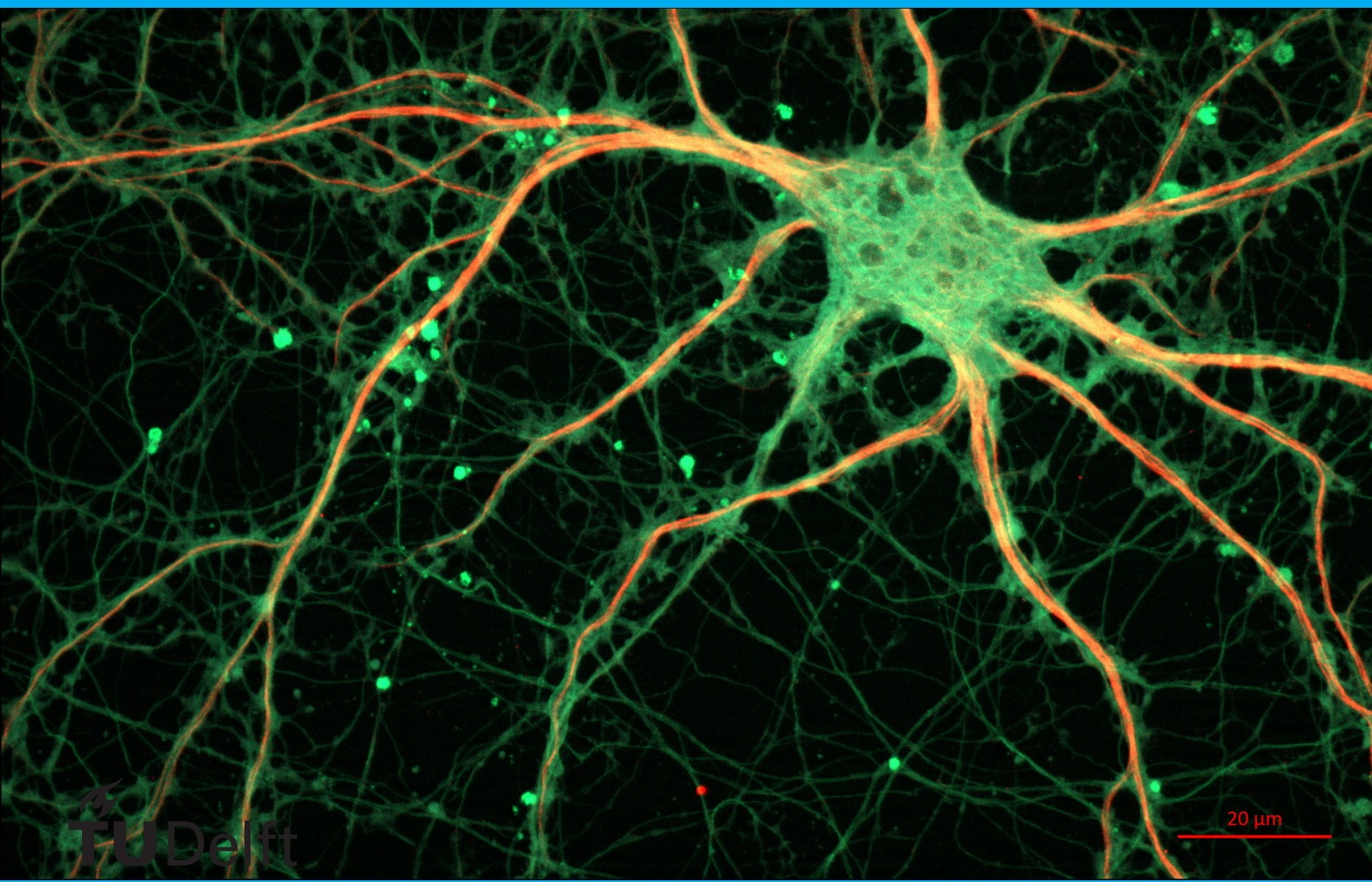
Deepfake Detection Using Convolutional Neural Networks

Working Towards Understanding the Effects of Design Choices

B.X. van Leeuwen



Deepfake Detection Using Convolutional Neural Networks

Working Towards Understanding the Effects of Design Choices

by

B.X. van Leeuwen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday 19 November 2020 at 13:30.

Student number:	4316983	
Project duration:	20 January 2020 – 19 November 2020	
Thesis committee:	Prof. dr. ir. H. X. Lin,	TU Delft, supervisor
	Prof. dr. ir. A. W. Heemink,	TU Delft, supervisor
	Dr. ir. M. B. van Gijzen,	TU Delft
	Ir. S. Q. Dijkhuis,	Cleverbase B.V., supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

When building a convolutional neural network, many design choices have to be made. In the case of Deepfake detection, there is no readily implementable recipe that guides these choices. This research aims to work towards understanding the effects of design choices in the case of Deepfake detection, using the Python library `Keras` and publicly available datasets. The choices analysed are dataset composition, preprocessing, dropout rate, batch size, network architecture, and specific dataset. We also analyse the difference between training a network on original images and DCT-residuals of images. Lastly, we analyse the networks' generalisation and robustness capabilities. The goal of these experiments is to work towards a readily implementable recipe for Deepfake detection algorithms. Furthermore, this research provides an overview of image manipulation algorithms, an overview of recent research into convolutional networks, and an extensive overview of the Deepfake detection research field.

To analyse dataset composition, we used different subsets of FaceForensics++, with different numbers of frames per video. We trained a shallow network, containing only four convolutional layers, on all three datasets. The dataset with one frame per video was the only one that did not result in immediate overfitting, although it contained less than two thousand frames in total.

We continued with the small dataset and tested different preprocessing settings and dropout rates on our shallow network. We found that preprocessing and dropout were not able to increase the maximum achievable accuracy, although they were able to curtail overfitting. It is possible that accuracy did not increase due to the high variety of artefacts in FaceForensics++. Batch size also does not have any effect on the maximum achievable accuracy. However, the runtime required for training a network increases considerably as the batch size decreases.

We tested DenseNet-121, Inception-v3, ResNet-152, VGG16, VGG19, Xception, and our shallow network on three different datasets: FaceForensics++, Celeb-df, and DeeperForensics-1.0. The goal of this experiment was to find what type of network is most suited for detecting Deepfakes with publicly available datasets of small size but high variation. We also wanted to see if there were differences in performance achieved on the different datasets.

Although all networks except the shallow one were pretrained on ImageNet, three of the different networks tested immediately overfitted on all three datasets used: DenseNet-121, Inception-v3, and XceptionNet. The other networks encountered most difficulties with Celeb-df, on which none of the networks managed to reach 70% accuracy before overfitting. The easiest network to train on was DeeperForensics-1.0, on which our shallow network achieved 92.5% accuracy. However, when testing the networks' robustness, none of the networks trained on DeeperForensics-1.0 reached an accuracy higher than random on Celeb-df or FaceForensics-1.0.

Shallow networks might be better suited for Deepfake detection on our small dataset. The shallow model and VGG16 achieved the highest accuracies. VGG19's performance is close to VGG16's. However, ResNet-152 is the deepest network used in this research and performs better than the shallower DenseNet, Inception-v3, and Xception.

Lastly, training our networks on DCT-residuals was supposed to help our network focus on statistical image content rather than semantical image content. However, performance on DCT-residuals was at best similar to performance on original images.

Our suggestions for continuation of this research are to experiment with different input sizes and other types of residual filters, collect larger (and higher quality) datasets with high variety, and to use interframe detection.

Preface

When I graduated secondary school, I still didn't believe I'd ever go to university. Consequently, when I received my diploma, I hadn't done any orientation as to what I'd most like to study. My maths teacher told me I should study maths, I told him that sounded like a very abstract study, and he suggested I enrol in Delft instead of in Leiden. So I did, that same afternoon, hardly knowing what I was signing up for.

And there I was, little over six years later, walking into Professor Arnold Heemink's office at the TU Delft, asking him whether it would be possible for me to graduate at Cleverbase B.V. on the detection of Deepfake videos; a research topic for which there was little expertise at TU Delft's Applied Mathematics faculty. Yet, the idea was fully supported, and I was assigned a graduation supervisor, Professor Hai Xiang Lin; who, although not experienced in the field of Deepfake detection, is an expert in the field of deep learning. He never hesitated to help me by sending me additional articles and he got me in touch with experts from other universities who might know more about computer vision specifically.

And here I am, almost another year later, ready to graduate; before you lies my master thesis on the detection of Deepfakes using convolutional neural networks. And what a journey it has been. I learned incredibly much on the topic of Deepfakes, convolutional networks, and deep learning in general. Moreover, I learned how to work with many new computer programmes such as Linux, Git, and AWS. Most importantly, I genuinely enjoyed the overall project. There were ups and downs (as seems to be the case with most graduation projects), but the sum of all emotions I experienced during this project is still positive. It even leaves me slightly sad to say goodbye to this project now.

Naturally, I didn't do any of this on my own. First off, I'm glad I could graduate with a company instead of internally at university, an opportunity for which I still have Lukas Spee to thank. Moreover, I would like to thank Sander Dijkhuis and Ellard van Eekelen for their daily supervision over the past year. The same holds for Jasper Remmerswaal, although he left the company before I finished my project. I would also like to thank Maarten Hoogendoorn, who helped me out with technical problems, like when I destroyed my SSD or when I uploaded every image in my training dataset to Git. Lastly, my thanks go out to Tessa Driessen for the creation of images 3.2, 3.6, and 3.9. It goes without saying that I also appreciate all other colleagues who cheered me on throughout the year.

I would also like to thank Hai Xiang Lin and Arnold Heemink for their trust in this self-suggested project and their more in-depth supervision on the subject matter. Lastly, I would like to thank my boyfriend, my friends, and my parents for their continuous mental support; they were always there to tell me that it's not unusual to take several months before producing results, or for your model to seemingly work only to end up not working two weeks later, and that all of those problems reflect on graduating in general rather than on my intelligence. In the end, it seems they were right (of course).

It only remains for me to say that I hope the reader of this thesis will enjoy reading it as much as I enjoyed writing it. I genuinely hope that I have been able to transfer enough of both the theory and my interpretations on paper for my thesis to be understandable for anyone interested.

B.X. van Leeuwen
Delft, November 2020

Contents

List of Figures

List of Tables

List of Abbreviations

AUC	Area Under Curve
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
EER	Equal Error Rate
ELBO	Evidence Lower Bound
GAN	Generative Adversarial Network
LSTM	Long Short-Term Memory
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SVM	Support-Vector Machine
VAE	Variational Autoencoder

1

Introduction

In our rapidly digitalising society, there is an increasing need for digital assurance with respect to the identities of the people we interact with online. Deepfake videos might pose a threat to this assurance. A Deepfake algorithm is able to create realtime video manipulations in which a source individual is replaced by a target individual. This replacement can be done with face swap, facial reenactment, or by generating a whole new video starring the target individual. The resulting videos vary from fake pornographic videos [91] to fake political speeches[1]. Although the quality of Deepfakes is not always stable, especially not when the video is dynamic, the need for Deepfake detection is omnipresent.

The detection of Deepfake videos can be considered a binary classification problem because every image is either 'real' or 'fake'. Such a binary image classification problem can be solved using convolutional neural networks. Convolutional networks teach themselves what filters are useful for the extraction of relevant patterns in images. They use the patterns extracted by such filters to decide whether an image is real or fake. This allows them to detect images scale and translation invariantly, contrary to plain feedforward networks. We use the Python deep learning library `Keras` for our implementations.

The Problem When designing a convolutional network for a specific task (in this case, Deepfake detection), there are many choices to be made. For example, there are different network architectures to choose from, and there are different hyperparameters to set. Moreover, there are different ways of letting an algorithm process an image or a video. For example, an algorithm can consider separate video frames and try to find spatial inconsistencies; or it can compare subsequent video frames to find temporal inconsistencies.

However, Deepfakes only surfaced around 2007 [1, 143], and convolutional networks only started dominating image classification research from 2012 onwards. This means the research field is relatively young. Hence, there is no readily implementable recipe on how to build a convolutional network for a specific task, nor on how to process images before feeding them to the network. The chaos that dominates the research field is problematic since Deepfakes and their detection need to be understood not only by technical researchers, but also by those who can protect against the societal threat they might pose.

This research aims to provide insight into these relatively new research fields and to work towards a readily implementable recipe for Deepfake detection algorithms.

Our Contribution This research serves as a stepping stone for those who are unfamiliar with this research field but still want to protect their products or processes from being compromised by Deepfakes. In order to do this, we first offer some handholds by

- providing an overview of research into convolutional networks from 2012 onwards;
- providing an overview of different kinds of deep learning algorithms that can perform video manipulation;
- providing an overview of different video manipulation detection approaches, by categorising and summarising methodologies of 53 papers on image and video manipulation detection.

[1] For example, see `https://www.youtube.com/watch?v=gLoI9hAX9dw`.

Then, we use the obtained knowledge to systematically work towards finding the most favourable combination of datasets, model architecture, and hyperparameters for detecting Deepfake videos with convolutional neural networks. We conduct experiments on dataset composition, preprocessing settings, dropout rates and batch sizes, dataset quality (using FaceForensics++ [104], Celeb-df [78], and DeeperForensics-1.0 [59]), and network architectures (a shallow network, DenseNet-121 [53], Inception-v3 [120], ResNet-152 [46], VGG16 and VGG19 [113], Xception [21]). We combine the convolutional bodies from all seven networks with two different classification networks. We also conduct experiments using the Discrete Cosine Transform (DCT) of images, hoping to help the convolutional network focus on the statistical content of images rather than the semantical context. Lastly, we test the networks' generalisation and robustness.

Our Goal Contrary to the 53 research articles we summarise, this report does not aim to improve the state of the art of Deepfake detection. Rather, we aim to understand the effects of different choices made in dataset composition, network architecture, and network hyperparameters on network performance; and why they affect performance the way they do. This way, we hope to provide insight into the behaviour of the underlying networks. We do not aim to praise or discredit any particular algorithm or approach; instead, we use our understanding of their differences and similarities to point us in the direction of what technical specifications Deepfake detection requires, and why. This will help us systematically work towards a Deepfake detection algorithm, using our fundamental understanding of the underlying networks.

Although the results of our experiments are not always as hoped or expected, we hope to take the reader on a journey that will provide understanding of the behaviour of convolutional neural networks, as well as insight into the effects different design choices have on this behaviour.

Our Methodology To achieve the above, we conducted seven separate experiments. Each experiment explored the effects of the mentioned design choices on network performance. We used three criteria to measure network performance: maximum accuracy achieved, minimum loss achieved, and whether overfitting occured.

The first experiment used subsets of FaceForensics++ containing different amounts of frames per video. This way, the different subsets were of different sizes and contained different levels of variety in data. We trained our shallow model on all three of them to compare the results on network performance.

The second, third, and fourth experiments trained our shallow network with different preprocessing settings, dropout rates, and batch sizes respectively. We analysed the effects on performance and used the most favourable preprocessing settings, dropout rates, and batch sizes for the remainder of our research.

The fifth experiment analysed what network architecture and publicly available datasets are most suited for Deepfake detection. To this end, we implemented the seven different networks mentioned earlier and trained them on FaceForensics++, Celeb-df, and DeeperForensics-1.0. Except for our shallow networks, all networks were pretrained on ImageNet. The idea is that having been trained on ImageNet provides the networks with filters that allow them to understand most real-world objects. This should make them less prone to overfitting on our small dataset. We analysed the networks' performance on the different dataset to point us into the direction of the network type best suited for Deepfake detection on the provided datasets. Moreover, we compared the three different datasets: do we observe differences in performance on the different datasets, or do all networks perform the same on every dataset?

The sixth experiment took the networks that performed best in experiment five and trained them on DCT-residuals of images. We compared their performance on these images to the performance on the original images. If DCT-residuals indeed help the network to focus on statistical rather than semantical content, this should cause an improvement in performance.

Experiment seven took the same networks as experiment six and tested their robustness. To this end, we let the networks detect Deepfakes on the two datasets on which they were not trained. Networks that do not perform well on unseen data are not useful in real-world scenarios, because it is unlikely that any images the networks will see in the real-world are from the dataset they were trained on.

Structure of This Report Two background chapters follow this introduction. Chapter 2 provides a general background of neural networks, only sometimes referring to computer vision tasks to increase understanding of the topics at hand. For the reader experienced with neural networks, this will merely serve as a reference. For the reader inexperienced with neural networks, this will set up the framework required for understanding the remainder of this report. Chapter 3 provides background information on video manipulation and its detection, building on the knowledge base set up in the general background chapter.

Chapter 4 defends the selection of literature included in this research; elaborates how we chose our categorisations of detection algorithms; elaborates the networks and datasets used for practical research. It also explains how we executed our experiments and why we executed them that way.

Chapter 5 presents the results on the effects of dataset composition, preprocessing, dropout rate, batch size, network architecture, and datasets used. It explains what these results mean and how they relate to the literature. Chapter 6 summarises the conclusions of this research.

2

Background I: A Brief Overview of Neural Networks

Artificial neural networks are inspired by biological neural networks [32]. Artificial neural networks have been around for nearly a century. For example, the McCulloch-Pitts neuron dates back as far as 1943 [18]. In the 1960s, researchers started combining several neural networks to create depth; however, as computers were not up to the heavy computational tasks, interest soon dwindled [18]. In the 1980s, this interest sparked again due to two newly introduced concepts: error backpropagation and the neocognitron, which would later evolve into the convolutional network [18]. Research combining the two yielded a network capable of classifying handwritten digits [18].

Before 2006, convolutional networks (and their predecessors) were the only deep networks that could be trained successfully [41]. In 2006, the Deep Belief Network (DBN) changed this [41]. Subsequently, in 2006 the term 'deep learning' was first introduced [18]. Since then, neural networks have come a long way.

Although neural networks might seem magical and futuristic, getting from an idea of something that could be solved by a neural network to a working neural network that can perform the task in mind can take a while. Fig. 2.1 shows the steps necessary to create a working neural network: data needs to be collected (and sometimes labelled) and preprocessed, and in the meantime an algorithm has to be designed. Once that is done, the algorithm has to be trained. Training is not always stable; the process might have to be repeated several times until the results are satisfactory. Moreover, a network that performs well on a training dataset need not necessarily perform well on unseen data (see Section 2.3.5).

Throughout this research, the terms model, network, neural network, and algorithm will be used interchangeably to indicate artificial neural networks.

There are numerous types of neural networks (some of which will be discussed in section 3.2). All neural networks have one thing in common: they learn their own parameters. Apart from that, it is difficult to talk about the properties of 'a neural network' without loss of generality. This background chapter will talk

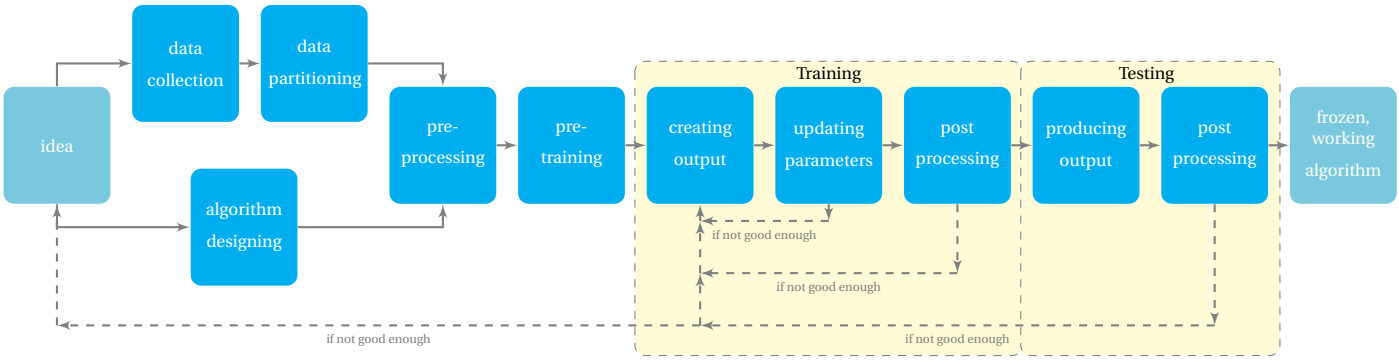


Figure 2.1: A graphical summary of the process of creating a deep learning algorithm.

about neural networks in general; this means that when technical details are concerned, it will assume 'basic' feedforward and recurrent networks for simplicity.

