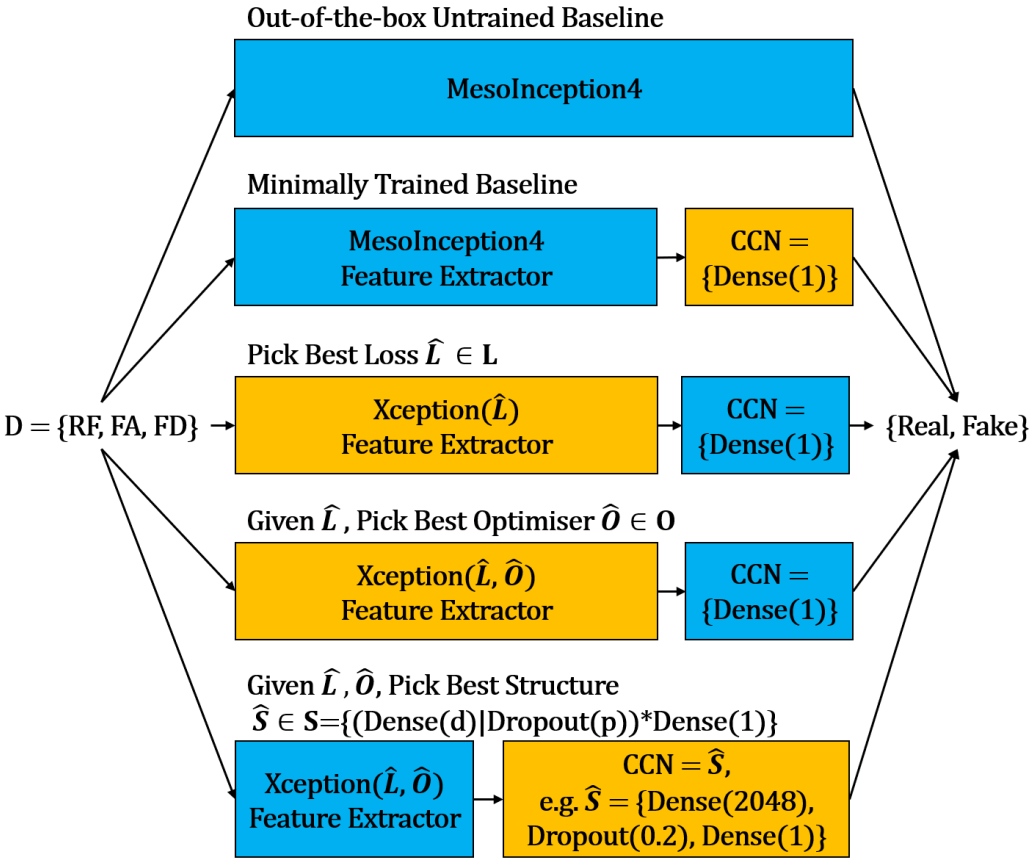

Figure 5.1: **Outline of main experiments**: untrained baselines, minimally trained feature extractor baselines, and process to find optimal custom classifying network, CCN

pays attention to good hyperparameter candidates but still distributes resources randomly in-hopes not to overfit to hyperparameter that performs well early on.

We set Hyperband's objective function to maximise validation AUC and give it a subset of hyperparameters to consider at each round of experimentation and carrying on with the best. More concisely, for some subset of hyperparameters $\lambda \subseteq O \times L \times CCN$, we use Hyperband($\lambda$) for $|\lambda| \times 100$ epochs to find the best hyperparameter, $\hat{\lambda} \in \lambda$ (Seen in Figure 5.1). We then use the architecture determined by Hyperband and train to 100 epochs (with early stopping and patience of 10 epochs, i.e. if the model does not improve in 10 non-consecutive epochs, we terminate) and evaluate. This 100 epoch training and early stopping are also applied to feature extractor baselines, we do this in order to eliminate unnecessary computation.

5.4 Loss Function

We experiment with two loss functions, Binary Cross-Entropy (BCH) and Sigmoid Focal Cross-Entropy, also known as Focal Loss (FL) [25].

If we consider ground-truth of a class to be $y \in \{-1, 1\}$ and the model's prediction probability of class $y = 1$ to be $p \in [0, 1]$ then we can define BCH loss for binary

classification can be as:

$$\mathrm{BCH}(p,y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise} \end{cases}$$

This distance may range outside of the interval $[0,1]$, so we normalise, we use Rectified Linear Unit (ReLU) to do this for non-classifying layers and the Sigmoid activation for the final classifying layer. This is a commonly used loss in many DL-based computer vision tasks. For mathematical convenience, we introduce p_t as we can rewrite $\mathrm{BCH}(p,y) = \mathrm{BCH}(p_\mathrm{t}) = -\log(p_\mathrm{t})$ where:

$$p_\mathrm{t} = \begin{cases} p & \text{if } y = 1 \\ 1^I - p & \text{otherwise} \end{cases}$$

Due to the 9.6 fake to 1 real class imbalance problem, we used class weights as it was beneficial in preliminary results. This is a simple oversampling method whereby every fake image will only be considered in conjunction with 9.6 real images, meaning that the gradient produced by a fake image is scaled by 9.6^{-1}. This can also be thought of as adding a normalising coefficient α to $\mathrm{BCH}(p_\mathrm{t}) = -\alpha\log(p_\mathrm{t})$.

To further ease the problem of class imbalance, we consider FL, which was developed to help ease the problem of extreme data imbalance in dense object detection [25]. FL can be defined as:

$$\mathrm{FL}\,(p_\mathrm{t}) = -\alpha_\mathrm{t}\,(1-p_\mathrm{t})^\gamma \log\,(p_\mathrm{t})$$

Where α and γ are biased towards the imbalance class and a penalisation parameter; Notice: this loss is similar to BCH with a balancing factor. FL sees state-of-the-art results in one-shot object detection in images. We hypothesise that FL will aid in easing class imbalance compared to BCH. We run Hyperband(L) for 200 epochs each dataset to determine $\hat{L}$.

5.5 Optimiser

Once we have determined the optimal loss, $\hat{L}$, we will use this to determine the optimal optimiser, $\hat{O}$, given $\hat{L}$.

We experiment with three standard optimisers, Adam [18], Root Mean Square Propagation (RMSprop) [14], and Stochastic Gradient Descent [31]. All of these algorithms are gradient-based optimisers, meaning that they use the derivative of loss to infer the "direction" of optima. Adam and RMSprop utilise momentum as well as gradient, which allows them to escape local optima. None of these algorithms guarantee global optima, however, they do guarantee convergence to some local optima given enough iterations and appropriate learning rates. Similar to finding $\hat{L}$, we will use Hyperband with 300 epochs of exploration for each dataset and pick the best performing optimiser to determine $\hat{O}$.

