# A.I. Vision for Social Good: Deepfake Video Detection

*Sakib Ahamed*

**MInf Project (Part 2) Report**
Master of Informatics
School of Informatics
University of Edinburgh
2022

# Abstract

Many state-of-the-art Deepfake video detectors use Convolutional Long Short-Term Memory-based networks (ConvLSTM) as they take video as direct input, and can inherently capture spatio-temporal information within the video and use it for detection. We opt not to use this approach and instead utilise dataset pre-processing protocols in order to transform video to image input for use with image classification algorithms, inspired by the successful field of image classification for non-image tasks. We consider three architectures, including Khalid et al.'s OC-FakeDect1 (Convolutional Variational Autoencoder) trained only on real images which detects Deepfakes by treating them as anomalies. We compare this to traditional binary image classification approaches including Afchar et al.'s MesoNet (Convolutional Neural Network) and Dosovitskiy et al.'s Vision Transformers. We train these architectures using several pre-processing protocols, evaluation methods, and datasets in order to deduce which settings work best and when. We offer deep analysis of all combinations of settings, models and datasets, which may aid the field of Deepfake detection. This results in us creating a Vision Transformer model which approaches state-of-the-art performance on the CelebDFv2 dataset, with a completely fair testset with no train-test split actor overlap. This was achieved via utilising our pre-processing protocols and evaluation methods; transforming videos to images via averaging over frames or randomly selecting frames, and predicting on each frame or averaging predictions over all frames for each video. We test this model on several unseen Deepfake datasets to showcase its superior generalisability compared to baselines.

# Acknowledgements

I would like to acknowledge my supervisor, James Garforth, for all the insightful conversations, which helped me tremendously with how to best improve my project. I would also like to acknowledge my friends for helping me cope throughout my final year, 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

The term "Deepfake" is a portmanteau of the words "Deep Learning" and "Fake". Deepfakes refer to a type of synthetic media created via Deep Learning (the predominant method to generate this media [26]), where a person in an existing image or video is replaced with someone else's likeness using A.I. algorithms. Deepfakes are often used to create fake celebrity porn or to spread false information. Deepfake technology has significantly improved to the point that they can be indistinguishable from real videos while requiring significantly fewer data to create [40]. This means the prevalence and malicious usage of ultra-realistic Deepfakes will increase, in turn increasing the likelihood of political and social unrest.

When we can not discriminate genuine videos from forged ones, we opt for a fight AI with AI approach, and use Deepfake detectors. This work concerns itself with such algorithms.

## 1.1 Motivation

Due to the immense damage that could be inflicted on society via the misuse of easily accessible, ultra-realistic Deepfake video, we are motivated to create detection algorithms. Many state-of-the-art (SOTA) Deepfake detectors take sequences of images as input to leverage spatio-temporal correlations to make their predictions [34] [40]. This is similar to how we as humans can tell if a video is fake (flickering/artifacts in facial expression over time). Most models employed for detection do not take video as direct input, and models which do take extremely long to train, making them uneconomical (e.g. Convolutional Long-Short Term Memory Networks [34]). Therefore, we also investigate a variety of pre-processing and evaluation techniques that we believe may encode temporal information.

However, since none of our models encode for time directly, we opt to use pre-processing to add this information in image format. The field of image classification for non-image tasks indicates that it is possible to map data points from one modality (videos) to another (images) via pre-processing to perform classification. We choose to do this by averaging frames over subsections of video and we will compare this to randomly selecting a single frame over the same subsection. Averaging frames should result in a blurring effect which is a clear indicator of movement (although not the direction of movement Figure 4.2).

We hypothesise this will aid the model's performance by allowing it to pick up on artifacts present in fake videos. Furthermore, we consider two evaluation methods: predicting overall frames from a single video and averaging to generate a single overall prediction for the video, and compare this to the traditional approach of predicting each frame across all videos independently.

### 1.1.1 MInf Parts 1 & 2

In part 1 of this project, we explored the same problem of Deepfake detection. After reviewing the literature, we found the area of image classification for non-image datasets promising, e.g. sound classification using spectral signatures [33] or malware classification with machine code bitmaps [12]. We hypothesised this approach could be leveraged for detection and we devised a novel video pre-processing protocol, frame differencing (FD), and compared it against others with baselines. FD consisted of computing the difference of frames over subsections. We compare FD to the averaging and randomly pre-processing mentioned prior.

We applied this pre-processing to the ultra-realistic CelebDFv2 dataset [23] and trained Afchar et al.'s MesoInception4 (an existing commonly used Deepfake detector baseline) via Inductive Transfer Learning. We performed extensive hyperparameter tuning of all trained models using Hyperband [22], a reinforcement learning-based (RL) approach outperforming classical Bayesian Optimisation. We compared our three models fine-tuned on our three pre-processed datasets with their untrained counterparts and found poor test AUCs among all models, i.e. all six models had AUCs of $\sim 0.5$, equivalent to randomly guessing. Indicating no clear improvement in performance using our novel method. We concluded part 1 by deducing that the dataset we chose to use was too difficult for our models to correctly classify. However, on review of our code from part 1 (at the start of part 2), we found that we were calculating AUC incorrectly and our novel method did indeed yield good results, with AUC far above 0.5. Despite this, we chose not to pursue FD and instead focus on new approaches.

We build on top of part 1 and explore Convolutional Neural Networks (CNN), Vision Transformers (ViT), and Convolutional Variational Autoencoders (ConvVAE) architectures with a set of varied datasets, pre-processing protocols, and evaluation methods for the purposes of Deepfake video detection. We consider the current SOTA in image classification, Vision Transformers [6] and employ Transfer Learning to fine-tune it for the task of Deepfake detection. We also consider Khalid et al.'s ConvVAE which exclusively utilises the real class to determine the legitimacy [14], circumventing the cat-and-mouse of Deepfake generation and detection. We compare our results against Afchar et al.'s MesoNet [1], four CNNs especially trained for detection as baselines.

Please note that our only overlap from part 1 is that we use the same AF and RF pre-processing protocols and the same baseline. However, in this work, we update our pre-processing protocols based on the work done in part 1 and consider several baselines and models. Thus, all trained models, experiments, results, etc. shown in this project are new to this work.

### 1.1.2 Contributions

We would like to draw the reader's attention to the following novel contributions in this paper:

• Created a generalisable Vision Transformer model approaching SOTA tested on several datasets.
  ◦ Determined generalisability of models via testing on various unseen datasets.

• Created a completely fair dataset, eliminating the possibility of *cheating* by models encoding individual's identities across training and test splits.
  ◦ Reinforcing our claims of generalisability and SOTA performance across models.

• Created an intelligent and efficient method of isolating target faces and cropping them, while excluding unwanted third party faces within the same subsections of frames.

• Comparisons, analysis, and statistical tests of dataset pre-processing, evaluation methods, and architectures on several datasets not yet considered by the literature.
  ◦ e.g. Optimal dataset pre-processing and evaluation method determined per architectures per dataset.

• Trained and tested several existing Deepfake detection models and compared them against existing baselines.

## 1.2 Research Objective

In this work, our aim is to create several Deepfake detection systems, which take pre-processed images (from video) as input and output a prediction of real or fake. We would like to deduce which architecture is best, as well as which architecture is most generalisable to new unseen Deepfake algorithms. We would also like to find which pre-processing and evaluation methods yield the best improvement in performance.

We hypothesise that ViT models will be the best overall architecture due to their recent success in computer vision [6] and that averaging frames and averaging over predictions will be the best pre-processing protocol and evaluation method, respectively. We believe this as they intuitively encode the most temporal information. In order to investigate this, we will train our architecture on a variety of different Deepfake datasets in conjunction with pre-processing techniques. We also investigate the efficacy of one-class ConvVAE based detectors that only use real images to train, and compare this approach with the binary image classification approach employed by baselines and ViT.

We conduct a cross-product style test on all models, datasets, pre-processing protocols, and evaluation methods to conclusively determine the best settings for detection. This methodology should allow us to provide analysis not yet seen within the literature e.g. what settings work optimally and when. Our main objective is to create and find the most generalisable model architecture for detecting unseen Deepfakes.

# Chapter 2

# Background

## 2.1 Deep Learning

Deep Learning (DL) is a subset of Machine Learning (ML) that is used to create models that can learn from data to make predictions. DL leverages Artificial Neural Networks (ANN), which can fit any continuous function to an arbitrary degree of accuracy with only a single (potentially infinite) hidden layer [5]. However, this ANN would be very prone to overfitting as the training examples would be remembered by the weights [7]. ”Deep” in DL refers to the depth of the network, and it has been shown that deeper networks learn slower and thus are less prone to overfitting [7], with each subsequent layer learning features at increasing levels of abstraction [17]. DL learns from data via vector operations proceeded by non-linear differentiable activation functions. The network approximates the underlying function described by these data, such that we can compute the gradient of the error function w.r.t. the weights, removing the need for hand-crafted features and allowing for gradient-based optimisation methods e.g. Stochastic Gradient Descent.

DL has been shown to be very successful in a variety of tasks, including image classification [18], object detection [30], and semantic segmentation [25]; these were achieved using CNNs. This suggests that these algorithms can pick up on the complexities of natural images and perhaps could be leveraged for detection, not just the creation, of Deepfakes.

## 2.2 Types of Deepfake Videos

There are three types of facial manipulation algorithms.

- **Fully Synthesised**: All components of the image are fake, including the person’s identity. These are often produced by Generative Adversarial Networks (GANs) with prominent examples from Nvidia’s StyleGAN [13]; examples can be seen at This Person Does Not Exist.

- **Facial Expression Transfer**: Making a target’s face digitally mimic a source actor’s emotion and facial expressions and other minor features (e.g. hairstyle) [43], while preserving identity [39].

- **Identity Transfer**: Digitally transplanting a source individual's entire face onto a target individual, where both source and target individuals are real people [2] (face-swapping).

The majority of fully synthesised Deepfake algorithms only produce images, with malicious actors having little control over the specifics of the images (only the ability to vary latent features to change basic attributes such as hair colour [13]) making it difficult to do real harm.

Facial expression transferring Deepfakes has more potential to do harm as they can be used to create fake videos. However, this technique requires the malicious actors to use a pre-existing video of the target, which could be traced back and proven fake, foiling any attempts to do harm.

Identity transferring Deepfakes do not have any of these issues and are arguably the most dangerous, thus this work concerns itself with this type of Deepfake. The majority of Deepfakes present online are of this type, mostly used for pornography ($\sim 96\%$ [37]), usually face-swapped with celebrities. This is due to the large amount of facial data available for these individuals. However, the prevalence of Deepfakes using everyday people's faces (via social media) will increase as SOTA Deepfake generators use fewer data to generate convincing fakes [35]. Recent applications of this type of Deepfake often use actors who closely resemble the source individual's face as the target video. This is followed by applying identity transferring Deepfake algorithms which eliminate the possibility of the video being traced back. An example of this is Channel 4's Her Majesty's 2020 Christmas (Deepfake) Address.

## 2.3 Convolutional Neural Network

Convolutional Neural Networks (CNN) are a special type of neural network which rose to prominence with AlexNet in 2012 [17]. This model achieved top-five accuracy of 84.7% test accuracy ($\sim$11% greater than the next runner-up) on the ImageNet challenge, a 1000-class image classification problem.

CNNs classify images by using convolution operations which filter and reduce inputs by a set of learnable kernels. Convolutional and Pooling layers extract spatial correlations in the image and reduce the size of input that is fed into Dense layers. Successive convolutions allow for feature extraction over patches of the images and the Pooling layer rejoins the patches for further feature extraction. These features are then passed into a final Dense network for classification.

## 2.4 Variational Autoencoders

Autoencoders can be split into two neural networks, an Encoder, and a Decoder. The encoder transforms the input into a latent representation, i.e. intermediate features of the input. The task of the decoder is to re-create the input using only the latent representation, trained to minimise reconstruction error (difference between the input and the output). The latent representation is typically smaller than the input to force the model to learn dense hidden representations. Autoencoders are analogous to (lossy)

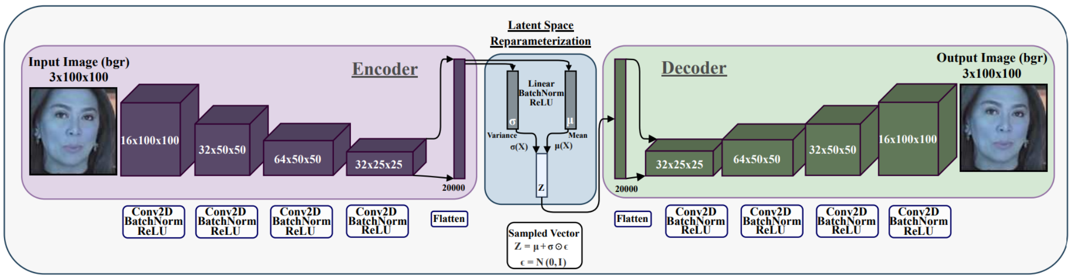


Figure 2.1: OC-FakeDect1 (ConvVAE) architecture [14]. Reconstruction Score is defined as $\mathrm{RMSE}(x,x')$, where $x$ is the Encoder input and $x'$ is the Decoder output.

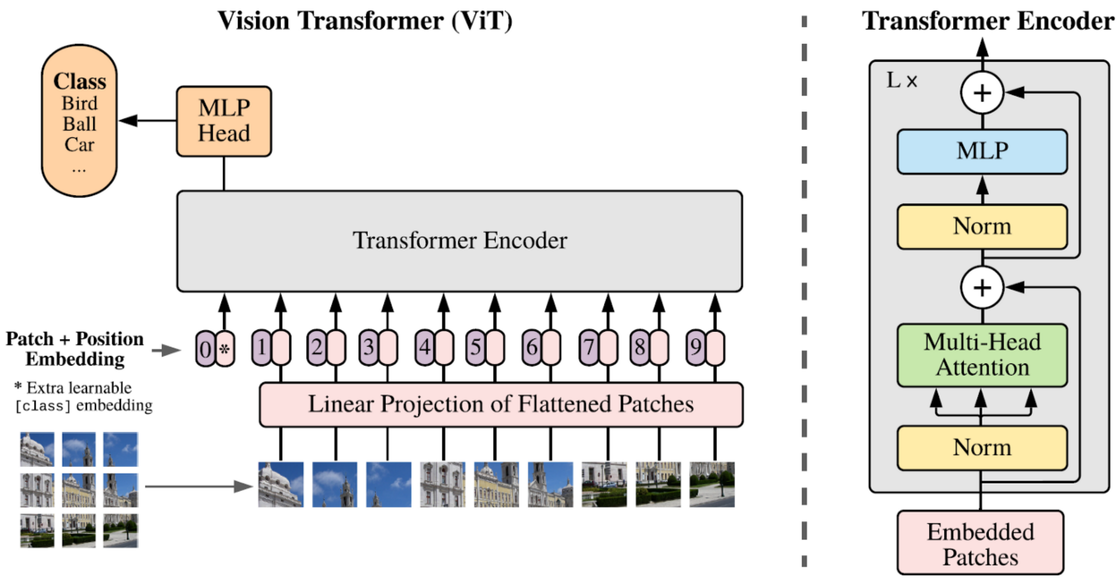


Figure 2.2: Vision Transformer architecture [6], each $16 \times 16$ image patch is treated similarly to a token in NLP.

data compression. They are often used to learn features [14], denoise [10], or generate new data [27].

Variational Autoencoders (VAE) build on top of this concept by adding a latent space. Instead of the encoder outputting the latent representations for the input, we output a latent sample using the (learned) latent mean and log-variance. This makes it possible to interpolate between points in the latent space and generate new data. Variational Autoencoders are trained by minimising the reconstruction error and the KL Divergence between the latent space and a prior distribution. Convolutional Variational Autoencoders (ConvVAE) further this by incorporating traits of CNNs so image inputs can be used efficiently, an example of a ConvVAE can be seen in Figure 2.1.

## 2.5 Vision Transformer

Dosovitskiy et al. introduced a Vision Transformer-based architecture for image recognition (ViT) [6]. This architecture achieved SOTA performance on multiple bench-

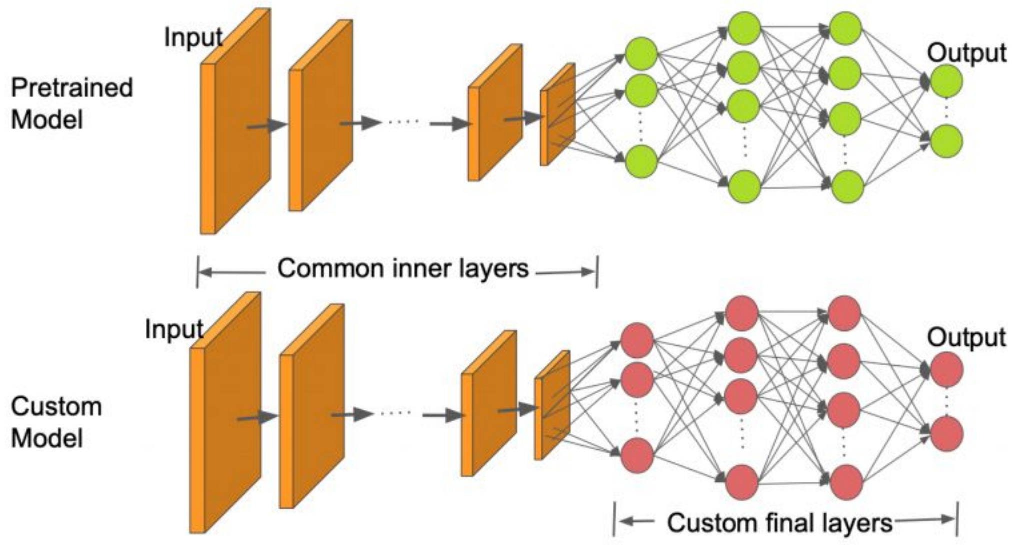


Figure 2.3: An example of Inductive Transfer Learning, with the addition of custom final layers for fine-tuning (used in our ViT models).

marks including ImageNet. Derivatives of this architecture are even used in Tesla's autopilot for their camera-input [38]. ViT splits the input image into patches (analogous to tokens in NLP). These patches are linearly projected into lower dimensions (similar to PCA). Positional embeddings (i.e. which part of the image the patch came from) are concatenated, this is followed by feeding the input into the standard transformer encoder outlined in Vaswani et al.'s seminal paper *"Attention is all you need"* [41]. This encoding step is performed several times using stacked Transformers. The output of this is then fed into a multi-layer perceptron (Dense layers) for classification, as seen in Figure 2.2. ViTs are pre-trained using a large supervised dataset and then fine-tuned on the specific classification task, in the same fashion as Transductive Transfer Learning.

ViT architecture leverages Attention, i.e. relationships between pairs of input patches. In computer vision, (Self-)Attention plays the role of capturing spatial information across the image. This is similar to convolutions, but with more flexibility to capture global trends across the image, not just local ones (Figure 2.5). ViTs take substantially more data to train from scratch (hence the Transductive Transfer Learning applied in its training process) compared to CNNs. This is due to ViTs having little inclination to fit the idiosyncrasies of particular training examples, which is what allows CNNs to quickly fit datasets, at the cost of generalisability [29]. This lack of bias means ViTs take longer to train (from scratch), but absorb global correlations over the dataset aiding in generalisability. Due to this specific feature of ViT, we consider this architecture for Deepfake detection and only perform Inductive Transfer Learning to fine-tune the Dense layers of Dosovitskiy et al.'s ViT-B16.

## 2.6 Transfer Learning

Transfer Learning is the process of using a pre-trained model already trained on a source domain and re-purposing it for a new target domain. This allows for the transfer of knowledge via learned features common to both domains which can be leveraged for more efficient training and better performance given the same compute. Without Transfer Learning, it would be infeasible to complete this project due to the costs entailed by training large computer vision models from scratch [28].

There are two types of Transfer Learning, Inductive and Transductive. Inductive Transfer Learning is when the model is trained on the source domain and then applied to the target domain. Transductive Transfer Learning is training on both the source and target domains simultaneously.

We choose to use Inductive Transfer Learning for our model for computational reasons. We freeze the pre-trained model's weights and remove the final classification layer, substituting it for our own. Meaning we only train on these few (Dense) layers proceeding the penultimate layer of the pre-trained model (Figure 2.3). We only do this for one architecture, ViT, which is discussed in-depth in Section 6.3.

## 2.7 Deepfake Video Detection & Related Work

### 2.7.1 MesoNets (CNN)

Afchar et al. introduced a set of 4 lightweight CNN-based Deepfake detection models in 2018 [1], achieving an AUC of 0.98, and 0.95 on the DeepFake (DF) and Face2Face (F2F) datasets, respectively. They also consider a variety of compression rates and showcase the robustness of their algorithms in the wild. In their work, they consider 2 model architectures trained on 2 datasets to produce 4 models, Meso4 and MesoInception trained on DF and F2F. Afchar et al. also provide all of their code to create, initialise and run their models. They include an example dataset, DFDB, which has already been pre-processed such that all the data points are extracted faces in image format via the Viola-Jones algorithm [42]. Pre-trained weights for all models can also be acquired, making testing on this baseline simple and ensure that our comparisons are valid, with no bugs which could skew results.

Despite these models no longer being SOTA, we deem it important to compare against them as they are a commonly used baseline among most papers in the field.

### 2.7.2 OC-FakeDect1 (ConvVAE)

Deepfake detection algorithms usually require real and fake data points, the latter of which is ever-changing as video forgery algorithms improve, making it difficult to create up-to-date detectors.

Khalid et al. introduced a ConvVAE for Deepfake detection [14] using only one class. Their approach only uses the real class to determine legitimacy by leveraging the reconstruction error disparity between real (which the network was trained on) and fake faces. Khalid et al. define reconstruction score as the root mean squared error (RMSE) between the input and generated output. Since the ConvVAE was only

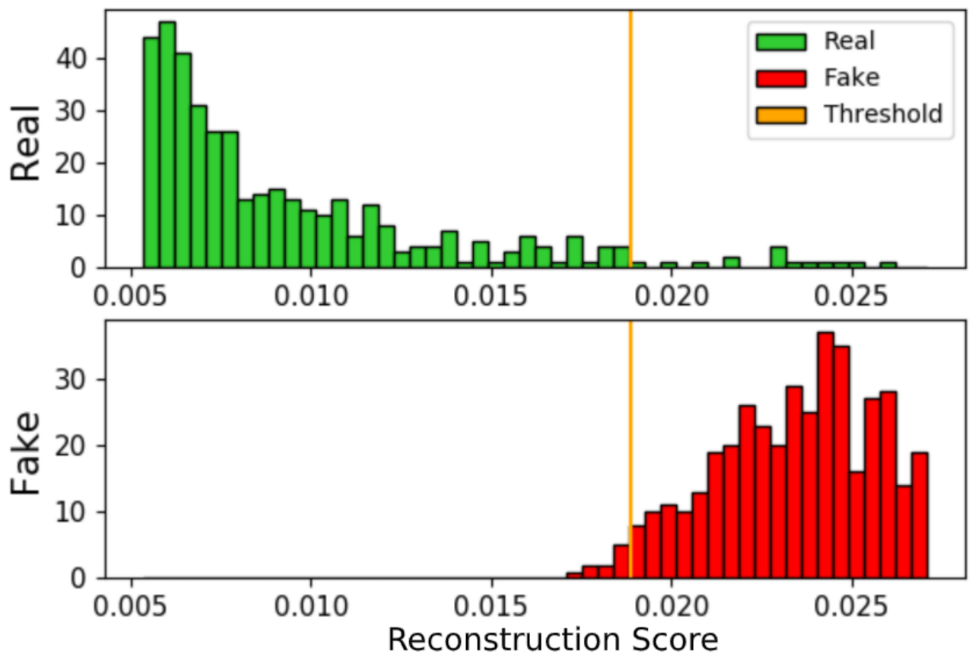


Figure 2.4: Thresholding technique on Reconstruction Score (RMSE) used by Khalid et al.'s models to classify [14].

trained on the real class, the model generates worse images when given fake faces as input (distributional shift). Thus, they can quantify the quality of generated images using RMSE and classify them using a scalar threshold on this metric. An example of this thresholding can be seen in Figure 2.4.