The remainder of this chapter explains the basic concepts required to understand neural networks, starting by explaining their general architecture in section 2.1. Section 2.2 goes deeper into the architecture by elaborating on activation functions. Once those sections have introduced the basic building blocks of neural networks, section 2.3 will conceptually explain how a neural network is trained; section 2.4 will provide a technical explanation of the algorithms used during training. Section 2.5 concludes this chapter with an explanation of performance measures.

Chapter 3 will build on the concepts introduced in this chapter to explain convolutional networks, image generation and manipulation, and image manipulation detection.

2.1. General Architecture and Terminology

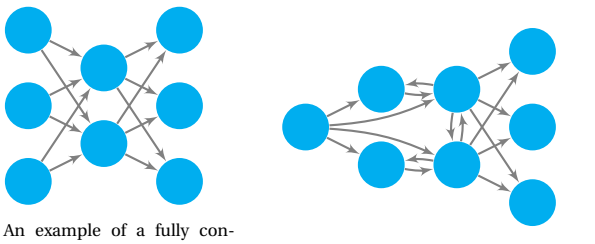
This section will first introduce basic terminology, and then provide an example that will clarify what all these terms mean in practice. This section aims to help the reader understand the general architecture of neural networks, i.e., the standard building blocks that build a neural network, their function, and how they relate to each other.

A basic neural network can be thought of as a directed weighted graph: nodes are connected through weighted edges, abbreviated to *weights*. These weights are summarised in a *weight matrix*. The value of a node is equal to the sum of all its weighted inputs (see example 1). We will use the term *parent node* to indicate a node that feeds information to its *child node*.

In neural networks, nodes are grouped in *layers*. These can be thought of as different stages of the network through which information passes: the input first passes through the first layer of the network, then through the second layer, and so on towards the last layer. The positions and directions of connections between nodes determine what type of neural network is concerned.

The first layer of a neural network is called the *input layer*; the last layer of a neural network is called the *output layer*. The layers in between are called *hidden layers*. The amount of layers a neural network has, is its *depth*; the amount of nodes it has in each layer is its *width*. A *deep learning* algorithm is a network with multiple hidden layers, which have nonlinear activation functions [32]. The opposite of a deep network is a shallow network. Deep networks perform better than shallow networks on tasks concerned with content in which patterns are repeated [32]. It has been shown that a multilayer perceptron[1] (MLP) with two hidden layers can perform at least one task an MLP with one hidden layer cannot (for details, see [22]). It has also been shown that there are several types of functions that a deep network can represent more easily than a shallow network [32].

Largely, neural networks can be divided into two categories[2]: *feedforward* networks and *recurrent networks* [32]. In feedforward networks, a parent node can only have child nodes in successive layers [32]. In a recurrent network[3], there is at least one feedback connection [32]: at least one parent node has a child in a previous layer, or in its own layer. This means a parent node can also be its own child[4].



(a) An example of a fully connected feedforward neural network.

(b) An example of a recurrent neural network.

Figure 2.2: Examples of neural networks.

In *fully connected* feedforward networks, a node is a parent to every node in the successive layer [32]. In a fully connected recurrent network, every node is a parent to every node, including (a recurrent connection

[1] A specific feedforward network with more than one hidden layer.
[2] There are also hybrid methods; however, these are not relevant for this report. We refer the interested reader to Du and Swamy [32].
[3] Recurrent networks will be explained elaborately in section 3.2.4.
[4] This does not lead to computational difficulties, since in recurrent networks a node's value is calculated iteratively.

into) itself [32]. Fig. 2.2 provides an example of a fully connected feedforward network (Fig. 2.2a) and a recurrent network (Fig. 2.2b).

In convolutional networks, we often speak of *fully connected layers*. This means that each node of the previous layer is a parent to every node in this layer. Fully connected layers are also called *dense* layers, e.g. in the `Keras`-library.

Example 1 (Terminology in Practice) *Let's take a look at a neural network using the tools discussed thusfar. We will consider the fully connected feedforward network depicted in Fig. 2.3. The network has an input layer, two hidden layers, and an output layer. The network can be seen as a function $f : \mathbb{R}^3 \to \mathbb{R}$ that is represented by the composition $f(x) = (u \circ V \circ W)(x)$, where the weight matrices u, V, W are defined by*

$$u = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}, V = \begin{bmatrix} v_{11} & v_{21} & v_{31} \\ v_{12} & v_{22} & v_{32} \\ v_{13} & v_{23} & v_{33} \end{bmatrix}, W = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \end{bmatrix}.$$

The entries of these weight matrices are the network's weights. In this example, x_1 is a parent node to h_{11}, h_{12}, h_{13}; $\hat{y}$ is a child node to h_{21}, h_{22}, h_{23}.

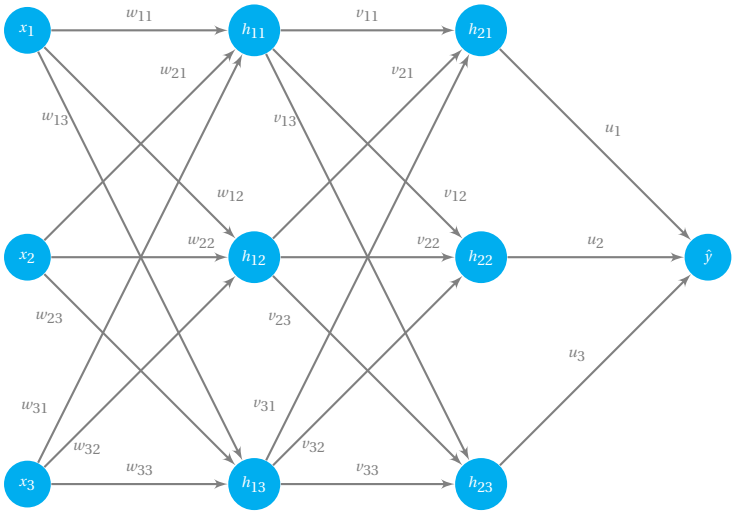


Figure 2.3: A fully connected feedforward neural network.

Example 1 suggests that stacking layers is pointless; if we write $z = uVW$, then $z \in \mathbb{R}^{1\times 3}$, and we might as well have set $f(x) = zx$ from the start. Indeed, an especially important tool we have not yet considered, is the *activation function*. These are (usually non-linear) functions that are applied after the weighted sum of a node's input has been calculated. This means that in Fig. 2.3 instead of $h_1 = w_{11}x_1$, we would have $h_1 = \sigma(w_{11}x_1)$ for some nonlinear σ, where $h_1 = [h_{11}, h_{12}, h_{13}], x = [x_1, x_2 x_3]$. More on activation functions will follow in section 2.2.

2.2. Activation Functions

As we saw in section 2.1, we need activation functions to make adding layers to a network useful. Activation functions are usually used in hidden and output layers. There are many types of activation functions; they enjoy active research still[5] [41]. This section will not provide an exhaustive overview of activation functions; we will only discuss the most popular ones: the linear function, the (logistic) Sigmoid, the hyperbolic tangent, the rectified linear unit (ReLU), the leaky rectified linear unit (leaky ReLU), Softplus and softmax.

[5] Since activation functions are being so actively researched, there is no clarity on what activation function works best for a specific network [41]. Choosing one beforehand is often not possible; it has to happen through trial and error [41]. Currently, the ReLU and its variants are most popular; before their discovery, the hyperbolic tangent and Sigmoid function were most popular [41]. The ReLU function is still actively being improved, see for example RPReLU [81].

Sigmoid The Sigmoid function, also called the logistic Sigmoid, is defined by $\sigma(x) = \frac{1}{1+e^{-x}}$ [33], see Fig. 2.4a. It is prone to vanishing gradients (see section 3.2.4) [32]: around the origin, the gradient is large, but for large or small x, the gradient is nearly 0. Hence, the Sigmoid function is not often used as activation function in hidden layers. However, the Sigmoid function is useful in the output layer of classification models [41]. It will return a prediction of the sample belonging to each available label [100].

Softmax The softmax function is defined by $\sigma : \mathbb{R}^n \to \mathbb{R}^n, \sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$ [41]. The Sigmoid function is a special case of the softmax function [33]. Their uses are also alike: softmax is used in the output layer of networks designed for classification problems with more than two classes [41]. The softmax function also returns the probability that a sample belongs to a certain class [41]. Contrary to the Sigmoid function, its predictions over all classes will add up to 1 [33].

Hyperbolic tangent The hyperbolic tangent is defined by $\sigma(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)}$, see Fig. 2.4b. It is particularly popular in recurrent neural networks [32]. The hyperbolic tangent often performs better than the Sigmoid function [41]. This is because the hyperbolic tangent is approximately linear around the origin [41].

ReLU The ReLU is defined by $\sigma(x) = \max(0, x)$, see Fig. 2.4c. It is not differentiable[6] at 0, but in practice this is not a problem (for details, see [41]). The main disadvantage of the ReLU function is that it can cause nodes to die because the gradients for $x < 0$ are always zero [32]. The leaky ReLU is designed to fix this.

Leaky ReLU The leaky ReLU is defined by $\sigma(x) = \max(0, x) + \alpha \min(0, x)$ for some set $0 < \alpha << 1$ [41], see Fig. 2.4d. There are also variants that consider α a learnable parameter [41].

Linear activation functions Linear activation functions are of the form[7] $\sigma(x) = x$, see Fig. 2.4e. Linear activation functions are often used in output layers [41].

As described in section 2.1, if a network contains subsequent layers with linear activation functions, those could be reduced to a single layer. However, sometimes it can still be beneficial to use linear activation functions for the sake of dimensionality reduction [41]. We demonstrate this by expanding example 1 using the explanation Goodfellow et al. [41] provide. To this end, suppose we have a fully connected feedforward network that takes input size $x \in \mathbb{R}^n$, and the input layer is followed by a hidden layer $h \in \mathbb{R}^m$. Then $h = Wx$ for some $W \in \mathbb{R}^{m \times n}$, meaning that W has mn parameters. However, if h were replaced by two hidden layers, $h_1 \in \mathbb{R}^q$ and $h_2 \in \mathbb{R}^m$, then the output size of h_2 would be the same as that of h. Yet, $h_2 = UVx$ for some $V \in \mathbb{R}^{q \times n}, U \in \mathbb{R}^{m \times q}$ for some arbitrary q. Consequently, U and V together contain only $q(n+m)$ parameters. If $q << n, m$, this reduces the amount of parameters considerably. Although this dimensionality reduction leads to loss of information, in practice it often suffices [41].

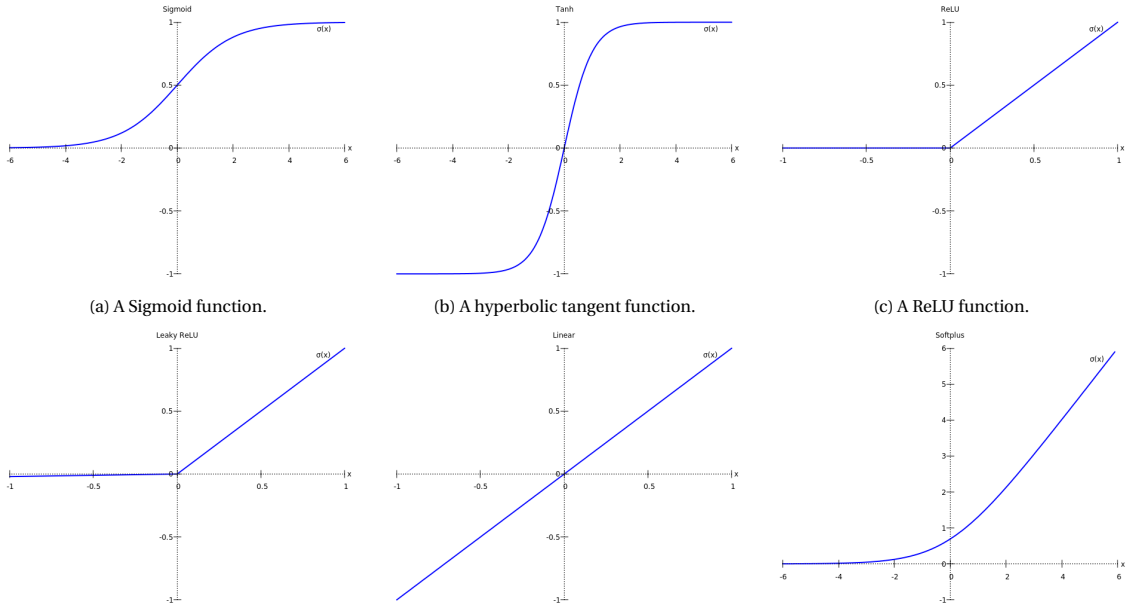
Softplus The Softplus function is defined by $\sigma(x) = \log(1 + e^x)$ [41], see Fig. 2.4f. Although the Softplus function looks more promising than the ReLU function because it is differentiable and it is less prone to saturation, in practice it does not outperform ReLU [41]. Therefore, Softplus is usually not used for hidden layers, only for output layers [41].

2.3. Training

A neural network has two types of parameters: *learnable* parameters, and *hyperparameters*. The hyperparameters are manually chosen by the designers of the algorithm [18]. It is also possible to have a learning algorithm find the best hyperparameters for another learning algorithm [41]. The learnable parameters are learned during *training* [18]. Mathematically, learning can be seen as non-linear curve fitting [32].

[6] Differentiability is a desired property, because training neural networks requires the calculation of gradients, see section 2.4.

[7] A mathematician might note that a general linear function σ is defined by $\sigma(x) = ax$ for some arbitrary $a \in \mathbb{R}$. However, since a node's weighted sum is given by $wx + b$, where w, b are learned by the network, there is no gain in computing $a(wx + b)$ rather than $wx + b$.



(a) A Sigmoid function.

(b) A hyperbolic tangent function.

(c) A ReLU function.

(d) A leaky ReLU function with $\alpha = 0.02$.

(e) A linear function.

(f) A softplus function.

Figure 2.4: Activation functions.

Example 2 (Hyperparameters) *Examples of hyperparameters are batch size, dropout rate, and learning rate; these will be explained in sections 2.4.1, 2.4.6, and 2.4. All entries of u, V, W in example 1 are learnable parameters.*

During training, networks learn feasible values for their learnable parameters. This *learning* is done with a *gradient descent* algorithm, which is explained in section 2.4. In the remainder of this report, the learnable parameters and hyperparameters will sometimes simply be called 'parameters'. It will be clear from the context which of the two we mean.

2.3.1. Pretraining

In networks with many layers, training can be separated into two parts: *pretraining* and *fine-tuning* [32]. As the name suggests, pretraining precedes fine-tuning or 'regular' training. The advantage of pretraining is that the network parameters are already at a local minimum at the start of fine-tuning, saving computational costs [32]. The result is that optimisation is both faster and produces better results [41].

There are two approaches to pretraining. One approach is to train a network one layer at a time by a greedy algorithm[8] [32], which can be executed by training only one layer (or a small part of the network's layers) at a time [41]. For implementation details, we refer the reader to Goodfellow et al. [41].

Another approach is to retrain a network that has been trained on another, yet similar dataset. This approach is especially useful in computer vision when the dataset at hand is small [20]. A network pretrained on a large dataset such as ImageNet [106] (which contains 1000 object classes such as trees, fish, and utensils[9]), can be considered as having a broad understanding of everyday objects, and can be used to further specialise on a smaller, more specific dataset [20]. Since the network has already seen a large variety of samples, this will result in better performance than only training directly on the small dataset [20].

If no pretraining is applied, the initial network parameters are set manually (usually randomly) and the fine-tuning phase is simply called training. It is also possible to *freeze* part of the network, meaning those parameters are no longer updated, and only fine-tune the other part of the network. For example, if we trained a convolutional network on ImageNet, we could freeze its first convolutional layers, since those will be simple filters useful for interpreting any real world object; the later convolutional layers could be fine-tuned to obtain the more complicated filters suited for the specific task at hand. The difference between convolutional filters in early and late layers is discussed in section 3.4.

[8] Greedy is an optimisation algorithm that does not search for a global optimum, but for the local optima of separate components of, in our case, a network. This is computationally cheap and yields acceptable pretrained weights [41].

[9] See `http://image-net.org/explore.php`.

2.3.2. Supervised versus Unsupervised Learning

Neural network learning can be roughly divided into two categories[10]: supervised and unsupervised. In the case of *supervised learning*, the network receives labelled data. This means data is of the form (x, y), where x is the actual input (e.g. an image) and y is the corresponding desired output [32]. A single input $x \in X$ is called a *sample*; output is often a label $y \in Y$, e.g. in the case of forgery detection, $y \in \{\text{real,fake}\}$. As we recall from section 2.1, a neural network can be seen as a function $f : X \to Y$, which maps inputs to their respective outputs.

The network uses input x to produce output $\hat{y} = f(x)$; this is an estimated label. The difference between y and $\hat{y}$ is calculated (using a loss function, see section 2.3.3) and the learnable network parameters are updated accordingly. This update is executed with gradient descent (see section 2.4).

When *unsupervised learning* is concerned, the network has no access to labels. The algorithm has to decide what categories seem useful, and what samples belong in what category. K-means (see example 3) is an example of an unsupervised learning algorithm. This algorithm is not directly relevant for the remainder of this research, but it should help the reader understand what it means that an algorithm decides upon categories itself.

Example 3 (K-means) *K-means*[11] *is an unsupervised learning algorithm that divides a set* $X = \{x_1, ..., x_n\} \subsetneq \mathbb{R}^d$ *into* K *clusters* $c_1, ..., c_K$ *such that the sum of the squared error between the means of each cluster and the points in that cluster is minimal [57]. That is, if we denote the mean of cluster* c_i *by* μ_i*, K-means minimises the cost function*

$$\sum_{i=1}^{K} \sum_{x_j \in c_i} \|x_j - \mu_i\|^2.$$

We will illustrate this with a simple applied example in Fig. 2.5: suppose we have

$$X = \{(0,-0.1), (0,0.1), (0,0.9), (0,1.1), (1,-0.1), (1,0.1)\} \subseteq \mathbb{R}^2.$$

We set our hyperparameter to $K = 3$*, meaning we let the algorithm find three clusters. Let the initial clusters be*

$$c_{purple} = \{(1,-0.1), (1,0.1)\}, c_{red} = \{(0,-0.1), (0,0.1), (0,0.9)\}, c_{blue} = \{(0,1.1)\},$$

as illustrated in Fig. 2.5a. First, we have to calculate the clusters' means[12]*, which are* $\mu_{purple} = (1,0), \mu_{red} = (0,0.3), \mu_{blue} = (0,1.1)$*; see Fig. 2.5b. Now, to minimise the cost function, the algorithm will place every data point in the cluster to whose mean it's closest. It is clear from Fig. 2.5b that there is only one data point that has to be moved:*

$$\|(0,0.9) - \mu_{red}\|^2 = \|(0,0.9) - (0,0.3)\|^2 = \|(0,0.6)\|^2 \geq \|(0,0.2)\|^2 = \|(0,0.9) - (0,1.1)\|^2 = \|(0,0.9) - \mu_{blue}\|^2.$$

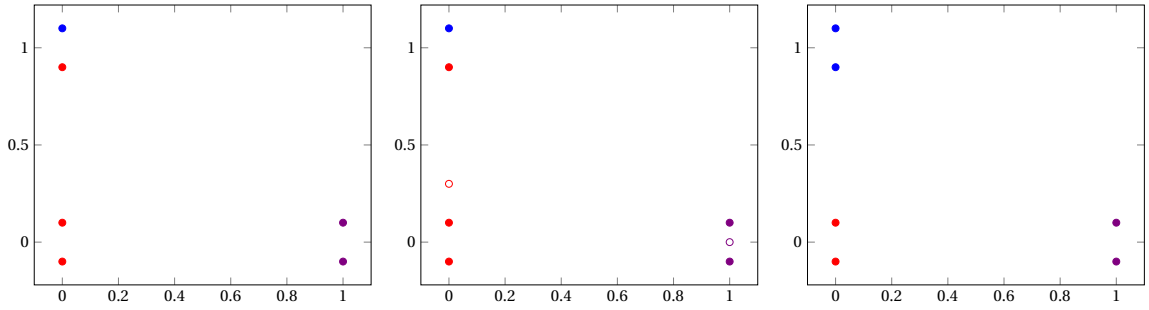
Hence, $(0,0.9)$ *has to be moved from* c_{red} *to* c_{blue}*. This results in the clustering shown in Fig. 2.5c. It is clear that the cost function has been minimised.*

When it comes to image forgery detection, supervised learning is preferable because we are very specific about the categories we want the algorithm to learn: 'real' and 'fake'. Therefore, it is best to provide the network with labelled data. On the other hand, when it comes to image generation, network output is an image rather than a label. We have no single image y in mind that we want the network to produce; there is a range of acceptable generated images. Hence, we cannot compare the input to any specific desired output. Therefore, image generation is an unsupervised learning process.