5.6 Fine-tuning

At this point, we know the optimal loss, $\hat{L}$, and optimiser, $\hat{O}$, for our Xception Feature Extractor when our CCN is a single Dense unit, with Sigmoid activation (since it is the final binary classification layer). We then experiment with structure CCN = $\{(\text{Dense}(d)|\text{Dropout}(p))*\text{Dense}(1)\}$ meaning (Dense layer with d units OR Dropout layer with dropout probability p) repeated zero or more times followed by a Dense layer with a single unit

However, this implies that we could potentially have an infinite number of layers or units for CCN, which would be infeasible. To reduce the search space to a finite and discrete set, we consider the search space as follows: Eight layers maximum, $|\text{CNN}| = 8$. For layers one to eight: Dense layers will have d units, where $d \in D = [1024, 4096]$ with step size 256, i.e. $d \in \{1024, 1280, ..., 4096\}$. Dropout layers will have Dropout probability, p, where $p \in P = [0, 1]$ with step size 0.05, i.e. $p \in \{0, 0.05, ..., 1\}$. For the final layer: We must have a Dense layer with a single unit, Dense(1). In regular expression notation, this can be written as
$(\text{Dense}(d)|\text{Dropout}(p))\{0-8\}\text{Dense}(1)$.

The choice of eight layers is completely arbitrary but most CCNs in the literature tend to have eight or fewer layers [8] [30]. Discretisation of settings results in 678,724,137,931 total possible network architectures ($\sum_{n=0}^{8}(|D|+|P|)^n$). We then pass these discretised settings for the structure to Hyperband to determine the optimal structure, $\hat{S}$. Due to computational constraints, we run Hyperband(S) for 500 epochs as it would be infeasible to $6.79 \times 10^{11} \times 100$ epochs of exploration.

Tables outlining Hyperband results in Evaluation and Results have validation AUCs as their score. We should not pay much attention to this as the 500 epochs are a quota for exploration, and will not be allocated to just one candidate model. The flow of experiments can be seen in Figure 5.1. We then evaluate all models, with the same test sets for each given dataset. Each dataset contains the pre-processed version of the exact same videos. Due to the actor-isolation, we expect lower test AUCs compared to validation AUCs because of the unseen actors, this is ideal as, if we gain high test AUC, we know that the model has generalised well.

Chapter 6

Evaluation and Results

6.1 Untrained MesoInception4 Baselines

Dataset	*Random Frame (RF)*				*Frame Average (FA)*				*Frame Difference (FD)*			
Pre-Trained Weights	*Face2Face (F2F)*		*DeepFake (DF)*		*Face2Face (F2F)*		*DeepFake (DF)*		*Face2Face (F2F)*		*DeepFake (DF)*	
Model Name	*U-F2F-RF*		*U-DF-RF*		*U-F2F-FA*		*U-DF-FA*		*U-F2F-FD*		*U-DF-FD*	
Class	*Real*	*Fake*	*Real*	*Fake*	*Real*	*Fake*	*Real*	*Fake*	*Real*	*Fake*	*Real*	*Fake*
Precision	0.09	0.92	0.08	0.92	0.08	0.92	0.09	0.92	0.08	0.92	0.09	0.92
Recall	0.02	0.99	0.22	0.78	0.62	0.41	0.14	0.87	0.71	0.31	0.03	0.97
F1	0.03	0.95	0.12	0.85	0.15	0.56	0.11	0.90	0.15	0.47	0.05	0.94
Test AUC	0.496		0.492		0.510		0.501		0.510		0.492	

Table 6.1: **Untrained Baselines (U-F2F & U-DF)** classification report of all untrained out-of-the-box baselines, evaluated with both weights (F2F & DF) and all datasets

We look at six untrained out-of-the-box baseline models, for each of our three datasets and two loadable weights for MesoInception4. All models show poor performance on all datasets with AUCs close to 0.5 (as good as randomly guessing), as expected. This is due to the fact that these baselines are being evaluated on datasets very different from what they were trained on. Unusually, we find that the best (tied) performing untrained baseline was U-F2F evaluated on the FD and FA datasets, both reaching AUCs of 0.51.

Table 6.1 (RF Baselines left) shows that performance on the RF dataset is also poor. By comparing the real and fake metrics, we can see that the model tends to predict fake, far more often than real. This is expected and due to class imbalance and is a common trait among most models. Oddly, the RF dataset performs the worst compared to the other untrained baselines. With the lowest AUC of 0.492. Despite this, RF baselines yield marginally higher real class precision, over the other baselines.

Table 6.1 (FA Baselines middle) again shows poor performance, with a slightly better than random test AUC of 0.510 with F2F weights. We find that FA consistently performs better than RF, regardless of which weights were loaded; validating our hypothesis that frame averaging pre-processing may aid performance.

This is strange as these baselines were trained on a still-image dataset analogous to RF. One would think that the averaging of frames over one-second intervals may blur important details but these results show otherwise. Although F1 significantly improves compared to U-F2F-RF and U-DF-RF, F1 for the fake class has fallen to 0.56, implying that we are now misclassifying fakes far more often than before. If

we consider all baselines, we see that, excluding the RF dataset, baselines loaded with the F2F weights perform consistently better, with higher real class F1s and test AUCs, compared to baselines loaded with DF weights. This suggests that the models trained on the F2F dataset generalise better to our new datasets.

Considering Table 6.1 (FD Baselines right), we oddly find that baselines evaluated on the FD dataset loaded with F2F (U-F2F-FD), performs identically to its FA evaluated counterpart, both achieving the highest test AUC of 0.510.

This is still strange as we would assume that the FD dataset would be drastically different from the F2F dataset this baseline was originally trained on. FD, as seen in Figure 4.3 contains unnatural images far removed from the images found in any other DeepFake dataset, and yet U-F2F-FD is just as good as U-F2F-FA.

6.2 Feature Extractor MesoInception4 Baselines

Dataset	*Random Frame (RF)*		*Frame Average (FA)*		Frame Difference (FD)	
Model Name	*FE-RF*		*FE-FA*		*FE-FD*	
Class	*Real*	*Fake*	*Real*	*Fake*	*Real*	*Fake*
Precision	0.09	0.92	0.08	0.92	0.08	0.92
Recall	0.39	0.63	0.62	0.40	0.51	0.50
F1	0.14	0.75	0.15	0.56	0.14	0.65
Test AUC	0.503		0.505		0.498	

Table 6.2: **Feature Extractors (FE)** classification report of all feature extractor MesoInception F2F baselines, evaluated on all datasets

Due to the poor AUC, we opt to apply basic Transfer Learning to our baseline, regardless of the dataset; untrained baseline with F2F weights, U-F2F. We only train the last layer of U-F2F on all datasets to obtain a series of minimally trained feature extractor MesoInception4 networks loaded with F2F weights and evaluate.