They define two architectures OC-FakeDect1 and the SOTA OC-FakeDect2. We only consider OC-FakeDect1 (Figure 2.1) as we had difficulty implementing OC-FakeDect2 and OC-FakeDect1 yields only marginally worse performance compared to their SOTA model, and apply a series of novel pre-processing, evaluation methods, and datasets to this architecture.

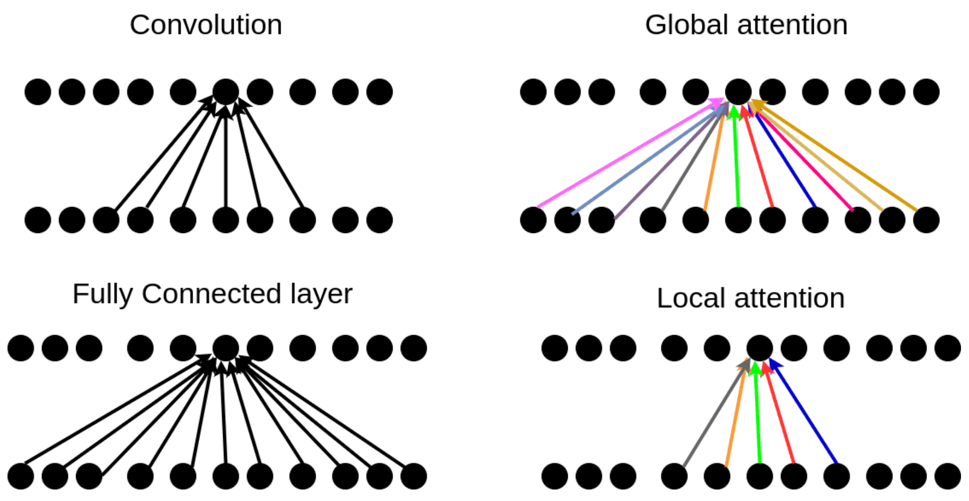


Figure 2.5: Diagram of Attention compared to Dense, Convolutional and Attention layers. The black lines indicate weights which update via an optimiser e.g. SGD, whereas the colourful lines indicate the Attention weights which change w.r.t. every input.

# Chapter 3

# Datasets

## 3.1 CelebDFv2

CelebDFv2 (CDFv2) [23] is the second generation of SOTA Deepfake video dataset using celebrities as source and target actors. Released 11-2019, it contains 590 (225,400 frames) and 5,639 (2,116,800 frames) of genuine and manipulated videos sequences, respectively.

There are 62 celebrities of various genders and ethnicities, who have been Deepfaked among each other, based on gender, resulting in an average of 9.6 fake videos for every real video. Both FF++ (Section 3.2) and CDFv2 have the advantage of offering ground truths for all fake videos. That is, every fake video has a corresponding real video. These ground truths make it possible for us to conduct a fair test, whereby we can isolate actors so that they only appear in the testset, removing the possibility of models remembering faces across train-test splits (see Section 4.1 for more details).

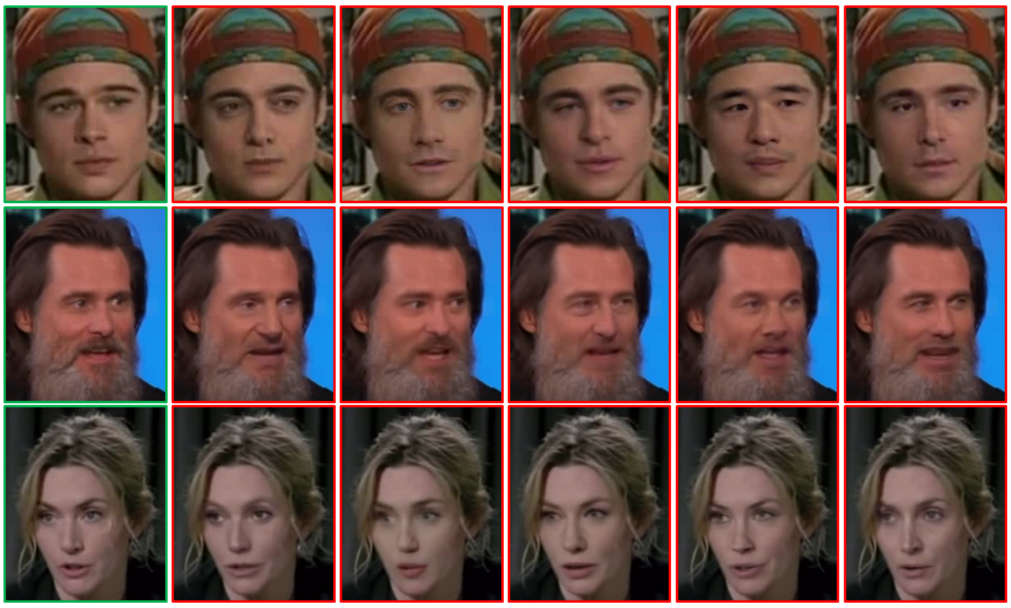


Figure 3.1: **CelebDFv2** [23] where the green boxes are real images and red boxes are their corresponding fakes.

This dataset provides the most convincing Deepfakes, with CDFv2 showing the highest Mask Structural Similarity Index Measure score [23] compared to its counterparts; indicating far superior quality compared to the other datasets mentioned. When we conducted visual inspections on the forged videos from each dataset, we could often immediately determine if a video was legitimate. However, with CDFv2 we found it difficult in many cases to do this. This further reinforced the value of this dataset.

## 3.2 FaceForensics++

FaceForensics++ (FF++) is a dataset collection [32] containing $1,000$ video sequences altered using four methods: Face2Face (F2F), DeepFakes (DF), FaceSwap (FS), NeuralTextures (NT). This dataset collection was released 01-2019 using 977 real videos gathered from genuine YouTube videos at a resolution of $256 \times 256$, with manipulation methods applied, resulting in a total of $509,900$ video frames per class.

In 2020 FF++ was improved with the addition of a new manipulation method, FaceShifter (FSHFT). This method is able to generate higher quality forged videos in comparison to the previous methods. In addition, FSHFT deals with facial occlusions using a second synthesis stage consisting of a Heuristic Error Acknowledging Refinement Network (HEAR-Net) [21]. This dataset mostly comprises frontal faces with minimal obstruction, allowing for best-case video forgeries which may aid in the generalisability of models trained with this dataset.

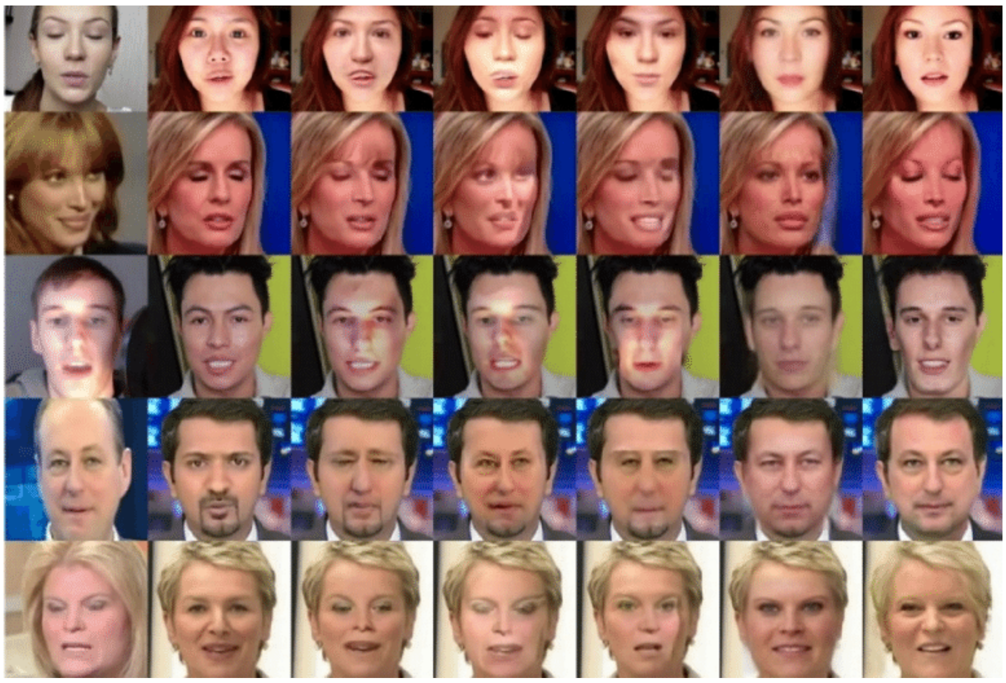


Figure 3.2: **FaceForensics++** [32] where left-most column are source faces and all other faces are Deepfake algorithms used in FF++.

| **Dataset** | **Year** | **Overall Number of Real Videos** | **Overall Number of Fake Videos** | **Faces Extracted Per Second (30 Frames)** | **Real Images After Pre-Processing** | **Fake Images After Pre-Processing** |
|---|---|---|---|---|---|---|
| CDFv2 | 2020 | 590 | 5,639 | 2 | 7227 | 67901 |
| FSHFT | 2020 | *1389*** | 1000 | 2 | 26138 | 16465 |
| F2F | 2019 | *1389*** | 1000 | 2 | 26138 | 16449 |
| DF | 2019 | *1389*** | 1000 | 2 | 26138 | 16444 |
| FS | 2019 | *1389*** | 1000 | 2 | 26138 | 13032 |
| NT | 2019 | *1389*** | 1000 | 2 | 26138 | 13025 |
| DFDB | 2019 | - | - | - | *11509** | *8000** |

Table 3.1: * DFDB was not pre-processed by us, we obtained it in image format (we can only pre-process video). ** Originally FF++ used 977 real videos to create 1000 fake videos (per forgery algorithm), but as of 2021, they use 1389 to the same number of fakes. $\{$FSHFT, F2F, DF, FS, NT$\} \in$ FF++. Please see Chapter 4 for details on what pre-processing refers to.

## 3.3 DeepFake Database

Afchar et al.'s original MesoNet [1] provided a download link for the dataset used, as well as pre-trained weights for each MesoNet as discussed in Subsection 2.7.1. The downloadable dataset, DeepFake Database (DFDB), provided is already pre-processed meaning that each data point is a `jpg` image and not an `mp4` video like all the other datasets mentioned in this paper.

For this reason, we cannot conduct our own pre-processing (as there are no videos to do this on) and thus we can not create averaged and random frame versions of this dataset as well as perform running average evaluation on our models. Despite these drawbacks, we deem it important to compare against this dataset as it is the one used by Afchar et al. and it gives us a good reference point for comparison between models.

The major ethical concern among all these datasets is that the individuals in these datasets did not consent to their faces being used for Deepfakes, let alone for others to benefit via academic research using their identities. These datasets contain faces of celebrities and individuals who chose to upload their faces onto YouTube. Furthermore, all of these datasets required us to sign up through academic portals and disclose our intent of usage before gaining access. We do not create any new Deepfakes as part of this project, only using the existing ones found online.

Despite these concerns, we choose to reluctantly carry on, as the majority of the literature has used these datasets. For the purposes of fair comparison and to minimise the number of people used in the field of Deepfake detection, we choose to use these datasets over newer ”in the wild” datasets [45].

# Chapter 4

# Dataset Pre-Processing Protocol & Evaluation Methods

This chapter focuses on the pre-processing protocols used to convert video datasets to image datasets; all 9 models used in this paper take images as input. In short, we create images by averaging the frames over a subsection of video or selecting a random frame from the same subsection to obtain images used for training and prediction. We also cover the different evaluation methods employed on these pre-processed datasets. This means either using a single image as a data point and predicting, or predicting on all frames from a given video and averaging over them to obtain a single prediction for the video.

## 4.1 Train-Test Actor Isolation

As mentioned in Section 3.1 both CDFv2 and FF++ have ground-truth videos, allowing us to isolate certain individuals for train and test purposes only. We do this in order to ensure that our models cannot learn actors' faces across train-test splits. CDFv2 has a consistent video file naming scheme (target actor + source actor + scene number). Whereas FF++ has an inconsistent naming scheme. Without a consistent naming scheme, it is impossible to isolate actors in this way. For this reason, we could only perform actor isolation on CDFv2.

We do this to keep the test as fair as possible, meaning our claims for generalisability later in this work are more concrete compared to other papers. We know our architectures will remember faces since many models in the field of ML remember data points instead of learning [36]. Overfitting itself is caused by models encoding training examples in their weights (remembering), which means they do not generalise well to unseen data.

In essence, we are making this dataset more difficult compared to the literature, and thus claims of SOTA performance are more reliable. We isolate 9 male and female actors from CDFv2, with these 18 actors used exclusively for test purposes only and the other 44 being used exclusively for training. This results in an approximately 30%: 70% train test split. To the best of our knowledge, train-test actor isolation has not been considered in the field.

## 4.2 Pre-Processing

We only have the ability to pre-process video datasets. Our motivation for doing this comes from numerous works where image classification CNNs are employed for non-image datasets via careful pre-processing. We do this in hopes to encode temporal information within our time-independent models.

Our pre-processing pipeline consists of extracting all frames from a video, followed by extracting a single face from each frame. Each video is at 30fps, and we choose to pre-process faces every half a second. We group $k = 15$ non-overlapping frames and either average over the group to produce an **Averaged Frame (AF)**, or select a **Random Frame (RF)** from the group. In this case, a group is $k$ consecutive (cropped) detected faces.

Doing this has the issue of inconsistent image sizes within the group (due to cropping), meaning we can not average over the group without dead zones. We fixed this in MInf part 1 by only considering the largest bounding box which contains all faces in the group of size $k$. However, this worked poorly in the edge case where multiple faces are detected in the scene, say at the edge of the frame, the largest bounding box over the group may encompass the entire frame.

We remedy this in part 2, by keeping track of the average face location within the group and only consider the largest bounding box which is $b$ pixels away from the centroid. We arbitrarily determined $b$ by considering the worst offenders of this edge case and found that $b$ equal to the 75th percentile of the most common face distance among all distances over the group worked best. That is, we use the $T_{70}$ of the interquartile range of face distances; i.e. sort all distances from the centroid and use the $75^{\text{th}}$ percentile for the new largest bounding box over the group to determine when to crop, and discard all frames which all outside this range. This gave us the best balance between not accidentally including unwanted faces while keeping tight face crops, which is important as model input sizes could be as low as $100 \times 100$ (OC-FakeDect1). This means that if a new face is detected within the scene, we only focus on the most common actor. This works well as all scenes among all datasets tend to be interview-style videos with minimal movement. A clear example of this can be seen in Figure 4.1 where $k = 4$ and using only frames that fall into $T_{70}$ of face distances from the average face location results in isolation of the target face (with only a single discarded target face crop).

Notice in Table 3.1 that datasets from FF++ have 26,138 real frames after pre-processing, whereas fakes have approximately 16,000 frames. This was because FF++ datasets do not use the entire video sequence to create the forged videos. The quality of Deepfakes among FF++ datasets seems to vary the number of fake images produced after pre-processing. This indicates our face extraction system did not always find the same faces, despite all FF++ videos being derived from the same real videos. We hypothesise this is due to the CNN we use to extract faces from video frames, which was trained to detect (real) human faces and segment them [8]. Since this CNN was trained to detect real faces, it would struggle to pick up on the unconvincing faces which are so poor in quality that the CNN does not deem them a face. As mentioned prior, FSHFT was a new addition to the FF++ dataset and Table 3.1 shows that this was the dataset the CNN extracted the most faces for, a testament to its realism.

## 4.3 Evaluation Methods

This pre-processing has allowed us to convert each video into a sequence of images, in such a way that we can recover which video each image was from (via filename). This allows us to vary the way we make predictions at inference time. We try two methods of evaluation. **Standard Evaluation (SE)**, which is where we predict every single image in our testset and predict as normal, treating every image independently from the last. We also try **Running Average Evaluation (RAE)** where we average over the predictions of images from a given video to provide a single prediction per video. Thus, RAE requires our pre-processing, doing RF or AF.

For example, say we are evaluating 2 videos, one real and one fake, that yielded 10 images each after pre-processing was applied. SE would independently predict on all 20 images to provide 20 predictions, one for each image. Whereas RAE would average the predictions over the 10 images from each video to provide 2 final predictions. We hypothesise that RAE will yield better performance compared to SE (which the majority of the literature use), as RAE may have the ability to pick up on the idiosyncrasies of fake videos; which tend to showcase flickering at the edges of faces.

```
def classify(
      preprocessed_images_from_video: list,
      model: (ViT | Meso4 | MesoInception | OCFakeDect1),
      evaluation_method: str
    ) -> list:
    """
    ====> Psuedocode for performing RAE/SE <====
    """
    # * `preprocessed_images_from_video` is all the pre-processed images
    # from a given (single) video

    # * `model` is the model being considered

    predictions = []
    for image in preprocessed_images_from_video:
        # Append model's prediction on image to predictions list
        predictions += [ model.predict(image) ]

        if evaluation_method == "Running Average Evaluation":
            # Average the predictions over images from the given video
            predictions = [ sum(predictions) / len(predictions) ]

    # Either len(predictions) == 1 or
    # len(predictions) == len(preprocessed_images_from_video)
    return predictions
```

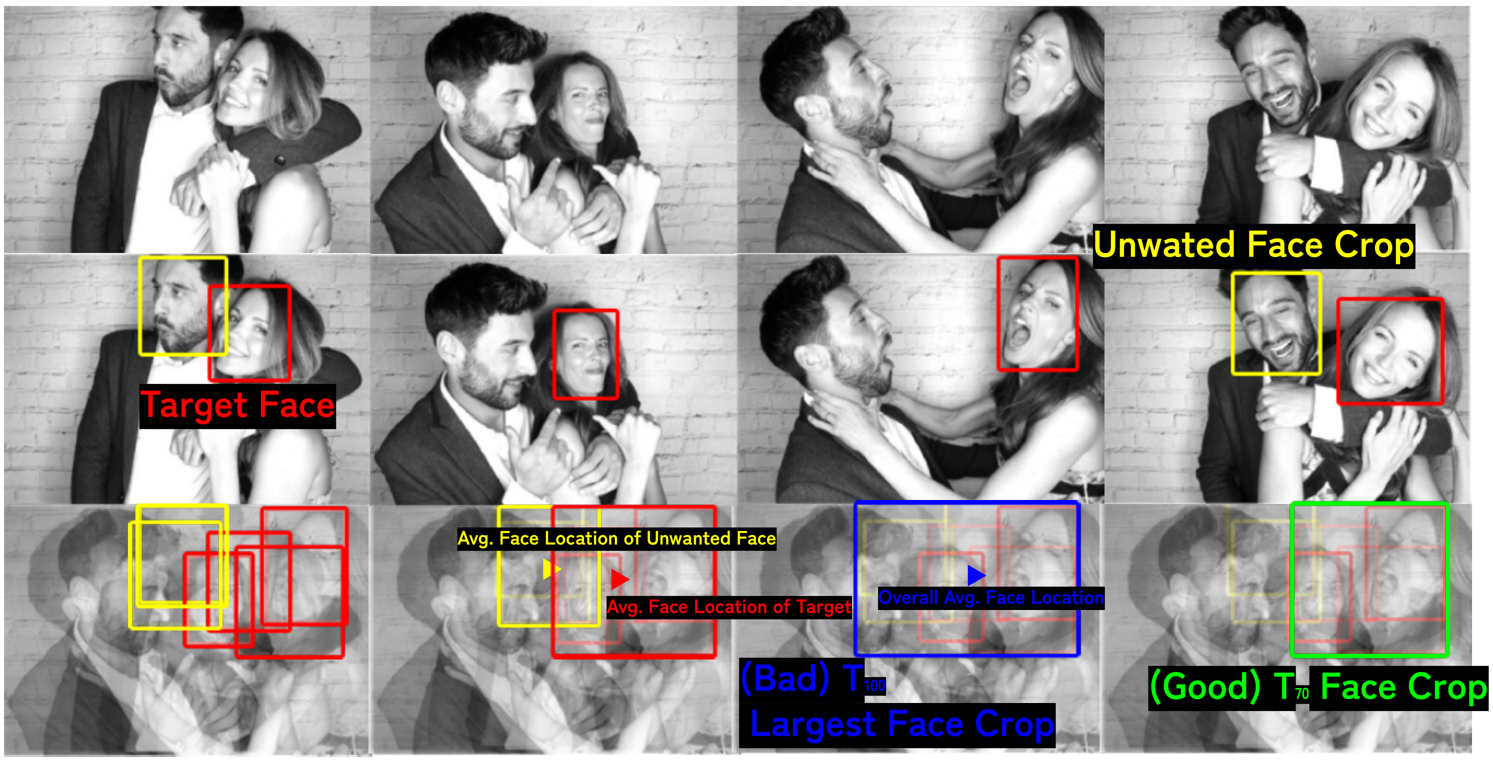

Unwated Face Crop
Target Face
Avg. Face Location of Unwanted Face
Avg. Face Location of Target
Overall Avg. Face Location
(Bad) T100
Largest Face Crop
(Good) T70 Face Crop

Figure 4.1: Face cropping protocol, using $T_{70}$ of the interquartile range of face distances from the overall average face location in order to isolate the target face (red), and exclude unwanted detected faces (yellow). This results in a good face crop containing only the target face (green) compared to the largest bounding box approach used in MInf Part 1 (blue).

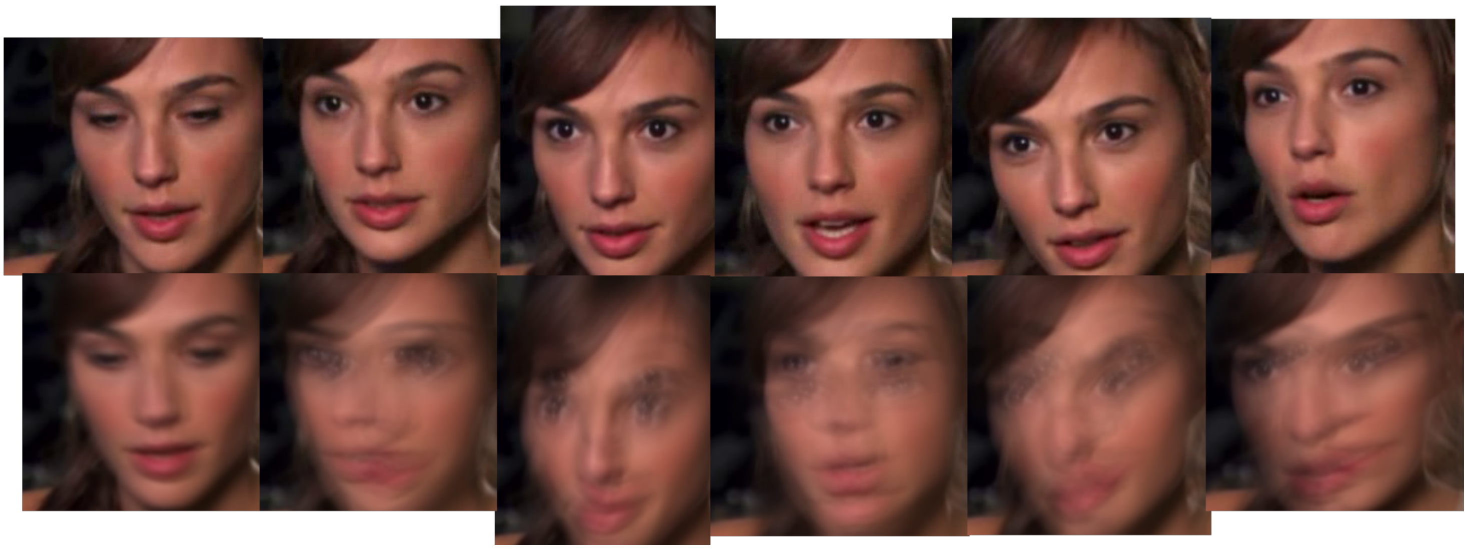


Figure 4.2: Examples of RF (upper row) and AF (lower row) pre-processed images. Notice how we as humans can use AF to infer movement, but not direction of movement in time.