[10] There are also hybrid methods that aim to get the best of both worlds. For example, *semi-supervised learning* uses some supervised data points to improve generalisation, but trains mostly on unlabeled data. *Reinforcement learning* is a supervised learning method that is inspired by dopamine-releasing neurons: it works with rewards and punishments [32]. The exact y to test $\hat{y}$ against is unknown; the network only receives information on whether its output ($\hat{y}$) is close to the desired output (y). Reinforcement learning does not require computation of derivatives, but it is a relatively slow learning process [32].
Another type of learning is *pool-based active learning*, which is a type of supervised learning. However, the network can only access labels after it 'pays' for them [32]. This approach intends to reduce the expenses of manually labelling data by labelling only the data that require labelling [32].

[11] Example inspired by Jain et al. [57].

[12] For which we will assume Euclidian distance.



(a) The initial clusters. (b) Calculating the clusters' means. (c) The updated clusters.

Figure 2.5: An applied example of the unsupervised K-means algorithm, with $K = 3$. For visualisation, the culters are red, blue, and purple. The solid dots are samples; the open dots are cluster means. The blue cluster's mean is not visible because it coinsides with the blue sample.

2.3.3. Loss Functions

During training, a learning algorithm aims to minimise a *loss function* (or *cost function* or *training criterion*) [18]. A loss function is a measure for the difference between the network's output and the network's desired output. It is minimised using a provided training dataset. The loss function is usually based on an error measure[13].

A Deepfake detection algorithm can be seen as a binary classification model. Its output is the probability that a provided sample is real. In classification models, *cross-entropy*[14] is the default choice of loss function [20]; since Deepfake detection is a binary classification task, we use binary cross-entropy. In `Keras`, it is implemented as

$$BCE(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})).$$

2.3.4. Capacity, Generalisation, and Robustness

Every neural network has a specific *capacity* [41]. This capacity is the network's ability to learn specific (types of) functions $f : X \to Y$, where X denotes the input set and Y the output set [41]. The amount of learnable parameters a network has, as well as network depth and width, add to its capacity [41]. The capacity required for a specific classification task depends on the amount, size and complexity of data in the dataset. For example, we can image that distinguishing cats from dogs might be less complex than distinguishing labradors from golden retrievers.

Neural networks aim to minimise the error on the data they receive during training. However, we also want the network to perform well on data that is similar to but not included in the training dataset. To this end, the data will be split into two datasets: *training* and *validation* data. The network is told what ratio to wield[15], and it will split the data randomly. During training, the network uses only the training data to update its parameters. The validation data is used to check how well the network performs on previously unseen data. A network's capability of performing its task on the validation dataset is called *generalisation*. Pretraining can improve generalisation [32].

Sometimes, we can use a third type of dataset: *test* data. This dataset contains data that is similar but not identical to the training or validation data. A network's capability of performing its task on test datasets is called *robustness*.

We illustrate the difference between generalisation and robustness in example 4.

Example 4 (Generalisation and Robustness) *Suppose we want to build a classification network to distinguish between cats and dogs. To train it, we can take pictures of our own cats and dogs; the network can dutifully split this collection of images into a training dataset and a validation dataset. If, after training has*

[13] The choice of error measure is often tricky, depending on, i.a., the type of data and the task the network has to perform [41]. For example, if a network is designed for a classification task with two categories, the error measure only needs to tell the network whether a data point is classified correctly or not. However, when a network is designed to generate images, a more sophisticated error measure will be required. Sometimes it even happens the quantity we desire to measure cannot be measured, and workarounds need to be used [41].

[14] All loss functions that are a negative log-likelihood of the found probability distribution and the actual probability distribution are cross-entropies [41]. In deep learning terminology, cross-entropy often means specifically the negative log-likelihood of a Bernoulli distribution [41].

[15] This ratio is also a hyperparameter.

completed, the network can distinguish between our cats and dogs in the validation data, we say the network generalises well.

However, we want to know for sure the network is capable of distinguishing between any types of cats and dogs; not just ours. Therefore, we can create a test dataset containing images of our neighbour's cats and dogs. If the network is able to perform the classification task on these images as well, we say it is robust.

2.3.5. Overfitting and Underfitting

If a network's capacity is not suited for the task at hand, this will result in problems in training stability. Two of these problems are *underfitting* and *overfitting*.

Overfitting occurs when a network's capacity exceeds the capacity needed for the task at hand [41]. The network will be able to memorise the samples in the dataset, instead of having to understand the patterns in the data that correspond to the different classes samples belong to. The result is that the network will perform well on the training dataset, but will fail completely on validation and test data. In section 2.4.6, we will discuss some tricks that help prevent overfitting.

Underfitting occurs when a network's capacity is lower than the capacity required for the task at hand [41]. The network will not be able to minimise the cost function sufficiently. This means it will not be able to find a meaningful relation between samples and their classes even on the training dataset. Example 5 explains a possible cause for underfitting.

Example 5 (Underfitting) *Continuing example 4, suppose we made a mistake in designing the algorithm, causing it only ever to be able to see a patch of the image sized 1% of the original input image. At this point, we have not yet acquired the technical tools to understand how this can happen. However, we can understand that the network's classification has become more difficult. If the network is lucky and the 1% of the image it receives is a cat's nose or eye, it might still be able to classify the image correctly. However, if it receives a piece of background or even fur, it will have to resort to classifying randomly: it will not be able to find meaningful relations between the input images and their respective classes. This will cause the network to underfit.*

2.4. Gradient Descent

Gradient descent is the algorithm that is usually used to update network parameters. It is based on the idea that when minimising a function, the directional derivative shows what direction to update the parameters in. This section first introduces the necessary terminology (section 2.4.1), then explains different types of gradient descent (section 2.4.2), then elaborates the algorithm used for computing gradients (section 2.4.3), then formulates gradient descent (section 2.4.4).

2.4.1. Batches and Epochs

A *batch* is a random subset of the training dataset which contains a fixed amount of images. This fixed amount is called *batch size*. A network updates its parameters after having seen one batch of images. An *epoch* is a full run of all training data through the algorithm once [32]. So if our training dataset contains 100.000 samples, and we set the batch size to 1.000, an epoch will be completed once the algorithm has seen 100 batches, and changed updated its parameters 100 times accordingly.

2.4.2. Batch, Mini-Batch, and Stochastic Gradient Descent

There are three different types of gradient descent: *batch learning*, *stochastic learning*, and *mini-batch learning* [112]. When batch learning is used, the batch size is equal to the size of the training set; the network parameters are updated once per epoch [32]. When stochastic learning is used, the batch size is 1; the network parameters are updated after every sample [112]. When mini-batch learning is used, the batch size is somewhere in between [41].

Batch learning is deterministic; stochastic learning and mini-batch learning are stochastic: since the batches are chosen at random, every run of the algorithm will contain different images in each batch. Therefore, the resulting outputs, gradients, and suggested updates will differ per run.

The advantage of stochastic learning is that it is faster and requires less memory storage than batch learning; the disadvantages are that stochastic learning is difficult to parallelise, and does not converge as well as batch learning [32]. Mini-batch learning combines the advantages of both methods.

2.4.3. Backpropagation

Backpropagation is the algorithm used to compute the gradient of a loss function with respect to the network parameters. Suppose our network[16] has $n \in \mathbb{N}$ parameters, receives an input vector $x \in \mathbb{R}^k$, and produces an output $\hat{y} \in \mathbb{R}^m$. We denote our network as $f : \mathbb{R}^k \to \mathbb{R}^m$ and our loss function as $J : \mathbb{R}^{2m} \to \mathbb{R}$. We order all network nodes such that the output can be computed sequentially. This means that nodes $u_1, ..., u_{n_k}$ correspond to the input nodes; $u_{n-m}, ..., u_n$ to the output nodes; and u_{n+1} to the cost. We denote $u = [u_1 \cdots u_n]^T$ the vector containing all network parameters. We define

$$P_i = \{u_j : j < i, u_j \text{ is a parent node to } u_i\}.$$

Now we have a cost u_{n+1}, and we want to know the corresponding gradient with respect to the network parameters. We can compute this using the chain rule, which states that

$$\nabla_u u_{n+1} = \left(\frac{\partial f(u)}{\partial u}\right)^T \nabla_{f(u)} u_{n+1},$$

where $\left(\frac{\partial f(u)}{\partial u}\right)$ denotes the Jacobian of the network. Backpropagation uses this to compute the gradient of the cost u_{n+1} with respect to any node u_j in the network:

$$\frac{\partial u_{n+1}}{\partial u_j} = \sum_{i:u_j \in P_{u_i}} \frac{\partial u_{n+1}}{\partial u_i} \frac{\partial u_i}{\partial u_j}.$$

2.4.4. Formulating Gradient Descent

Gradient descent uses these computed gradients to determine in what direction the gradient of the cost function is smallest, and the network parameters are updated to move in that direction. In order to do this, the directional derivative is used [41]. The directional derivative of $(J \circ f)(u)$ in direction v is defined as

$$\frac{\partial}{\partial \alpha}(J \circ f)(u + \alpha v)\Big|_{\alpha=0} = v^T \nabla_u (J \circ f)(u).$$

Hence, the gradient descent algorithm calculates what unit vector v minimises $v^T \nabla_u (J \circ f)(u)$; this will be the unit vector pointing in the opposite direction of the steepest negative gradient. The network parameters are then updated: $u \leftarrow u - \eta \nabla_u (J \circ f)(u)$, where η is the *learning rate*. The learning rate is a hyperparameter [41].

2.4.5. Adam and RMSProp

Contrary to regular batch learning, stochastic and mini-batch learning cannot use a fixed learning rate η. Instead, they require the learning rate to change after every iteration [41]; otherwise, due to the randomness in selecting the mini-batches, SGD need not converge [41]. Popular examples of algorithms that automatically change the learning rate are *Adam* [64] and *RMSProp*. Both methods "have been proposed to automatically tune the learning rate η_t by using second-order moments of historical gradients" [142].

Both algorithms use learning rate $\eta_t = \alpha \cdot \frac{m_t}{\sqrt{v_t}}$, where α is a hyperparameter, $\hat{m}_t$ is a first moment estimate, and $\hat{v}_t$ is a second raw moment estimate [64, 142]. The difference between Adam and RMSprop is the way m_t is updated: if we denote g_t the gradient at iteration t, then for Adam, $m_{t+1} = \beta m_t + (1-\beta) g_{t+1}$, with β another hyperparameter [142]. For RMSProp, $\beta = 0$ [142].

2.4.6. Regularisation: Dropout and Stochastic Depth

Regularisation is the term used to describe anything that reduces the generalisation error of an algorithm, but not its training error [41]. Regularisation aims to prevent an algorithm from overfitting [41].

[16] This network is necessarily feedforward. As we have discussed in section 2.1, recurrent neural networks have less restrictions on the connections that connect graphs than feedforward networks have. As a consequence, a recurrent network is a cyclic graph [14]. This means that the backpropagation algorithm discussed in this section cannot be used [14]. Instead, a variant called Backpropagation Through Time (BPTT) is used [14]. This algorithm first 'unfolds' the neural network for a finite number of timesteps, creating a feedforward neural network [14]. This feedforward network can then be trained as a regular feedforward network, with the additional constraint that all entries representing the same operation need to have the same weight [14].

A popular way of regularising is using *dropout*[41], which randomly turns off nodes during training to prevent *co-adaptation* [32]. Co-adaptation means that different nodes depend on each to correct their mistakes; preventing this forces the individual nodes to perform well regardless of the mistakes or corrections of other nodes [115]. The probability of a node being dropped is the *dropout rate*. In practice, dropout is often implemented as a dropout layer, meaning there is one (or several) layer(s) in the network of which the nodes randomly switch off during training. Fig. 2.6 shows such a dropout layer. Section 3.4.2 discusses some possible positions of the dropout layer(s) in the network.

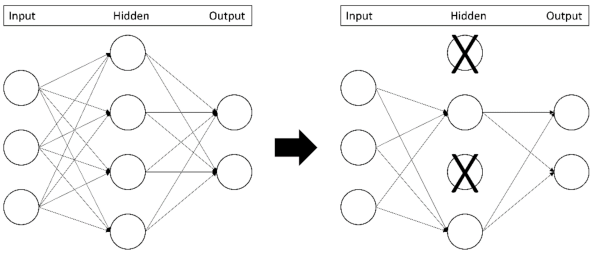


Figure 2.6: A dropout layer as depicted by Sultan et al. [117]. The hidden layer is a dropout layer; the right image shows this means that nodes and their connections are randomly turned off during training. When a node is switched off, all its weights are 0, preventing information from passing through it.

Whereas dropout can be viewed as dropping parts of the network on a micro-scale, it is also possible to do so on a macro-scale, e.g. with *stochastic depth* (see section 3.4.2.3). This also works as regularisation [54].

2.5. Performance Measures

There are different ways of measuring performance. This section discusses some standard performance measures. Additional performance measures used in the literature this research is based on, can be found in Appendix B.1. All performance measures depend on four values: true positives[17], true negatives, and false positives, false negatives (see Table 2.1). These values can be collected in a *confusion matrix*, which is of the from of (2.1) [34].

Table 2.1: The values on which performance measures are based.

Value	Abbreviation	Description
True Positives	TP	Amount of samples correctly classified as positive.
True Negatives	TN	Amount of samples correctly classified as negative.
False Positives	FP	Amount of samples incorrectly classified as positive.
False Negatives	FN	Amount of samples incorrectly classified as negative.

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix} \tag{2.1}$$

The choice of performance measure depends on what type of algorithm will be used. However, it also depends on what the goal of the algorithm is. For example, in the case of Deepfake detection, we can imagine a scenario in which it would be worse not to detect a Deepfake than to falsely mark a pristine video as possibly manipulated. For example, there might be extra checks in the case a video is labelled as suspicious. In this case, the actual Deepfake would be prevented from doing harm, and the pristine video would pass the extra checks. The downside of these extra checks is that they would require extra time.

Possibly the most often used performance measure is accuracy; for its formula, see Table 2.2. An algorithm's accuracy is the fraction of samples it classified correctly. Hence, an accuracy of 1.00 represents a perfect algorithm. However, accuracy weighs false positives as heavily as false negatives. Now suppose a Deepfake detection algorithm that receives 10,000 samples for testing, 10 of which are Deepfakes, and 9,990 of which are real videos. If it classified all samples as real, its accuracy would still be 0.999. An algorithm that classifies all ten Deepfakes as fake, as well as an additional ten real videos, will also have an accuracy of 0.999. Nevertheless, we can imagine the latter is probably the preferable algorithm.

There are several different performance measures available that express these different preferences. They can be found in Table 2.2. Accuracy, sensitivity, specificity, precision, fall-out, and miss rate are rates and therefore have values between 0 and 1. For accuracy, sensitivity, specificity, and precision, a high score is desirable; for fall-out and miss rate, a low score is desirable. The F_1-measure, although less straight forward,

[17] We will consider a fake sample 'positive'.

Table 2.2: Standard performance measures, taken from [32, 34, 111].

Measure	Formula	Description
Accuracy (ACC)	$\frac{TP+TN}{TP+TN+FN+FP}$	Rate of correctly classified samples.
Sensitivity/True Positive Rate (TPR)/ Recall/Hit rate	$\frac{TP}{TP+FN}$	Rate of positive samples classified as positive.
Specificity/True Negative Rate (TNR)	$\frac{TN}{TN+FP}$	Rate of negative samples classified as negative.
Precision	$\frac{TP}{TP+FP}$	Rate of samples classified as positive that are indeed positive.
Fall-out/False Positive Rate (FPR)/ False Rejection Rate (FRR)[a]	$\frac{FP}{TN+FP}$	Rate of negative samples classified as positive.
Miss rate/False Negative Rate (FNR)/ False Acceptance Rate (FAR)[a]	$\frac{FN}{FN+TP}$	Rate of positive samples classified as negative.
F_1	$\frac{2\cdot\text{Precision}\cdot\text{Sensitivity}}{\text{Precision}+\text{Sensitivity}}$	Combines sensitivity and precision.

[a] Assuming that accepting a sample means labeling it negative.

is also a rate (see Appendix C.2) for which a high rate is desirable. It is to be noted that the F_1-measure does not consider True Negatives. Hence, if we are interested in the true negatives a model returns, it might be best to use a different performance measure.

Example 6 provides an example of a confusion matrix and the resulting performance measures.

Example 6 *Consider the two deepfake detection algorithms described at the beginning of this section. Both algorithms receive 10,000 test images, 9990 of which are real, 10 of which are fake. Algorithm A has confusion matrix (2.2), algorithm B has confusion matrix (2.3). The corresponding performance scores can be seen in Table 2.3.*

$$\begin{bmatrix} 9990 & 10 \\ 0 & 0 \end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix} 9980 & 0 \\ 10 & 10 \end{bmatrix} \tag{2.3}$$

Table 2.3: The performance scores of algorithms A and B.

Measure	Algorithm A's score	Algorithm B's score
Accuracy	0.999	0.999
Sensitivity	1	0.999
Specificity	0	1
Precision	0.999	1
Fall-out	1	0
Miss rate	0	0.001
F_1	2	2

Indeed, we see that it will depend on what we deem important what algorithm we will wish to use. Both have the same accuracy and F_1-score. Algorithm B has a (slightly) worse sensitivity and miss rate; algorithm A has a (slightly) worse precision, and (considerably) worse specificity and fall-out.

3

Background II: Video Manipulation and Its Detection

Video manipulation is a broad term that could encompass a number of techniques. We would like to clearly distinguish between:

- video *generation*, which is the creating of *fully synthesised* videos from scratch,
- video *manipulation*, which is the altering of existing video content,
- video *translation*, which is the altering of video style[1].

Different types of video generation and manipulation will be discussed in section 3.5.3. Video manipulation *detection* is a binary *classification* task: we provide a neural network with an image[2] and it tells us whether it expects it to belong to the category 'real' or 'fake'.