Table 6.2 shows the results of the trained models on our test sets. We see that despite fine-tuning on the last layer, these models are not much better than our untrained baselines in terms of test AUC. Looking closer at the real class F1, there is a clear improvement with the highest real class F1 belonging to FE-FA. Here, the trend of FA models performing better than the others continues. However, FE-RF is now performing slightly better in comparison to FE-FD. Although a marginal improvement, FE-FA is indeed consistently yielding better results compared to FE-RF; U-F2F-FD was one of the best performing untrained baselines whereas FE-FD was the worst minimally trained baseline, therefore no clear conclusions for FE-FD can be made as of yet.

Figure 6.1 shows training history, notice how we did not train all models for the same number of epochs. This is due to the early stopping mechanism which will terminate training if the model shows 10 instances of decreasing validation AUC. We can see that all models show smooth training curves. However, the validation curves for all metrics are volatile, suggesting that FE models are having trouble finding generalisable optima.

Although FE-FA reached the highest test AUC (top right of Figure 6.1). FE-RF shows lower false positives (FP) at 156 versus FE-FA's 420. We care about FP as the model keeps predicting fake due to the severe class imbalance. If we get low FP and high AUC, this means the class imbalance problem is being solved.

FE-RF's validation loss is decreasing in tandem with train loss, whereas all other FE models do not show convergent validation losses. This means that FE-RF had little trouble adapting to our dataset, with all metrics smoothly converging. This makes sense as the RF dataset is the closest to the datasets used in MesoNet. In brief, FEs showed marginal gains over U-F2Fs, with FE-FA being the best on test AUC, although FE-RF showed better validation statistics implying more training may yield better results with this model. Despite these marginal gains over the untrained baselines, we find FA performs better than its counterparts, further implying that FA pre-processing has a positive impact on performance.

6.3 Determining Optimal Loss

Dataset	*Random Frame (RF)*	*Frame Average (FA)*	*Frame Difference (FD)*
Best Loss	Binary Cross-Entropy	Binary Cross-Entropy	Binary Cross-Entropy
Epochs of Exploration	200	200	200
Time Taken	20 Hours 14 Mins		
Score (Val-AUC)	0.621	0.678	0.666

Table 6.3: **Best Loss,** $\hat{L}$, determined by Hyperband($\{\mathsf{BCH}, \mathsf{FL}\}$)

Table 6.3 shows the results of running Hyperband for 200 epochs per dataset, with two choices for loss, Hyperband($\{$BCH, FL$\}$). All top ten models achieve a validation AUC of 0.62 or higher, regardless of the dataset. All top models prefer Binary Cross-Entropy, over Focal Loss. Examining closer, Focal Loss had no gain in performance compared to randomly guessing. Thus, $\hat{L} = \text{BCH}$.

Although not the metric we are focusing on, validation AUC is relatively high, being higher than any test AUCs achieved by untrained baselines (Table 6.1). We find a similar trend: RF slightly worse than FD, with FA being the winner in terms of validation AUC. This is promising but we can not make any concrete conclusions using validation AUC, as the model is indirectly "training" using the validation set [34]. We do not evaluate Hyperband models as they are too numerous and we are only using them to determine optimal hyperparameters.

6.4 Determining Optimal Optimiser

Dataset	*RF*	*FA*	*FD*
Best Optimiser	Adam	SGD	SGD
Epochs of Exploration	300	300	300
Time Taken	29 Hours 24 Mins		
Score (Val-AUC)	0.623	0.658	0.660

Table 6.4: **Best Optimiser,** $\hat{O}$, determined by Hyperband($\{$Adam, RMSprop, SGD$\}$) given $\hat{L} =$ BCH

Table 6.4 shows the results of using Hyperband for 300 epochs for each dataset, Hyperband($\{$Adam, RMSprop, SGD$\}$) given $\hat{L} =$ BCH. We see that two out of the three models for each dataset prefer SGD over Adam. SGD classically does not utilise momentum [31] whereas Adam [18] does. However, in the TensorFlow implementation, SGD incorporates a default initial momentum of 0.9 which decays. RMSprop was not present in any top tens for any of the datasets. In preliminary tests, RMSprop showed noisy convergence behaviour when considering training and validation loss; Adam also showed this trend, whereas SGD did not.

Due to the decaying momentum found in SGD, we hypothesise that the model was able to find a better optimum by avoiding bad local minima with higher loss via overshooting with momentum. All models use a standard constant learning rate of $1e-3$ as the literature suggests that this is optimal for a variety of problems [21] [5], hence we did not consider multiple learning rates in conjunction with our optimisers. Due to computational constraints, we only create a single model for our final evaluation. A vote of our top optimisers determines that the optimal optimiser is $\hat{O} =$ SGD.

6.5 Determining Optimal Structure

Table 6.5 shows the top models for each dataset for finding optimal structure using Hyperband((Dense(d)|Dropout(p))0-8) for 500 epochs, given $\hat{L} =$ BCH and $\hat{O} =$ SGD. All models show to prefer seven penultimate layers, i.e number of layers after Xception output and before final Dense unit, seen in Figure 5.1. RF preferred a CCN with a Dense layer size of 2816 followed by another Dense layer with 1280 units. This is strange as this means that the network is compressing the learned features in the layer before (similar to Autoencoders [4]), then the network prefers more units with dropout layers with 3328 or more dense units.

Both FA and FD preferred the exact same network. This may be due to the fact that we used the same random seed for all experiments, thus Hyperband considered the same order of "random" initialisations for structure. With the initial layer containing 2304 units, with dropout throughout the networks excluding the last layer leading to the final Dense unit. This initial layer is followed by an increased number of Dense units, 3328, then a constant number of 1280 dense units.

Dataset	*Random Frame (RF)*	*Frame Average (FA)*	*Frame Difference (FD)*
Penultimate Layers	7	7	7
Dense Units 0	2816	2304	2304
Dropout Probability 1	*N/A*	0.30	0.30
Dense Units 2	1280	3328	3328
Dropout Probability 3	0.10	0.15	0.15
Dense Units 4	3584	1280	1280
Dropout Probability 5	0.30	0.15	0.15
Dense Units 6	3328	1280	1280
Dropout Probability 7	0.1	*N/A*	*N/A*
Epochs of Exploration	500	500	500
Time Taken	50 Hours 27 Mins		
Score (Val-AUC)	0.641	0.667	0.651

Table 6.5: **Best Structure, $\hat{\text{CCN}}$**, determined by Hyperband((Dense(d)|Dropout(p)){0-7}Dense(1)) given $\hat{L} = \text{BCH}$ and $\hat{O} = \text{SGD}$. *N/A* indicates that the model chose not to use this layer.