# Chapter 5

# Methodology

In this chapter, we discuss the methodology of our experiments, specific models, and their settings used in this work. In total, we consider 9 models:

- 4 Baseline models which we do not train, only evaluate.
  ```
  [Meso4 with DF pre-trained weights,
   Meso4 with F2F pre-trained weights,
   MesoInception with DF pre-trained weights,
   MesoInception with F2F pre-trained weights]
  ```
- 2 Vision Transformer models which we exclusively train on CDFv2
  ```
  [ViT fine-tuned on CDFv2 pre-processed with RF,
   ViT fine-tuned on CDFv2 pre-processed with AF]
  ```
- 3 OC-FakeDect1 models which we train on CDFv2 and DFDB
  ```
  [OC-FakeDect1 trained on CDFv2 pre-processed with RF,
   OC-FakeDect1 trained on CDFv2 pre-processed with AF,
   OC-FakeDect1 trained on DFDB]
  ```

Each of these 9 models is tested 25 times, resulting in 225 total experiments. This is because there are 7 datasets in total, 6 of which were obtained in video format, and thus they could be pre-processed with the protocols discussed in Section 4.2 (RF, AF) into 12 datasets. These 12 datasets can then be evaluated image by image (SE) or as a video (RAE) to obtain 24 evaluation sets plus the extra dataset which was already in image format, DFDB. Recall, that we can only perform RAE when we can perform RF or AF, this is discussed in depth in Section 4.3.

$25 = ((6 \times 2 \times 2) + 1)$ = ((Video Datasets $\times$ Pre-Processing Protocols $\times$ Evaluation Methods) + Image Dataset). Where:
Video Datasets = `[CDFv2, F2F, DF, FSHFT, FS, NT]`
Pre-Processing Protocols = `[Random Frame (RF), Average Frame (AF)]`
Image Dataset = `[DFDB]`
Evaluation Methods = `[Standard Evaluation (SE),`
`Running Average Evaluation (RAE)]`

After these 9 models are trained, we test them on all testsets, essentially conducting

Table 5.1: **Left Table:** Model descriptions. Includes the model architecture (Model), training set (Trainset), and its associated pre-processing protocol (Trainset Pre-Processing). Rows where Trainset Pre-Processing contain a dash (-) indicate that we could not conduct any pre-processing (DFDB). Trainsets with asterisks (*) refer to the pre-trained weights provided by Afchar et al.

| **Description of All Models** | | | **Description of All Test Sets** | |
|---|---|---|---|---|
| **Model** | **Trainset** | **Trainset Pre-Processing** | **Testset** | **Testset Pre-Processing** |
| MesoInception | DF* | - | CDFv2 | AF |
| MesoInception | F2F* | - | CDFv2 | RF |
| Meso4 | DF* | - | DF | AF |
| Meso4 | F2F* | - | DF | RF |
| ViT | CDFv2 | RF | DFDB | - |
| ViT | CDFv2 | AF | F2F | AF |
| OC-FakeDect1 | DFDB | - | F2F | RF |
| OC-FakeDect1 | CDFv2 | RF | FS | AF |
| OC-FakeDect1 | CDFv2 | AF | FS | RF |
| | | | FSHFT | AF |
| | | | FSHFT | RF |
| | | | NT | AF |
| | | | NT | RF |

Table 5.2: **Right Table**: Testset descriptions. Includes testset and its associated pre-processing protocol (Testset Pre-Processing). DFDB does not have a pre-processing protocol as we obtained it with prior pre-processing (DFDB already came as still images). Thus, testsets are (`[CDFv2, F2F, DF, FSHFT, FS, NT]` × `[AF, RF]) + [DFDB]`

a cross-product of models, datasets, pre-processing, and evaluation methods where applicable. Table 5.1 and Table 5.2 showcases all models and datasets (with pre-processing protocols but not their evaluation methods). Notice that in Table 5.2 there are only 13 testsets when in reality there are 25 when we include the evaluation method (SE, RAE).

Of these 9 models, 4 are MesoNet baselines which we do not train, only test. The other 5 are the ViT and OC-FakeDect1 models outlined above.

## 5.1 Models

The following section covers the experiment settings for each of our architectures. We try our best to mimic the settings outlined by Afchar et al. (for ViT models) and Khalid et al. (for OC-FakeDect1) when appropriate. We do this to ensure that the test is as fair as possible, however, we could not always keep to this protocol. In hindsight, we believe we may have hindered our model's performance by doing this but we still manage to create models approaching SOTA.

### 5.1.1 Untrained Baselines: Meso4 & MesoInception

**MesoNets were *not trained at all*, as pre-trained weights were provided.** We use Meso4 & MesoInception as untrained baselines. These models come pre-trained with two sets of weights for the DF and F2F datasets (from FF++). In addition, they also provide an image dataset called DFDB which we use as a testset. However, since DFDB has already been pre-processed into images, we can not easily recover the source video of each image (naming convention of DFDB image files has no bearing on the video source). Thus we can not test evaluation methods (RAE, SE).

Afchar et al.'s version of DF and F2F datasets were pre-processed slightly differently compared to ours. They utilised the Viola-Jones algorithm to detect faces [42] [1], which tends to work optimally for frontal faces, with $\sim 50$ faces extracted from each scene. Whereas we use a pre-trained facial recognition CNN [9] which works well no matter the face orientation, and we extract a face every 15 frames (0.5 seconds) resulting in an average of 12 faces extracted per video (for CDFv2). This is less than the expected 20 faces per video as not every frame had a detectable face. We believe that these pre-processing differences may skew results in favor of our baselines as the CNN can pick up more difficult to detect faces that the Viola-Jones algorithm can not [20] [9], in turn making our version of the dataset more difficult and hence harder for our models to classify correctly.

We considered applying Inductive Transfer Learning and fine-tuning each of these baselines but we decided not to as this would be repeating work from MInf Project (Part 1). Note: When DF and F2F are mentioned as a testset in this paper, it refers to the video datasets pre-processed to AF and RD images by us. When DF and F2F are referred to in the context of trainset we mean DF *weights* and F2F *weights* and we denote this with an asterisk e.g. DF*.

### 5.1.2 OC-FakeDect1

OC-FakeDect1 Training Settings, unless stated otherwise, are the same as Khalid et al. **OC-FakeDect1 was *trained from scratch*, as we did not have pre-trained weights for this model**. We train OC-FakeDect1 twice, we did this due to abnormal training loss curves and to fix a bug within our code. We used the following settings for our second and final training iteration for OC-FakeDect1 models:

- **Trainset:** {CDFv2 (AF), CDFv2 (RF), DFDB}, CDFv2 with actor-isolated train-set split and both pre-processing protocols plus DFDB.
- **Batch Size:** 128
- **Image Input Size:** $100 \times 100$, this can not be varied without changing the entire architecture. In their paper, convolution strides, kernel sizes, etc. result in the intermediate latent vector being 20,000 dimensions, therefore changing the input size would force us to vary these parameters to keep the dimensionality of the latent vector consistent Figure 2.1.
- **Training Epochs:** $100 \rightarrow 300$, increased due to noisy training loss (Khalid et al. use 100 training epochs). We increased the number of epochs due to the noisy training loss during training iteration 1 (Figure 5.1).

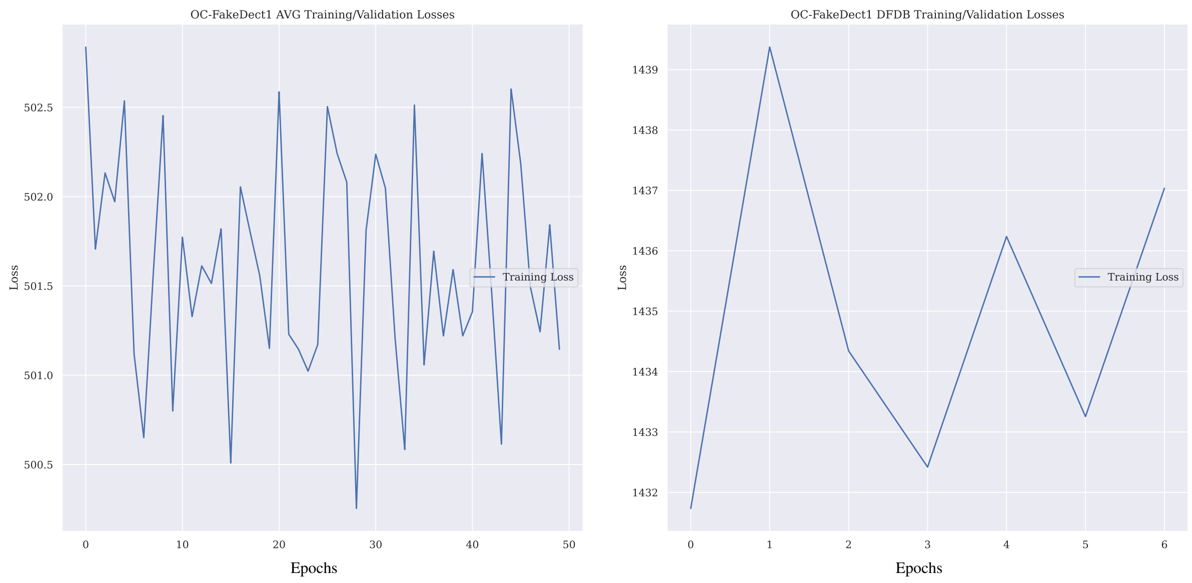


Figure 5.1: **Botched Training Iteration 1**: OC-FakeDect1 training metrics for DFDB [**Right**] and CDFv2 (AF) [**Left**]. Please note: Full training losses of this model being trained on CDFv2 (RF) was lost due to bad code management (overwritten plots).

• **Learning Rate, $\lambda$:** $1 \times 10^{-3} \rightarrow 1 \times 10^{-4}$, (Khalid et al. use a constant $\lambda = 1 \times 10^{-3}$), similar to training epochs, we lowered the learning rate to help fix training loss.

• **Train-Test Split:** $7.4\% : 92.6\%$ *(Trainset Reals : Testset Reals + Testset Fake + Trainset Fakes)* when testing on CDFv2 (due to actor-isolated train-set split). This skewed train-test split is caused by CDFv2 having $\times 9.6$ number of fakes compared to real images. $25\% : 75\%$ split on all other datasets as FF++ datasets have an equal number of real and fake videos (see Figure 5.4). Recall we are only training on the real class and using the rest to determine thresholds. (Khalid et al. use 30,000 real images for training and 4,000 real and fake images to determine thresholds).

• **Loss Function:** Mean Squared Error (MSE), $\sum_{i=1}^{D}(x_i - y_i)^2$.

• **Optimiser:** Adam [15].

• **Data Augmentation:** {Original, Horizontal Flip, Vertical Flip}, we apply the same data augmentation (generating *new* data points via linear transformations on existing data points) strategies as employed by Khalid et al.

• **Callbacks:** Early Stopping and Model Checkpoint $\rightarrow$ None. During the first training iteration, we used callbacks to prematurely end training (after 10 epochs of no improvement) and save model weights (when model improves), where improvement is defined by getting a lower training loss compared to the lowest training loss so far. This worked poorly due to the noisy training loss, hence the low epoch numbers on the x-axis of Figure 5.1. We dropped this in favor of no callbacks, instead of taking the model as is after the 300th epoch.

We were unable to recreate Khalid et al.'s SOTA OC-FakeDect2 due to GPU VRAM

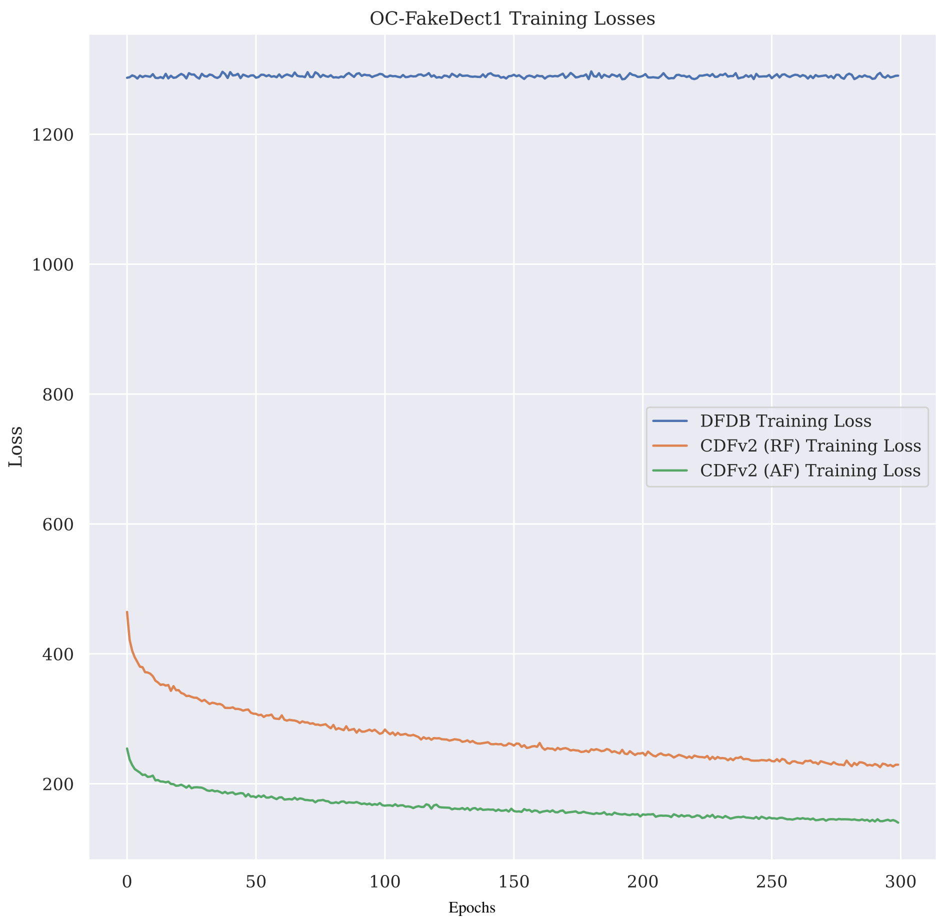


Figure 5.2: **Final Training Iteration 2**: OC-FakeDect1 training losses (MSE).

limitations and general ML library difficulties. However, OC-FakeDect1 still yields performance metrics $\sim 2\%$ lower than the SOTA OC-FakeDect2. This is why we only consider OC-FakeDect1. As mentioned prior, OCFakeDect models only train on a single class (real images) and use the RMSE discrepancy between classes to classify. This means that there is no validation set needed. Figure 5.4 showcases how we convert our binary datasets for use with this one-class model while still utilising all of the data available to us.

#### 5.1.2.1 Determining Classification Threshold

We use the real class from our trainset for training. Then, we combine the fakes from trainset and testset, and use the reals from the testset in order to determine the optimal threshold for separation. Figure 5.4 illustrates this process. Khalid et al. determine their threshold by calculating the inter-quartile range (IQR) to mark the 80% quartile ($T_{80}$) of the distribution (seen in Figure 2.4). We also considered calculating the point of intersection if the two RMSE distributions were normal curves, or by looping over all thresholds between the minimum and maximum RMSEs and picking the one that yields the highest F1 score.

However, we found that all of these methods yielded unreasonable thresholds. We opt to determine our threshold using $T_{50}$ between the means of the two distributions

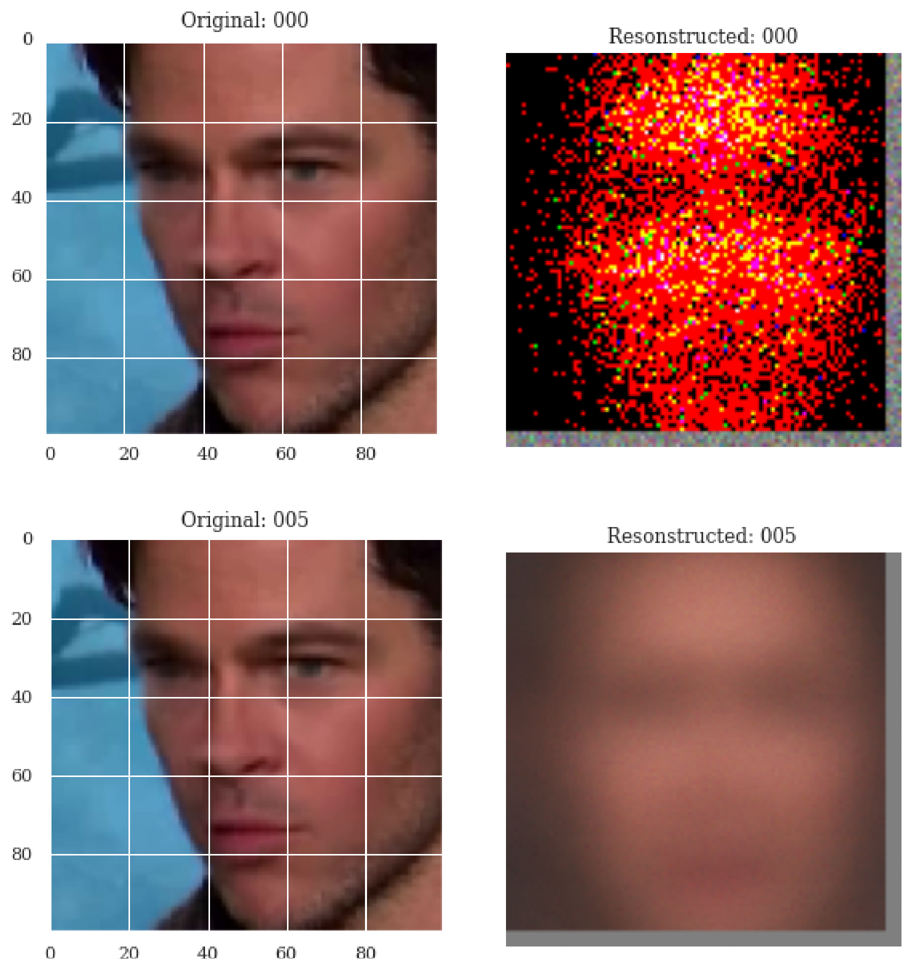


Figure 5.3: Reconstructed face (from single held-out validation image) via OC-FakeDect1 at 1st epoch vs 6th epoch.

(maximal separation between means) as this yielded the most sensible thresholds (example seen in the left-hand plot of Figure 5.5). One could argue that this is cheating in some sense, as we are determining a threshold on this testset and then testing whether it is a good fit. But this is the approach adopted by Khalid et al. Furthermore, the metric of AUC also considers all possible thresholds in order to aid in threshold selection in binary image classification, thus we do not deem this cheating. All thresholds are found on the trainset, even for new unseen datasets.

#### 5.1.2.2 Botched Training Loss (Training Iteration 1)

Figure 5.1 shows the training metrics of OC-FakeDect1 on DFDB and CDFv2 (AF) on our first training iteration. We see that training loss is erratic, with no clear downward trend. This type of training loss curve is an indicator of the learning rate being too high [3]. For this reason and the fact that this model takes substantially less time to train compared to ViT, we chose to break our commitment to making this model close to the baselines/original paper; and conducted light hyperparameter tuning. That is, we tested a variety of learning rates with this model and saw the same erratic loss.

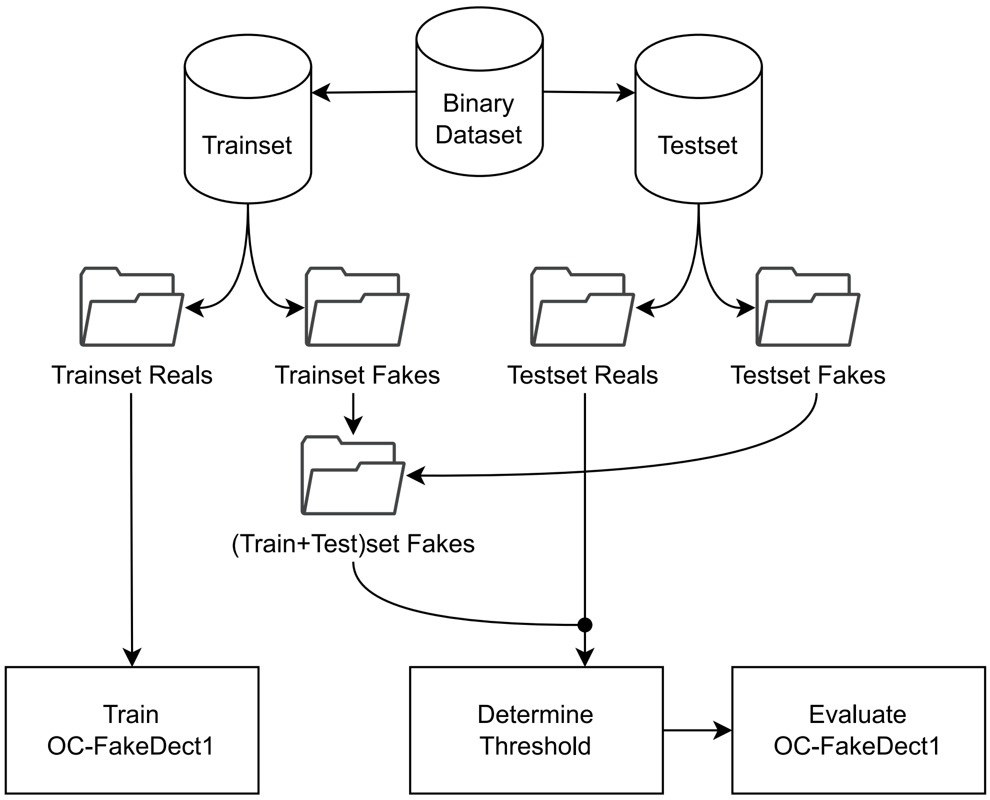


Figure 5.4: One-Class training protocol (how we convert our binary datasets for use with OC-FakeDect1)

We followed this by sanity checks whereby we train this model on a single batch of 64 images for 1000 *"epochs"* and we saw a clear reduction in training loss (classical decaying loss curves on the single batch).

This suggests that we are not training for enough epochs (when considering all images) for the model to learn good latent representations of the input faces. Thus, we increased the total number of training epochs from Khalid et al.'s 100 to 300. In addition to this, we lowered the learning rate by an order of magnitude to $1 \times 10^{-4}$, we did this to ensure that our second training iteration would be successful.

#### 5.1.2.3 Final Training Iteration

Due to the reasons discussed above and the fact that we found a major bug in our code, we had no option but to retrain our model. Hence, we have two sets of loss metrics Figure 5.1 and Figure 5.2 for the first botched and the second final training iteration, respectively. The training settings above are for our final model, and all further results are from the final model (second training iteration). We omit the results from our first bugged training iteration as they are too numerous, but these can be found via our links in our Appendices.