Since we will not be taking sound into account, the videos we consider are technically nothing but sequences of images. Image manipulation and its detection are both examples of *computer vision* tasks: tasks that require computers to interpret real-world images and their spatial correlations. However, there are some important differences between videos and images that need to be considered when trying to generate, manipulate or detect videos rather than images:

- videos are more heavily compressed than images, and compression may hide traces of forgery [97];
- videos are sequences of frames with many temporal constraints, meaning that simply projecting a series of images, no matter how realistic the images, need not result in realistic videos: the correlation between subsequent images is at least as important as the quality of the individual images.

We will use the terms *image* and *video frame* interchangeably. It should be clear from context whether the term 'image' indicates an independent image or a video frame.

Throughout this report, video manipulation and its detection will be the main focus. However, to prevent unnecessary complications, we will sometimes talk about image manipulation and its detection. This is because video manipulation algorithms have to alter videos frame by frame, meaning that if we understand how images are manipulated, we also understand how video frames are manipulated. Similarly, detection algorithms need to analyse videos frame by frame, meaning that if we understand how manipulated images are detected, we can understand how manipulated video frames can be detected. We will talk about video manipulation and its detection when temporal constraints are involved; more on this follows in section 3.5.4.

The remainder of this chapter will first explain how images are prepared for being processed by neural networks (section 3.1). Then, it will elaborate on specific types of neural networks used for video manipulation (section 3.2), moving on to Deepfake datasets (section 3.3), then explaining convolutional networks (section 3.4), and concluding by giving a detailed review of different types of detection strategies (section 3.5).

[1] Examples of translation are image-to-emoji or image-to-painting. Translation is not considered in the remainder of this research. We refer the interested reader to, e.g., Isola et al [56], Taigman et al. [121], and Zhu et al. [141].

[2] Or with a sequence of images, in the case of interframe detection, see section 3.5.4.

3.1. Preprocessing

One of the difficult aspects of Deepfake detection is that, compared to other fields of computer vision, the available data is limited. For general computer vision tasks, there are more, larger, and more varied datasets available. However, even when data availability is not a problem, storing the data requires much memory space. A trick that has been designed to circumvent this problem, is *preprocessing*. Preprocessing is the generation of new images from existing images in the dataset, using various operations. Throughout this research, we apply the following preprocessing operations: rotation, shearing, horizontal flipping, vertical and horizontal shifting, and zooming. These operations are visualised in Fig. 3.1.

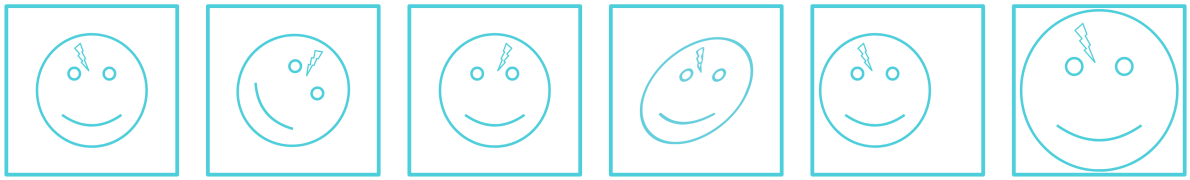


Figure 3.1: Examples of preprocessing operations. From left to right: the original image, a rotated image, a horizontally flipped image, a sheared image, a horizontally shifted image, and a zoomed image.

3.2. Neural Networks for Image Generation

As explained in section 2.3.3, loss functions are often based on probability distributions. This is because neural networks often model probability distributions. In the case of image generation, this probability distribution will help the model generate new samples. The better the model probability distribution approximates reality, the more likely each produced sample is to look realistic. In the case of image manipulation detection, we are asking the algorithm the question, 'How likely is it that this image is real?'

Mathematically, these concepts can be described as follows: Suppose $x \in X \subseteq \mathbb{R}^n$ is an input vector[3]. Then x is regarded a sample of an unkown probability distribution $p^*(x)$. We want to find a *model distribution* $p_\theta(x)$ with parameters θ that approximates $p^*(x)$. During training, the network learns the parameters θ that help approximate $p^*(x)$ as closely as possible.

When the generation has to satisfy any constraints other than producing an image that would fit in dataset X, we are concerned with *conditional* probabilities. The unknown conditional probability distribution would then be $p^*(x|y)$, where y would, for example, be the image of which x needs to be a manipulation. It is clear that in the case of generating/manipulating images of human faces, both the conditional and the unconditional distributions are unknown; there are no available probability distributions describing any person's face. Therefore, the model learns the distribution $p_\theta(x)$ (resp. $p_\theta(x|y)$), which approximates the real distribution. This will allow the algorithm to generate plausible samples of the person's face, without knowing the real-world probability distribution of the face. Example 7 elaborates on conditional probabilities.

Example 7 (Conditional Probability) *Consider a neural network trained to generate faces with a given facial expression. Then X will be the dataset containing images of people's faces; the input vector x represents a single image of some person's face. The labels y would be facial expressions, so the unconditional unknown probability distribution $p^*(x)$ would result in samples with arbitrary facial expressions; the conditional probability distribution $p^*(x|y)$ would result in samples with prescribed facial expressions.*

We have now built a toolbox sufficient to understand the neural networks used most often for image generation and manipulation. We will discuss them in the remainder of this section. For each type of network, we will first explain how they work, and then provide a brief overview of their history. The aim of this historical overview is twofold:

- to offer the interested reader enough handholds to dive deeper into the subject matter if desired;
- to help the reader understand other texts on video manipulation and generation, by already being able to separate the most often mentioned networks.

[3]The complete dataset $X = \{x_1, ..., x_m\}$ should consist of m i.i.d. samples of $p^*(x)$ (because then $\log p_\theta(X) = \sum_{x \in X} \log p_\theta(x)$) [66].

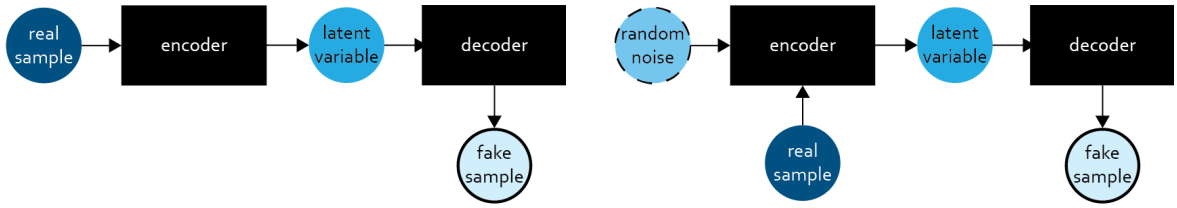
We will not go into hybrid models, although they are also popular. We refer the interested reader to VAE-GAN [80], and would like to point out that in the context of image manipulation, GANs often have convolutional networks for their encoder and decoder networks. Otherwise, we feel the information provided below should suffice to provide the reader with a solid basis with which hybrid methods, should they be encountered in literature, can also be understood.

3.2.1. Autoencoders

An autoencoder is a neural network consisting of two subnetworks: an *encoder* and a *decoder*, see Fig. 3.2a. The encoder receives a real sample image and encodes it into the latent variable. The decoder attempts to decode this latent variable back to the input image. Mathematically, an autoencoder takes an input $x \in \mathbb{R}^n$. Its encoder is a function $f : \mathbb{R}^n \to \mathbb{R}^m, x \mapsto h$ for some $m < n$. The decoder is a function $g : \mathbb{R}^m \to \mathbb{R}^n, h \mapsto \hat{y}$.

Autoencoders were first introduced by Rumelhart et al. [105] in 1986 [13]. Up until recently, autoencoders were mostly used for reducing dimensionality and learning features; nowadays, they are also used as a generative model [41]. Generally, the interesting part of the autoencoder is the encoder; the decoder only serves to make sure that the encoder maps important features into the latent variable [41].

During training, an autoencoder minimises the loss between x and $\hat{y}$ using backpropagation and gradient descent. The loss function used by autoencoders depends on the task for which they are being trained. Autoencoders are designed to ensure their outputs cannot be perfect copies of their input [41]. For example, when an encoder has to reduce the dimensionality of its input, the decoder is forced to make approximations when mapping the latent variable back to the original dimensionality.



(a) An autoencoder.

(b) A variational autoencoder.

Figure 3.2: Autoencoders versus variational autoencoders.

3.2.2. Variational Autoencoders

Variational Autoencoders (VAEs) and regular autoencoders have similar network structures. However, a VAE's encoder does not deterministically map a real sample to a latent variable; it maps a randomly perturbed real sample to a latent variable, see Fig. 3.2b. This means that VAEs are stochastic, whereas autoencoders are deterministic. This allows them to create new content (e.g. images) based on the random noise they receive for input.

The posterior probability density $p_\theta(h|x)$, mapping x to latent variable h, is defined by

$$p_\theta(h|x) = \frac{p_\theta(x,h)}{p_\theta(x)},$$

where $p_\theta(x)$ is the marginal distribution, and $p_\theta(x,h)$ the joint density [66]. The marginal distribution, defined by $p_\theta(x) = \int p_\theta(x,h)dh$, is often intractable [66]. As an immediate consequence, the posterior is also intractable ($p_\theta(x,h)$ is tractable) [66].

The VAE's encoder is a probability distribution $q_\varphi(h|x)$ that approximates the intractable $p_\theta(h|x)$ [66], and the decoder is a probability distribution $p_\theta(x|h)$, mapping h back to x. The VAE's training criterion is equivalent to maximising the loglikelihood of $p_\theta(x)$. We will provide [66]'s derivation of the criterion:

$$\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{q_\varphi(h|x)}(\log p_\theta(x)) \\
&= \mathbb{E}_{q_\varphi(h|x)}\left(\log\left(\frac{p_\theta(x,h)}{p_\theta(h|x)}\right)\right) \\
&= \mathbb{E}_{q_\varphi(h|x)}\left(\log\left(\frac{p_\theta(x,h)}{q_\varphi(x|h)}\frac{q_\varphi(x|h)}{p_\theta(h|x)}\right)\right) \\
&= \mathbb{E}_{q_\varphi(h|x)}\left(\log\frac{p_\theta(x,h)}{q_\varphi(x|h)}\right) + \mathbb{E}_{q_\varphi(h|x)}\left(\log\frac{q_\varphi(x|h)}{p_\theta(h|x)}\right).
\end{aligned}$$

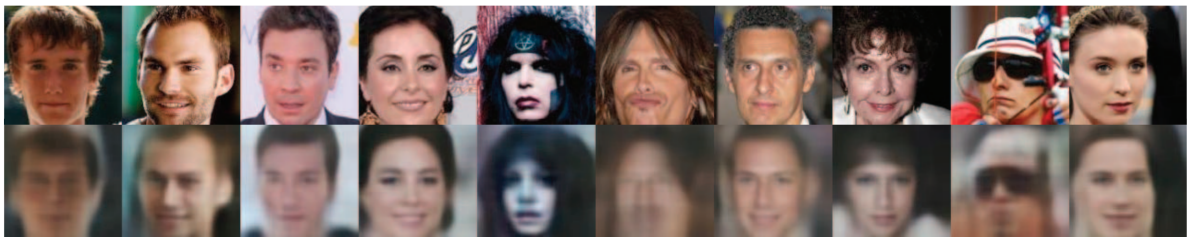


Figure 3.3: Examples of images reconstructed by a variational autoencoder as provided by [50]. The top row contains real images from CelebA; the bottom row contains the images the variational autoencoder reconstructed.

The first term is called the Evidence Lower Bound (ELBO), also called variational lower bound; the second term the *Kullback-Leibler divergence* (KL-divergence). KL-divergence measures the difference between two probability distributions [41]. Hence, maximising the ELBO (with respect to θ and φ), will maximise $p_\theta(x)$ and minimise the difference between $q_\varphi(h|x)$ and $p_\varphi(h|x)$, thereby improving the VAE [66].

We denote the ELBO $L_{\theta,\varphi}(x)$. The ELBO is maximised using SGD. However, $\nabla_{\varphi,\theta} L_{\theta,\varphi}(x)$ is intractable. For the unbiased estimator w.r.t. θ, we provide the derivation from [66]:

$$\begin{aligned}
\nabla_\theta L_{\theta,\varphi}(x) &= \nabla_\theta \mathbb{E}_{q_\varphi(h|x)}\left(\log \frac{p_\theta(x,h)}{q_\varphi(x|h)}\right) \\
&= \nabla_\theta \mathbb{E}_{q_\varphi(h|x)}(\log p_\theta(x,h) - \log(q_\varphi(x|h)) \\
&= \mathbb{E}_{q_\varphi(h|x)} \nabla_\theta(\log(p_\theta(x,h) - \log q_\varphi(x|h)) \\
&\simeq \nabla_\theta(\log p_\theta(x,h) - \log q_\varphi(x|h)) \\
&= \nabla_\theta \log p_\theta(x,h)
\end{aligned}$$

For the unbiased estimator w.r.t. φ, we need something called the *reparametrisation trick* (because ∇_φ and $\mathbb{E}_{q_\varphi}(h|x)$ cannot be interchanged) [66]. Instead of defining $h \sim q_\varphi(h|x)$, we define $h = f(z,\varphi,x)$, where z is a sample from a probability distribution which is independent of x and φ [66]. Consequently, we can rewrite

$$L_{\theta,\varphi}(x) = \mathbb{E}_{q_\varphi(h|x)}\left(\log p_\theta(x,h) - \log q_\varphi(h|x)\right) = \mathbb{E}_{p(z)}\left(\log p_\theta(x,h) - \log q_\varphi(h|x)\right).$$

Kingma et al. [65] first introduced variational autoencoders. Fig. 3.3 shows examples of images reconstructed by variational autoencoders. The image is taken from Hou et al. [50], who trained their variational autoencoder on CelebA [82]: a dataset containing over two hundred thousand images of over ten thousand celebrities.

3.2.3. Normalised Flow

Normalised flow has the same structure as a VAE; the difference is that it uses invertible functions in its encoder. Using the reparametrisation trick, it assumes a random variable $z_0 \sim p(z)$, and maps (x, z_0) to the latent variable h recursively: $f(z_0, x) = (f_T \circ \cdots \circ f_1)(z_0, x)$, where $f_i, i \in \{1, ..., T\}$ is invertible [66]. Consequently, we can write the Jacobian as $\frac{dh}{dz_0} = \sum_{t=1}^{T} \frac{dz_t}{z_{t-1}}$ [66]. This in turn yields

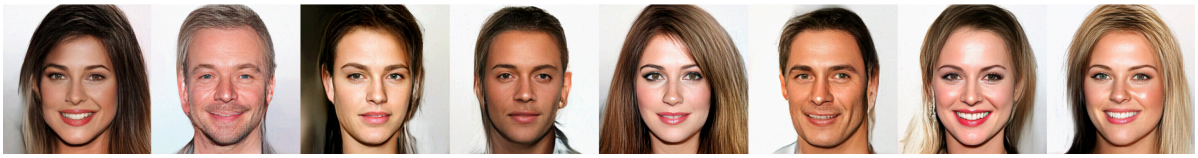
$$\log\left|\det\left(\frac{dh}{dz_0}\right)\right| = \sum_{t=1}^{T} \log\left|\det\left(\frac{dz_t}{dz_{t-1}}\right)\right|.$$

Hence, normalised flow results in tractable probability densities.

The first flow-based model was NICE, introduced by Dinh et al. [29] in 2015. In 2017, Dinh et al. [30] presented an improved version of the model called realNVP, using an affine transformation rather than an additive one. Fig. 3.4a shows images reconstructed by realNVP[4]. Kingma et al. [67] improved this model further yet, and called their algorithm Glow; their results are shown in Fig. 3.4b. Another improvement, Flow++, was published by Ho et al. [47], but they focussed mostly on non-human objects. Recently, Kumar et al. [69] extended realNVP and Glow to a flow-based video generation algorithm called VideoFlow.

[4]Taken from https://laurent-dinh.github.io/2016/07/13/real-nvp-visualization.html.

(a) Examples of images reconstructed by realNVP.



(b) Examples of images reconstructed by Glow as provided by Kingma and Dhariwal [67].

Figure 3.4: Examples of images reconstructed by flow-based models.

3.2.4. Recurrent Networks

As described in section 2.1, recurrent neural networks (RNNs) are networks with recurrent connections, or feedback connections: connections between nodes in the same layer, or in previous layers.

Theoretically, it is possible to train an RNN to represent any dynamical system [14]. However, since training RNNs requires BPTT (see section 2.4.3), vanishing and exploding gradients[5] can affect recurrent networks when many iterations through time are executed [14, 41]. A popular solution to this problem is Long Short-Term Memory (LSTM) [41] (see Fig. 3.5).

An LSTM network stores its state inside a so-called 'memory cell' [43]. The information flow into and out of this cell is controlled by gate nodes [43]. Other popular features LSTMs can have are a forget gate and a peephole; depending on the specific task the LSTM will be used for, desired features can be selected [43].

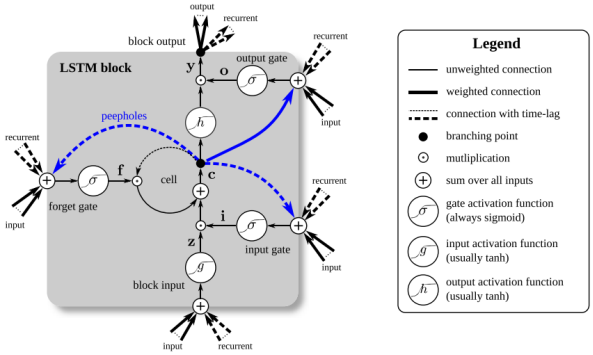


Figure 3.5: An LSTM-block (taken from Greff et al. [43]).

The input gate decides how much of the state of time $t-1$ should be updated with the input of time t [14]. Similarly, the output gate decides how much of the state should make up the output [14]. At time t, the forget gate can choose to discard data stored in the state from time $t-1$; the forget gate operates before the input gate does [14]. The peephole was introduced to provide the input, output, and forget gates access to the state; this solves the problem that with a closed output gate, the gates do not receive any information about the current state [38].

Long Short-Term Memory was first introduced in 1997 by Hochreiter et al. [48]. Forget gates were introduced by Gers et al. [37]; peephole connections by Gers et al. [36]. The first generative recurrent model was pixelRNN, introduced by Van Den Oord et al. [128], and improved by Van Den Oord et al. [127]. The model is inspired by i.a. NADE [72], and uses LSTM. Methods such as PixelRNN (and PixelCNN, see section 3.2.6) and NADE are also called *autoregressive* methods, because they model one colour channel[6] of one pixel at a time [99]. Autoregressive methods have *tractable*[7] likelihoods [47].

3.2.5. Generative Adversarial Networks

Generative adversarial networks (GANs) consist of two neural networks. Although autoencoders and VAEs also consist of two neural networks, we can see in Fig. 3.6 that they are nothing like GANs.

GANs consist of a generative network and a discriminative network. These two networks play an

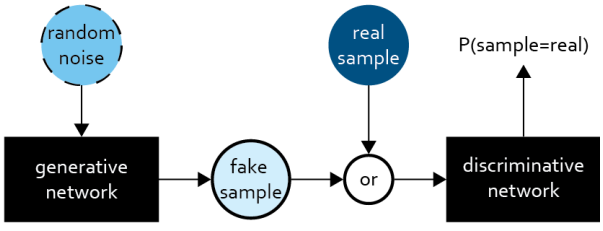


Figure 3.6: A generative adversarial network.

[5] In deep networks, as backpropagation moves through layers, errors can become so small they *vanish* [32]. This problem is called *vanishing gradients*, and it makes updating weights in the first layers of deep networks difficult [32]. Another gradient related problem is *exploding gradients*, where the slope of the gradient is too large [14]. The result is that the gradient descent algorithm will update the weights with such large amounts per iteration, that it passes the loss function's minima altogether [41].