All models preferred dropout probabilities of 0.3 or less. This is promising and consistent with the literature where higher values are rarely used [36]. All models prefer to take Xception's 2048 dimensional feature vector and pass it into a Dense layer first. Models had the option of using eight layers, however, they all chose to use seven instead. All models mostly preferred to have an alternating Dense and Dropout arrangement, suggesting that this arrangement was helpful for maximising validation AUC, and when a layer was not included, it was always a Dropout layer. Due to the fact that two out of our three best performing models per dataset choose identical parameters, we determine the optimal structure of our custom classifying network to be $\hat{\text{CCN}}$ = {Dense(2304), Dropout(0.3), Dense(3328), Dropout(0.15), Dense(1280), Dropout(0.15), Dense(1280), Dense(1)}.

6.6 Our Models

We now train a our custom classifying network determined by Hyperband, $\hat{L} = \text{BCH}$, $\hat{O} = \text{SGD}$ and $\hat{\text{CCN}}$ = {Dense(2304), Dropout(0.3), Dense(3328), Dropout(0.15), Dense(1280), Dropout(0.15), Dense(1280), Dense(1)} with the feature vector of Xception as input.

Dataset	*Random Frame (RF)*		*Frame Average (FA)*		*Frame Difference (FD)*	
Model Name	*OM-RF*		*OM-FA*		*OM-FD*	
Class	*Real*	*Fake*	*Real*	*Fake*	*Real*	*Fake*
Precision	0.08	0.92	0.08	0.92	0.08	0.92
Recall	0.71	0.31	0.53	0.48	0.78	0.23
F1	0.15	0.47	0.14	0.63	0.15	0.37
Test AUC	0.501		0.502		0.497	

Table 6.6: **Our Models (OM)** classification report of our final models, evaluated on all datasets

Table 6.6 shows that all models performed poorly, with OM-FA performing marginally better than OM-RF in terms of test AUC with OM-FD being the worst with sub 0.5

AUC. All of our models show high fake class precision at 0.92, with low fake class recall, this indicates that our models are still mostly predicting real to the detriment of fake class predictions. OM-FA achieves the highest cumulative F1 over both classes at 0.77, with OM-RF a second at 0.61 and OM-FD with the lowest at 0.51. These metrics show a clear improvement over our untrained baselines, however, they are marginally worse than our FE models.

Despite the class imbalance and low AUCs, confusion matrices (Table 6.7) show that OM-RF and OM-FA predicted real far more often than fake, whereas FA does the opposite. The fact that some of our models predicted real more often, even with the heavy imbalance towards fake, suggesting class imbalance seems to be circumvented here. Perhaps ensemble methods may yield better results as different models have picked on different features to distinguish DeepFakes. Ensembling is when we take the predictions of many different models and combine them (e.g. with a vote) to reach performance better than any single model [44].

	OM-RF Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	1136	464
True Fake	12431	5607
	OM-FA Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	352	1248
True Fake	3908	14130
	OM-FD Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	1253	347
True Fake	13811	4227

Table 6.7: **OM** Confusion Matrix for all our final models

Figure 6.2 shows metrics during training, we can clearly see that the early stopping mechanism has significantly hindered our performance as training metrics have not plateaued. Most of our models only trained to around 30 epochs before early stopping terminates them. This is somewhat of an unfair comparison as our models clearly did not converge to optimas; early stopping has caused some models to train for far more epochs compared to others, e.g. FE-FA models being trained for over 60 epochs which may explain its improved performance compared to other models.

Figure 6.2 also shows the same trend of noisy validation metrics which do not converge. All validation metrics of our model are significantly worse compared to FE models. Our models had 770 false positives and the lowest accuracy at 0.626 at the worst case, whereas FE models had 420 maximum false positives and the lowest accuracy 0.745, indicating that FE models are better than our models in every case. Our models show that FD pre-processing yielded no significant benefits on performance compared to RF, although throughout all experiments, we have seen that FA-trained models consistently beat RF-trained ones.

We are getting low AUCs close to 0.5 for all models, despite our training metrics showing much higher validation AUCs and accuracy, sometimes close to 0.9. We hypothesise this is due to two reasons, our actor-isolated test set and non-voting based prediction on videos. This test set is much harder than a typical test set as we took care to isolate 9 male and 9 female actors strictly for testing purposes. This was done to ensure that the models did not memorise the faces and scenarios of fake actors, making the test set harder and thus we achieve low AUCs. Another reason is the way we choose to evaluate our models. When we pre-process for a given dataset, we create around 13 images per video (since the interval, $k = 30$ and videos are on average 13 seconds). This naturally increases the amount of data the model has access to - more instances to train on but also more instances to get wrong, and without taking into account what video these images are originally from, we unfairly penalise the model, even if it is making meaningful predictions.

For example, in our setup, consider a real video. When it is pre-processed there will be 13 images corresponding to this video. This means that there are now 13 times more instances to potentially predict wrong. If the model incorrectly predicted that seven or more of these images were fake, a simple vote over the predictions per video would lead to a single (incorrect) verdict of fake and the model would only be penalised once, instead of six more times which is the case currently. This would also mean that creating more images (i.e. increasing k) would have no effect on the total number of labels to predict. We do not currently vote on images with respect to the original video's label as part of the evaluation, but we hypothesise that using this method would lead to far better performance. AUC curves and confusion matrices for all models discussed in the main body of this project can be seen in Appendices.