During test time on a dataset the model has not been trained on, we perform the same thresholding technique on its train split (used as a validation split). Classification is carried out on the test split using the threshold determined on the training set (see

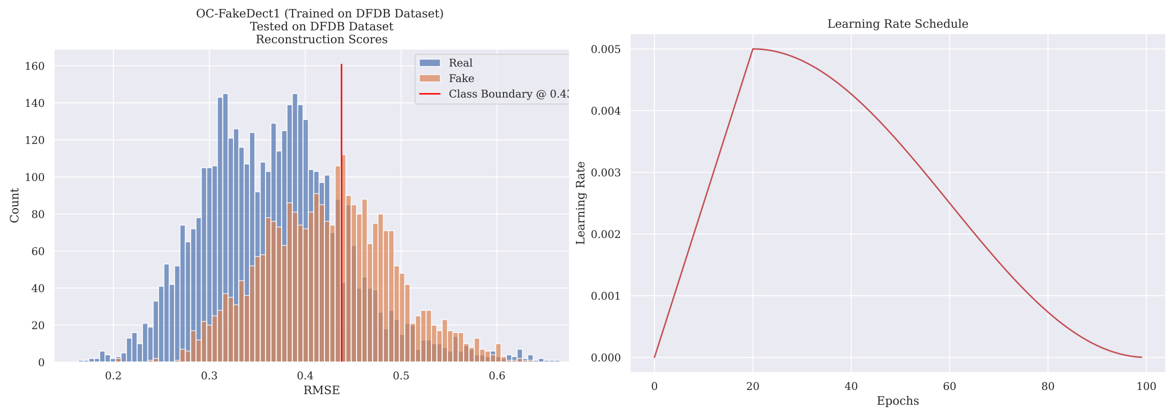


Figure 5.5: **Left:** Highest F1 threshold for OC-FakeDect1 trained and tested on DFDB w/ SE (first training iteration) **Right**: Cosine learning rate schedule for ViT models

Figure 5.4).

### 5.1.3 ViT

ViT Training Settings, unless stated otherwise, are the same as Dosovitskiy et al.'s original ViT paper, otherwise, we use Afchar et al.'s baselines settings where possible. **ViTs were *not trained from scratch*, we use ImageNet 2012 pre-trained weights and only fine-tune the Dense final layer**:

• **Trainset:** {CDFv2 (AF), CDFv2 (RF)}, CDFv2 with actor-isolated train-set split and both pre-processing protocols.

• **Batch Size:** 64. Baselines and original ViTs use batch sizes of 75 and 512, respectively, neither of which fit onto our GPU's VRAM.

• **Image Input Size:** $256 \times 256$. Same as baselines.

• **Training Epochs:** 100. Same as baselines.

• **Max Learning Rate, $\lambda_{\max}$:** $5 \times 10^{-3}$, We use a Cosine Annealed learning rate used by Dosovitskiy et al., which starts at 0 linearly climbing to $\lambda_{\max}$ 20% through training and decays based on a cosine curve (see right-hand plot of Figure 5.5). This ensures convergence to local minima.

• **Validation Split:** 20% of Trainset.

• **Train-Test Split:** 73.9% : 26.1% when testing on CDFv2 (due to actor-isolated train-set split). 20% of the trainset was used for validation metrics, thus validation metrics are not actor-isolated, unlike the testset.

• **Loss Function:** Binary Cross-Entropy $-(y\log(p)+(1-y)\log(1-p))$ where $y \in \{0,1\}$.

• **Optimiser:** Stochastic Gradient Descent (SGD) [31] with 0.9 momentum and Cosine Annealed learning rate schedule. SGD was used to train the final Dense unit from ViT feature extractor. Original ViT was optimised using Adam [16].

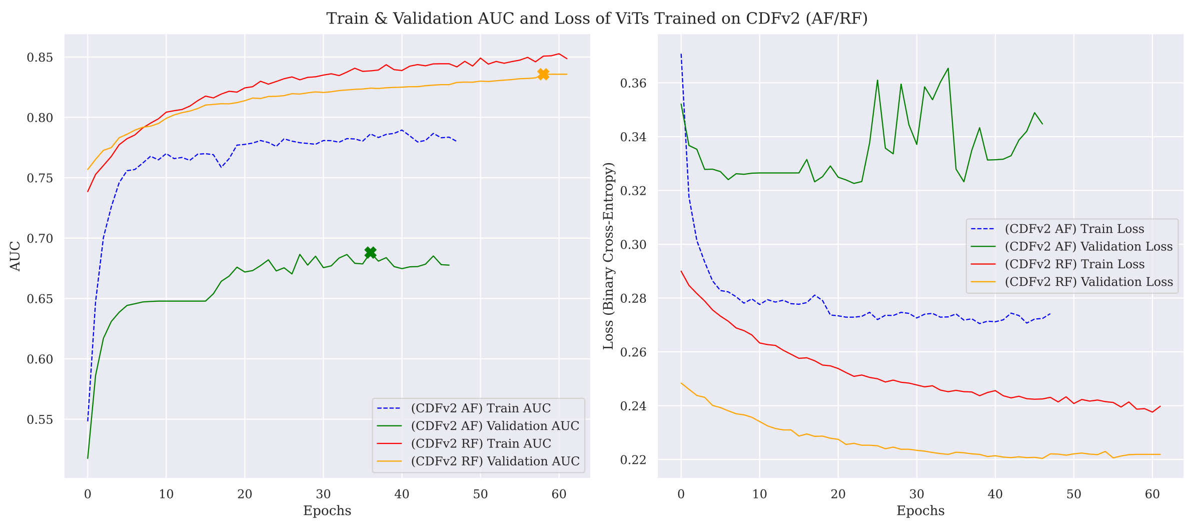


Figure 5.6: ViT training metrics. Orange and green dots signify the highest validation AUC ViT models we chose to use for evaluation: ViT (CDFv2 RF) and ViT (CDFv2 AF), respectively.

- **Callbacks:** Same Early Stopping and Model Checkpointing used as OC-FakeDect1.

Dosovitskiy et al. outlined several ViT models in their original paper. When we say ViT, we are referring to ViT-Base with $16 \times 16$ input patch size. We had the option of using the SOTA ViT-Large and ViT-Huge models with 307 million and 632 million parameters, respectively; however, these proved too unwieldy to use. We choose to use Inductive Transfer Learning and use pre-trained weights from ImageNet 2012. We freeze all weights of the network and detach the final dense layer (which had 1000 units corresponding to the 1000 classes in ImageNet) and replace it with our own network. To minimise compute constraints we only choose to add a single trainable layer truncating the features from the penultimate layer into a single Dense unit for classification. This single Dense unit has Sigmoid activation corresponding to the probability of the input being fake. We only fine-tune this final layer resulting in 769 and 85.8 million trainable and non-trainable parameters, respectively. We hypothesise that adding extra trainable layers would vastly improve ViT performance beyond the results showcased in Section 6.3, but we leave this as future work. In most applications of Inductive Transfer Learning, several trainable layers are employed for fine-tuning, including Dropout, Batch-Norm, etc in addition to Dense layers [28].

Both Deepfake detection and ImageNet classification concern themselves with (semi-)natural images. This means the first few layers of the network tend to be the same regardless of domain. Weights in early layers distinguish general features (lines/-gradients/edges). Intermediate layers combine features learned in earlier ones to learn new increasingly complex features (simple shapes/curves/corners), with final layers learning domain-specific features (real/fake faces in our case). Examples of this can be seen in Figure 5.7.

Since we use the same random seeds in all cases, our training datasets are mirror copies of each other; everything is the same from the network initialisation to the order that we train images, with the only difference being the pre-processing protocol applied. Thus, both ViTs start at the same position in the loss landscape and we can make direct comparisons between the two. Notice that we trained OC-FakeDect1 on three datasets {CDFv2 (AF), CDFv2 (RF), DFDB} Whereas we only train ViT on two datasets (not DFDB). This was because training the OC-FakeDect1 model was relatively quick in comparison to ViT and we deemed it more important to train on the harder datasets for which we had spent time pre-processing/isolating actors.

Throughout the training process, in addition to keeping track of validation loss, we also keep track of validation AUC, this allows us to retrieve the weights of the model with the best validation AUC and use it for evaluation. We choose to stop our model based on maximal validation AUC instead of minimal validation loss since in preliminary experiments, this yielded better *non*-actor-isolated test AUC on CDFv2 (RF). However, we only did this once as it is bad practice to optimise to our testset like this. Furthermore, we use actor-isolated test splits for all other evaluations, so this did not skew our results.

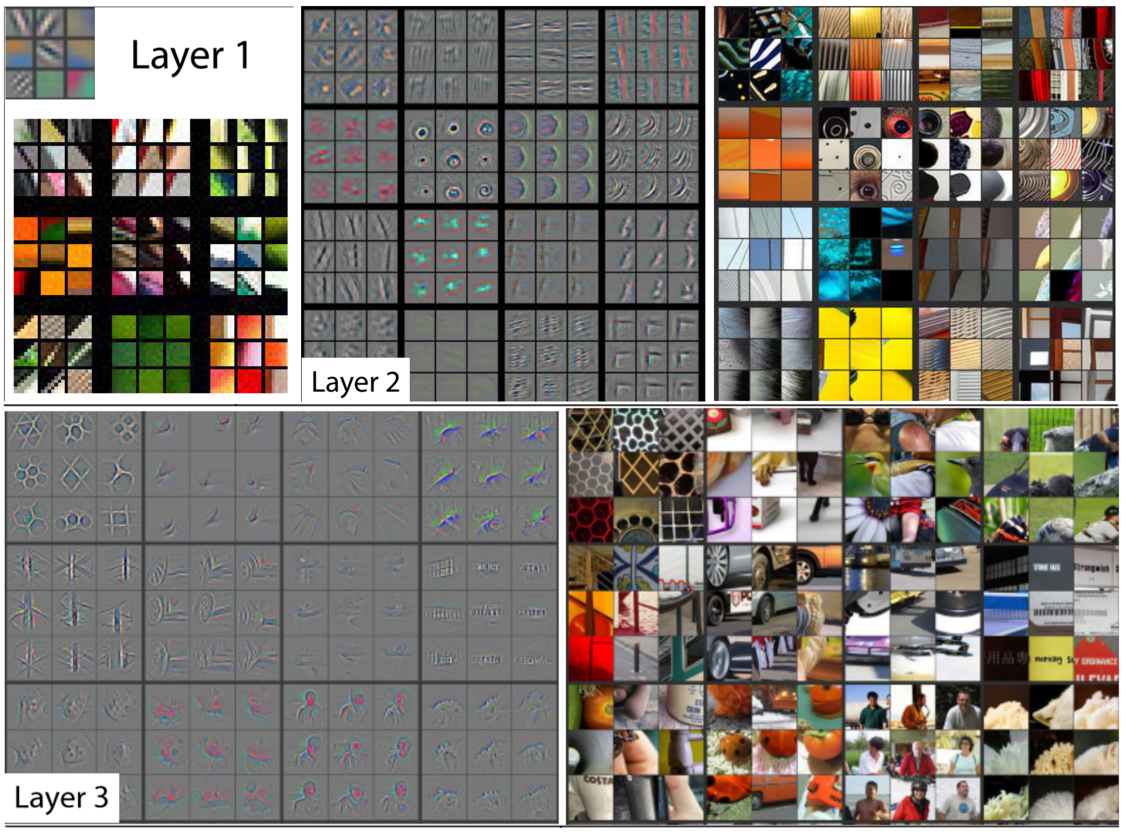


Figure 5.7: Visualisation of learned features from weights in AlexNet for ImageNet 2012 [44]

## 5.2 Overview

In this chapter, we have outlined both the methodology and experiment settings for all architectures. The majority of our time was spent pre-processing and refining our OC-FakeDect1 models, with ViT models working well with minimal alterations. Training each ViT model took approximately 48 hours, twice as long as training OC-FakeDect1s. However, we only trained one of the two architectures on all three datasets, resulting in a total of five models trained by us (not six as we did not train ViT on DFDB due to compute constraints).

All of this resulted in 225 experiments. We chose to do this cross-product style experiment method to provide analysis in Chapter 6 and Chapter 7 which has not yet been seen by the Deepfake detection literature. This was not a trial-and-error/see-what-sticks style experiment method, as all results have their use and can be compared with their counterparts. This methodology allows us to vary independent variables (per architecture) and apply the scientific method. Each architecture's results were trained and tested in the exact same way with only the independent variables changing. The bulk of our results come from testing on all possible testsets for each model. We do this in order to concretely verify the effects of pre-processing protocols and evaluation methods on our models, allowing us to determine which is best and when.

We use a variety of carefully crafted statistical tests in order to draw conclusions about the validity of our approach and which combinations of settings work best. To the best of our knowledge, no other papers consider an actor-isolated test split for CDFv2. However, doing this allows us to make stronger claims about model generalisability as the model can not use remembered faces or background scenes in order to cheat.

# Chapter 6

# Results & Evaluation

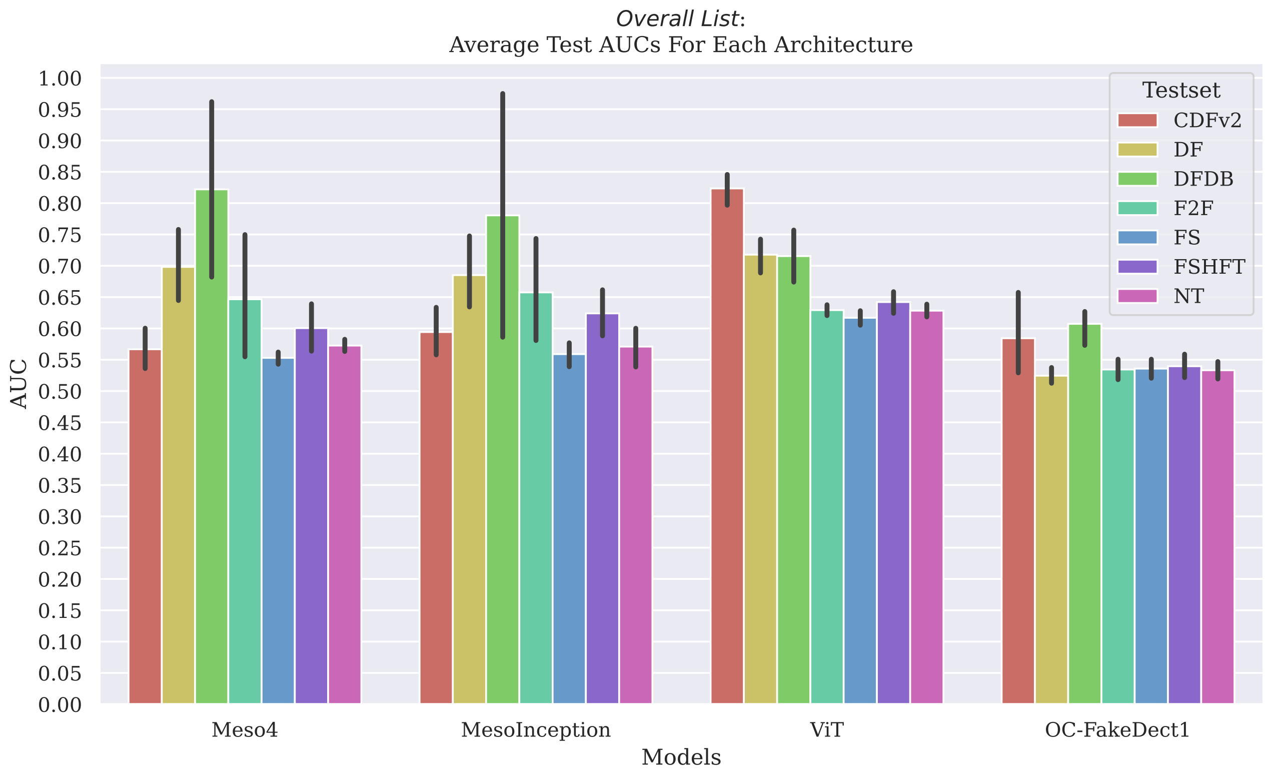
We showcase results and perform an objective analysis with surface-level explanations in this chapter. Subjective conclusions, further comments, and trends are included in Chapter 7. All code to create figures, tables, t-tests, and all 225 experiments can be viewed in the Appendices of this paper. Please note that:

**Baselines & DF/F2F Trainsets:** Any table where the model is Meso4/MesoInception and trainset is DF/F2F, these models were not trained on DF/F2F by us. Instead, they use pre-trained weights fine-tuned on the DF/F2F datasets by Afchar et al., we denote this with an asterisk (*). Any other mention of trainset being DF/F2F outside of these baselines refers to the actual datasets and not weights.

**F1, Precision & Recall:** We purposefully choose not to draw any conclusions from F1, Precision, or Recall as all of these metrics use a threshold of 0.5 which is not the optimal threshold for each model. This is why some models show high AUC but low F1. We realised this a few days before the submission deadline and retesting all models would be infeasible, this is why we do not include any confusion matrices. Furthermore, AUC tends to give a fuller picture of a model's performance as it considers all thresholds. This being said, this is an academic exercise; if we were to deploy these models in the real world for Deepfake detection, we would take great care to pick the most optimal thresholds to obtain the best results on unseen data. i.e. AUC has no bearing on real-world performance, we have to pick a threshold for classification.

**Dependent t-test for Paired Samples:** We use `scipy.stats.ttest_rel` which calculates t-test on *two related* samples of scores. To eliminate any correlation between the train and test splits, all t-tests performed are done using the full *generalisability list* of models (Figure 6.1, Table 6.3) i.e. where trainset $\neq$ testset; testing on new unseen datasets that are not the dataset we trained on. This is a test for the null hypothesis that two related or repeated samples have identical mean values, where variance is unknown. We make no assumptions about the distributions of the two groups.
One could argue that we can not meaningfully conduct a t-test on AUC as there is a correlation we are not accounting for, as we tested on the same dataset we trained on. However, we eliminate this correlation by filtering out all models where trainset $\neq$ testset, the *generalisability list* (a subset of which can be seen in Table 6.3) and thus we can use a dependent t-test for paired samples. For each t-test we vary the alternate hypothesis, $a$, from $a \in \{<, \neq, >\}$ which will test if the mean of the distribution underlying the first sample ($\mu_{\text{GroupA}}$) is alternate to the mean of the distribution underlying

the second sample ($\mu_{\text{GroupB}}$). We use a p-value significance threshold of 0.05. E.g. if $a \in \{>\}$ then $h_0 : \mu_{\text{GroupA}} = \mu_{\text{GroupB}}$, and $h_1 : \mu_{\text{GroupA}} > \mu_{\text{GroupB}}$.



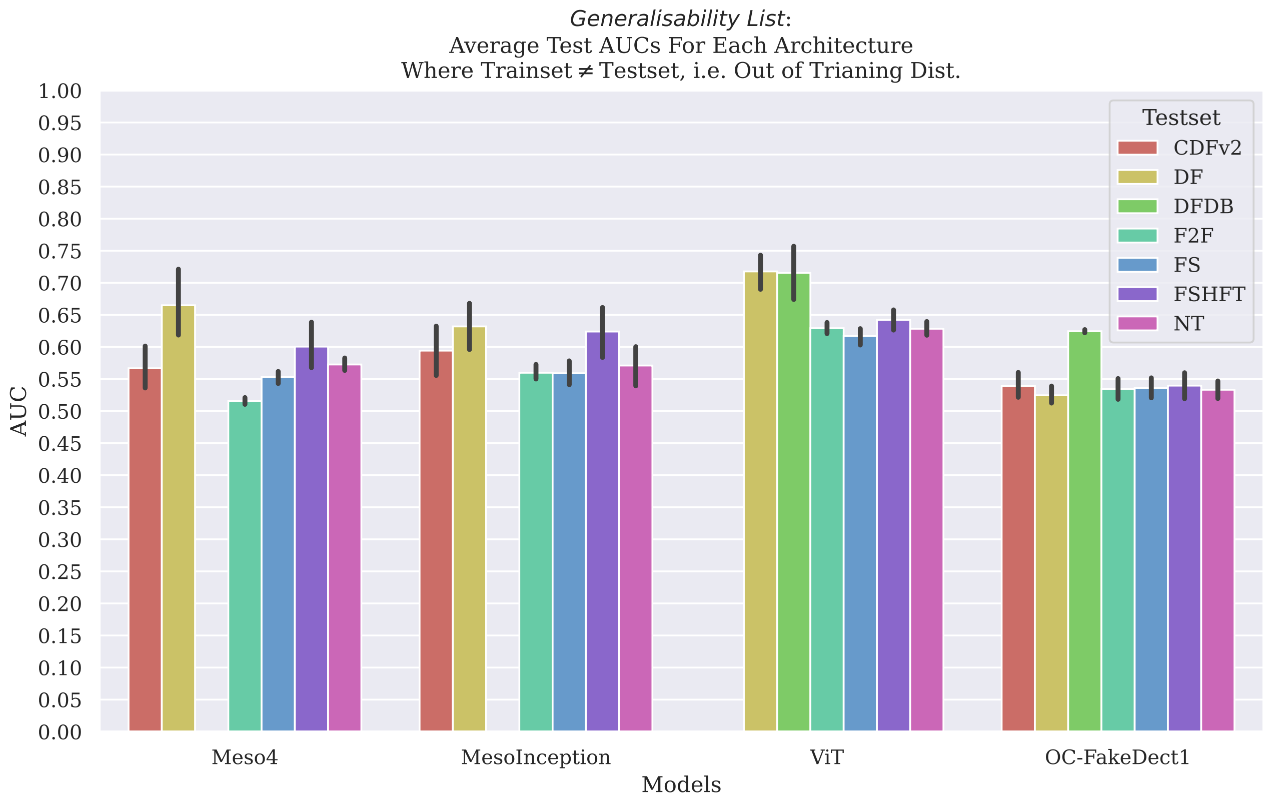


Figure 6.1: **Top:** Average test AUCs for each model. **Bottom:** *Generalisability list* of models, average test AUCs for each model where trainset ≠ testset. Black lines indicate standard error.