[6] That is: Red, Green, Blue, the three RGB-channels.

[7] Contrary to, for example, VAEs; see section 3.2.2.

Figure 3.7: Images generated by ProGAN as provided by Karras et al. [62].

adversarial game. The generative network receives random noise ($x \in \mathbb{R}^n$) and attempts to transform this into a specific variable. In the case of image manipulation, this variable would be an image in a specific domain (e.g., cats). The discriminative network receives for input either the variable generated by the generative network, or an actual variable from the intended domain (e.g., a real image of a cat). The discriminative network's job is to tell which of the two it received. It does this by returning a value between 0 and 1, where 1 means it is absolutely certain it received the real sample; 0 means it is absolutely certain it received the generated sample. The aim is to maximally confuse the discriminative network, i.e. have it always return 0.5.

If we denote the generative network with network parameters θ by $g_\theta : \mathbb{R}^n \to \mathbb{R}^n, x \mapsto h$ and the discriminative network with network parameters $\tilde{\theta}$ by $d_{\tilde{\theta}} : \mathbb{R}^n \to [0,1], h \mapsto p$, then the training criterion of GANs can be described by [41]

$$\min_{g_\theta} \max_{d_{\tilde{\theta}}} \left(\mathbb{E}_{p_{\text{data}}} \log d_{\tilde{\theta}}(x) + \mathbb{E}_{p_{\text{data}}} \log \left(1 - d_{\tilde{\theta}}(g_\theta(h))\right)\right).$$

GANs were first introduced by Goodfellow et al. [42]. An important improvement was the Wasserstein GAN, which stabilised training, introduced by Arjovsky et al. [9]. Donahue et al. [31] started working towards using GANs for 'unsupervised learning of rich feature representations for arbitrary data distributions' with BiGAN. Kim et al. [63] introduced DiscoGAN, which improved robustness 'to the mode collapse problem' for 'discovering cross-domain relations'. Saito et al. [109]'s TGAN and Kurutach et al. [71]'s CIGAN generate videos rather than images. Zhu et al. [141] introduced CycleGAN, improving training stability by imposing *cycle-loss*: requiring the GAN to map its output back to its input. Bansal et al. [10] extended CycleGAN to Re-CycleGan, which further improves cycle-loss and incorporates temporal constraints to enable ReCycleGAN to generate videos. Tulyakov et al. [126] separate inputs' 'motion' and 'content' in their MoCoGAN, processing the two separately. Karras et al. [62]'s ProGAN offers a breakthrough in the image size GANs are able to generate by increasing the amount of hidden layers and the resolution of images during training, progressively scaling up images (for their results, see Fig. 3.7). Choi et al. [19]'s StarGAN reduced the amount of data needed for training by learning distributions across more than two domains, allowing the network to build on the knowledge already gained on other domains when learning a new domain.

3.2.6. Generative Convolutional Networks

Convolutional networks use *filters* to gain understanding of spatial correlations in images. They are the main building block of Deepfake detection and will be explained in section 3.4. As generative models, CNNs were introduced as PixelCNN, along with PixelRNN, by Van Den Oord et al. [128] and improved to Gated PixelCNN in [127]. Their results for Gated PixelCNN can be seen in Fig. 3.8. Salimans et al. [110] published another improvement: PixelCNN++.

Figure 3.8: Images generated by Gated PixelCNN as provided by Van Den Oord et al. [127]. The left image was used as input; the other images are generated by Gated PixelCNN.

3.3. Deepfake Datasets

Several Deepfake datasets have been published over the years. We will only discuss the three we use for our research. For a more complete overview, we refer the reader to, e.g., Johnston and Elyan [60] and Verdoliva [129].

3.3.1. FaceForensics++

FaceForensics++ [107] is based on 1,000 real videos, downloaded from YouTube by the authors. For the Deepfake subset of their dataset, the authors use a faceswap implementation available on github[8]. This implementation "is based on two autoencoders with a shared encoder that are trained to reconstruct training images of the source and the target face, respectively". The resulting face "is then blended with the rest of the image using Poisson image editing".

3.3.2. Celeb-df

Celeb-df [78] is based on 590 real videos, downloaded from YouTube by the authors. They explain that the Deepfake algorithm they use is an autoencoder with "improv[ed] resolution of the synthesised face". The authors "apply a colour transfer algorithm between the synthesised donor face and the input target face", increasing the realism of the Deepfakes. They also "create a smoothing mask based on the landmarks on eyebrows and interpolated points on cheeks and between lower lip and chin", yielding Deepfakes with less visual artefacts left by the mask used. Lastly, the authors "reduce temporal flickering of synthetic faces in the Deepfake videos by incorporating temporal correlations among the detected face landmarks". These adjustments all increase the visual quality of the Deepfakes.

3.3.3. DeeperForensics-1.0

DeeperForensics-1.0 [59] uses the real videos provided by FaceForensics++ as target videos; they collect their source videos from 100 paid actors. For the generation of Deepfakes, they use a variational autoencoder. They "disentangle structure (i.e., expression and pose) and appearance (i.e., texture, skin colour, etc.) of a face". They remove obvious artefacts in, e.g., skin colour using a "masked adaptive instance normalisation module". To reduce flickering and other temporal inconsistencies, they incorporate temporal constraints in the shape of Markov assumptions and optical flow assumptions.

DeeperForensics-1.0 also purposely incorporates distortions unrelated to images being Deepfakes or not. These distortions are, as listed by the authors:

- saturation adjustments,
- insertion of blocks of pixels (see e.g. 026_W008 in Fig. B.10),
- colour contrast adjustments,
- Gaussian blur,
- white Gaussian noise (see e.g. 123_W012 in Fig. B.10),
- JPEG compression,
- video constant rate factor adjustments.

[8] https://github.com/deepfakes/faceswap

3.4. Convolutional Networks

As we will find in section 3.5.4, most image manipulation detection makes use of convolutional neural networks. This section will first explain why they are useful for computer vision, and how they work. Then, in section 3.4.2, we will provide an overview of research h

Convolutional neural networks (CNNs/ConvNets) are feedforward neural networks [32] that use convolution instead of weight matrix multiplication in one or more layers [41]. A simple feedforward network such as we saw in Fig. 2.3 needs to receive input vectors, meaning $n \times m$ images need to be transformed to $mn \times 1$ vectors to be processed by the network. Convolutional models, on the other hand, can capture any spatial correlations across images rather than being restricted to correlations that are visible in a vector. This is why the convolutional network is able to understand that a rotated or horizontally flipped face (see Fig. 3.1) are also faces.

In convolutional neural networks, there are generally two types of filters: convolutional filters and pooling filters. The remainder of this section will explain these two types of filters.

First, we will explain the convolutional operation theoretically. To that end, suppose a CNN takes an image $X \in \mathbb{R}^{n \times m}$ for input. The convolutional operation uses a kernel, which in this case would also be two dimensional, say $K \in \mathbb{R}^{k \times \ell}, k \leq n, \ell \leq m$ [41]. The convolution takes the form of

$$(X * K)_{i,j} = \sum_{\kappa} \sum_{\tau} X_{\kappa,\tau} K_{i-\kappa, j-\tau}.$$

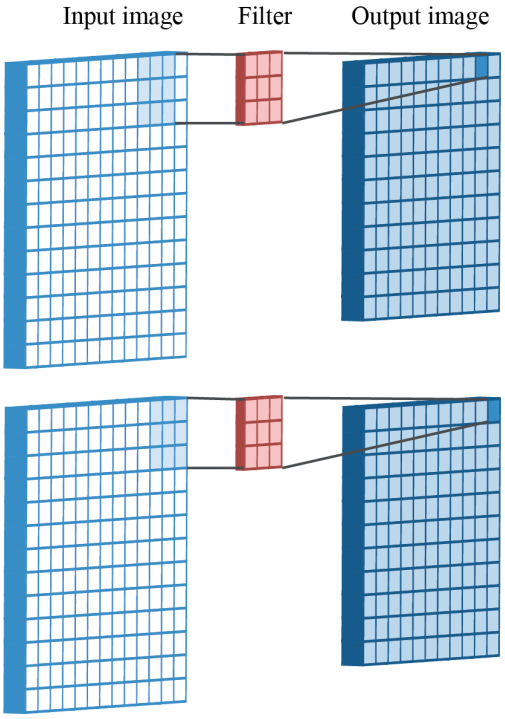


Figure 3.9: Applying filters to images in convolutional neural networks. The filter slides over the input image from left to right (top to bottom), summarising pixel values. These images show the last two iterations on the top row of the input image with a 3×3 filter; next, the filter will slide back to the right side of the image, but one pixel row lower, to continue summarising the remainder of the image. In theory, the order in which the filter passes over the image is arbitrary.

In practice, this comes down to the operation shown in Fig. 3.9, where the filter functions as follows: for the first iteration, the filter F would be placed over the top right corner of the image, say X^1. If we denote

$$F = \begin{bmatrix} F_{11} & F_{21} & F_{31} \\ F_{12} & F_{22} & F_{32} \\ F_{13} & F_{23} & F_{33} \end{bmatrix}, X^1 = \begin{bmatrix} X_{11} & X_{21} & X_{31} \\ X_{12} & X_{22} & X_{32} \\ X_{13} & X_{23} & X_{33} \end{bmatrix},$$

then the resulting pixel in the output image would be calculated as follows:

$$F(X^1) = \sum_{i=1}^{3} \sum_{j=1}^{3} F_{ij} X_{ij}.$$

Depending on the values of F_{ij}, convolutional filters can detect different patterns in images. Examples of such patterns are horizontal edges, vertical edges, or circles. Such relatively easy patterns are often found in early layers; deeper in the convolutional network, filters combine these simpler shapes to detect more difficult patterns, such as eyes. In convolutional neural networks, the entries F_{ij} are learnable parameters. This is how the network determines what patterns are useful for the task it is being trained for. For example, a neural network distinguishing different car models would have no use for recognising patterns corresponding to eyes. Fig. 3.10 shows an image of a four on which a filter is applied that detects horizontal edges. The filter used to achieve this is

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}.$$

Whereas convolutional filters are used to emphasise specific patterns, pooling filters are used to reduce dimensionality without altering the semantic content of the image. The most popular pooling operations are average pooling and max-pooling. If we denote the max-pooling operation by $F_{\max}$, and average pooling by F_{avg}, then continuing with X^1 as defined earlier in this section, we have

$$F_{\max}(X^1) = \max_{i,j\in\{1,2,3\}}(X_{ij}), F_{\text{avg}}(X^1) = \frac{1}{9}\sum_{i=1}^{3}\sum_{j=1}^{3} X_{ij}.$$

For RGB-images, convolution and pooling are applied to each colour channel separately [20].

3.4.1. Classification Networks

In this research, convolutional networks will be used for image classification tasks. However, convolutional layers are only able to find filters that can be used to find relevant patterns in data. They cannot decide how the presence (or absence) of specific patterns relates to the content of an image. To this end, a *classification network* has to be stacked on top of the convolutional network. We will use the term *convolutional body* to refer to the convolutional layers underneath the classification network.

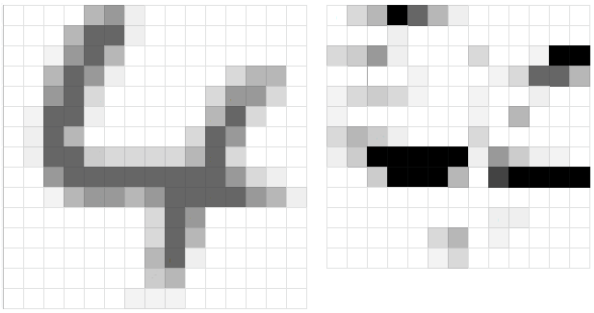


Figure 3.10: The result of applying a convolutional filter to an image: the left image shows the original input, the right shows the result after convolution with a 3×3 filter to detect horizontal edges is used. The images are in grey scale, and their brightness corresponds to a value between 0 and 1. Indeed, in the left image we see the horizontal edge is emphasised, whereas the remainder of the image is .

A classification network consists of one or more fully connected layers. The output layer of the classification network will have a Sigmoid or Softmax activation function (see section 2.2), which will result in output in the shape of the probability of an image belonging to a particular class. If the classification network contains more than one fully connected layer, the other fully connected layers will use ReLU or improved ReLU activation functions.

3.4.2. Well-Known Convolutional Networks and Ongoing Research

Since LeCun et al. [74] introduced modern convolutional networks forty years ago, they have been improved considerably. However, many fundamental questions still remain unanswered, such as how to design a network that will work optimally on a given task, what effects architectural changes will have on network performance, or what the best approach is to scale up networks. Hence, research into the design of neural networks is abundant. This research has different aspects, such as speeding up the training process or achieving higher performance scores. This section will elaborate on some well known convolutional networks, along with other research highlights. It is not an exhaustive study; the research field moves too rapidly to do an exhaustive study at this point. Rather, this sections aims to provide a feeling of how young this research field is, and how little practices have moved beyond hypothesising. It should also provide the reader with enough knowledge of convolutional networks to be able to read additional literature without too much trouble.

As an example of how little evidence is available to support certain conclusions, we provide an elaboration of convolutional network's robustness against compression in Appendix B.2.

3.4.2.1. 2012: AlexNet

In 2012, Krizhevsky et al. [68] introduced AlexNet, which is a relatively shallow network with five convolutional layers and three fully connected layers. It was the convolutional network that re-sparked research into using convolutional networks for image recognition. Krizhevsky et al. [68] found that removing a convolutional layer degraded performance.

3.4.2.2. 2015: GoogLeNet, Inception Modules, VGG, Highway Networks

In 2015, Szegedy et al. [119] introduced GoogLeNet. GoogLeNet is made up of *Inception modules* (see Fig. 3.11), making the network sparser[9] to

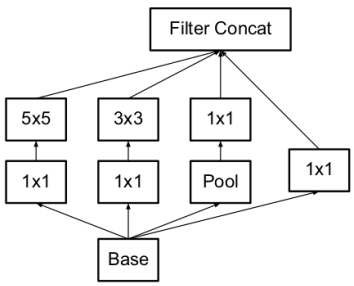


Figure 3.11: An Inception module as proposed by Szegedy et al. [119].

[9] This sparsity is implemented by using convolutional layers that are not fully connected. As can be seen in Fig. 3.11, each convolutional filter only sends its output to a selected few convolutional filters in the next layer, rather than to all convolutional filters as would have

curtail overfitting and reduce computational costs. This was necessary because recent trends were adding more layers to networks, implementing larger layers in networks, and using dropout. Inception modules incorporate a 1×1 convolutional layer before any 3×3 or 5×5 convolutional layer to reduce computational costs and use more large filters in higher layers to capture higher abstraction. GoogLeNet is 12 times smaller than AlexNet (in terms of parameters), yet more accurate.

In the same year, Simonyan et al. [113] introduced VGG-networks of different depths (see Fig. 3.12); VGG-networks are relatively simple networks with a purely feedforward flow. Simonyan et al. [113] state that the Local Response Normalisation that Krizhevsky et al. [68] used does not improve performance, but does increase memory use and computational costs. They also state that two 3×3-layers have the same receptive field as one 5×5-layer, although the latter results in a less discriminative decision function (due to fewer non-linearities) and considerably more parameters. Furthermore, they use 1×1 convolutional filters to further increase the non-linearity of the decision function without affecting receptive fields.

Simonyan et al. [113] stress that although their network has more parameters and is deeper than Szegedy et al. [119]'s, they converge faster due to "implicit regularisation imposed by greater depth and smaller convolutional filter sizes; pre-initialisation of certain layers". They combined the scores of several networks by averaging the softmax posteriors. Interestingly, combining two networks resulted in better performance than combining seven.

Another initiative to solve the issues with training deep networks, *Highway Networks*, was published by Srivastava et al. [116]. These networks implement "an LSTM-inspired adaptive gating mechanism that allows for computation paths along which information can flow across many layers without attenuation".

3.4.2.3. 2016: Residual Learning, Network Scaling, Inception-v3, and Stochastic Depth

In 2016, He et al. [46] confirmed that accuracy saturates and then decreases upon increasing network depth, but not due to overfitting. He et al. [46] attempt to prevent this accuracy drop by using *residual learning* (see Fig. 3.12): "Denoting the desired underlying mapping as $\mathcal{H}(x)$, we let the stacked nonlinear layers fit another mapping of $\mathcal{F}(x) := \mathcal{H}(x) - x$. The original mapping is recast into $\mathcal{F}(x) + x$". This is inspired by Srivastava et al. [116]'s Highway Networks, except Highway Networks have learnable connections, and residual learning uses identity mappings.
Residual learning should make optimisation easier,

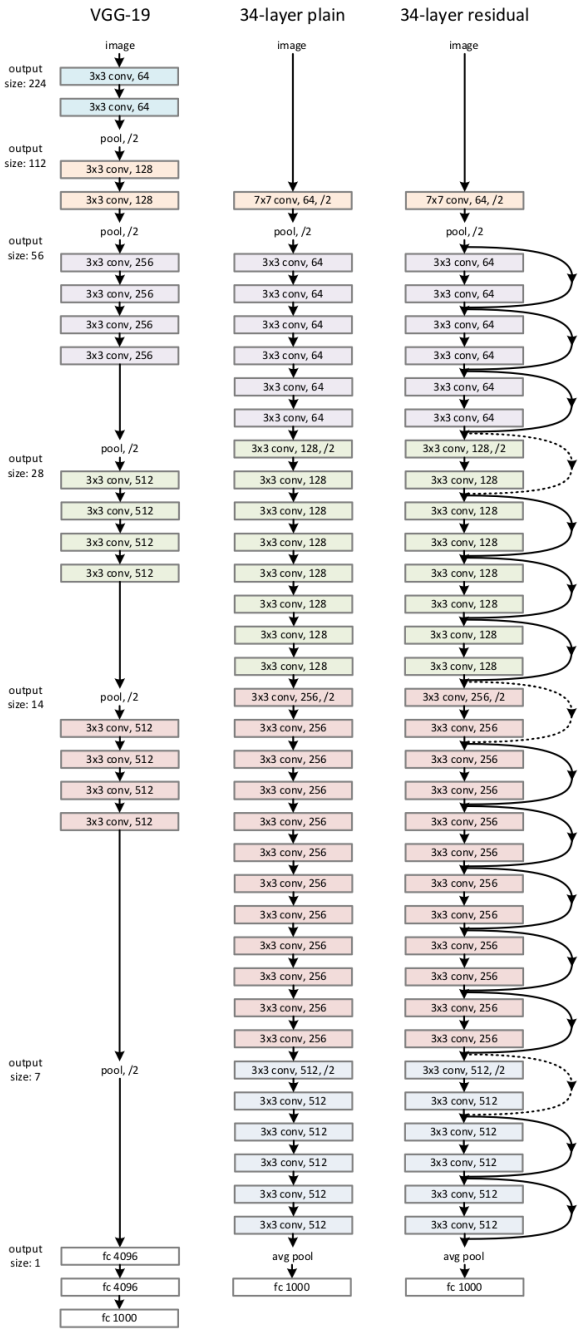


Figure 3.12: The VGG19 and ResNet structures as provided by He et al. [46]. The network on the left is VGG19. VGG16 is similar, but with three layers less: VGG19 has five blocks (blue, orange, purple, green, red), of which the latter three contain four convolutional layers; in VGG16, those three blocks contain only three convolutional layers. The network in the middle is a plain 34-layer network. The network on the right is a 34-layer residual network, with identity connections skipping the convolutional layers. ResNet-152 also consists of four 'blocks', but the blocks are larger: they contain 9, 24, 102, and 9 convolutional layers. The convolutional layers themselves are also different. For example, 2/3 of the layers use 1×1 filters. For further details, see He et al. [46].

been the case in a fully connected layer [119].

and can be implemented with *shortcut connections*, which skip at least one layer. He et al. [46] use identity shortcut connections, meaning they do not increase the amount of learnable parameters, nor computational costs. Consequently, Resnet-152 still has lower complexity than VGG. ResNets also "have excellent generalisation performance".