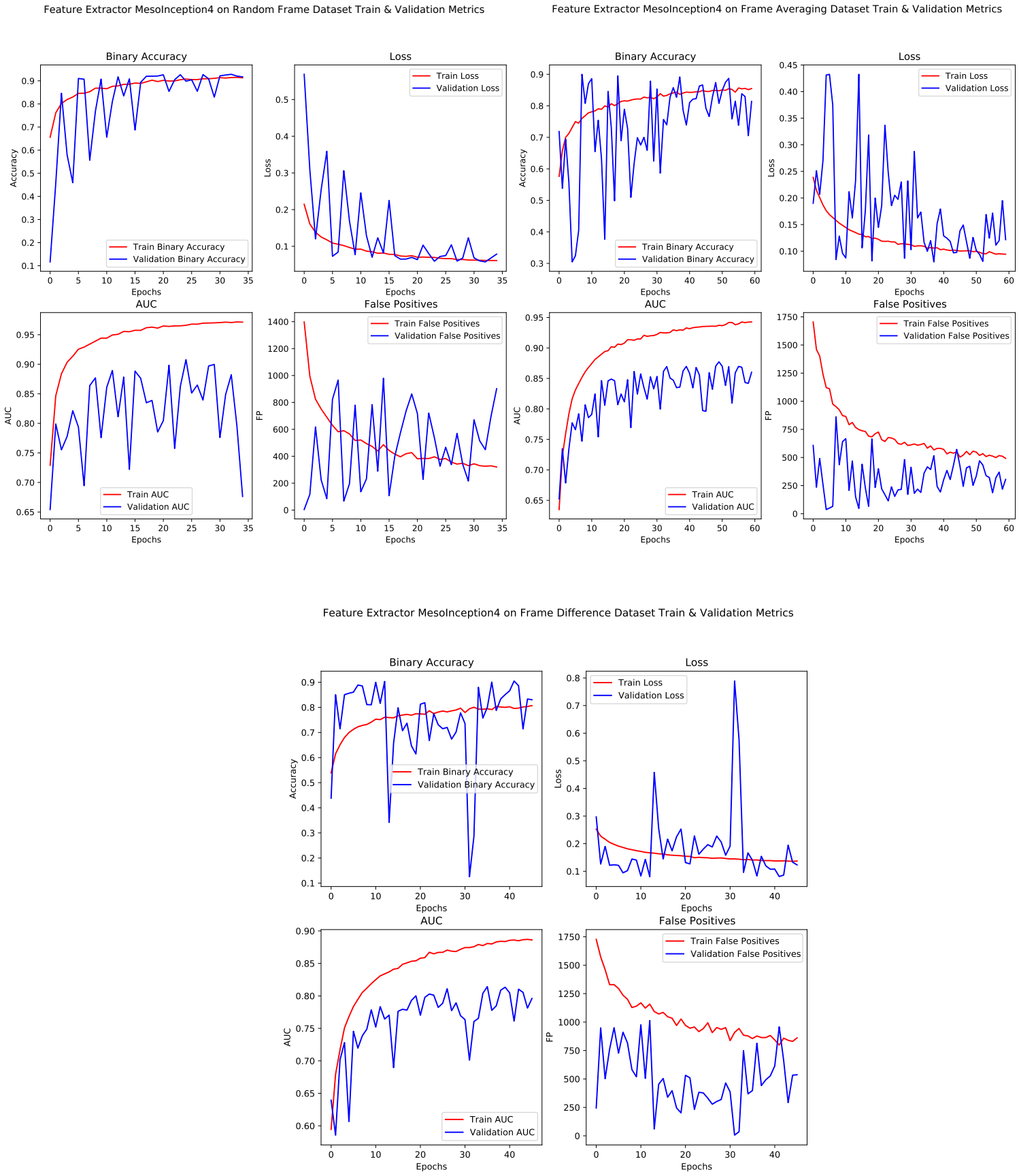

Figure 6.1: ***Feature Extractor MesoInception4 Baselines Training Metrics.***
Top Right: **FE-RF**, Validation Metrics: Loss=0.125, Acc=0.823, AUC=0.935, FP=156.
Top Left: **FE-FA**, Validation Metrics: Loss=0.134, Acc=0.806, AUC=0.861, FP=420.
Bottom: **FE-FD**, Validation Metrics: Loss=0.163, Acc=0.745, AUC=0.923, FP=121.

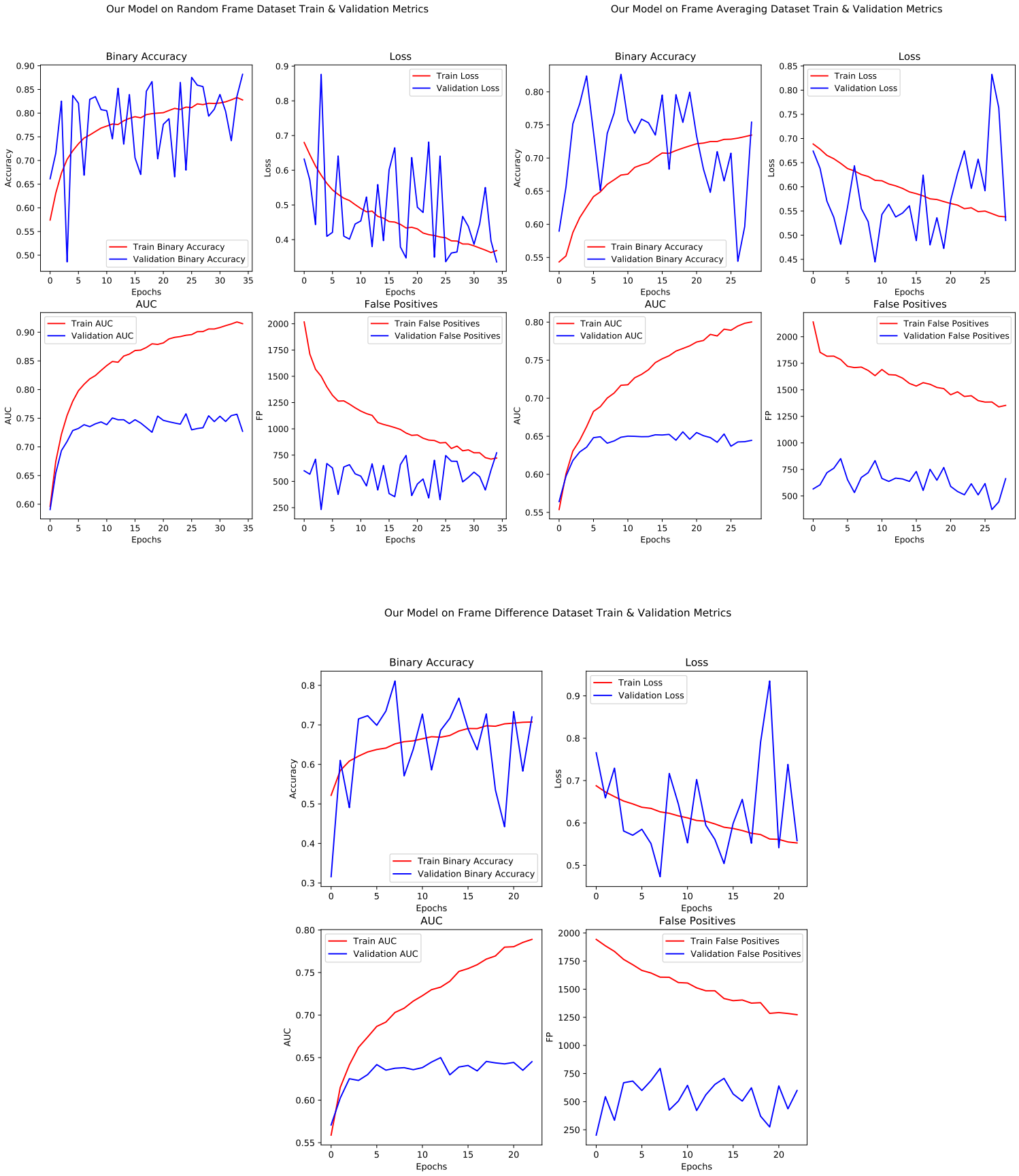

Figure 6.2: ***Our Models' Training Metrics.***
Top Right: **OM-RF**, Validation Metrics: Loss=0.728, Acc=0.626, AUC=0.805, FP=296.
Top Left: **OM-FA**, Validation Metrics: Loss=0.517, Acc=0.765, AUC=0.732, FP=770.
Bottom: **OM-FD**, Validation Metrics: Loss=0.585, Acc=0.690, AUC=0.737, FP=532.