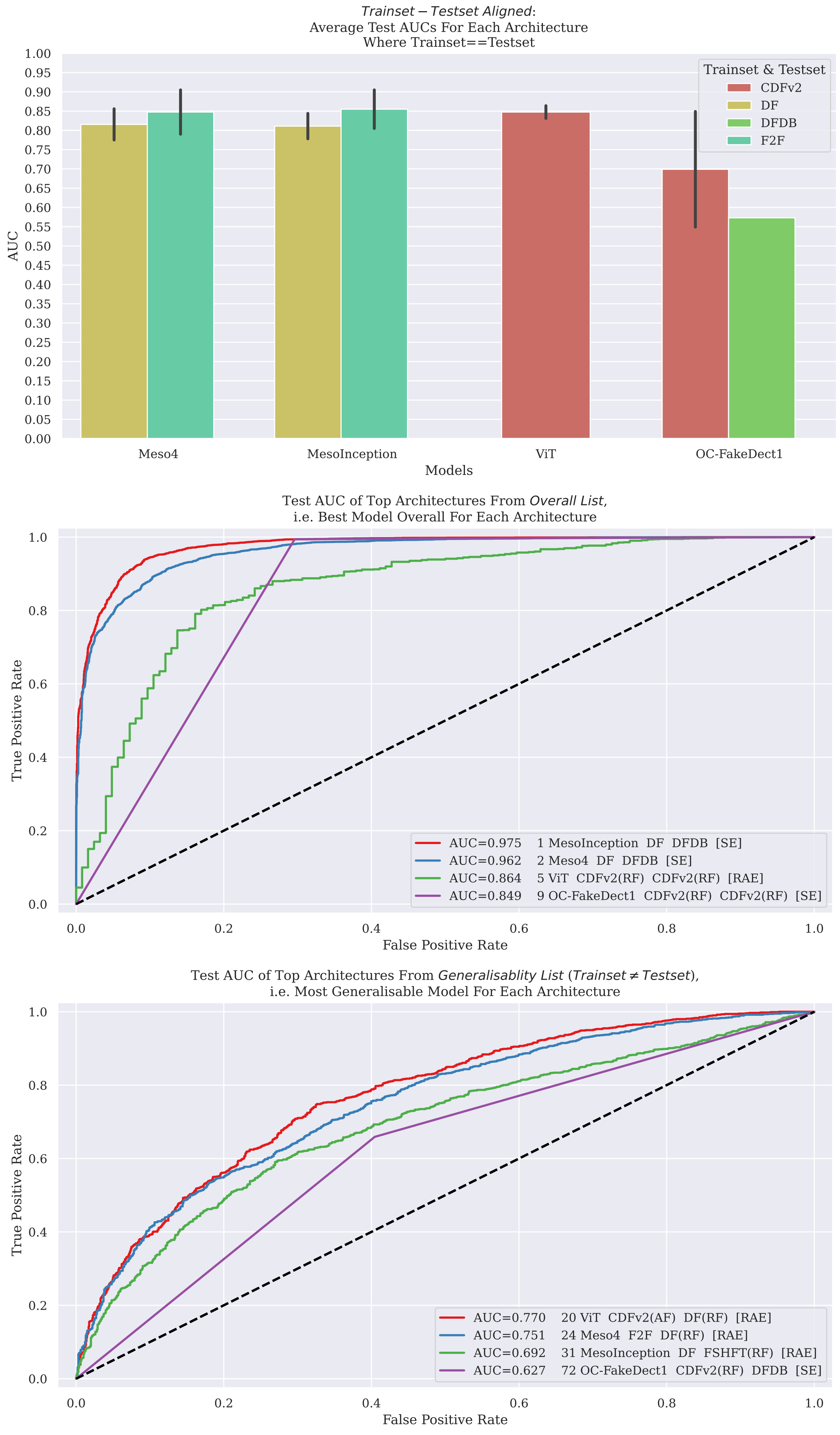


Figure 6.2: **Top:** Average test AUCs of 7 main models (not considering our AF and RF pre-processing). **Middle:** Each architecture's best test AUCs for from overall list (Table 8.1. **Bottom:** Each architecture's best test AUCs for *generalisability list* (subset at Table 6.3)

| | **Model** | **Trainset** | **Testset** | **AUC** | **F1** | **Precision** | **Recall** | **Evaluation Method ( 0=SE, 1=RAE )** | **Trainset Pre-Processing ( 0=RF, 1=AF )** | **Testset Pre-Processing ( 0=RF, 1=AF )** |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | MesoInception | DF* | DFDB | 0.975 | 0.925 | 0.953 | 0.900 | 0 | - | - |
| 3 | MesoInception | F2F* | F2F | 0.905 | 0.318 | 0.252 | 0.430 | 1 | - | 0 |
| 8 | Meso4 | DF* | DF | 0.856 | 0.175 | 0.159 | 0.194 | 1 | - | 0 |
| 30 | Meso4 | DF* | FSHFT | 0.704 | 0.380 | 0.316 | 0.476 | 1 | - | 0 |
| 40 | MesoInception | F2F* | CDFv2 | 0.673 | 0.908 | 0.915 | 0.902 | 1 | - | 1 |
| 70 | MesoInception | DF* | NT | 0.628 | 0.445 | 0.363 | 0.573 | 1 | - | 0 |
| 95 | MesoInception | DF* | FS | 0.602 | 0.211 | 0.287 | 0.166 | 1 | - | 1 |

Table 6.1: Best baselines by testset, where asterisk (*) in testset denotes that we are using pre-trained weights and these models were not trained by us.

| | **Model** | **Trainset** | **Testset** | **AUC** | **F1** | **Precision** | **Recall** | **Evaluation Method ( 0=SE, 1=RAE )** | **Trainset Pre-Processing ( 0=RF, 1=AF )** | **Testset Pre-Processing ( 0=RF, 1=AF )** |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | MesoInception | DF* | DFDB | 0.975 | 0.925 | 0.953 | 0.900 | 0 | - | - |
| 3 | MesoInception | F2F* | F2F | 0.905 | 0.318 | 0.252 | 0.430 | 1 | - | 0 |
| 5 | ViT | CDFv2 | CDFv2 | 0.864 | 0.963 | 0.930 | 0.999 | 1 | 0 | 0 |
| 8 | Meso4 | DF* | DF | 0.856 | 0.175 | 0.159 | 0.194 | 1 | - | 0 |
| 30 | Meso4 | DF* | FSHFT | 0.704 | 0.380 | 0.316 | 0.476 | 1 | - | 0 |
| 51 | ViT | CDFv2 | NT | 0.650 | 0.594 | 0.597 | 0.592 | 0 | 1 | 0 |
| 63 | ViT | CDFv2 | FS | 0.633 | 0.597 | 0.603 | 0.592 | 0 | 1 | 0 |

Table 6.2: Overall best test AUC models by testset.

## 6.1 Untrained Baselines

Table 6.1 shows the best baseline models (Meso4/MesoInception) for each testset. The numbers on the left-most column indicate the total ranking in terms of AUC among all models. Table 6.2 shows that the best performing models (regardless of trainset/testset) are the baselines, with MesoInception tending to outperform Meso4. Recall that both baselines came with DF/F2F pre-trained weights, they were not trained on the DF/F2F datasets by us (denoted via *). Thus, we'd expect baselines to do extremely well on the DFDB dataset (as this is what the baselines were fine-tuned to) and on both DF/F2F testsets. This is because they are all in the same training distribution.

Table 6.3 shows the highest test AUC models by testset to gauge model generalisability. This can more clearly be visualised by Figure 6.1 where we can see the average AUC of all models tested on datasets that they were not trained on. The results indicate that baselines perform extremely well when trainset-testset are aligned (top of error bars on green/yellow in the upper bar chart of Figure 6.1). However, they perform extremely poorly in terms of generalisability AUC when tested on Deepfakes outwith their training distribution. Furthermore, AUC of baselines tested on datasets corresponding to their pre-trained weights is lower compared to metrics stated in their paper. We hypothesise this is due to our face extraction method making our dataset harder compared to Afchar et al.'s Viola-Jones approach. e.g. Meso4 with DF weights yields 0.856 AUC on our pre-processed version of the DF dataset. Most baselines gained test AUC when RAE was applied, with the only exception to this being DFDB, the dataset provided by the paper.

| | Model | Trainset | Testset | AUC | F1 | Precision | Recall | Evaluation Method ( 0=SE, 1=RAE ) | Trainset Pre-Processing ( 0=RF, 1=AF ) | Testset Pre-Processing ( 0=RF, 1=AF ) |
|---|---|---|---|---|---|---|---|---|---|---|
| 20 | ViT | CDFv2 | DF | 0.770 | 0.661 | 0.511 | 0.933 | 1 | 1 | 0 |
| 23 | ViT | CDFv2 | DFDB | 0.757 | 0.958 | 0.922 | 0.997 | 0 | 0 | - |
| 30 | Meso4 | DF* | FSHFT | 0.704 | 0.380 | 0.316 | 0.476 | 1 | - | 0 |
| 40 | MesoInception | F2F* | CDFv2 | 0.673 | 0.908 | 0.915 | 0.902 | 1 | - | 1 |
| 51 | ViT | CDFv2 | NT | 0.650 | 0.594 | 0.597 | 0.592 | 0 | 1 | 0 |
| 54 | ViT | CDFv2 | F2F | 0.646 | 0.601 | 0.440 | 0.946 | 1 | 0 | 0 |
| 63 | ViT | CDFv2 | FS | 0.633 | 0.597 | 0.603 | 0.592 | 0 | 1 | 0 |

Table 6.3: [Subset of *generalisability list*] most generalisable model by test AUC by testset (where generalisability means that training dataset $\neq$ testing dataset, i.e. out of distribution testing).

| | Model | Trainset | Testset | AUC | F1 | Precision | Recall | Evaluation Method ( 0=SE, 1=RAE ) | Trainset Pre-Processing ( 0=RF, 1=AF ) | Testset Pre-Processing ( 0=RF, 1=AF ) |
|---|---|---|---|---|---|---|---|---|---|---|
| 5 | ViT | CDFv2 | CDFv2 | 0.864 | 0.963 | 0.930 | 0.999 | 1 | 0 | 0 |
| 20 | ViT | CDFv2 | DF | 0.770 | 0.661 | 0.511 | 0.933 | 1 | 1 | 0 |
| 23 | ViT | CDFv2 | DFDB | 0.757 | 0.958 | 0.922 | 0.997 | 0 | 0 | - |
| 37 | ViT | CDFv2 | FSHFT | 0.675 | 0.621 | 0.463 | 0.940 | 1 | 1 | 1 |
| 51 | ViT | CDFv2 | NT | 0.650 | 0.594 | 0.597 | 0.592 | 0 | 1 | 0 |
| 54 | ViT | CDFv2 | F2F | 0.646 | 0.601 | 0.440 | 0.946 | 1 | 0 | 0 |
| 63 | ViT | CDFv2 | FS | 0.633 | 0.597 | 0.603 | 0.592 | 0 | 1 | 0 |

Table 6.4: Overall best ViT models by testset.

## 6.2 Best Models

The upper bar chart of Figure 6.1 shows the average test AUCs of each model for all models, whereas the lower chart shows the same but when testset $\neq$ trainset (hence the missing bars). Table 6.2 and Table 6.3 show the same (but only top results per testset). Notice that OC-FakeDect1 has been tested on all datasets, as all testset bars are present, this is because OC-FakeDect1 was trained on both CDFv2 and DFDB, and the bar for CDFv2 indicates the model performance on DFDB and vice-versa. This is not a mistake. Similarly, Meso4 and MesoInception both still have bars for DF/F2F despite them using pre-trained weights for these datasets, these bars are average test AUCs from the opposite trainsets; E.g. the yellow bar (DF) in the lower chart plot in Figure 6.1 indicates DF performance on F2F* (for MesoNets). Overall the best model is MesoInception with DF weights tested on DFDB. We fully expect to see this baseline outperform our models as Afchar et al. conducted a full hyperparameter search whereas we did not. Recall that each model tries to mimic the training protocol (input size, learning rate decays, etc) of the Meso baselines where possible, despite this approach (probably) not being optimal for each model.

Table 6.3 and the lower chart of Figure 6.1 both clearly indicate that our ViTs are the most generalisable, achieving test AUCs just under 0.78 on the unseen datasets of DF and DFDB. We believe this is due to the more convincing Deepfakes present in CDFv2 allowing ViT to pick up on features the other models struggle to. OC-FakeDect1 struggles to classify well on any unseen datasets except on DFDB, most likely due to the unconvincing Deepfakes present in DFDB, allowing for the ConvVAE to pick up on ”low-hanging” latent features which do not generalise well to other datasets.

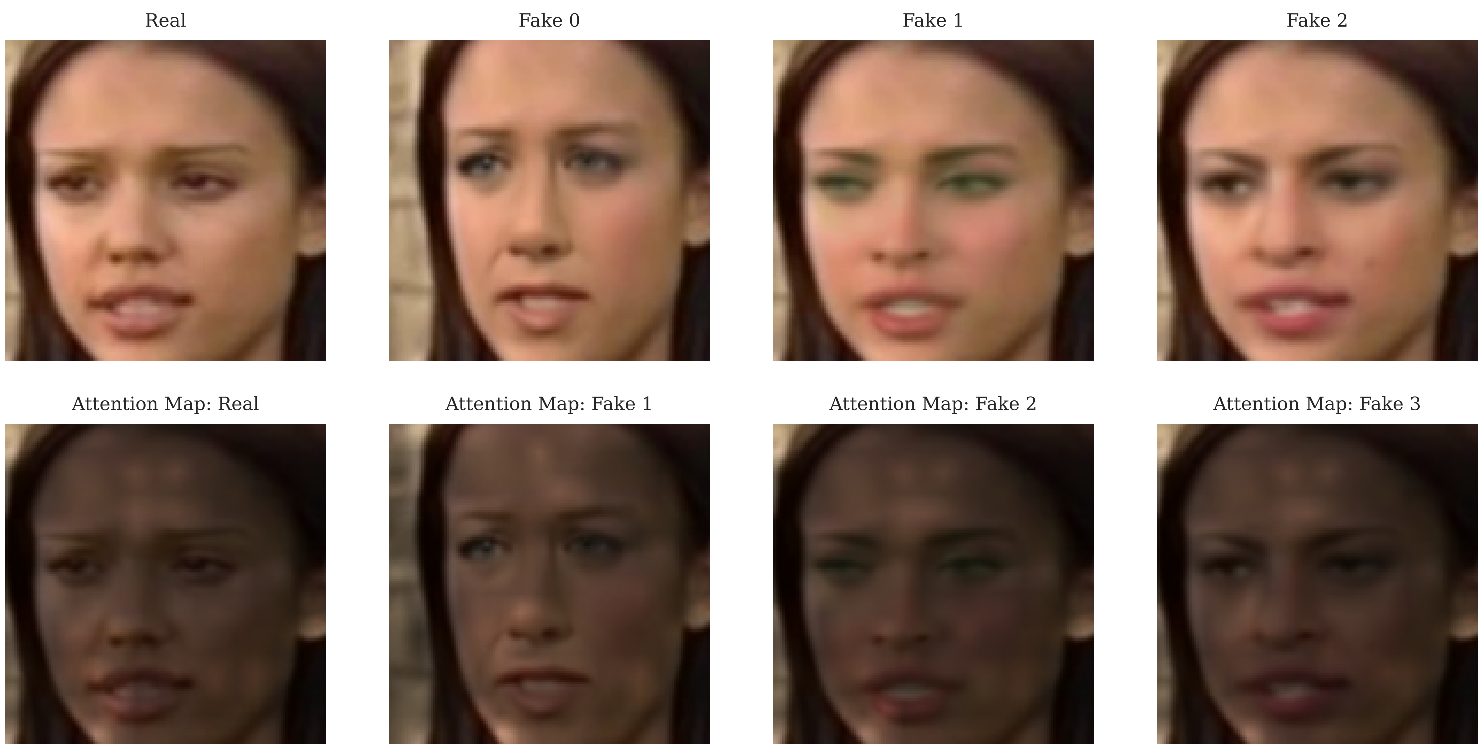


Figure 6.3: Attention map of real and fake test image from ViT trained on CDFv2 with RF pre-processing.

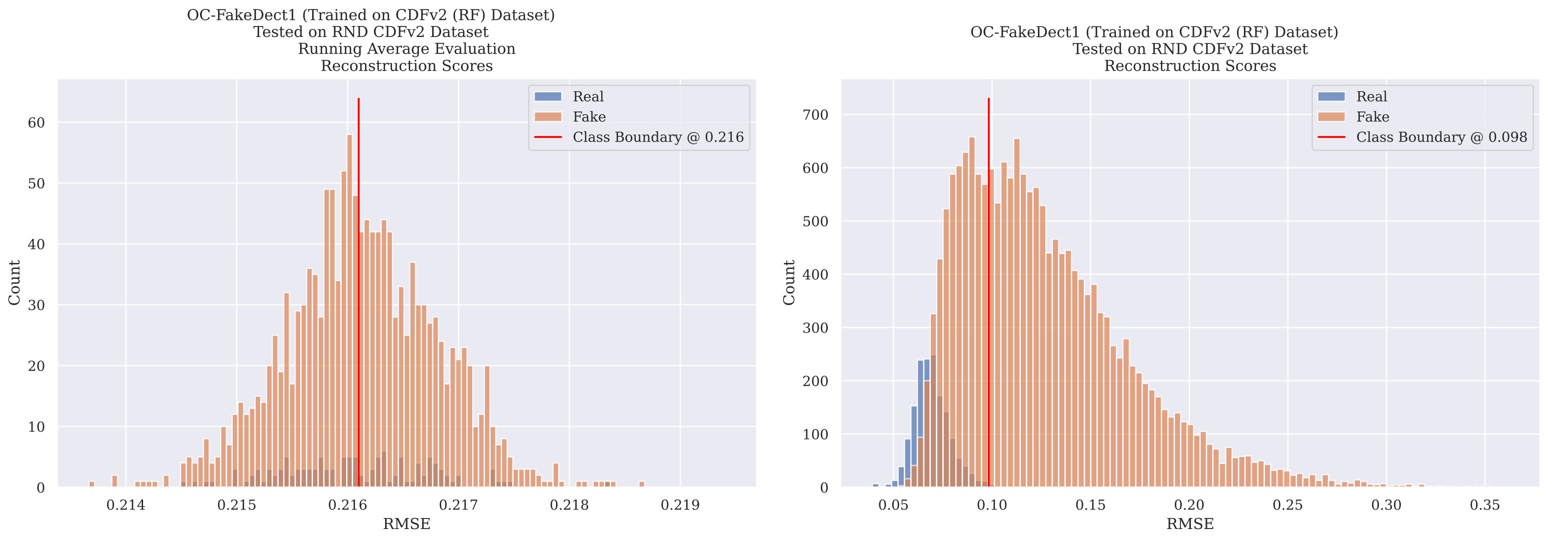


Figure 6.4: OC-FakeDect1 trained on CDFv2 (RF) thresholds when **Left:** RAE vs **Right:** SE.

| | Model | Trainset | Testset | AUC | F1 | Precision | Recall | Evaluation Method ( 0=SE, 1=RAE ) | Trainset Pre-Processing ( 0=RF, 1=AF ) | Testset Pre-Processing ( 0=RF, 1=AF ) |
|---|---|---|---|---|---|---|---|---|---|---|
| 9 | OC-FakeDect1 | CDFv2 | CDFv2 | 0.849 | 0.428 | 0.770 | 0.296 | 0 | 0 | 0 |
| 72 | OC-FakeDect1 | CDFv2 | DFDB | 0.627 | 0.338 | 0.291 | 0.405 | 0 | 0 | - |
| 101 | OC-FakeDect1 | DFDB | FSHFT | 0.590 | 0.552 | 0.460 | 0.690 | 0 | - | 0 |
| 117 | OC-FakeDect1 | DFDB | FS | 0.574 | 0.495 | 0.389 | 0.681 | 0 | - | 0 |
| 118 | OC-FakeDect1 | DFDB | F2F | 0.573 | 0.536 | 0.446 | 0.671 | 0 | - | 0 |
| 124 | OC-FakeDect1 | DFDB | DF | 0.571 | 0.535 | 0.444 | 0.675 | 0 | - | 0 |
| 129 | OC-FakeDect1 | DFDB | NT | 0.568 | 0.489 | 0.385 | 0.669 | 0 | - | 0 |

Table 6.5: Overall best OC-FakeDect1 models by testset.

## 6.3 Vision Transformer

Table 6.2 shows the highest AUC (0.864) model which was not a baseline is ViT, when trained and tested on CDFv2; which is a substantially harder dataset compared to the others. As mentioned prior, Table 6.3 indicates that ViTs are most generalisable. The most generalisable ViTs used a variety of trainset pre-processing and evaluation methods, but most seem to yield better performance with RF testsets. The best overall ViT used RAE with RF pre-processing applied to both trainset and testset, whereas the most generalisable ViT prefers an AF trainset instead.

Self-Attention allows ViT to visualise which parts of the image the model ”focuses” on to make predictions. Figure 6.3 shows the Attention maps for a real and fake test image from ViT trained on CDFv2 (RF). Attention weights focus on the same parts of the image, whether real or fake, with more Attention placed on the fake image (especially around the nose and upper lip). We hypothesise that with proper hyperparameter tuning and modernised training protocols (e.g. input masking [24] [19]) we would be able to achieve the new SOTA ($\sim$0.95 AUC [4]). Recall that we perform actor isolation on our CDFv2 testset, this reinforces our claims of SOTA performance.

## 6.4 OC-FakeDect1

Despite the marginal inferiority of OC-FakeDect1 compared to OC-FakeDect2, we found that no ConvVAE models achieved the best test AUC on any of our testsets. In Table 6.5 our best OC-FakeDect1 ranks 9th overall and reaches an AUC of 0.849, just under ViT’s best of 0.864. All OC-FakeDect1s preferred RF train and testset pre-processing, in addition to SE. These models can not properly utilise RAE as we are averaging over RMSE meaning the RMSE/Count histogram is skewed as averages accumulate; bad re-creation RMSEs propagate over the predictions. This can clearly be seen in Figure 6.4.

Our intuition that these models would perform better when trained on the less realistic DFDB proved false, as the best ConvVAEs are trained on the more realistic CDFv2. Training on this dataset was the most generalisable option but only for DFDB with all others yielding AUCs marginally higher than random. The results above suggest that OC-FakeDect1 does well when trained on crisp (RF) high-quality Deepfakes, but it does not generalise well. One reason for this could be due to the VAE using the latent space to create distinguishing features to re-create the input images and thus high-quality Deepfakes (DFDB) aid the VAE in learning latent representations and re-creating the input.

Training losses for this model (Figure 5.2) never settle for DFDB, so we expect poor performance when trained on this dataset. However, we found that training losses plateaued at a lower level for CDFv2 when pre-processed with AF compared to RF. Despite this the RF variant was the best performing model, indicating that our models were overfitting to training reals.

## 6.5 Pre-Processing: AF vs RF

Figure 6.6 shows the average test AUCs split by trainset and testset pre-processing protocol for each model. It suggests that models tested on AF datasets yield a small improvement in test AUC. To determine which pre-processing protocol is most beneficial to model performance, we consider a set of dependent t-tests for paired samples as described at the start of this chapter on the *generalisability list* (trainset $\neq$ testset) mentioned prior. We set groups A and B to all non-matching combinations of trainset and testset pre-processing protocols to obtain Table 6.6.

There is strong evidence to suggest that testing on AF datasets yields higher mean test AUC compared to RF, mostly when trainset pre-processing differs from testset. It also suggests that there is no significant difference in test AUC means among AF and RF train tested models. i.e. when trainset and testset have the same pre-processing applied there is no difference in mean test AUC. These findings found in Table 6.6 match what is shown in Figure 6.6.

Although we can use Table 6.6 to conclude that some pre-processing protocols are better than others (first 3 rows are better than the bottom 3 rows). We can not use p-value rankings to make any claims about which protocol is better compared to others that also yield improved AUC (i.e. within the top 3 rows).

## 6.6 Evaluation Method: RAE vs SE

Figure 6.5 shows the average test AUC of all models split by whether the evaluation method was RAE or SE. RAE improved model performance in all cases except OC-FakeDect1. We used the same t-test setup mentioned prior.

We isolate all models where the evaluation method was RAE (group A) and their equivalent model where the evaluation method was SE (group B), with the alternative hypothesis being $a = \{<\}$ i.e. group A has a mean less than group B. This results in a p-value 0.0873 which is strong evidence to suggest that the two distributions are equal. However, this is not what Figure 6.5 shows. On average all models have improved AUC when evaluated with RAE with exception of OC-FakeDect1. We decided to further isolate the two groups by filtering out OC-FakeDect1 models, with the alternative being $>$, i.e. RAE models have greater mean AUC than SE models when excluding OC-FakeDect1. This resulted in a 0.0004 p-value (several orders of magnitude less than the previous p-value) which is strong evidence to suggest that the average test AUC of RAE models is greater than SE models when the model is not OC-FakeDect1.