Zagoruyko et al. [135] performed further research on ResNets. They state that there are three "simple" ways to improve residual networks: adding layers per block, widen the existing layers, and increasing filter sizes. Their own research focusses on width. Previous research kept width purposely minimal, because "as gradient flows through the network there is nothing to force it to go through residual block weights and it can avoid learning anything during training". However, the authors found a way to increase the width of the network without degrading performance; they built a network fifty times smaller and twice as fast as previous residual networks, without diminishing accuracy. The amount of parameters is reported to stay roughly the same. They also found that the type of convolutional layers inside residual blocks did not make much difference, as long as the amount of parameters was kept constant.

Contrary to previous research, Zagoruyko et al. [135] insert dropout between convolutional layers (instead of after the last convolutional layer), increasing performance. Furthermore, their research undermines an earlier claim that residual network depth causes regularisation, as well as the earlier claim that residual networks are effective due to their depth. They state that ResNets generalise better than Inception networks. Additionally, wide networks should be more efficient to train on GPUs, for they allow more room for parallelisation.

Szegedy et al. [120] published another paper on Inception modules, in which they admit that the complexity of their modules complicates changing their network. They contribute GoogLeNet's success mostly to dimensionality reduction. Szegedy et al.'s [120] adapt the same idea as Simonyan et al. [113]: that convolutional filters larger than 3×3 are inefficient. Moreover, Szegedy et al. [120] find that even 3×3 filters can be replaced by the cheaper alternative of a 3×1 filter followed by a 1×3 filter (see Fig. 3.13). The resulting network, Inception-v3, has 42 layers, but "computation cost is only about 2.5 higher than that of GoogLeNet and it is still much more efficient than [VGG]". Inception-v3 outperforms ResNet, while it is "six times cheaper computationally and using at least five times less parameters". Furthermore, Szegedy et al. [120] propose a method called *label-smoothing regularisation* that improves regularisation, making the model "less confident" by randomly changing label distributions.

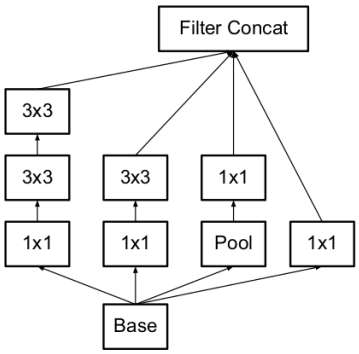


Figure 3.13: The improved version of the Inception module as provided by Szegedy et al. [120].

Lastly, Szegedy et al. [120] conducted experiments with reducing network complexity when increasing resolution, to keep computational costs constant. The more complex networks with lower resolution were slower to train, but their performance is hardly bested by the less complex networks with higher resolution.

Huang et al. [54] introduced convolutional networks of stochastic depth. This aims to curtail vanishing gradients and reduce training time and loss in information flow by "creating deep Residual Network [46] architectures with sufficient modelling capacity; however, during training we shorten the network significantly by randomly removing a substantial fraction of layers independently for each sample or mini-batch". It turns out stochastic depth also improves regularisation.

3.4.2.4. 2017: Inception-v4, Inception-ResNet, Xception, FractalNet, DenseNet

In 2017, Szegedy et al. [118] published yet another paper on Inception networks, presenting Inception-v4, and Inception-ResNet. The former is a deeper and wider version of previous Inception networks; the latter implements He et al. [46]'s residual learning in an Inception network, "since Inception networks tend to be very deep". Inception-ResNet uses cheaper Inception modules than GoogLeNet: "each Inception block is followed by filter-expansion layer (1×1 convolution without activation) which is used for scaling up the dimensionality of the filter bank before the residual addition to match the depth of the input". When the network contained more than 1000 filters, the network suffered from dying nodes. This could be prevented by scaling residuals; this showed to be more effective than He et al. [46]'s suggestion of initially training with small learning rate. They find that Inception-v4 can achieve similar performance as Inception-ResNet,

although residual learning does help speeding up training.

Chollet [21] interpreted "Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation". The paper explains that provided with an RGB-image, a convolutional network finds patterns in three dimensions: height, width, and channels of the image. Inception modules assume "cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly"; Chollet sharpens this hypothesis by stating that the two are completely separable. The resulting network is called Xception[10] (see Fig. 3.14), and is more linearly designed than Inception networks. Xception outperforms Inception-v3 and ResNets; Xception and Inception-v3 are of similar size. Chollet [21] finds that residual connections are paramount for convergence speed and for the performance of Xception. Chollet [21] also finds that "the depth of the intermediate feature spaces on which spatial convolutions are applied [may be] critical to the usefulness of the non-linearity: for deep feature spaces the non-linearity is helpful, but for shallow ones it becomes harmful, possibly due to a loss of information".

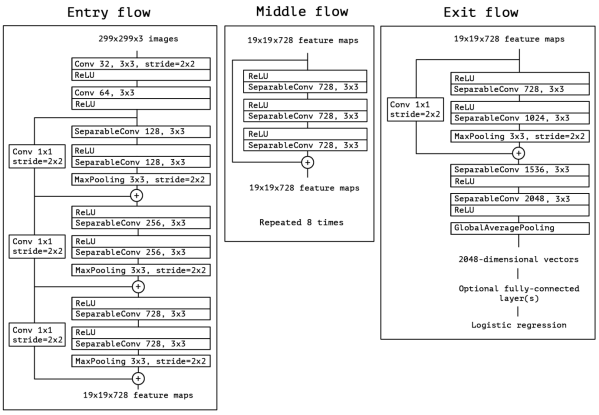


Figure 3.14: A schematic depiction of Xception as provided by Chollet [21].

Larsson et al. [73] introduced FractalNet: a deep network that does not need "explicit residual learning" for training. Their implicit residual learning differs from He et al. [46]'s by considering every signal as important as the others, by using drop-path regularisation (which can prevent sub-networks from co-adapting), and by extracting "high-performance subnetworks consisting of a single column [...], produc[ing] a signal to which a residual could not be added". FractalNet outperforms ResNet.

Huang et al. [53] published DenseNets as a solution to the vanishing gradient problem in deep convolutional networks (see section 3.2.4). DenseNet is shown in Fig. 3.15. The network owes its name to the connections between *all* layers: every layer sends its output to all subsequent layers. Whereas residual connections sum features, DenseNet uses concatenation. This results in less parameters than preceding convolutional networks; probably because older networks learn more filters than strictly necessary. It also reduces the required learning rate for training. DenseNet is more robust to overfitting than preceding networks, and outperforms preceding networks on several datasets while containing less parameters and being computationally cheaper than, e.g., ResNet. DenseNet also implements the 1×1 convolution bottleneck layers introduced by Szegedy et al. [120], which is found to be useful. The model size is further reduced by reducing "the number of feature-maps at transition-layers".

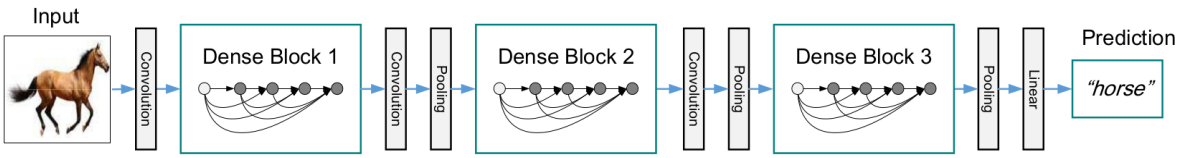


Figure 3.15: A schematic depiction of DenseNet as provided by Huang et al. [53]. DenseNet-121 contains four dense blocks, with 12, 24, 48, and 32 convolutional layers (from left to right).

3.4.2.5. 2018: MesoNet

In the specific context of Deepfake detection, MesoNet [1] is often used as a benchmark network. MesoNet is relatively small, with only four convolutional layers and one fully connected layer. Still, the network reaches 90% accuracy by considering single video frames.

3.4.2.6. 2019: EfficientNet

In 2019, Tan et al. [122] published EfficientNet, which "uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients". This builds heavily on Zagoruyko et al. [135], "quantifying the

[10] Xception stands for Extreme Inception.

relationship among all three dimensions of network width, depth, and resolution".

3.5. Different Types of Detection Methods

This section explores different detection strategies. It is based on the articles that are selected as described in section 4.1. This section is divided into the following sections:

- Section 3.5.2: What technologies are available?
- Section 3.5.3: What types of manipulation are to be detected?
- Section 3.5.4: What strategies can be used to detect manipulations?

It might have been expected that this section would include a categorisation tree based on the type of network used. However, it is not possible to categorise network types using Definition 2. Most detection makes use of convolutional networks; some use GANs, but often these GANs have convolutional layers as well. Even capsule networks, which are presented as alternatives for convolutional networks [93], use convolutional layers themselves.

Similarly, no categorisations were made based on whether or not a network is fully connected: there are also networks that have some fully connected layers, and some other layers. Hence, this classification would introduce a class of hybrids that would need a classification of its own (e.g., locations and amount of fully connected layers), which would be too specific to be useful.

First, section 3.5.1 provides the definitions we used throughout this section. Then, sections 3.5.2, 3.5.3, and 3.5.4 will provide different options in the form of categorisations following definition 2, and provide the advantages and disadvantages of the different categories. All three sections first elaborate the different options available, then explain the advantages and disadvantages of these options. We conclude this section by explaining its consequences for this research.

3.5.1. Definitions

This section provides definitions we will be working with throughout the remainder of this research. The reason for formulating our own definitions is that there is no consensus in the literature as to what definitions are attached to certain terms. Different papers use different definitions, which often have to be derived from context. To avoid confusion, we will clearly state here with what definitions we will be working.

We came to the definitions we are working with by combining all definitions found in literature and moulding them into the definitions that best fit the scope of our research.

Definition 1 (Detection methods) *A video or image manipulation detection method is any method that takes as input (part of) a video or image, respectively, and returns a judgement on the integrity of its visual content.*

Detection methods that produce additional output to the judgement of the images' integrity (e.g., the location of the forgery [107], the GAN which forged the image [85], or the original image the forged image originates from [5]) are beyond the scope of the provided categorisations. The algorithms are taken into account, but only their methods for judging integrity; additional outputs will not be taken into account in the categorisation process.

Definition 2 (Category) *If n sets of detection methods are a partition of their parent category, they are called categories. The root category is defined as 'detection methods'.*

Definition 3 (Deepfake) *A deepfake is a computer generated video that has the goal of forging a target individual's identity.*

Deepfakes can be, but need not be, generated to malicious ends. Definition 3 is specifically ours and is not a definition used throughout all of literature. Some publications consider Deepfakes to be any deep learning video manipulation (e.g., Kumar et al. [70]), some consider Face2Face to create Deepfakes (e.g., Kumar et al. [70]), some take it to be faceswap videos (e.g., Li et al. [77]), and others use the term Deepfake to refer only to videos generated by algorithms called Deepfake (e.g., Cozzolino et al. [27]). Some sources even consider image manipulation to be Deepfakes (e.g., Hsu et al. [52]).

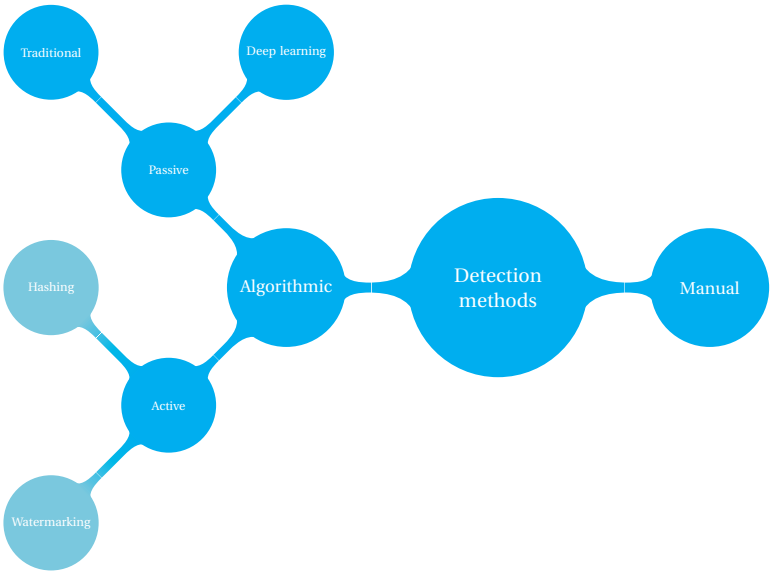


Figure 3.16: Categorisation of detection technologies.

3.5.2. Available Technologies

This section provides a categorisation tree of manipulation algorithms based on available technologies in Fig. 3.16. With no or little computational power available, one will have to settle for manual detection. With computational power available, algorithmic detection can be used. This can then be split into active detection, which requires prior information embedded into an image at the source [40]; and passive detection (also called blind detection [129] or multimedia forensics [104]), which can be used when no prior information is available [40]. Examples of active detection are

- watermarking, which detects manipulations due to traces of alteration in imperceptible watermarks [138]; and
- image hashing, which allows image authentication by means of shared secret keys [138].

This report focusses solely on passive detection since most devices do not currently have the software implemented that would support active detection of images they produce.

Passive detection can be split into

- deep learning methods, which are data-driven, meaning they are fed a large dataset and find their own reasons for deciding an image is real or not; and
- traditional methods, which are either taught these reasons by humans, or manage to highlight traces of manipulation but leave it to humans to decide on the integrity of the image.

3.5.2.1. Advantages and Disadvantages of Different Techniques

As to manual detection with the naked eye, two user studies show that Deepfakes are rapidly becoming too realistic for humans to detect them:

- Rössler et al. [107] performed a user study with the Face2Face algorithm (published by Thies et al. [125]; not a deep learning algorithm). For the version that produces the more realistic videos, human accuracy without compression was 60.57%; with hard compression, human accuracy was 48.93%. In their previous research, Rössler et al. [104] show that even for raw and high-quality images, human accuracy does not exceed 80%.
- Jiang et al. [59] perform a user study in which users have to say whether an image is real or not. Of Rössler et al. [104]'s FaceForensics++, test subjects consider 8.4% to be real; of Li et al. [78]'s Celeb-df, test subjects consider 61.0% real; of Jiang et al. [59]'s own DeeperForensics-1.0, test subjects consider up to 64.1% to be real.

As to the difference between active and passive detection, active detection is generally more robust than passive detection [138], but its applicability is limited [3, 60], because they require special equipment [39]; for fully generated images (see section 3.5.3), there will be no watermark to extract in the first place, because the forgery is not based on a pristine image [52]. Johnston and Elyan [60] also acknowledge that most existing videos lack watermarks.

Passive detection methods can have low accuracy and can be computationally expensive [138]. For fully generated images, passive detection methods may find no traces of manipulation, because no modification of images has taken place [52].

As to deep learning methods, problems lie in them being time-consuming and data demanding [88]. A deep learning algorithm trained on a specific type of manipulation will not detect a different type of manipulation, even if the two types of manipulations are semantically similar [27]. Deep learning methods also suffer the disadvantage that their decision rules are not interpretable for humans [88]. This also results in uncertainty of what deep learning's potential is in the area of image manipulation detection [6]. Moreover, deep learning methods are likely to be biased to a training dataset, meaning they will recognise an image stems from a particular dataset instead of recognising whether it is pristine or not [27]. Another concern about deep learning detection methods is that they might be included in the GAN's training, teaching the GAN's discriminator (see section 3.2.5) to circumvent such methods [89].

On the other hand, traditional methods perform well but depend on assumptions made when designing the detection algorithm [27]. It is also not agreed whether traditional detection methods work well on deep learning manipulation methods. For example, Afchar et al. [1] state that deep learning "disrupts traditional signal processing approaches".

3.5.3. Types of Manipulation

Whereas some researchers aim to detect all types of manipulations (e.g., Nguyen et al. [96]), many only try to detect one type of manipulation. Fig. 3.17 shows the different types of video manipulation that are considered in the literature. The first level of subcategories divides manipulation into manual manipulation and computer-generated manipulation. Computer-generated methods can be:

- Deep learning, which is a data-driven approach: the algorithm receives a large dataset and teaches itself what implies an image is real or fake. Deep learning can be further categorised into
  - attribute (e.g., hair colour, age, gender) change,
  - facial expression manipulation (also called facial reenactment),
  - face swap, and
  - fully synthesised imagery.
- Traditional methods, which need human assistance to build a model they can use to perform video manipulations; they can not learn a model solely from data. However, once they have learned the model, they can manipulate video without further human assistance. An example is Thies et al. [125]'s Face2Face.

Manual video manipulation also makes use of computers. However, in manual manipulation, a human being is involved at all stages of the manipulation process. Manual manipulation has two main categories [60]:

- intraframe manipulation, which can be [60]
  - copy-move manipulation, which is the copying of an object from one location in an image to another location within the same image [60];
  - re-touching, which is, for example, applying a colour filter to an image [60]; and
  - splicing, which is the inserting of an object from one image into another image [60].
- interframe manipulation, which can be [61]
  - frame deletion,
  - frame insertion, or
  - frame shuffling.

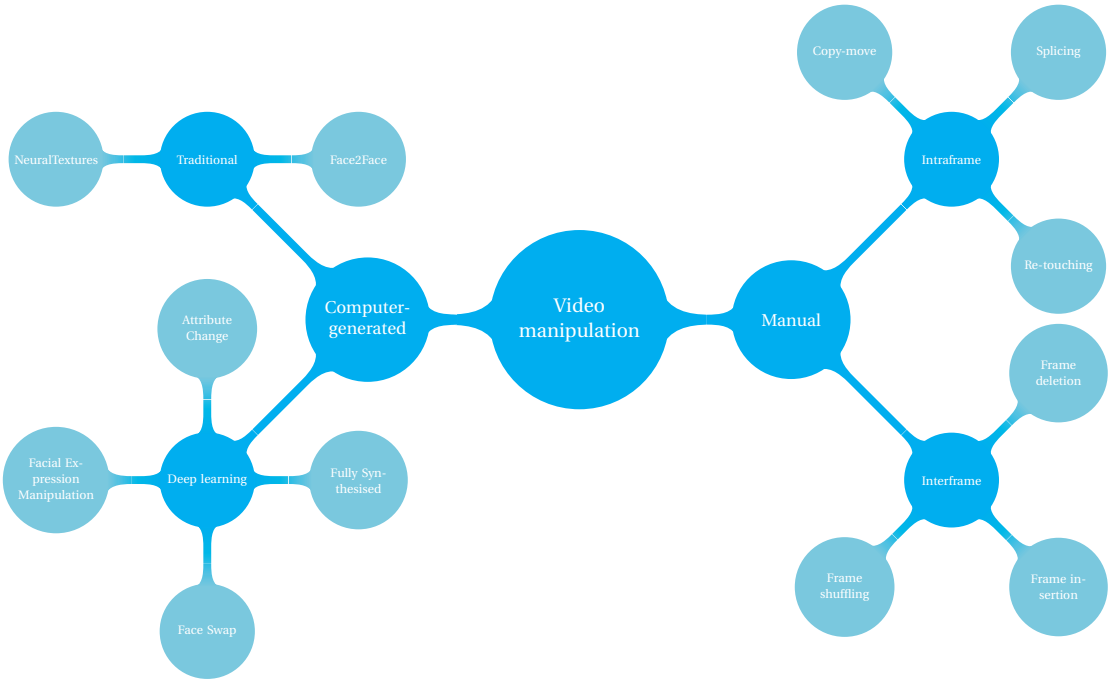


Figure 3.17: Categorisation of video manipulation methods. The cyan blue nodes represent categories; the sky blue nodes represent examples of those categories.