Chapter 7

Discussion

7.1 Conclusion

We set out to apply pre-processing to the CelebDFv2 DeepFake video dataset in hopes to leverage temporal information in an image for use with CNNs. We created three datasets via frame averaging (FA), frame differencing (FD), and randomly selecting frames (RF). Two of the three datasets, FA and FD, encapsulated time information in some meaningful way; FA does this via blurs and FD via colour intensity, whereas RF did not capture any and can be thought of as a snapshot in time. We do this in hopes of picking up on the idiosyncrasies present in many DeepFake videos.

We consider a series of untrained out-of-the-box MesoInception4 baselines (U-F2F and U-DF), which showed poor performance on all datasets. Oddly, untrained baselines evaluated on the FA dataset yielded the best performance despite the fact they are trained on datasets analogous to RF, suggesting these models gained a marginal boost in performance due to this pre-processing.

We then applied basic Transfer Learning on the best performing untrained baseline (U-F2F), by treating this network as a feature extractor and only training the last layer to fine-tune to our datasets, to arrive at a series of minimally trained feature extractors (FE).

We found that these FE models performed the best compared to any other networks, with marginally higher AUCs compared to untrained models. This continues the same trend of FA-trained models performing slightly better than RF-trained models.

We then proposed our own model which used the penultimate output of Xception, with ImageNet weights, as a feature vector and passed it to our own custom classifying network (CCN). We used Hyperband to find optimal settings for our CCN including loss, optimiser, and network structure given our datasets.

We then used these settings to create our own models (OM). We found that these models perform slightly better than our untrained baselines but worse than our FE models. This poor performance was most likely due to the varying training epochs among all networks. Early stopping made it so that some models trained more than others, which may have skewed our results. The training metrics for OMs showed they had not fully converged to optima when we evaluated them. Despite this, we again found that FA models performed better than RF, and FD showed slightly worse performance.

We hypothesise that FE models gained performance is due not only to the extra epochs of training but the fact that this network had already been trained on another DeepFake dataset (F2F dataset), whereas OMs had been trained from ImageNet, which does not have as many faces as F2F.

We see poor test performance among all models, despite high validation performance and we speculate this is due to the harder actor-isolated test set and the means of evaluation. To conclude, we found that all models trained or evaluated on FA showed slightly higher performance compared to RF-trained models, whereas FD-trained models showed no such improvements.

7.2 Future Work

At the start of this project, we wanted to combine FA and FD pre-processing methods, by overlaying the averaged frame on top of the differenced one. This would result in the averaged frame plus highlighted areas which would show where the movement had occurred. We opted not to do this so that we could assess each method individually. A natural place to improve upon this work is to combine pre-processing methods and investigate whether this further improves performance.

In addition, We see that class imbalance played a major role in hindering our performance. In the future, we would want to explore methods of easing class imbalance in more detail. We chose the naive approach of just creating more data, and although this can sometimes work, it also affected our test set (more instances to possibly get wrong). We would like to specifically explore ensembling and other voting-style methods, which could potentially combine the predictions of many different models to yield better performance compared to any single model.

We would like a constant number of epochs per model and perhaps more complex training setups. We hypothesise that we can achieve a much better test AUC by finding a better optimum. This might include trying a series of random seeds, using learning rate decay, and/or more aggressive data augmentation. In preliminary results, we heavily experimented with automatically reducing learning rates based on the loss over a given interval, this allows us to take smaller steps in the loss landscape when close to an optimum. Despite success with this, we chose not to include it in our model as we are focusing on determining whether the pre-processing had any tangible effects on performance.

Bibliography

[1] Keras applications. https://keras.io/api/applications/. (Accessed on 03/28/2021).

[2] Module: tf — tensorflow core v2.4.1. https://www.tensorflow.org/api_docs/python/tf/. (Accessed on 04/11/2021).

[3] Darius Afchar, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Mesonet: a compact facial video forgery detection network. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7. IEEE, 2018.

[4] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021.

[5] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[6] Dmitri Bitouk, Neeraj Kumar, Samreen Dhillon, Peter Belhumeur, and Shree K Nayar. Face swapping: automatically replacing faces in photographs. In *ACM SIGGRAPH 2008 papers*, pages 1–8. 2008.

[7] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[8] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017.

[9] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[10] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) dataset, 2020.

[11] Gleb Esman. Splunk and tensorflow for security: Catching the fraudster with behavior biometrics — splunk, April 2017. (Accessed on 02/19/2021).

[12] Adam Geitgey. Github - ageitgey/face_recognition: The world's simplest facial recognition api for python and the command line. https://github.com/ageitgey/face_recognition. (Accessed on 03/09/2021).

[13] Javier Hernandez-Ortega, Ruben Tolosana, Julian Fierrez, and Aythami Morales. Deepfakeson-phys: Deepfakes detection based on heart rate estimation, 2020.

[14] Geoffrey Hinton. lecture_slides_lec6.pdf. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. (Accessed on 03/29/2021).

[15] Justin M Johnson and Taghi M Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):1–54, 2019.

[16] Mahmoud Kalash, Mrigank Rochan, Noman Mohammed, Neil DB Bruce, Yang Wang, and Farkhund Iqbal. Malware classification with deep convolutional neural networks. In *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)*, pages 1–5. IEEE, 2018.

[17] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[21] Suki Lau. Learning rate schedules and adaptive learning rate methods for deep learning — by suki lau — towards data science. (Accessed on 04/09/2021).

[22] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[23] YD Li, ZB Hao, and Hang Lei. Survey of convolutional neural network. *Journal of Computer Applications*, 36(9):2508–2515, 2016.

[24] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. Celeb-df: A large-scale challenging dataset for deepfake forensics. In *IEEE Conference on Computer Vision and Patten Recognition (CVPR)*, 2020.