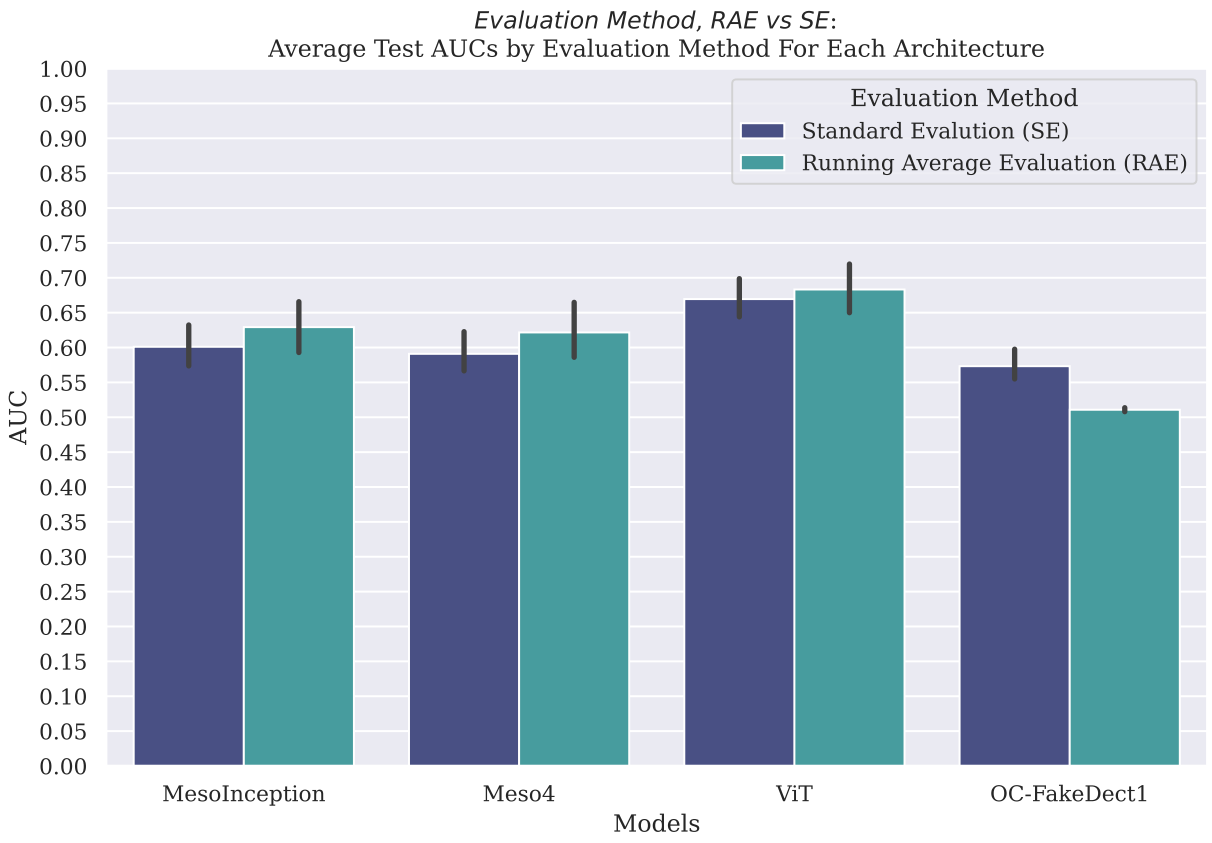


Figure 6.5: *RAE vs SE:* Average test by evaluation method for each model.

| | **Group A (Train Pre-Proc', Test Pre-Proc')** | **Group B (Train Pre-Proc', Test Pre-Proc')** | **p-Value** | **Accepted Alternate Hypothesis** $\mu_{\text{train}}, \mu_{\text{test}}$ | **Accepted Null Hypothesis** $\mu_{\text{train}}, \mu_{\text{test}}$ |
|---|---|---|---|---|---|
| 1 | (RF, RF) | (RF, AF) | 0.000700 | (RF, RF)>(RF, AF) | - |
| 2 | (AF, RF) | (RF, AF) | 0.003730 | (AF, RF)>(RF, AF) | - |
| 3 | (AF, AF) | (AF, RF) | 0.005005 | (AF, RF)>(AF, AF) | - |
| 4 | (AF, AF) | (RF, AF) | 0.062791 | - | (AF, AF)==(RF, AF) |
| 5 | (RF, RF) | (AF, RF) | 0.213224 | - | (RF, RF)==(AF, RF) |
| 6 | (RF, RF) | (AF, AF) | 0.262400 | - | (RF, RF)==(AF, AF) |

Table 6.6: Trainset-testset pre-processing t-tests (performed on generalisability list) to deduce which combinations are better than others. Note that the two left most columns indicate the means of the pre-processing distributions, e.g. (RF, RF)>(RF, AF) $\iff \mu_{\text{(RF, RF)}} > \mu_{\text{(RF, AF)}}$.

## 6.7 Pre-Processing vs Evaluation Method

Please note, the width of the Violin plots in Figure 6.7 indicates the proportion of models which achieved a given AUC (similar to a probability density function) with the black lines within the plot showing individual data points. The heights of the opaque and transparent violins indicate the actual and hypothetical maximum AUCs for each variable, respectively. The main focus should be the opaque violins, where the colour and x-axis denote categorical variables. We also use (X, Y) as shorthand to denote X pre-processing was applied to the trainset and Y pre-processing on the testset.

Figure 6.7 reinforces what has been shown in Figure 6.5, Figure 6.6, and Table 6.6. Models perform best with (RF, RF) with RAE. The most significant improvement in AUC is achieved by using RAE when (AF, AF). We already concluded this pre-processing arrangement was superior to (AF, RF). Overall, the top plot indicates that RAE closes the gap between ($x$, RF) and ($x$, AF), where $x \in \{\text{AF}, \text{RF}\}$. The bottom plot shows that RAE always improved the AUC with higher variance among all pre-processing combinations.

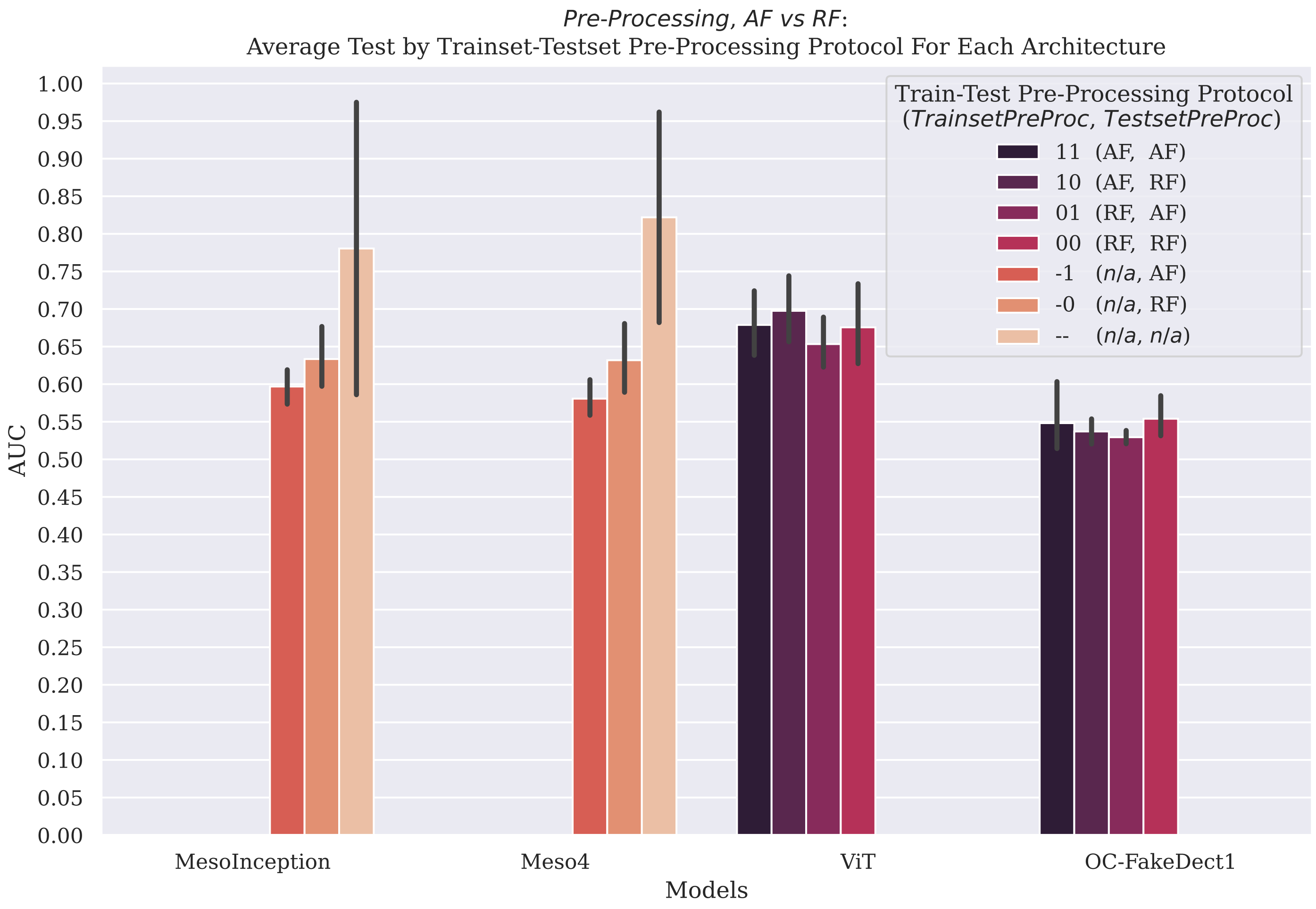
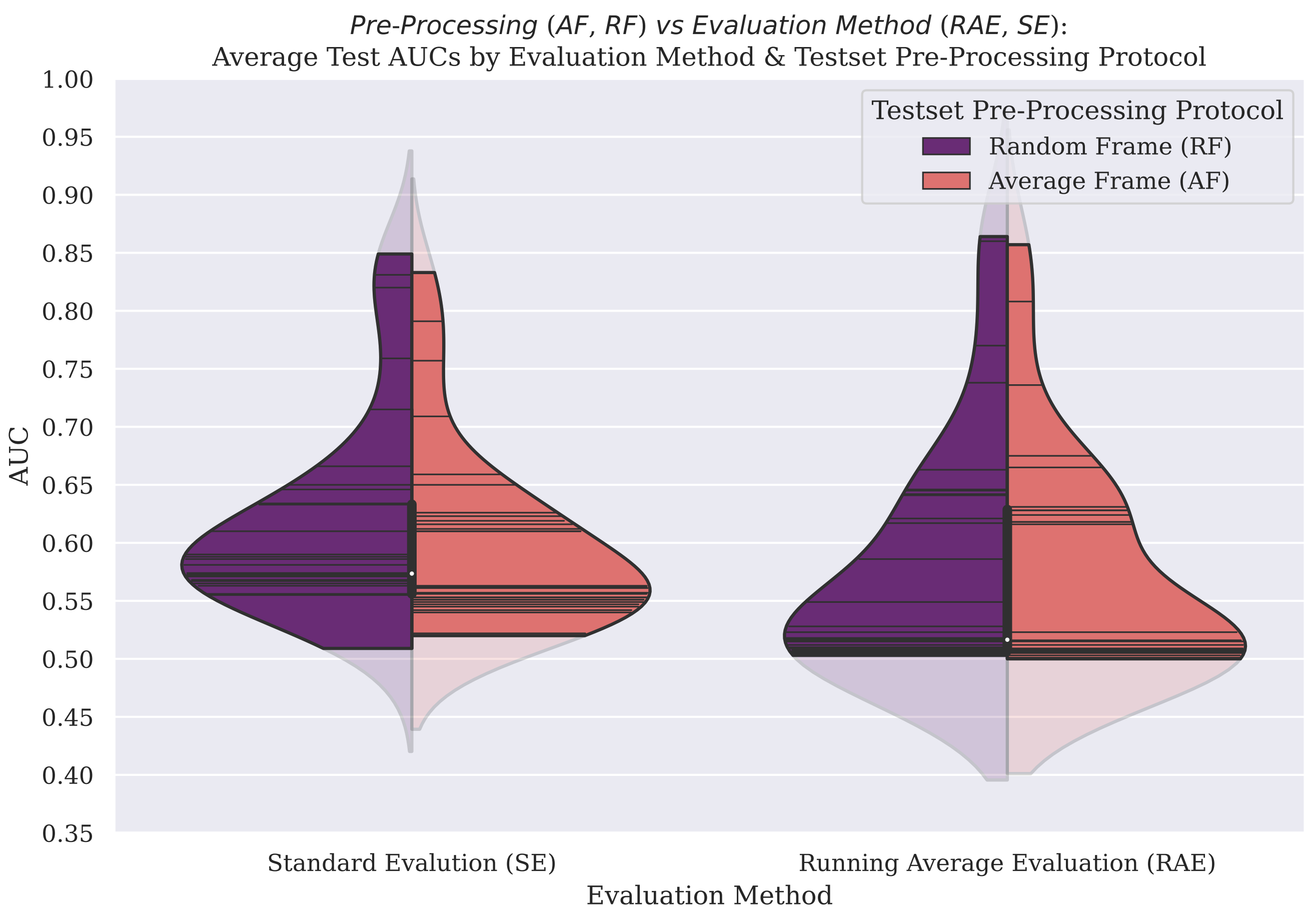


Figure 6.6: *AF vs RF:* Average test AUCs by trainset/testset pre-processing protocol for each model. $n/a$ defers to DFDB (no pre-processing by us, hence top of error bars for baselines is performance on DFDB).



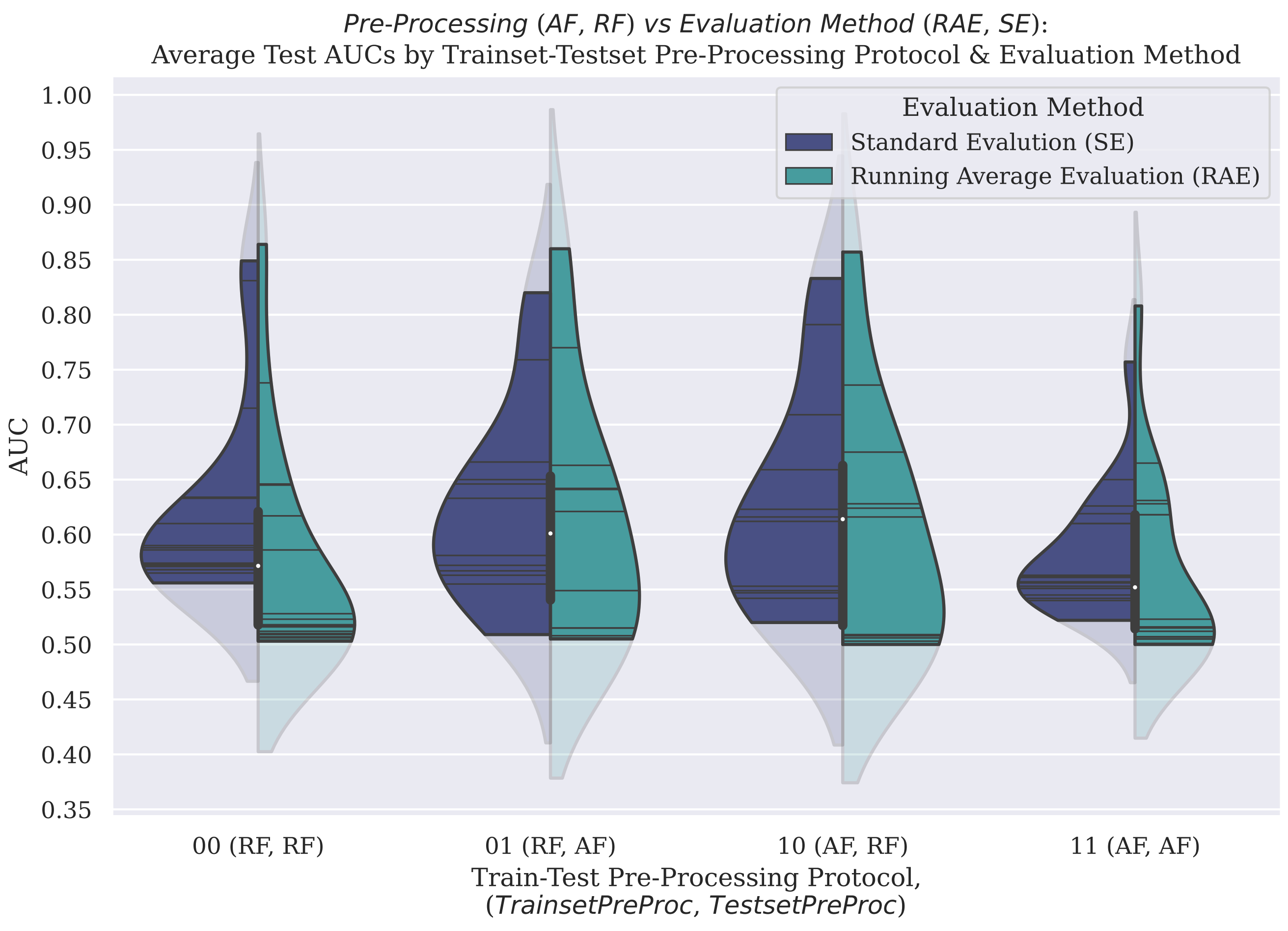


Figure 6.7: *(RAE vs SE) vs (AF vs RF):* Violin plots of average test AUCs by trainset-testset pre-processing protocol, evaluation methods, and their proportions. Notes on how to read these plots are given in Section 6.7.

# Chapter 7

# Discussion

## 7.1 Conclusion

We have shown that our ViT models outperform existing baselines in terms of generalisability (experiment no. 20) which was our main objective, and the objective of any ML model. Our hypothesis that ViT models in conjunction with RAE and AF would be best was partially correct, with our most generalisable model using this setup (but with a RF testset).

We have also shown our ViT's performance approaches SOTA on CDFv2 (experiment no. 5), falling 0.086 below the current SOTA [4] which did not use an actor-isolated train-test split. Our SOTA ViT achieves this by only training on a single Dense layer with only 769 trainable parameters, no hyperparameter tuning whatsoever, and what we believe to be a harder (actor-isolated) dataset. Thus, we fully expect our model to be the new SOTA with steps on how to achieve this in Subsection 7.2.1.

## 7.2 Future Work

### 7.2.1 Reaching & Surpassing SOTA

The following steps could be applied to make experiment no. 5 beat the current SOTA on the CDFv2: conduct a full hyperparameter search, explore classification networks architectures (i.e. how many layers, what layers to add, etc. to include in our trainable network), and increase the number of training epochs with varied network initialisations. Alternatively, use Transductive Transfer Learning. We recommend hyperparameter tuning via Hyperband [22], an automated multi-armed bandit RL approach used in part 1 which showed promising results (on validation AUC). We especially recommend this method as it provides constant exploration, which is important with a limited amount of compute given the extremely large search space. Bayesian approaches would also be permissible.

### 7.2.2 Improvements

We saw that pre-processing the FF++ datasets was difficult as it produced a varied number of fake frames despite them all being derived from the same real videos (Table 3.1). A simple way to fix this is to share the same face crop locations among all FF++ datasets which would result in an equal number of fake images produced. We grouped frames of video for pre-processing using the arbitrary number of $k = 15$ (0.5 seconds), we would like to see further work exploring how varying $k$ impacts performance on both binary and one-class detection algorithms.

In CDFv2, we only tested using our fair actor-isolated train-test split, which we believe to be harder than the traditional random shuffle used by other papers. However, we would like to see further work to test this and answer the question of whether our CNN-based face extraction method does indeed make detection datasets harder compared to the Viola-Jones approach used by Afchar et al. Perhaps performance on these datasets is more correlated to which faces were extracted compared to the algorithms used for detection. This would also help standardise the field, which is important as the speed and convincingness of new Deepfake creation methods increase.

We conducted RAE calculating the mean over the predictions, however, we do not be believe this to be the optimal way of classifying using RAE. We would advise doing this via an exponential weighting of consecutive fake predictions. E.g. when averaging over 20 predictions, having a sequence of 3 fake predictions should be a clearer indicator of a Deepfake than if there were 3 isolated fake predictions among the 20. This is similar to how human can distinguish fake videos in ultra-realistic cases, a split second of visual artifacts indicating that the video is fake despite the majority of the video fooling us.

### 7.2.3 Modern VAEs & One-Class Systems

We find Khalid et al.'s approach of determining thresholds dubious as it still requires fake examples despite aiming to just train on the real class. Furthermore, they use a latent vector of 20,000 dimensions when their input is only 100,000 dimensions. This means the model is not learning Dense hidden representations useful to classification. In addition, Reconstruction Scores have no consistent ranges and can not be used among different datasets. Normalising RMSE would solve this issue. However, if we still require fake examples as part of this model, why not train on both classes?

During this work we considered a Masked Autoencoder (MAE) [11], which masks parts of the input (similar to Self-Supervised Learning in NLP) and trains a Vision Transformer-based Autoencoder. It learns to reconstruct the missing pixels, and create generalisable hidden representations. After MAE has been trained, we discard the Decoder and perform linear probing, i.e. we use Dense layers to classify the hidden representations produced by the trained Encoder.

We actually created and trained this model on several datasets and it showed extremely promising results, with code being available in our GitHub (see Appendices). However, we chose not to consider this model as results would be too numerous and we thought that it would be better to focus on being able to thoroughly analyse the other models.

# 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. To view the table of all results ordered by test AUC, see Table 8.1.

## Code

We **strongly encourage** the reader of this paper to take a look at our GitHub, specifically `./Collate_Results.ipynb` which contains the bulk of the code used for analysing.

This GitHub contains all code used for experimentation. However, we exclude the datasets and saved weights for each trained model as they are too large in size to fit to onto our repository.

- `./_BASELINE_TESTS/` contains code for all baselines experiments.
  - `./_BASELINE_TESTS/Results/` contains individual AUCs, F1s, y_true, y_pred, etc. as text files for all baselines experiments.
- `./_DATASETS/` would usually contain all pre-processed datasets but this was omitted due to GitHub's repository file limit.
  - `./_DATASETS/FaceForensicspp/pipeline.py` contains our pre-processing algorithms as a set of helper functions within one python file. **Please see** `get_stable_faces()` function to see the $T_{70}$ protocol described in Section 4.2 and Figure 4.1.
  - `./FaceForensicspp/preprocess_ffpp.ipynb` contains the code to pre-process FF++ into AF and RF datsets. The same code was used to pre-process CDFv2.
- `./_PLOTS/` contains all plots generated for our analysis.
- `./_TRAINING/ViT/` contains all code to train and test our ViT models.
  - `./_TRAINING/ViT/Results/` contains individual AUCs, F1s, y_true, y_pred, etc. as text files for all ViT experiments.
- `./_TRAINING/OC-FakeDect-Implementation/` contains all code to train and test our OC-FakeDect1 models.

- ./_TRAINING/OC-FakeDect-Implementation/Results/ contains individual AUCs, F1s, y_true, y_pred, etc. as text files for all OC-FakeDect1 experiments.

- ./_WEIGHTS/ would usually contain all saved model weights but this was omitted due to GitHub's repository file limit.
- ./env.yaml is the environment used to produce all of these results, in contains libraries used and their source. All experiments were either done using a Tesla P100 via Google Colab Pro or an RTX 3070 at home, both utilising 16GB RAM.