Not all types of manipulation are necessarily malicious; for example, frames can be dropped automatically during compression. Our definition for Deepfakes (definition 3 in section 3.5.1) implies that attribute change does not count as a Deepfake, as the changing of hair colour, age, or even gender, will not allow a person to impersonate someone else. Face swap and fully synthesised videos do satisfy the definition; as does facial expression manipulation, because it allows a person to fit a target individual's facial expression (mouth, especially) to any desired speech.

3.5.3.1. Advantages and Disadvantages of Different Manipulation Techniques

There are several disadvantages to an algorithm focussing on one specific type of manipulation, such as needing many algorithms alongside each other to ward off all manipulations, or needing to know the type of manipulation applied in advance [11]. Therefore, some researchers (e.g. Cozzolino et al. [24]) consider it desirable to focus on algorithms that can detect any type of manipulation. On the other hand, the question is whether this approach is feasible at all: the No Free Lunch theorem (see section B.3) states that the better an algorithm is at one task, the worse it will be at every other task. Therefore, the question is whether detecting manipulations generated by different algorithms is indeed a combination of tasks or a single larger task. For example, it is possible to build a deep learning network that can distinguish between cats and dogs with over 95% accuracy [20], but it is also possible to design a deep learning network that can distinguish different types of dogs [16]).

Some detection methods depend on assumptions that are not satisfied by all types of manipulation. For example, D'Avino et al. [28] assume that at least the background of part of the video is pristine. Such techniques would not work on fully synthesised videos. On the other hand, Marra et al. [86] compute GAN-fingerprints with a technique that assumes a whole image is generated by the same algorithm; this might not work on faceswap algorithms, as they leave parts of an image untouched. Tariq et al. [124] also address that detection focussing on two images having merged will not detect fully synthesised imagery.

It seems GAN-fingerprints are model-specific; not even algorithm specific, but specific to the particularly trained algorithm [134]. This is alarming because much research on Deepfake generation is not executed by scientific researchers and therefore not necessarily published [88]. Moreover, even if an algorithm is published, this means one only has to re-train it to make sure its fingerprints are unknown.

3.5.4. Passive Detection Strategies

This section builds upon section 3.5.2's categorisation, taking 'passive detection algorithms' as root node. Most passive detection algorithms are designed to focus on a specific type of detection traces; the full tree is depicted in Fig. 3.18. There are two main categories to distinguish between: interframe and framewise detection. Interframe detection uses relations between subsequent frames to detect forgery. This can be done using

- semantical artefacts such as
  - too little eye blinking, detected by LSTM [76],
  - discrepancies in the heart rate of a recorded individual, e.g. by measuring small differences in skin colour [35],
  - facial landmark analyses by a model built with authentic videos of high profile people [2];
- statistical inconsistencies, which can be detected by, e.g.,
  - discrepancies in the optical flow of subsequent frames, detected by LSTM [7],
  - comparing image key-points in subsequent frames [143],
  - detecting temporal inconsistencies by applying an LSTM to image residuals of the frames [28, 136],
  - multimedia stream descriptors [45];
- applying a deep learning model to raw RGB-data, letting the model find their own correlations between sequence frames for indications of manipulations. This consists mostly of
  - using long-short term memory to analyse temporal discrepancies between CNN-filtered video frames [44, 108, 114].

Framewise detection extracts single frames from videos and treats them as independent images. Algorithms that use framewise detection and then average the output score over several frames, such as Cozzolino et al. [25], are considered framewise detection. Framewise detection can be divided into:

- statistical detection, which uses low-level features. Examples of such features are
  - local feature descriptors, extracting e.g., image key-points or edges [4],
  - warping artefacts, which are inconsistencies in resolution where a swapped face is merged back into the source image [77],
  - co-occurrence matrices, extracted from the raw RGB-data [92],
  - frequency of saturated and under-exposed pixels, fed into an SVM[11] [89],
  - intensity histograms, showing the amount of pixels per intensity level [8],
  - differences in predicted and actual compression levels [61],
  - camera model artefacts [15, 23, 25, 26],
  - GAN-based artefacts [86],
  - found by transforming input images to the hexadecimal system so they can be considered amino acids sequences, so these sequences can be searched for signatures that suggest an image is (Deepfake) pornography [91];
- semantical detection can be based on
  - missing details in teeth and eyes [88],
  - mismatched eye colour [88],
  - differences in the 3D face models based on landmarks of the central face region versus the whole face, due to imperfect merging of swapped faces into source images [132],

[11] A Support-Vector Machine: a feedforward network that has classes as output, rather than the probability that a sample belongs to a class [41].

- locations of facial landmarks, letting an SVM pick up unnatural deviations [133],
- differences in expression of specific emotions [8],
- missing reflections [88],
- wrongly modelled geometry, causing artefacts around the nose and edge of the face [88],

- deep learning on raw RGB-data, simply feeding a large dataset of unedited images into a deep learning algorithm. The algorithm is not told what traces of manipulation to focus on; it finds its own patterns, which need not be understandable for humans. Some approaches are:
  - purely convolutional networks, which may differ slightly from strategy:
    - Afchar et al. [1] designed the popular shallow convolutional network MesoNet,
    - Huh et al. [55] use siamese convolutional networks to determine whether two images have the same metadata, with EXIF[12] metadata as training supervision,
    - Jeon et al. [58] use pretrained AlexNet,
    - Kumar et al. [70] use five convolutional networks in parallel: four for different facial regions and one for the whole image,
    - Majumdar et al. [84] use VGG,
    - Marra et al. [87] implement iCaRL[13] for classification,
    - Rahmouni et al. [103] compare image patches,
    - Raghavendra et al. [101] combine VGG19 and AlexNet,
    - Tariq et al. [123] generate their own human-created forged images to mimic Deepfakes,
    - Tariq et al. [124] designed their own shallow convolutional model,
    - Yu et al. [134] use an 8-layered convolutional network,
    - Zhang et al. [137] use networks that were pretrained on ImageNet,
    - Zhou et al. [139] use ResNet-101,
    - Zhou et al. [140] use Inception-v3;
  - convolutional autoencoders [94],
  - capsule networks (which are convolutional networks with a dynamic routing algorithm on top) [93, 96],
  - anomaly detection, which only requires pristine data during training and afterwards will spot manipulated samples as being anomalies [49],
  - using LSTM after convolutional layers to locate discrepancies in the spatial relationships between neighbouring pixels [11, 51];

- deep learning on image residuals, or noise residual, envisioned by high pass filters which remove semantic content, and can be used to,
  - feed into a purely convolutional network [84, 90, 134],
  - feed into a binary classifier [24, 75, 140],
  - feed into a SVM [139],
  - perform anomaly detection on third-order derivative filters [28],
  - feed third-order derivative filters into a convolutional autoencoder [27],
  - find discrepancies in compression levels, made visible with Discrete Cosine Transform (DCT) [6, 136],
  - detect median filtering [83].

[12] EXIF metadata are "camera specifications that are digitally engraved into an image file at the moment of capture and are ubiquitously available" [55].

[13] Incremental Classifier and Representation Learning: allows a detection network to learn extra classes, without forgetting the ones it has already learned [87].

One might have noticed that several articles are mentioned in more than one example. This is because some articles compare different approaches (e.g., Zampoglou et al. [136], Yu et al. [134]), and some articles combine several approaches to obtain more reliable results (e.g., Majumdar et al. [84], Zhou et al. [139]). Some of these articles (e.g. Amerini et al. [6], Majumdar et al. [84], Zhou et al. [139], Zhou et al. [140]) are in the 'purely convolutional' category as well as in another; this might seem contradictory, but it is not: it merely means the detection algorithm has at least one purely convolutional stream analysing raw RGB-data.

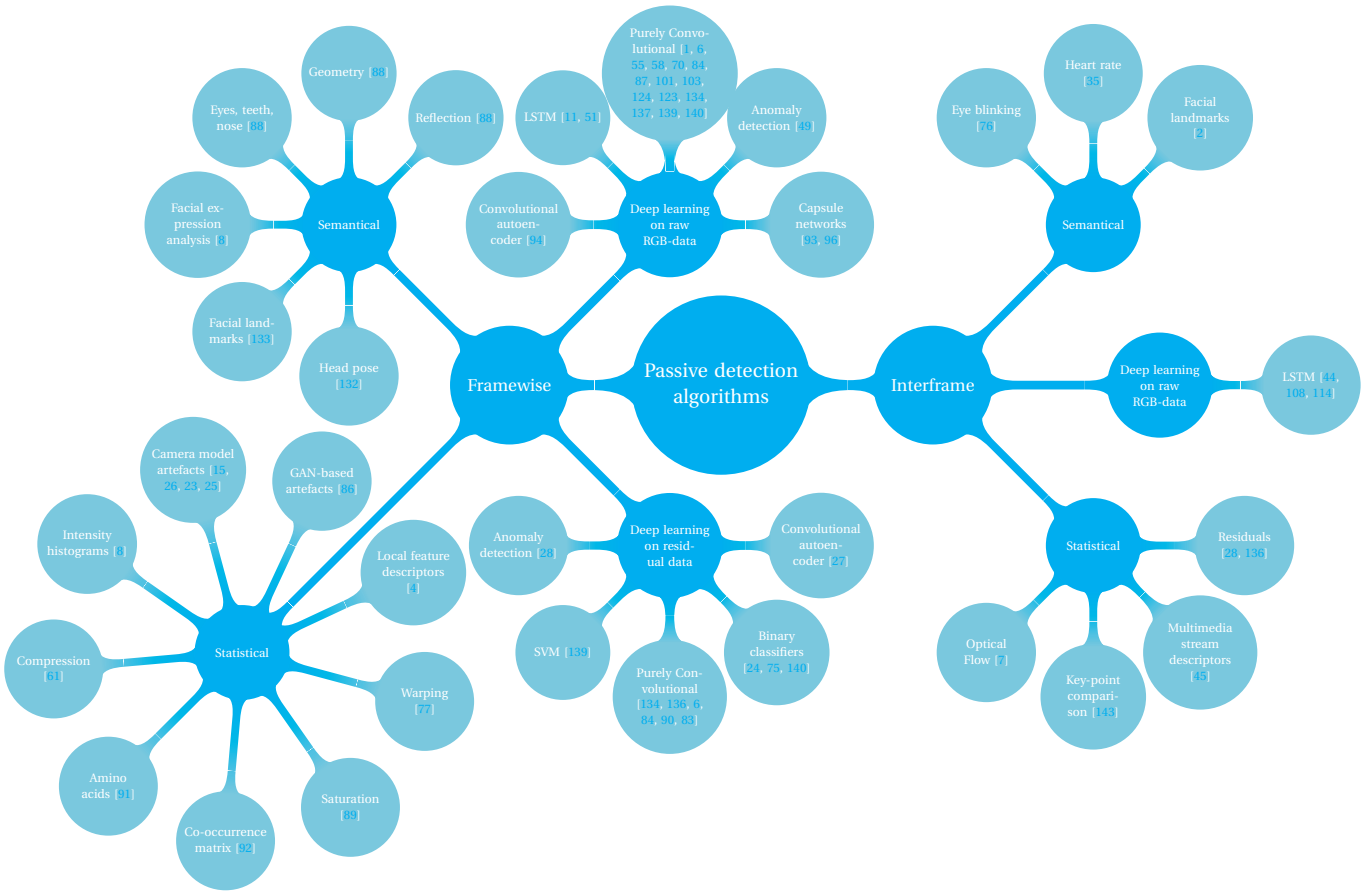


Figure 3.18: Categorisation of detection algorithms: the focus of the detection method. The cyan blue nodes represent categories; the sky blue nodes represent examples of those categories.

3.5.4.1. Advantages and Disadvantages of Different Detection Strategies

Semantical detection is based on, e.g., physical or physiological artefacts. This means these strategies are conceptually easy to understand for humans: it concerns the type of artefacts we would spot ourselves (if they were clear enough). A disadvantage of methods based on specific visual artefacts is that they might not be universally applicable [88]; for example, spotting blurred teeth is only possible in images in which the recorded individual has their mouth open [88]. Moreover, visual artefacts are unlikely to be useful in the future, since the quality of Deepfakes is rapidly improving [88]. For example, Li et al. [76]'s strategy of detecting lack of eye-blinking, was circumvented by adding images with closed eyes in the training phase of generation algorithms [77].

Methods focussing on physical artefacts are dependent on specific modelling assumptions [27]. They are more robust than pixel-level detection, but in realistic situations, pixel-level methods outperform physical artefact based methods [27].

Specifically, Agarwal et al. [2]'s building of a model a person to detect discrepancies in facial landmarks is effective; however, it requires the building of a model for every individual it is to be applied to. Hence, it is not feasible to apply this method to every person in every video.

Detection using image noise, image residuals, or (other) statistical image features are promising because image generation changes image statistics [92]. Opinions on how difficult it will be to circumvent such approaches differ. Marra et al. [86] praise statistical methods because they "rely on the long trail of subtle traces left in each image by the acquisition devices, traces that can be hardly disguised even by a skilled attacker." On the other hand, Tariq et al. [124] state that detection in the residual domain will fail when manipulated images' edges are properly smoothed, and that purely convolutional detection in the residual domain can be "easily" circumvented by skilled attackers. Marra et al. [86] point out that methods detecting device or even camera model fingerprints will likely fail on fully generated imagery since GAN-generated images do not have the necessary camera artefacts.

An advantage of detection based on noise residuals is that these methods can be trained on real data only [25]. Cozzolino et al. [25, 23] explain these results as the methods being applicable to any type of manipulation. Some noise residual-based detection methods do not need to be trained at all, because they can be analysed using traditional algorithms rather than deep learning algorithms [28]. Experiments performed by Zhou et al. [140] indicate that their network using residual images outperforms their method working with raw RGB-data. Zhou et al. [139] on the other hand work with four datasets and on two these datasets, the raw RGB-network outperforms the residual network; on the other two datasets it is the other way around. The downside of such detection methods is that they only work on videos with static backgrounds [28].

A much-discussed type of image residual is Photo Response Non-Uniformity (PRNU), which can be seen as a device fingerprint. Whereas the method works on image statistics, it suppresses most camera artefacts [25]. It works only on dark and textured images, and although the training set does not need to contain manipulated data, it does need an abundance of real imagery to learn PRNU patterns [25, 23, 28]. Another disadvantage of PRNU is that it is sensitive to other image noise [23]. On the other hand, detection based on PRNU enjoys stability, and generality [25].

Another specific residual is Noiseprint, which can be seen as a camera model fingerprint[14]. The method is not limited to cameras of which images are contained in the training dataset [25]. A downside of Noiseprint in the context of video manipulation detection is that the automatic stabilisation cameras apply, suppresses the artefacts the Noiseprint detector uses for detection [25].

Possibly the biggest downside of detection by convolutional models on purely RGB-data is that the networks' findings are not readily explicable for humans [89]. Moreover, when sophisticated post-processing is applied, this type of detection might not work [140]. In line with this concern, Babby et al. [11] express doubt as to whether there are (or will be) sufficient visual clues in Deepfakes for purely convolutional networks to spot them. Marra et al. [87] also point out that convolutional networks would have to be retrained every time a new generation algorithm is published; this is problematic, given the pace with which new generation algorithms are published. However, even if purely convolutional networks are not reliable enough for Deepfake detection, convolutional networks are still worth being researched: they are also the backbones of many statistical detection methods, so understanding them is crucial.

Rahmouni et al. [103] compare the filters their convolutional layer found to filters used in traditional image forensics. Some filters, e.g. horizontal and vertical difference, appeared in both sets of filters. However, the convolutional network also found some more complicated filters that had not yet been discovered by traditional image forensics researchers; possibly, Rahmouni et al. [103] explain, because they are too counterintuitive to be found by human researchers.

The advantage of interframe detection is that not only spatial but also temporal inconsistencies can be detected. Such temporal inconsistencies exist because Deepfake algorithms generate each video frame individually [114], and intuitively make interframe detection sound more promising than framewise detection. However, D'Avino et al. [28] find that the performance of their algorithm only decreases slightly when they use framewise detection instead of interframe detection.

By using framewise detection, image forensics methods that already existed can be readily applied. Nevertheless, this does not mean that these methods perform well on video frames without adjustments; video frames are generally more difficult to process due to their high level of compression [1, 25, 136].

[14] The difference between a model fingerprint and a device fingerprint is that model fingerprints are based on statistical image characteristics caused by the algorithms chosen for image processing in a model; device fingerprints are based on statistical image characteristics caused by manufacturing imperfections.

3.5.5. Consequences for This Research

Regarding the categories provided in section 3.5.2, due to the test results from Jiang et al. [59], Rössler et al. [104], and Rössler et al. [107], manual detection will be considered insufficient, and will not be considered in the remainder of this research. Active detection will not be considered because it will not be available in the short term. Hence, the focus of this research will be on passive detection algorithms.

Regarding the categories provided in section 3.5.3, this research will focus on the detection of face swapped images generated by deep learning algorithms, since most video generation algorithms seem to use this approach. Therefore, this is the type of manipulation most likely to be encountered by the detection algorithm. Moreover, it is the type of manipulation used in most publicly available datasets.

Regarding the categories provided in section 3.5.4, this research will focus on framewise deep learning detection, both using raw RGB-data and using image residuals.

4

Research Methodology

This chapter elaborates the methodology applied throughout this research. Section 4.1 explains the selection criteria and relevance checks used to obtain articles for literature study. Section 4.2 explains how we selected articles to base section 3.5 on. Then, section 4.3 will explain what networks, datasets, and experiments we used to come closer to understanding the effects of design choices when building a convolutional neural network for Deepfake detection. Lastly, section 4.4 explains how we tested the robustness of the designed networks.

4.1. Selecting Articles for Literature Study

Since the research field of Deepfake detection is young, it is not yet clear what papers are important milestones, and what papers are pursuing dead ends. This means there is no clear selection of articles that provide coverage of the whole research field. It is also not clear what related research fields are relevant: detection methods for some non-deep learning manipulation techniques are nearly matured, but it is not yet clear which of those (if any) can be incorporated in Deepfake detection. Hence, our article selection is carefully structured for two reasons:

- to make sure our research does not overlook large areas of the research field;
- to provide clarity on what related research fields are taken into account and which are not.

By elaborating on these choices, this section should allow the future reader to put this research into perspective.

4.1.1. Initial Article Selection

On February 13, 2020, Google Scholar was consulted to find articles about Deepfake detection. Three consecutive searches were executed: two for Deepfake detection, with different parentheses; one for video manipulation detection. These searches amounted to a total of 625 articles. The details are summarised in Table 4.1. For all search terms, patents and citations were excluded.

Table 4.1: The used search terms and the number of hits they resulted in. The 'number of new results'-column contains the number of those hits that had not been turned up by the previous search terms.

Search Term	Number of Results	Number of New Results	Constraints on date
Deepfake detection	513	513	≥2019
"video manipulation" detection "deep learning"	116	92	None
"Deepfake detection"	66	20	None

The reason only one out of three search terms in Table 4.1 is combined with a constraint on the publishing date, is that the other two search terms only amounted to 112 articles in total even without the use of a date constraint. The first search term, however, yielded 513 articles with the date constraint applied; to reduce the amount of time needed to go through all titles and remaining abstracts, the choice was made to impose the date constraint on the first search term. Arguably, any relevant literature missed due to this constraint will

have been mentioned by enough of the considered articles as not to have escaped our notice, because we added articles through a systematic backwards snowballing track.

The resulting articles went through four rounds of relevance checks:

1. The titles were analysed to see whether articles were potentially relevant.
2. The abstracts were read to see whether articles were potentially relevant.
3. The format was checked to see whether articles were potentially relevant.
4. The trustworthiness was checked to see whether articles were relevant.

Each round comes with a set of rejection criteria. If the article did not meet any of these rejection criteria, it was considered (potentially) relevant.