[25] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[26] Mathworks. Ssim, 2020.

[27] Yisroel Mirsky and Wenke Lee. The creation and detection of deepfakes: A survey. *ACM Computing Surveys (CSUR)*, 54(1):1–41, 2021.

[28] Zohaib Mushtaq, Shun-Feng Su, and Quoc-Viet Tran. Spectral images based environmental sound classification using cnn with meaningful data augmentation. *Applied Acoustics*, 172:107581, 2021.

[29] Thanh Thi Nguyen, Cuong M Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, and Saeid Nahavandi. Deep learning for deepfakes creation and detection: A survey. *arXiv preprint arXiv:1909.11573*, 2019.

[30] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[31] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[32] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics: A large-scale video dataset for forgery detection in human faces. *arXiv preprint arXiv:1803.09179*, 2018.

[33] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*, pages 1041–1044, Orlando, FL, USA, Nov. 2014.

[34] Tarang Shah. Train, validation and test sets. `https://tarangshah.com/blog/2017-12-03/train-validation-and-test-sets/`, 2017. (Accessed on 04/11/2021).

[35] Simranjeet Singh, Rajneesh Sharma, and Alan F. Smeaton. Using gans to synthesise minimum training data for deepfake generation, 2020.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[37] Isabel Bush Stanford. Measuring heart rate from video. 2016.

[38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[39] Shahroz Tariq, Sangyup Lee, and Simon S. Woo. A convolutional lstm based residual network for deepfake video detection, 2020.

[40] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2387–2395, 2016.

[41] Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales, and Javier Ortega-Garcia. Deepfakes and beyond: A survey of face manipulation and fake detection. *Information Fusion*, 64:131–148, 2020.

[42] Cristian Vaccari and Andrew Chadwick. Deepfakes and disinformation: Exploring the impact of synthetic political video on deception, uncertainty, and trust in news. *Social Media+ Society*, 6(1):2056305120903408, 2020.

[43] Daniel Vlasic, Matthew Brand, Hanspeter Pfister, and Jovan Popovic. Face transfer with multilinear models. In *ACM SIGGRAPH 2006 Courses*, pages 24–es. 2006.

[44] Shuo Wang and Xin Yao. Diversity analysis on imbalanced data sets by using ensemble models. In *2009 IEEE symposium on computational intelligence and data mining*, pages 324–331. IEEE, 2009.

[45] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

Chapter 8

Appendices

Notice

This Chapter does not intend to infringe on the 40-page limit. Everything included from here onwards is supplementary and not vital to the understanding of this project.

Training Setup and Other Details

Most if not all training and evaluation was conducted on Google Colab which utilises Nvidia K80 GPUs. Google Colab presented a lot of problems due to server time-outs, thus with great difficulty, we managed to borrow a GTX 970. Despite this compute, we ran into many issues with TensorFlow and GTX 900 series cards. If we were to do this project again, we would highly recommend using PyTorch. For all training and testing purposes, we used a random seed of 1337, this seed was used for both NumPy, TensorFlow, model initialisations, and all data generators, anything that had an element of randomness. This ensured that we would get the same results no matter the run. We used a batch size of 64 for all models, as it was the same as the MesoNet papers [3]. No data augmentation was applied, as we deemed it unnecessary due to the size of our dataset. Default TensorFlow learning rates ($1e-3$), optimiser initialisation, etc. were used throughout and can be viewed on the TensorFlow API page [2]. All layers in our CCN used ReLU activation, except for the final Dense unit which used Sigmoid activation.

Throughout this project we use the term "preliminary results", this term refers to tests conducted early on in the project. These results were not included in this project as they were lost; due to malicious crypto-mining software present on our laptop, resulting in data loss. All the code can be found at the GitHub:
`https://github.com/Sakib56/MInf-DeepfakeDetection-FrameDifferencing`.
This GitHub will remain private until the day of project submission (12/04/2021), after this date this repository will be made public. However, the dataset will not be included in this repository as re-distributions and derivations of CelebDFv2 are not permitted [24]. If you would like access to the dataset, please do not hesitate to contact me at s1759855@ed.ac.uk.

Software, Libraries and Dependencies

Python 3.8.6 packaged by conda-forge
dlib 19.21.1
face-recognition 1.3.0
face-recognition-models 0.3.0
h5py 2.10.0
Keras 2.4.3
Keras-Preprocessing 1.1.2
keras-tuner 1.0.2
matplotlib 3.3.3
numpy 1.19.5
opencv-python 4.5.1.48
Pillow 7.2.0
scikit-learn 0.24.1
tensorboard 2.4.1
tensorboard-plugin-wit 1.8.0
tensorflow 2.4.1
tensorflow-addons 0.12.1
tensorflow-estimator 2.4.0
tqdm 4.56.0

Full Results of All Models

	U-DF-FA Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	229	1371
True Fake	2302	15736

	U-F2F-FD Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	627	973
True Fake	6739	11299

	U-DF-FD Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	53	1547
True Fake	562	17476

	FE-FA Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	994	606
True Fake	10785	7253

	U-DF-RF Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	352	1248
True Fake	3908	14130

	FE-FD Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	810	790
True Fake	8960	9078

	U-F2F-FA Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	988	612
True Fake	10715	7323

	FE-RF Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	621	979
True Fake	6674	11364

	U-F2F-RF Confusion Matrix	
	Predicted Real	*Predicted Fake*
True Real	25	1575
True Fake	240	17798