If you would like to access our saved model weights, pre-proccessed datsets, or any other part of this project. Please email at zsakib.ahamed@gmail.com and we will send you the relevant files. Please note that all these files total 63GB.

## Supplementary Plots

All plots can be found via the links above: ./_PLOTS/, ./_BASELINE_TESTS/Results/, ./_TRAINING/ViT/, and ./_TRAINING/OC-FakeDect-Implementation/Results/. However we choose to add the following these plots here to for clarity.

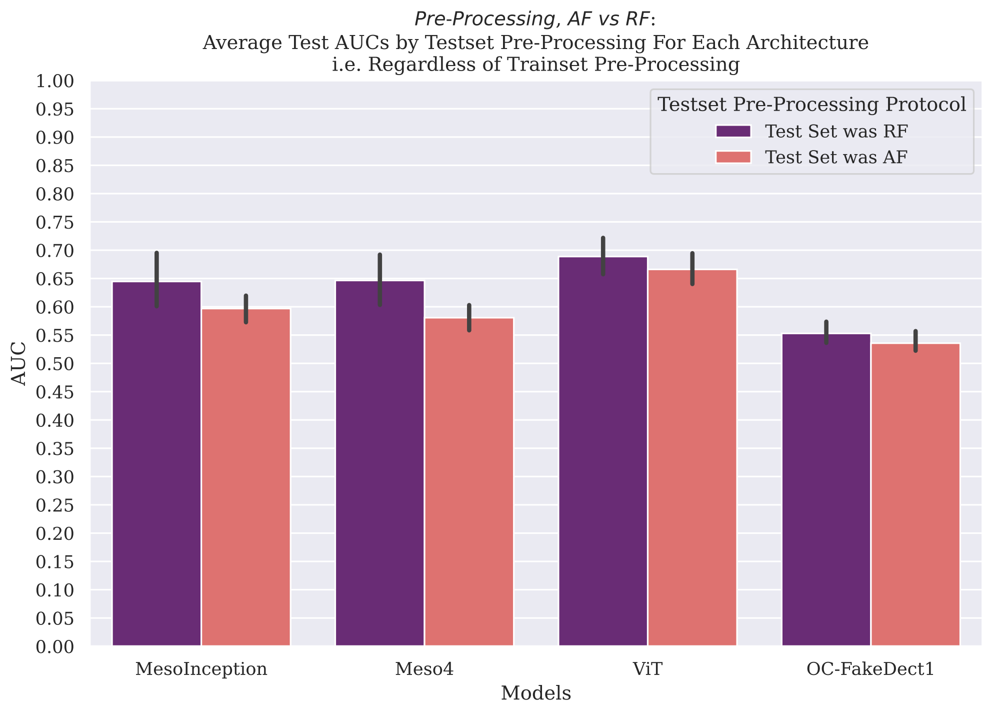


Figure 8.1: Pre-Processing, AF vs RF: Average Test AUCs by Testset Pre-Processing For Each Architecture i.e. Regardless of Trainset Pre-Processing

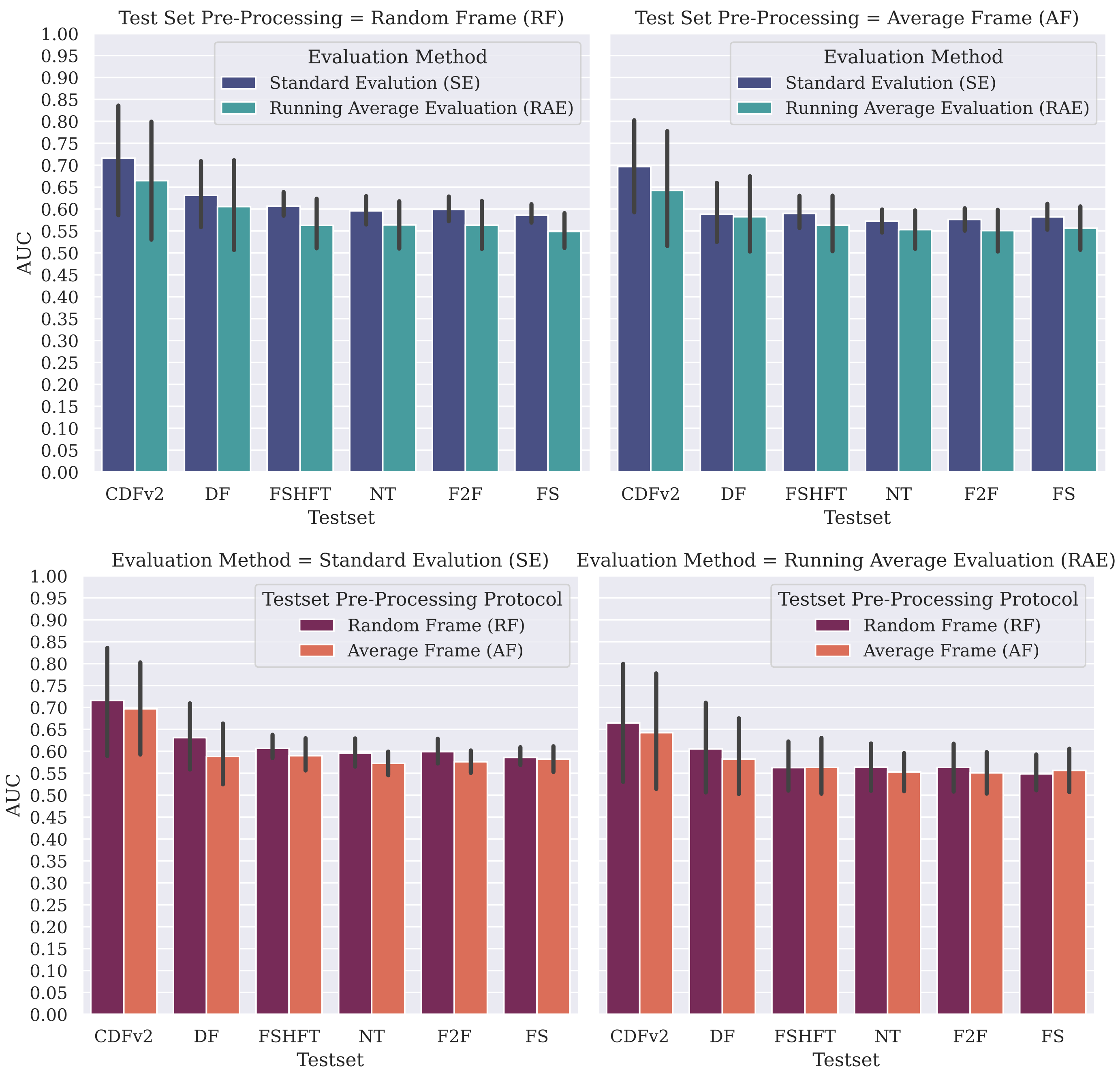


Figure 8.2: *(RAE vs SE) vs (AF vs RF):* Category plot of evaluation method, testset pre-processing method by dataset over all models. *Note: poor performance on OC-FakeDect1 skew these plots to suggest RF is better, which is not generally true.*

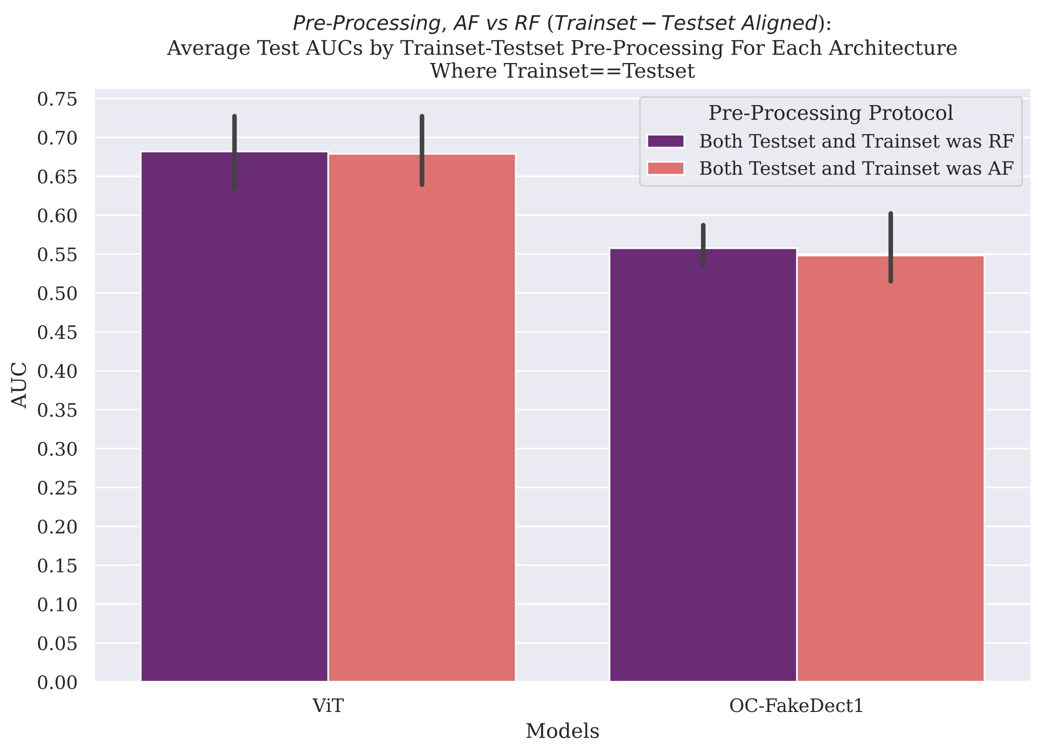


Figure 8.3: Trainset Testset Aligned: Average Test AUCs For Each Architecture Where Trainset==Testset

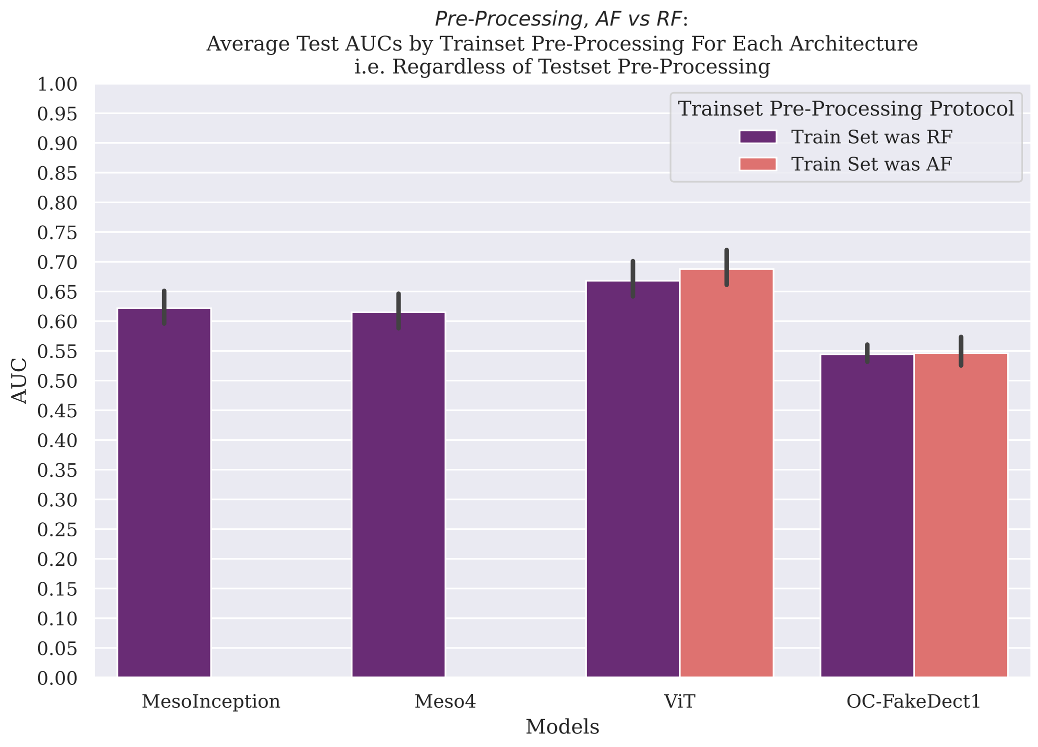


Figure 8.4: Pre-Processing, AF vs RF: Average Test AUCs by Trainset Pre-Processing For Each Architecture i.e. Regardless of Testset Pre-Processing

| | Model | Trainset | Testset | AUC | F1 | Precision | Recall | Evaluation Method ( 0=SE, 1=RAE ) | Trainset Pre-Processing ( 0=RF, 1=AF ) | Testset Pre-Processing ( 0=RF, 1=AF ) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | MesoInception | DF* | DFDB | 0.975 | 0.925 | 0.953 | 0.900 | 0 | - | - |
| 2 | Meso4 | DF* | DFDB | 0.962 | 0.904 | 0.930 | 0.880 | 0 | - | - |
| 3 | MesoInception | F2F* | F2F | 0.905 | 0.318 | 0.252 | 0.430 | 1 | - | 0 |
| 4 | Meso4 | F2F* | F2F | 0.905 | 0.128 | 0.118 | 0.139 | 1 | - | 0 |
| 5 | ViT | CDFv2 | CDFv2 | 0.864 | 0.963 | 0.930 | 0.999 | 1 | 0 | 0 |
| 6 | ViT | CDFv2 | CDFv2 | 0.860 | 0.953 | 0.944 | 0.964 | 1 | 1 | 0 |
| 7 | ViT | CDFv2 | CDFv2 | 0.857 | 0.963 | 0.932 | 0.996 | 1 | 1 | 1 |
| 8 | Meso4 | DF* | DF | 0.856 | 0.175 | 0.159 | 0.194 | 1 | - | 0 |
| 9 | OC-FakeDect1 | CDFv2 | CDFv2 | 0.849 | 0.428 | 0.770 | 0.296 | 0 | 0 | 0 |
| 10 | MesoInception | DF* | DF | 0.844 | 0.174 | 0.159 | 0.190 | 1 | - | 0 |
| 11 | OC-FakeDect1 | CDFv2 | CDFv2 | 0.833 | 0.454 | 0.785 | 0.319 | 0 | 1 | 1 |
| 12 | ViT | CDFv2 | CDFv2 | 0.831 | 0.960 | 0.930 | 0.993 | 0 | 0 | 0 |
| 13 | ViT | CDFv2 | CDFv2 | 0.820 | 0.948 | 0.943 | 0.953 | 0 | 1 | 0 |
| 14 | ViT | CDFv2 | CDFv2 | 0.808 | 0.960 | 0.922 | 1.000 | 1 | 0 | 1 |
| 15 | MesoInception | F2F* | F2F | 0.805 | 0.800 | 0.746 | 0.863 | 0 | - | 0 |
| 16 | ViT | CDFv2 | CDFv2 | 0.791 | 0.956 | 0.934 | 0.980 | 0 | 1 | 1 |
| 17 | Meso4 | F2F* | F2F | 0.790 | 0.742 | 0.803 | 0.689 | 0 | - | 0 |
| 18 | MesoInception | DF* | DF | 0.778 | 0.737 | 0.818 | 0.671 | 0 | - | 0 |
| 19 | Meso4 | DF* | DF | 0.775 | 0.735 | 0.792 | 0.685 | 0 | - | 0 |
| 20 | ViT | CDFv2 | DF | 0.770 | 0.661 | 0.511 | 0.933 | 1 | 1 | 0 |
| 21 | ViT | CDFv2 | DF | 0.759 | 0.549 | 0.512 | 0.592 | 0 | 1 | 0 |
| 22 | ViT | CDFv2 | CDFv2 | 0.757 | 0.958 | 0.922 | 0.997 | 0 | 0 | 1 |
| 23 | ViT | CDFv2 | DFDB | 0.757 | 0.958 | 0.922 | 0.997 | 0 | 0 | - |
| 24 | Meso4 | F2F* | DF | 0.751 | 0.339 | 0.286 | 0.417 | 1 | - | 0 |
| 25 | ViT | CDFv2 | DF | 0.738 | 0.617 | 0.450 | 0.982 | 1 | 0 | 0 |
| 26 | Meso4 | F2F* | F2F | 0.737 | 0.110 | 0.167 | 0.082 | 1 | - | 1 |
| 27 | ViT | CDFv2 | DF | 0.736 | 0.633 | 0.470 | 0.966 | 1 | 1 | 1 |
| 28 | ViT | CDFv2 | DF | 0.715 | 0.687 | 0.580 | 0.841 | 0 | 0 | 0 |
| 29 | ViT | CDFv2 | DF | 0.709 | 0.639 | 0.559 | 0.745 | 0 | 1 | 1 |
| 30 | Meso4 | DF* | FSHFT | 0.704 | 0.380 | 0.316 | 0.476 | 1 | - | 0 |
| 31 | MesoInception | DF* | FSHFT | 0.692 | 0.382 | 0.320 | 0.473 | 1 | - | 0 |
| 32 | MesoInception | DF* | DF | 0.692 | 0.145 | 0.211 | 0.110 | 1 | - | 1 |
| 33 | MesoInception | F2F* | DF | 0.687 | 0.550 | 0.402 | 0.870 | 1 | - | 1 |
| 34 | MesoInception | F2F* | F2F | 0.684 | 0.551 | 0.403 | 0.873 | 1 | - | 1 |
| 35 | Meso4 | F2F* | DFDB | 0.682 | 0.750 | 0.637 | 0.912 | 0 | - | - |
| 36 | Meso4 | F2F* | F2F | 0.678 | 0.527 | 0.793 | 0.394 | 0 | - | 1 |
| 37 | ViT | CDFv2 | FSHFT | 0.675 | 0.621 | 0.463 | 0.940 | 1 | 1 | 1 |
| 38 | MesoInception | F2F* | FSHFT | 0.675 | 0.609 | 0.439 | 0.995 | 1 | - | 0 |
| 39 | ViT | CDFv2 | DFDB | 0.674 | 0.501 | 0.490 | 0.511 | 0 | 1 | - |
| 40 | MesoInception | F2F* | CDFv2 | 0.673 | 0.908 | 0.915 | 0.902 | 1 | - | 1 |
| 41 | Meso4 | F2F* | DF | 0.672 | 0.695 | 0.701 | 0.689 | 0 | - | 0 |
| 42 | Meso4 | DF* | DF | 0.669 | 0.091 | 0.200 | 0.059 | 1 | - | 1 |
| 43 | Meso4 | F2F* | CDFv2 | 0.667 | 0.512 | 0.869 | 0.363 | 1 | - | 0 |
| 44 | ViT | CDFv2 | FSHFT | 0.666 | 0.563 | 0.536 | 0.592 | 0 | 1 | 0 |
| 45 | ViT | CDFv2 | DF | 0.665 | 0.600 | 0.430 | 0.996 | 1 | 0 | 1 |
| 46 | ViT | CDFv2 | FSHFT | 0.663 | 0.624 | 0.491 | 0.859 | 1 | 1 | 0 |
| 47 | MesoInception | DF* | FSHFT | 0.662 | 0.144 | 0.209 | 0.109 | 1 | - | 1 |
| 48 | MesoInception | DF* | FSHFT | 0.660 | 0.685 | 0.700 | 0.671 | 0 | - | 0 |
| 49 | ViT | CDFv2 | FSHFT | 0.659 | 0.644 | 0.567 | 0.745 | 0 | 1 | 1 |
| 50 | ViT | CDFv2 | DF | 0.650 | 0.734 | 0.602 | 0.940 | 0 | 0 | 1 |
| 51 | ViT | CDFv2 | NT | 0.650 | 0.594 | 0.597 | 0.592 | 0 | 1 | 0 |
| 52 | MesoInception | F2F* | DF | 0.649 | 0.738 | 0.636 | 0.881 | 0 | - | 1 |
| 53 | MesoInception | F2F* | CDFv2 | 0.648 | 0.945 | 0.918 | 0.974 | 1 | - | 0 |
| 54 | ViT | CDFv2 | F2F | 0.646 | 0.601 | 0.440 | 0.946 | 1 | 0 | 0 |
| 55 | ViT | CDFv2 | F2F | 0.646 | 0.567 | 0.544 | 0.592 | 0 | 1 | 0 |
| 56 | Meso4 | DF* | FSHFT | 0.645 | 0.683 | 0.680 | 0.685 | 0 | - | 0 |