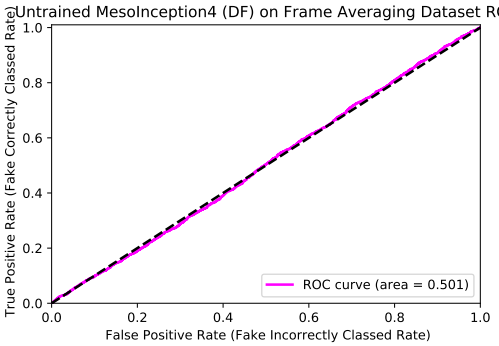

Figure 8.1: **U-DF-FA** AUC Curve

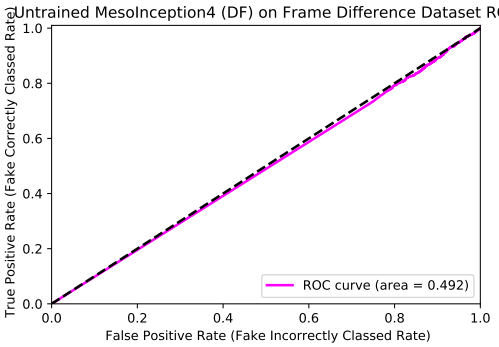

Figure 8.2: **U-DF-FD** AUC Curve

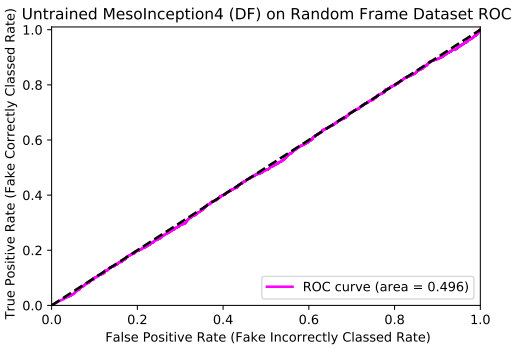

Figure 8.3: **U-DF-RF** AUC Curve

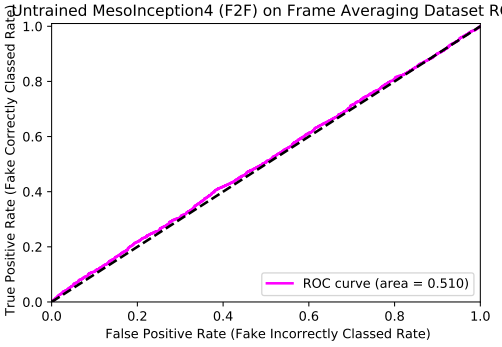

Figure 8.4: **U-F2F-FA** AUC Curve

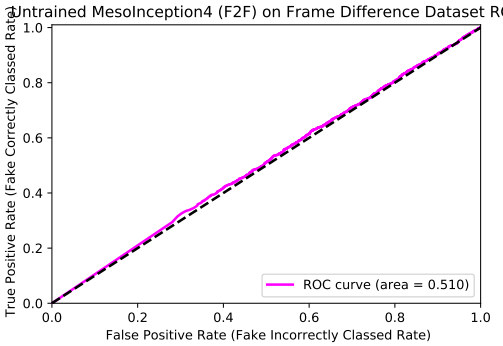

Figure 8.5: **U-F2F-FD** AUC Curve

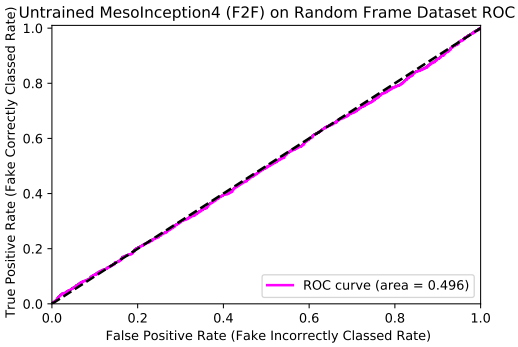

Figure 8.6: **U-F2F-RF** AUC Curve

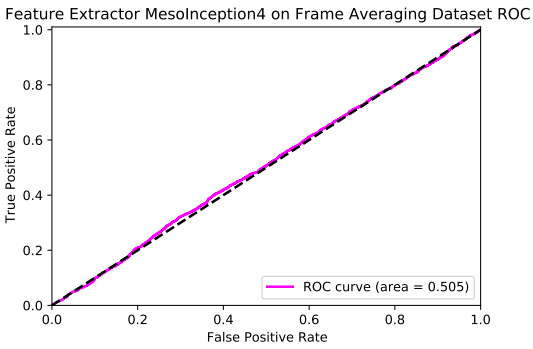

Figure 8.7: **FE-FA** AUC Curve

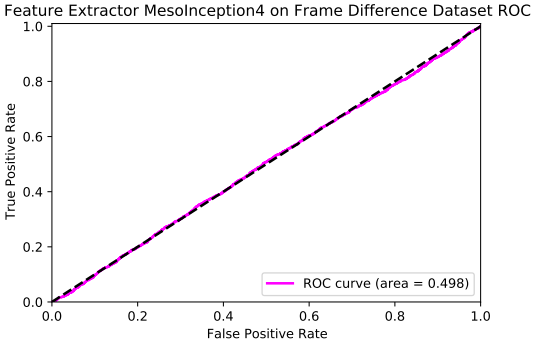

Figure 8.8: **FE-FD** AUC Curve

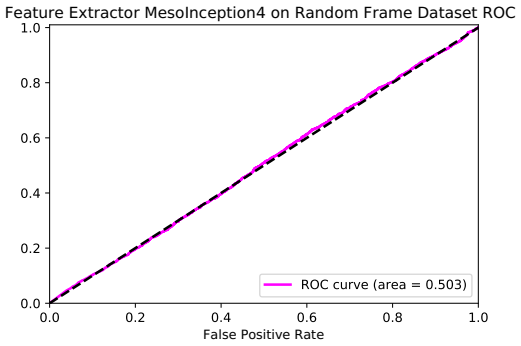

Figure 8.9: **FE-RF** AUC Curve

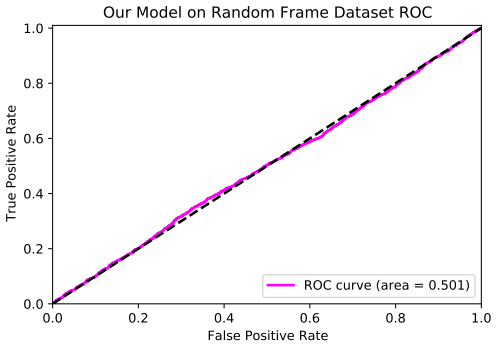

Figure 8.10: **OM-RF** AUC Curve

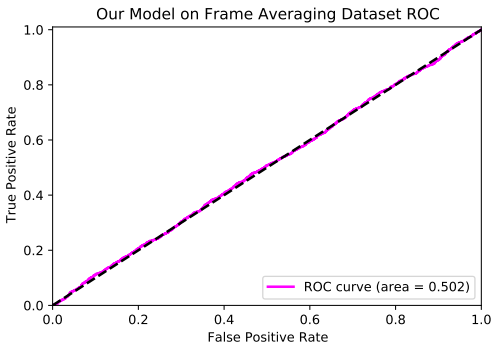

Figure 8.11: **OM-FA** AUC Curve

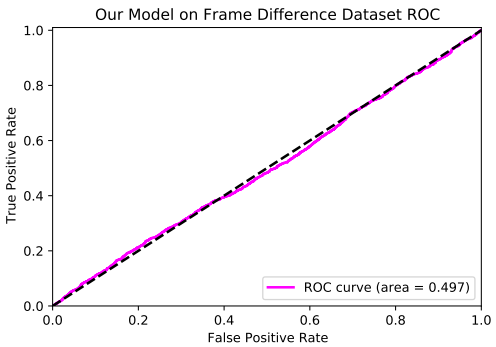

Figure 8.12: **OM-FD** AUC Curve