Table 8.1: All Results

| | **Model** | **Trainset** | **Testset** | **AUC** | **F1** | **Precision** | **Recall** | **Evaluation Method ( 0=SE, 1=RAE )** | **Trainset Pre-Processing ( 0=RF, 1=AF )** | **Testset Pre-Processing ( 0=RF, 1=AF )** |
|---|---|---|---|---|---|---|---|---|---|---|
| 57 | ViT | CDFv2 | NT | 0.645 | 0.602 | 0.441 | 0.948 | 1 | 0 | 0 |
| 58 | ViT | CDFv2 | NT | 0.642 | 0.606 | 0.480 | 0.822 | 1 | 1 | 0 |
| 59 | ViT | CDFv2 | F2F | 0.641 | 0.605 | 0.479 | 0.821 | 1 | 1 | 0 |
| 60 | MesoInception | DF* | DF | 0.639 | 0.462 | 0.784 | 0.328 | 0 | - | 1 |
| 61 | ViT | CDFv2 | NT | 0.634 | 0.730 | 0.645 | 0.841 | 0 | 0 | 0 |
| 62 | ViT | CDFv2 | F2F | 0.633 | 0.694 | 0.590 | 0.841 | 0 | 0 | 0 |
| 63 | ViT | CDFv2 | FS | 0.633 | 0.597 | 0.603 | 0.592 | 0 | 1 | 0 |
| 64 | Meso4 | F2F* | DF | 0.632 | 0.181 | 0.255 | 0.140 | 1 | - | 1 |
| 65 | ViT | CDFv2 | FS | 0.631 | 0.599 | 0.429 | 0.994 | 1 | 0 | 1 |
| 66 | Meso4 | DF* | FSHFT | 0.631 | 0.110 | 0.234 | 0.072 | 1 | - | 1 |
| 67 | MesoInception | F2F* | CDFv2 | 0.630 | 0.856 | 0.910 | 0.809 | 0 | - | 1 |
| 68 | ViT | CDFv2 | FS | 0.628 | 0.609 | 0.457 | 0.915 | 1 | 1 | 1 |
| 69 | ViT | CDFv2 | FSHFT | 0.628 | 0.600 | 0.430 | 0.996 | 1 | 0 | 1 |
| 70 | MesoInception | DF* | NT | 0.628 | 0.445 | 0.363 | 0.573 | 1 | - | 0 |
| 71 | MesoInception | F2F* | F2F | 0.627 | 0.742 | 0.641 | 0.881 | 0 | - | 1 |
| 72 | OC-FakeDect1 | CDFv2 | DFDB | 0.627 | 0.338 | 0.291 | 0.405 | 0 | 0 | - |
| 73 | ViT | CDFv2 | FS | 0.626 | 0.773 | 0.657 | 0.940 | 0 | 0 | 1 |
| 74 | Meso4 | DF* | DF | 0.626 | 0.379 | 0.772 | 0.251 | 0 | - | 1 |
| 75 | ViT | CDFv2 | F2F | 0.624 | 0.605 | 0.454 | 0.905 | 1 | 1 | 1 |
| 76 | ViT | CDFv2 | FS | 0.623 | 0.683 | 0.630 | 0.745 | 0 | 1 | 1 |
| 77 | MesoInception | F2F* | CDFv2 | 0.622 | 0.921 | 0.915 | 0.927 | 0 | - | 0 |
| 78 | OC-FakeDect1 | CDFv2 | DFDB | 0.622 | 0.348 | 0.297 | 0.420 | 0 | 1 | - |
| 79 | ViT | CDFv2 | FS | 0.621 | 0.604 | 0.478 | 0.818 | 1 | 1 | 0 |
| 80 | MesoInception | DF* | FSHFT | 0.620 | 0.458 | 0.760 | 0.328 | 0 | - | 1 |
| 81 | ViT | CDFv2 | FSHFT | 0.619 | 0.735 | 0.603 | 0.940 | 0 | 0 | 1 |
| 82 | ViT | CDFv2 | NT | 0.618 | 0.598 | 0.428 | 0.989 | 1 | 0 | 1 |
| 83 | ViT | CDFv2 | F2F | 0.618 | 0.597 | 0.428 | 0.989 | 1 | 0 | 1 |
| 84 | ViT | CDFv2 | FSHFT | 0.617 | 0.602 | 0.441 | 0.948 | 1 | 0 | 0 |
| 85 | ViT | CDFv2 | F2F | 0.616 | 0.650 | 0.577 | 0.745 | 0 | 1 | 1 |
| 86 | ViT | CDFv2 | NT | 0.616 | 0.599 | 0.451 | 0.892 | 1 | 1 | 1 |
| 87 | ViT | CDFv2 | NT | 0.612 | 0.684 | 0.632 | 0.745 | 0 | 1 | 1 |
| 88 | MesoInception | DF* | NT | 0.611 | 0.199 | 0.275 | 0.156 | 1 | - | 1 |
| 89 | ViT | CDFv2 | NT | 0.610 | 0.775 | 0.658 | 0.940 | 0 | 0 | 1 |
| 90 | ViT | CDFv2 | F2F | 0.610 | 0.736 | 0.605 | 0.940 | 0 | 0 | 1 |
| 91 | ViT | CDFv2 | FSHFT | 0.610 | 0.694 | 0.591 | 0.841 | 0 | 0 | 0 |
| 92 | MesoInception | DF* | NT | 0.609 | 0.694 | 0.719 | 0.671 | 0 | - | 0 |
| 93 | Meso4 | F2F* | DF | 0.605 | 0.511 | 0.728 | 0.394 | 0 | - | 1 |
| 94 | MesoInception | F2F* | FSHFT | 0.603 | 0.696 | 0.584 | 0.863 | 0 | - | 0 |
| 95 | MesoInception | DF* | FS | 0.602 | 0.211 | 0.287 | 0.166 | 1 | - | 1 |
| 96 | Meso4 | DF* | NT | 0.599 | 0.455 | 0.368 | 0.598 | 1 | - | 0 |
| 97 | MesoInception | F2F* | DF | 0.597 | 0.711 | 0.605 | 0.863 | 0 | - | 0 |
| 98 | Meso4 | DF* | FSHFT | 0.596 | 0.372 | 0.720 | 0.251 | 0 | - | 1 |
| 99 | MesoInception | F2F* | DF | 0.595 | 0.592 | 0.429 | 0.957 | 1 | - | 0 |
| 100 | Meso4 | F2F* | CDFv2 | 0.594 | 0.585 | 0.898 | 0.434 | 0 | - | 0 |
| 101 | OC-FakeDect1 | DFDB | FSHFT | 0.590 | 0.552 | 0.460 | 0.690 | 0 | - | 0 |
| 102 | Meso4 | F2F* | NT | 0.589 | 0.491 | 0.389 | 0.664 | 1 | - | 0 |
| 103 | MesoInception | DF* | CDFv2 | 0.589 | 0.490 | 0.900 | 0.336 | 1 | - | 0 |
| 104 | ViT | CDFv2 | FS | 0.588 | 0.731 | 0.647 | 0.841 | 0 | 0 | 0 |
| 105 | MesoInception | F2F* | DFDB | 0.586 | 0.734 | 0.613 | 0.914 | 0 | - | - |
| 106 | ViT | CDFv2 | FS | 0.586 | 0.600 | 0.440 | 0.945 | 1 | 0 | 0 |
| 107 | OC-FakeDect1 | CDFv2 | FSHFT | 0.586 | 0.549 | 0.456 | 0.690 | 0 | 0 | 0 |
| 108 | Meso4 | DF* | CDFv2 | 0.585 | 0.319 | 0.861 | 0.196 | 1 | - | 0 |
| 109 | MesoInception | F2F* | FS | 0.584 | 0.744 | 0.655 | 0.863 | 0 | - | 0 |
| 110 | MesoInception | DF* | NT | 0.584 | 0.459 | 0.766 | 0.328 | 0 | - | 1 |
| 111 | OC-FakeDect1 | CDFv2 | FSHFT | 0.581 | 0.544 | 0.452 | 0.683 | 0 | 1 | 0 |
| 112 | MesoInception | F2F* | NT | 0.580 | 0.608 | 0.438 | 0.992 | 1 | - | 0 |

| | **Model** | **Trainset** | **Testset** | **AUC** | **F1** | **Precision** | **Recall** | **Evaluation Method ( 0=SE, 1=RAE )** | **Trainset Pre-Processing ( 0=RF, 1=AF )** | **Testset Pre-Processing ( 0=RF, 1=AF )** |
|---|---|---|---|---|---|---|---|---|---|---|
| 113 | MesoInception | DF* | F2F | 0.580 | 0.223 | 0.301 | 0.177 | 1 | - | 1 |
| 114 | MesoInception | DF* | FS | 0.579 | 0.457 | 0.753 | 0.328 | 0 | - | 1 |
| 115 | Meso4 | F2F* | FSHFT | 0.577 | 0.508 | 0.400 | 0.694 | 1 | - | 0 |
| 116 | Meso4 | DF* | NT | 0.577 | 0.172 | 0.330 | 0.116 | 1 | - | 1 |
| 117 | OC-FakeDect1 | DFDB | FS | 0.574 | 0.495 | 0.389 | 0.681 | 0 | - | 0 |
| 118 | OC-FakeDect1 | DFDB | F2F | 0.573 | 0.536 | 0.446 | 0.671 | 0 | - | 0 |
| 119 | OC-FakeDect1 | DFDB | DFDB | 0.573 | 0.388 | 0.336 | 0.459 | 0 | - | - |
| 120 | OC-FakeDect1 | CDFv2 | F2F | 0.572 | 0.536 | 0.444 | 0.676 | 0 | 1 | 0 |
| 121 | OC-FakeDect1 | CDFv2 | F2F | 0.572 | 0.536 | 0.444 | 0.675 | 0 | 0 | 0 |
| 122 | OC-FakeDect1 | DFDB | CDFv2 | 0.571 | 0.766 | 0.935 | 0.648 | 0 | - | 0 |
| 123 | Meso4 | DF* | NT | 0.571 | 0.691 | 0.697 | 0.685 | 0 | - | 0 |
| 124 | OC-FakeDect1 | DFDB | DF | 0.571 | 0.535 | 0.444 | 0.675 | 0 | - | 0 |
| 125 | Meso4 | F2F* | FS | 0.570 | 0.273 | 0.352 | 0.222 | 1 | - | 1 |
| 126 | Meso4 | F2F* | NT | 0.569 | 0.690 | 0.690 | 0.689 | 0 | - | 0 |
| 127 | Meso4 | DF* | FS | 0.569 | 0.166 | 0.322 | 0.112 | 1 | - | 1 |
| 128 | OC-FakeDect1 | CDFv2 | FS | 0.568 | 0.491 | 0.384 | 0.679 | 0 | 0 | 0 |
| 129 | OC-FakeDect1 | DFDB | NT | 0.568 | 0.489 | 0.385 | 0.669 | 0 | - | 0 |
| 130 | OC-FakeDect1 | CDFv2 | FS | 0.567 | 0.489 | 0.383 | 0.677 | 0 | 1 | 0 |
| 131 | MesoInception | F2F* | FS | 0.566 | 0.595 | 0.429 | 0.971 | 1 | - | 1 |
| 132 | OC-FakeDect1 | CDFv2 | NT | 0.565 | 0.486 | 0.382 | 0.668 | 0 | 0 | 0 |
| 133 | OC-FakeDect1 | DFDB | FSHFT | 0.563 | 0.527 | 0.438 | 0.663 | 0 | - | 1 |
| 134 | Meso4 | DF* | CDFv2 | 0.563 | 0.499 | 0.896 | 0.346 | 0 | - | 0 |
| 135 | OC-FakeDect1 | CDFv2 | NT | 0.563 | 0.484 | 0.380 | 0.666 | 0 | 1 | 0 |
| 136 | Meso4 | F2F* | FSHFT | 0.562 | 0.658 | 0.630 | 0.689 | 0 | - | 0 |
| 137 | OC-FakeDect1 | DFDB | FS | 0.562 | 0.484 | 0.380 | 0.666 | 0 | - | 1 |
| 138 | Meso4 | F2F* | NT | 0.562 | 0.277 | 0.356 | 0.226 | 1 | - | 1 |
| 139 | OC-FakeDect1 | CDFv2 | CDFv2 | 0.561 | 0.650 | 0.901 | 0.508 | 0 | 0 | 1 |
| 140 | MesoInception | DF* | CDFv2 | 0.561 | 0.535 | 0.900 | 0.381 | 0 | - | 0 |
| 141 | Meso4 | F2F* | FS | 0.561 | 0.510 | 0.723 | 0.394 | 0 | - | 1 |
| 142 | Meso4 | F2F* | CDFv2 | 0.561 | 0.213 | 0.891 | 0.121 | 1 | - | 1 |
| 143 | MesoInception | DF* | F2F | 0.559 | 0.444 | 0.691 | 0.328 | 0 | - | 1 |
| 144 | Meso4 | DF* | NT | 0.558 | 0.373 | 0.724 | 0.251 | 0 | - | 1 |
| 145 | MesoInception | F2F* | FSHFT | 0.557 | 0.596 | 0.430 | 0.974 | 1 | - | 1 |
| 146 | Meso4 | DF* | FS | 0.557 | 0.559 | 0.433 | 0.787 | 1 | - | 0 |
| 147 | OC-FakeDect1 | DFDB | F2F | 0.557 | 0.521 | 0.433 | 0.654 | 0 | - | 1 |
| 148 | OC-FakeDect1 | CDFv2 | FSHFT | 0.556 | 0.522 | 0.431 | 0.661 | 0 | 0 | 1 |
| 149 | OC-FakeDect1 | CDFv2 | DF | 0.556 | 0.522 | 0.431 | 0.660 | 0 | 0 | 0 |
| 150 | Meso4 | F2F* | NT | 0.556 | 0.510 | 0.723 | 0.394 | 0 | - | 1 |
| 151 | OC-FakeDect1 | CDFv2 | DF | 0.555 | 0.521 | 0.430 | 0.659 | 0 | 1 | 0 |
| 152 | Meso4 | DF* | FS | 0.554 | 0.373 | 0.723 | 0.251 | 0 | - | 1 |
| 153 | MesoInception | F2F* | FS | 0.553 | 0.753 | 0.658 | 0.881 | 0 | - | 1 |
| 154 | OC-FakeDect1 | CDFv2 | FSHFT | 0.553 | 0.519 | 0.429 | 0.657 | 0 | 1 | 1 |
| 155 | OC-FakeDect1 | DFDB | NT | 0.553 | 0.475 | 0.373 | 0.653 | 0 | - | 1 |
| 156 | OC-FakeDect1 | CDFv2 | F2F | 0.551 | 0.517 | 0.427 | 0.655 | 0 | 0 | 1 |
| 157 | MesoInception | DF* | F2F | 0.551 | 0.506 | 0.403 | 0.678 | 1 | - | 0 |
| 158 | OC-FakeDect1 | CDFv2 | FS | 0.551 | 0.476 | 0.371 | 0.661 | 0 | 0 | 1 |
| 159 | Meso4 | DF* | FS | 0.549 | 0.668 | 0.651 | 0.685 | 0 | - | 0 |
| 160 | MesoInception | DF* | F2F | 0.549 | 0.651 | 0.633 | 0.671 | 0 | - | 0 |
| 161 | OC-FakeDect1 | CDFv2 | CDFv2 | 0.549 | 0.629 | 0.905 | 0.482 | 1 | 1 | 0 |
| 162 | OC-FakeDect1 | CDFv2 | FS | 0.549 | 0.473 | 0.369 | 0.659 | 0 | 1 | 1 |
| 163 | Meso4 | F2F* | CDFv2 | 0.549 | 0.327 | 0.895 | 0.200 | 0 | - | 1 |
| 164 | OC-FakeDect1 | CDFv2 | F2F | 0.547 | 0.513 | 0.424 | 0.652 | 0 | 1 | 1 |
| 165 | Meso4 | F2F* | FSHFT | 0.545 | 0.495 | 0.665 | 0.394 | 0 | - | 1 |
| 166 | OC-FakeDect1 | CDFv2 | NT | 0.545 | 0.469 | 0.367 | 0.651 | 0 | 0 | 1 |
| 167 | Meso4 | F2F* | FSHFT | 0.544 | 0.285 | 0.364 | 0.234 | 1 | - | 1 |
| 168 | OC-FakeDect1 | DFDB | CDFv2 | 0.542 | 0.725 | 0.929 | 0.594 | 0 | - | 1 |

| | Model | Trainset | Testset | AUC | F1 | Precision | Recall | Evaluation Method ( 0=SE, 1=RAE ) | Trainset Pre-Processing ( 0=RF, 1=AF ) | Testset Pre-Processing ( 0=RF, 1=AF ) |
|---|---|---|---|---|---|---|---|---|---|---|
| 169 | OC-FakeDect1 | CDFv2 | NT | 0.542 | 0.466 | 0.364 | 0.648 | 0 | 1 | 1 |
| 170 | MesoInception | DF* | FS | 0.541 | 0.554 | 0.434 | 0.769 | 1 | - | 0 |
| 171 | OC-FakeDect1 | DFDB | DF | 0.540 | 0.504 | 0.418 | 0.632 | 0 | - | 1 |
| 172 | MesoInception | F2F* | NT | 0.539 | 0.738 | 0.644 | 0.863 | 0 | - | 0 |
| 173 | Meso4 | F2F* | FS | 0.536 | 0.524 | 0.410 | 0.724 | 1 | - | 0 |
| 174 | Meso4 | F2F* | FS | 0.528 | 0.683 | 0.678 | 0.689 | 0 | - | 0 |
| 175 | MesoInception | DF* | FS | 0.528 | 0.663 | 0.657 | 0.671 | 0 | - | 0 |
| 176 | OC-FakeDect1 | DFDB | CDFv2 | 0.528 | 0.662 | 0.928 | 0.515 | 1 | - | 0 |
| 177 | Meso4 | DF* | F2F | 0.524 | 0.217 | 0.389 | 0.150 | 1 | - | 1 |
| 178 | MesoInception | F2F* | FSHFT | 0.523 | 0.714 | 0.601 | 0.881 | 0 | - | 1 |
| 179 | OC-FakeDect1 | CDFv2 | CDFv2 | 0.523 | 0.680 | 0.926 | 0.538 | 1 | 0 | 1 |
| 180 | OC-FakeDect1 | CDFv2 | CDFv2 | 0.523 | 0.641 | 0.913 | 0.494 | 1 | 0 | 0 |
| 181 | OC-FakeDect1 | CDFv2 | DF | 0.522 | 0.489 | 0.403 | 0.621 | 0 | 0 | 1 |
| 182 | OC-FakeDect1 | CDFv2 | DF | 0.520 | 0.488 | 0.402 | 0.621 | 0 | 1 | 1 |
| 183 | MesoInception | F2F* | FS | 0.519 | 0.601 | 0.434 | 0.977 | 1 | - | 0 |
| 184 | OC-FakeDect1 | CDFv2 | F2F | 0.518 | 0.441 | 0.406 | 0.484 | 1 | 0 | 0 |
| 185 | Meso4 | DF* | F2F | 0.518 | 0.359 | 0.632 | 0.251 | 0 | - | 1 |
| 186 | OC-FakeDect1 | CDFv2 | FS | 0.517 | 0.475 | 0.440 | 0.517 | 1 | 0 | 0 |
| 187 | OC-FakeDect1 | CDFv2 | NT | 0.516 | 0.474 | 0.439 | 0.517 | 1 | 0 | 1 |
| 188 | OC-FakeDect1 | CDFv2 | FSHFT | 0.516 | 0.446 | 0.407 | 0.492 | 1 | 0 | 0 |
| 189 | MesoInception | DF* | CDFv2 | 0.516 | 0.370 | 0.899 | 0.233 | 0 | - | 1 |
| 190 | OC-FakeDect1 | DFDB | CDFv2 | 0.515 | 0.667 | 0.924 | 0.521 | 1 | - | 1 |
| 191 | OC-FakeDect1 | CDFv2 | NT | 0.515 | 0.472 | 0.438 | 0.511 | 1 | 1 | 0 |
| 192 | OC-FakeDect1 | CDFv2 | FS | 0.515 | 0.469 | 0.439 | 0.504 | 1 | 1 | 0 |
| 193 | MesoInception | DF* | CDFv2 | 0.515 | 0.279 | 0.877 | 0.166 | 1 | - | 1 |
| 194 | OC-FakeDect1 | DFDB | FSHFT | 0.512 | 0.447 | 0.411 | 0.489 | 1 | - | 0 |
| 195 | OC-FakeDect1 | DFDB | FS | 0.512 | 0.443 | 0.411 | 0.481 | 1 | - | 1 |
| 196 | MesoInception | F2F* | NT | 0.511 | 0.754 | 0.659 | 0.881 | 0 | - | 1 |
| 197 | Meso4 | DF* | F2F | 0.511 | 0.643 | 0.606 | 0.685 | 0 | - | 0 |
| 198 | Meso4 | DF* | F2F | 0.510 | 0.545 | 0.425 | 0.761 | 1 | - | 0 |
| 199 | OC-FakeDect1 | DFDB | NT | 0.510 | 0.465 | 0.434 | 0.503 | 1 | - | 0 |
| 200 | OC-FakeDect1 | CDFv2 | CDFv2 | 0.509 | 0.694 | 0.916 | 0.558 | 0 | 1 | 0 |
| 201 | OC-FakeDect1 | CDFv2 | CDFv2 | 0.509 | 0.671 | 0.923 | 0.527 | 1 | 1 | 1 |
| 202 | OC-FakeDect1 | CDFv2 | DF | 0.509 | 0.447 | 0.414 | 0.485 | 1 | 0 | 0 |
| 203 | OC-FakeDect1 | CDFv2 | NT | 0.508 | 0.465 | 0.431 | 0.505 | 1 | 1 | 1 |
| 204 | OC-FakeDect1 | CDFv2 | F2F | 0.508 | 0.465 | 0.431 | 0.504 | 1 | 1 | 0 |
| 205 | OC-FakeDect1 | CDFv2 | FSHFT | 0.508 | 0.455 | 0.415 | 0.504 | 1 | 1 | 1 |
| 206 | Meso4 | DF* | CDFv2 | 0.508 | 0.231 | 0.908 | 0.132 | 0 | - | 1 |
| 207 | OC-FakeDect1 | CDFv2 | F2F | 0.507 | 0.467 | 0.430 | 0.512 | 1 | 0 | 1 |
| 208 | OC-FakeDect1 | DFDB | NT | 0.507 | 0.453 | 0.416 | 0.496 | 1 | - | 1 |
| 209 | OC-FakeDect1 | CDFv2 | DF | 0.507 | 0.451 | 0.417 | 0.491 | 1 | 0 | 1 |
| 210 | OC-FakeDect1 | CDFv2 | NT | 0.507 | 0.450 | 0.416 | 0.489 | 1 | 0 | 0 |
| 211 | Meso4 | DF* | CDFv2 | 0.507 | 0.053 | 0.830 | 0.027 | 1 | - | 1 |
| 212 | MesoInception | F2F* | NT | 0.506 | 0.594 | 0.429 | 0.969 | 1 | - | 1 |
| 213 | OC-FakeDect1 | DFDB | DF | 0.506 | 0.465 | 0.429 | 0.506 | 1 | - | 0 |
| 214 | OC-FakeDect1 | CDFv2 | FS | 0.506 | 0.456 | 0.417 | 0.502 | 1 | 1 | 1 |
| 215 | OC-FakeDect1 | CDFv2 | FSHFT | 0.506 | 0.453 | 0.418 | 0.495 | 1 | 1 | 0 |
| 216 | OC-FakeDect1 | CDFv2 | FSHFT | 0.505 | 0.467 | 0.428 | 0.515 | 1 | 0 | 1 |
| 217 | OC-FakeDect1 | CDFv2 | DF | 0.505 | 0.464 | 0.428 | 0.508 | 1 | 1 | 0 |
| 218 | OC-FakeDect1 | CDFv2 | FS | 0.505 | 0.461 | 0.428 | 0.500 | 1 | 0 | 1 |
| 219 | OC-FakeDect1 | DFDB | F2F | 0.505 | 0.452 | 0.418 | 0.492 | 1 | - | 1 |
| 220 | OC-FakeDect1 | DFDB | FS | 0.504 | 0.455 | 0.419 | 0.499 | 1 | - | 0 |
| 221 | OC-FakeDect1 | DFDB | F2F | 0.503 | 0.463 | 0.426 | 0.508 | 1 | - | 0 |
| 222 | OC-FakeDect1 | CDFv2 | DF | 0.503 | 0.453 | 0.420 | 0.490 | 1 | 1 | 1 |
| 223 | OC-FakeDect1 | DFDB | DF | 0.501 | 0.457 | 0.425 | 0.496 | 1 | - | 1 |
| 224 | OC-FakeDect1 | DFDB | FSHFT | 0.500 | 0.459 | 0.423 | 0.502 | 1 | - | 1 |
| 225 | OC-FakeDect1 | CDFv2 | F2F | 0.500 | 0.459 | 0.423 | 0.501 | 1 | 1 | 1 |

# Bibliography

[1] 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.

[2] 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.

[3] Thomas M. Breuel. The effects of hyperparameters on SGD training of neural networks. *CoRR*, abs/1508.02788, 2015.

[4] Hong-Shuo Chen, Mozhdeh Rouhsedaghat, Hamza Ghani, Shuowen Hu, Suya You, and C-C Jay Kuo. Defakehop: A light-weight high-performance deepfake detector. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2021.

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

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[7] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.

[8] 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).

[9] Adam (Ageitgey) Geitgey. Ageitgey/face_recognition: The world's simplest facial recognition api for python and the command line.

[10] Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246, 2016.

[11] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021.

[12] 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.

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

[14] Hasam Khalid and Simon S Woo. Oc-fakedect: Classifying deepfakes using one-class variational autoencoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 656–657, 2020.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

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

[17] 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.

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

[19] Hanting Li, Mingzhe Sui, Feng Zhao, Zhengjun Zha, and Feng Wu. Mvit: Mask vision transformer for facial expression recognition in the wild. *CoRR*, abs/2106.04520, 2021.

[20] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[21] Lingzhi Li, Jianmin Bao, Hao Yang, Dong Chen, and Fang Wen. Advancing high fidelity identity swapping for forgery detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

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

[23] 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.

[24] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[26] 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.

[27] Hiromitsu Nishizaki. Data augmentation and feature extraction using variational autoencoder for acoustic modeling. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1222–1227. IEEE, 2017.

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

[29] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *Advances in Neural Information Processing Systems*, 34, 2021.

[30] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[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] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

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

[36] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much, 2017.

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

[38] Tesla. Tesla ai day, 2021.

[39] 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.

[40] 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.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[42] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–I. Ieee, 2001.

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

[45] Bojia Zi, Minghao Chang, Jingjing Chen, Xingjun Ma, and Yu-Gang Jiang. Wild-deepfake: A challenging real-world dataset for deepfake detection. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 2382–2390, 2020.