4.1.2. Title Selection

The titles of these 625 articles were analysed as a first relevance check. Reasons to exclude articles were:

- Language: articles in languages other than Dutch, English, or German.
- Content: articles that are not relevant because they cover only social sciences (e.g., psychology, politics, law, religion, philosophy, ethics), or because they have a focus other than the detection of manipulated video or images (e.g., the detection of generated audio, detection of frame dropping, generation rather than detection of video). Articles based on image detection are accepted because they might be applicable to video frames. Similarly, detection of manual manipulation (as defined in section 3.5.3) such as splicing is accepted, because it might be applicable to Deepfakes; after all, face-swapping is a form of splicing. Manual detection of, e.g., frame shuffling is not accepted, because it does not seem applicable to Deepfakes.
- Duplicates: articles were considered duplicates if their titles and authors were identical.

In case of doubt about the scientific field (e.g., if the title did not immediately clarify whether the article was technical or political), the publishing journal or conference was taken into account. If this did not eliminate the doubt, or there was doubt about anything else, articles were not rejected. Table 4.2 shows the amount of articles rejected for each reason. The total number of rejected titles is 412. This leaves 213 titles for the second selection round.

Table 4.2: The reasons for rejecting titles and the amount of titles rejected for each reason.

Rejected for	Number of titles
Subject	361
Duplicate	16
Language	35

4.1.3. Abstract Selection

During abstract reading, the ‘content’ and ‘duplicate’ criterion from the previous section were kept; the ‘language’ criterion was no longer necessary because none of the remaining articles was in a language other than English or German. The operationalisation of ‘duplicate’ had to be extended to cover two new cases:

- articles that had different titles but identical abstracts, and
- chapters of books that were on the list when the whole book was also on the list.

Furthermore, it turned out two new reasons for rejecting articles had to be added:

- detection focus: detection methods based specifically on interframe manipulation (frame insertion, deletion or shuffling); and
- misdirecting titles, referring to presentation only’s or titles that referred to whole proceedings, instead of a single article.

In case the abstract was not clear enough to decide on the relevance of an article, the conclusion was also read. Table 4.3 shows the amount of articles rejected for each reason. The total number of rejected abstracts is 129, which leaves 84 relevant items.

Table 4.3: The reasons for rejecting abstracts and the amount of abstracts rejected for each reason.

Rejected for	Number of abstracts
Subject	114
Duplicate	5
Detection Focus	7
Misdirecting	3

4.1.4. Format Selection

Of the 84 items left, 10 more are rejected because they are not research articles contributions. Of these ten, one is a chapter from a book; nine are theses. This leaves 74 articles.

4.1.5. Trustworthiness Selection

Due to the rapid pace at which the research field is progressing, considering only peer-reviewed articles would lead to a gap between the considered research and the state of the art. In order to stick to trustworthy articles, without having the disadvantage of this lag, two types of articles will be considered trustworthy:

1. peer-reviewed articles, and
2. articles with at least one author who has another peer-reviewed article in the 74 articles under consideration.

This led to the rejection of 22 more articles, leaving 52 articles for the remainder of this research. Of these 52 articles, 44 are peer-reviewed; 8 are not peer-reviewed themselves but have at least one author with another peer-reviewed publication in the remaining selection.

4.1.6. Snowballing through Selected Articles

To make sure no literature was overlooked, backwards snowballing[1] was used to retrieve more relevant literature. We constructed a list of titles on image or video manipulation that were not yet included in our selection. Articles older than 2017 were not considered[2], nor were articles on active detection[3]. Articles on copy-move detection, frame dropping, frame deletion, or frame insertion were not considered. Articles previously rejected due to untrustworthiness were accepted.

For each title on this list, we counted how many of the original 52 selected articles cited it. The result was a list of 143 articles with an average of 2.44 citations. Articles with 5 citations or more were added to the 52 already accepted articles, resulting in 18 additional articles. This amounted to 70 articles in total. Of these 70 articles, 52 presented new detection approaches to some extent. The other articles were bibliometries (1), reviews/overviews (5), proposals of datasets (5), adversarial attacks (2), improvements of existing methods (4), and algorithms to categorise by what GAN an image was generated (1).

4.2. Categorisation of Detection Methods

This section contains the methodology used for the three different categorisations presented in section 3.5. These categories answer the following questions respectively:

1. What technologies are available?
2. What types of manipulation are to be detected?
3. What strategies can be used to detect manipulations?

[1] Studying references of selected articles [130].
[2] Since Deepfakes only surfaced in 2017.
[3] Since we cannot assume all mobile devices to support active detection, see section 3.5.2.

The chosen categories are a combination of categories already provided by literature, and our own categories. Those provided by the 70 selected articles were taken into consideration if and only if they met definition 2. Consequently, 'categories' that overlap each other are ignored, as are 'categories' that leave any detection method without a category. In case a small alteration could make a categorisation satisfy definition 2, the categories were accepted after these alterations were made. The resulting categorisations can be found in chapter 3. The 52 articles presenting new methods were sorted into appropriate categories in section 3.5.4.

4.3. Exploring the Effects of Design Choices on Network Performance

We systematically work towards finding the most favourable combination of datasets, model architecture, and hyperparameters for the detection of Deepfake videos with convolutional neural networks. First, section 4.3.1 elaborates the different network architectures used; section 4.3.2 elaborates the datasets used. Then, section 4.3.3 elaborates the experiments conducted.

4.3.1. Networks

For our experiments, we use seven different convolutional bodies and two different classification networks. These will be discussed in sections 4.3.1.1 and 4.3.1.2 respectively.

4.3.1.1. Convolutional Bodies

We use seven different convolutional bodies. Six of those are well-known models imported directly; one is inspired by the literature but built by ourselves.

The latter convolutional body is that of the shallow model presented by Chollet [20] (in listing 5.5). It has four convolutional layers, uses ReLu activation functions, and has a dropout layer. This model was trained from scratch. We will provide a schematic picture of the network together with its classification network in section 4.3.1.2.

The remaining six convolutional bodies are those of networks presented in section 3.4.2: DenseNet-121, Inception-v3, ResNet-152, VGG16, VGG19, Xception. The well-known convolutional models were chosen because they have been frequently iterated upon. This makes them promising networks to work with. From the networks discussed in section 3.4.2, we chose the networks that were available on `keras.applications`. We chose not to work with InceptionResNet because all other networks with inception modules overfitted.

Initially, we chose to work only with the most popular version of each model. Hence, we only included VGG19. However, when of all 2fc-models (see section 4.3.1.2), Shallow-2fc performed most stably on DeeperForensics-1.0, and VGG19-2fc reached the highest accuracy on DeeperForensics-1.0, we figured VGG16-2fc might be the perfect tradeoff between accuracy and stability. Therefore, VGG16 was included as well.

DenseNet-121, Inception-v3, ResNet-152, VGG16, VGG19, and Xception were imported pretrainedly from Keras; the pretraining dataset is ImageNet. This way, we expand the approaches of Zhang et al. [137] and Zhou et al. [140], who use Inception-v3 pretrained on ImageNet.

Chollet [20] explains that if the dataset used to pretrain the network is sufficiently large, "the pretrained network can effectively act as a generic model of the visual world, and hence its features can prove useful for many different computer-vision problems, even though these new problems may involve completely different classes than those of the original task." Chollet [20] mentions ImageNet as a dataset that satisfies this 'sufficiently large' condition and explains that re-using the convolutional body and only training the classification layer is a legitimate way of using pretrained networks, although with this approach the convolutional filters will not be finetuned on the training dataset.

A possible disadvantage of using ImageNet as a pretraining dataset is that ImageNet is a dataset with mostly non-human classes, meaning the convolutional filters might not be optimal for the detection of human features. On the other hand, as Zhang et al. [137] explain, this approach forces the network to focus on training its classifying layers, reducing the risks of it overfitting on a small dataset.

In some cases (especially when very deep networks are concerned) it is preferable to freeze only the first convolutional layers, because the later layers learn very specific filters [20]. However, due to time restrictions, we did not pursue this approach.

4.3.1.2. Classification Networks

Each convolutional body is used twice[4], with different classifying layers on top:

[4] At first, we only worked with the first classification network. When the networks started overfitting, we decided to look into the amount of (learnable) parameters they contained. This was when we realised that the classification layers were unrealistically large. When

- once, convolutional bodies are followed by a dropout layer and two fully connected layers (one with 512 nodes and ReLU activations, one with 1 node and a Softmax activation);
- once, convolutional bodies are followed by a global average pooling layer and one fully connected layer (with one node and a Softmax activation).

This first approach agrees with Chollet's [20] shallow model (provided in listing 5.5); the latter with all pretrained models except the VGG networks: those originally had three fully connected layers. For brevity, the networks with two fully connected layers will be denoted as Shallow-2fc[5], DenseNet-2fc, Inception-2fc, ResNet-2fc, VGG16-2fc, VGG19-2fc, Xception-2fc; the networks with one fully connected layer will be denoted as Shallow-1fc, DenseNet-1fc, Inception-1fc, ResNet-1fc, VGG16-1fc, VGG19-1fc, Xception-1fc.

Fig. 4.1 depicts Shallow-2fc. Table 5.9 provides an overview of the amount of (trainable) parameters each network has.

Table 4.4: The (trainable) parameters of the convolutional networks used for detection with raw RGB image data.

Models	**Parameters**			
	Two fc-layers		**One fc-layer**	
	Total	**Trainable**	**Total**	**Trainable**
Shallow	6,795,457	6,795,457	240,961	240,961
DenseNet-121	25,912,897	18,875,393	7,038,529	1,025
Inception-v3	38,581,025	16,778,241	21,804,833	2,049
ResNet-152	109,752,193	51,381,249	58,372,993	2,049
VGG16	24,152,897	9,438,209	14,715,201	513
VGG19	29,463,106	9,438,722	20,024,897	513
Xception	72,242,729	51,381,249	20,863,529	2,049

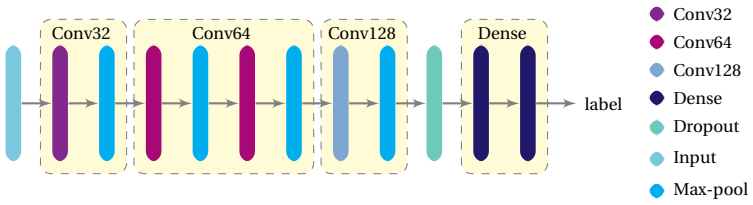


Figure 4.1: A schematic depiction of Shallow-2fc. Each convolutional layer is followed by a max-pooling layer with filter size (2,2). All convolutional layers have ReLU activation functions, and filters sized (3,3); Conv32 has 32 of these filters, Conv64 has 64 of these filters, and Conv128 has 128 of these filters. The convolutional layers are followed by a dropout layer. The first fully connected layer has 512 nodes with ReLU activation functions; the last fully connected layer has 1 node with a Sigmoid activation.

4.3.2. Datasets

Furthermore, we use five different datasets, composed from three different publicly available datasets. These public datasets are FaceForensics++ [104], Celeb-df [78], and DeeperForensics-1.0 [59]. We will elaborate the five resulting datasets in the remainder of this section.

FaceForensics++ From the FaceForensics++ dataset, only the Deepfake subset is used. Hence, unless stated otherwise, from now on we will use the term 'FaceForensics++' to refer exclusively to its Deepfake subset.
The FaceForensics++ dataset used in this research is constructed by downloading the 1000 Deepfake videos and 1000 original videos from the FaceForensics++ dataset (at compression level c40[6]). The frame extraction

digging into this, we found it more intuitive to use a global pooling layer followed by a classification layer containing a single node. This can be interpreted as each separate filter in the last layer of the convolutional body being represented by a single weight, allowing the network to decide what filters are important for distinguishing between 'real' and 'fake' classes. We expected to have to re-do all experiments because the 1fc-models would outperform the 2fc-models. When this was not the case, we decided to let both models exist alongside each other.

[5] The only difference between Shallow-2fc and the shallow model Chollet [20] provides in listing 5.5 is that Shallow-2fc has a dropout layer and takes differently sized input.

[6] FaceForensics++ is available raw, c23, and c40, which is respectively no, light, and heavy compression. Compression levels al relevant when building a detection algorithm, see appendix B.2.

script FaceForensics++ provides was altered to extract the first 25 frames of each video. These frames were cropped using `autocrop`[7] to create face crops of size 400 × 400. The frames `autocrop` failed to crop were not considered to keep the dataset uniform. In case `autocrop` failed to crop the first frame of a video, the next cropped frame was added to the dataset. In case no frames of a video had been cropped, nothing was added to the dataset. The final dataset consists of 944 fake frames and 953 real frames.

Larger FaceForensics++ This subset of the FaceForensics++ dataset, consisting of 30 fake videos and 31 real videos, was downloaded at compression level c23. The frames from these videos were extracted using the script FaceForensics++ provides. Frames 320 through 499 of fake video 638_640 were deleted because `Pillow`[8] could not open them. This resulted in 14,148 real frames and 15,844 fake frames. These remaining frames were cropped using `autocrop` to create face crops of size 400 × 400. The frames `autocrop` failed to crop were not considered to keep the dataset uniform. The final dataset consists of 13,128 real frames, and 15,327 fake frames.

Largest FaceForensics++ All 1000 Deepfake videos and all 1000 original videos from the FaceForensics++ dataset were downloaded (at compression level c40). The frame extraction script FaceForensics++ provides was altered to extract the first 25 frames of each video. These frames were cropped using `autocrop` to create face crops of size 400 × 400. The frames `autocrop` failed to crop were not considered to keep the dataset uniform. The final dataset consists of 23,039 fake frames and 23,301 real frames.

Celeb-df To construct this dataset, the first 25 frames were extracted from all real Celeb-df videos using `ffmpeg` (version 3.4.8). The frames were cropped using `autocrop` to create face crops of size 400 × 400. The frames `autocrop` failed to crop were not considered to keep the dataset uniform.
From the real videos, the first autocropped version of the first frame from each video was included in this dataset. For 70 real videos, autocrop failed to crop the first frame. For 34 videos, at least one other frame was successfully autocropped; for those videos, the first of those autocropped frames was added to this dataset. From the remaining 36 videos, no frames were added. In total, this dataset contains 842 real cropped frames. For the Deepfake counterpart, only the first frame was extracted for each fake Celeb-df video. These frames were autocropped using the same settings as for the real videos. Only part of these videos was added to the Deepfake counterpart of the Celeb-df dataset used[9]. The frames from these videos that were successfully cropped[10] we added to the dataset. The resulting dataset consists of 853 fake frames.

DeeperForensics-1.0 The real part of this dataset is the real part of FaceForensics++. The fake part of this dataset is based on the 1.000 Deepfakes of random-level distortions of the DeeperForensics-1.0 dataset. To construct it, the first 25 frames were extracted using `ffmpeg`. The frames were cropped using `autocrop` to create face crops of size 400×400. The frames `autocrop` failed to crop were not considered to keep the dataset uniform. The first autocropped version of the first frame from each video was included in this dataset. From 73 real videos, autocrop failed to crop the first frame. For 31 videos, at least one other frame was successfully autocropped; for those videos, the first of those autocropped frames were added to the dataset. From the remaining 42 videos, no frames were added. In total, the dataset contains 958 real cropped frames.

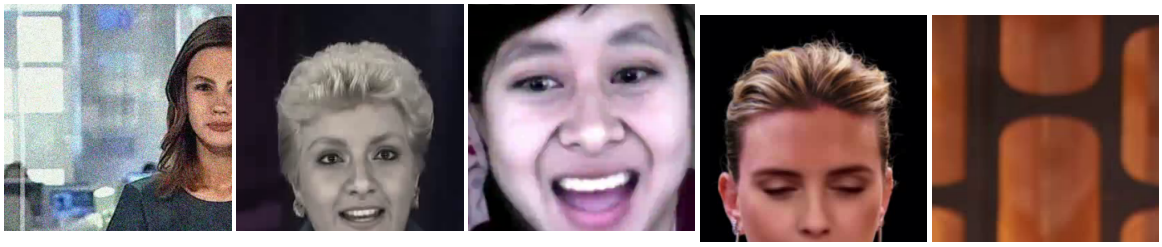
4.3.2.1. Cropping

The `autocrop` programme does not crop flawlessly. In case the obtained face crop was still recognisable as part of a face, it was accepted as part of the dataset. Fig. 4.2 shows examples of accepted and rejected video frames.

4.3.3. Experiments Conducted

We conduct several consecutive experiments to find the most favourable datasets, network architecture, preprocessing settings, and hyperparameters for Deepfake detection. Our study is not exhaustive; the research

[7] A Python programme that automatically crops portraits: https://pypi.org/project/autocrop/; we used version 1.0.2.
[8] A Python imaging library, allowing Python to perform image processing, see https://pypi.org/project/Pillow/; we used version 7.1.2.
[9] The extracted frames of videos with names ending in 0001, 0002, 0003.
[10] Which was 267, 298, and 288 times respectively.



(a) An accepted imperfect face crop from DeeperForensics-1.0.

(b) An accepted imperfect face crop from DeeperForensics-1.0.

(c) An accepted imperfect face crop from DeeperForensics-1.0.

(d) An accepted imperfect face crop from Celeb-df.

(e) A rejected imperfect face crop from Celeb-df.

Figure 4.2: Examples of imperfect face crops.

field is moving too rapidly to perform an exhaustive study at this point. However, we conduct experiments with enough variety to point us in the right direction of favourable settings for Deepfake detection.

The experiments are conducted consecutively. Each experiment continues with the settings that yielded the most favourable results in the previous experiment. What settings are considered most favourable depends on three criteria:

- maximum accuracy achieved,
- minimum loss achieved,
- training stability (e.g. overfitting, underfitting).

We use Python 3.8.0 and `Keras` 2.3.1 on `tensorflow` 2.2.0. All training is done with a 80%/20% training/validation data split. All training was done using binary cross-entropy as an error measure, and RMSProp as an optimiser. Accuracy is the only considered metric[11]. All runs are executed on an AWS instance `t3.xlarge`[12], until we had to switch to `c5.4xlarge` due to time restrictions; the descriptions of the experiments will indicate clearly when this was the case.

Due to time restrictions, the first four experiments are conducted on Shallow-2fc[13]. This was considered a suitable choice because according to Chollet [20], the network can successfully be trained to distinguish between cats and dogs (which is also a binary classification problem) using only a small dataset. Literature also shows that shallow networks can perform well on Deepfake detection tasks, see e.g. Afchar et al. [1], Rahmouni et al. [103], and Tariq et al. [123].

Also, due to time restrictions, every run will be executed only once. Appendix B.5 defends why this choice should not result in technical issues. Furthermore, we argue that since we do not aim to find 'the best' algorithm, but want to understand the effects different choices have on performance, one run will suffice to provide enough insights.

Appendix B.5 also explores the consequences this choice has for our research. We can summarise our findings as follows. When only one run has been executed, no conclusions can be drawn on account of

- underfitting,
- differences in accuracy smaller than 5%,
- differences in loss smaller than 0.05,
- degradation of the accuracy or loss curves after reaching the maximum accuracy or minimum loss if the training and validation curves degrade together.

Consequently, if a network underfits, it will be run again. If this still results in underfitting, the underfitting will be considered as occurring structurally. If the underfitting does not occur in the second run, we will include the second run in our results.

The remainder of this section will elaborate on the experiments conducted.

[11] This is because we aim to understand how to build a working algorithm. Once the algorithm works, decisions about performance measures can be taken into account. First, accuracy needs to be such that the model fits the problem, and does not overfit.

[12] This choice is explained in appendix A.

[13] Shallow-1fc is not more computationally expensive than Shallow-2fc. However, we only came up with Shallow-1fc after the first four experiments had already been conducted. We decided that there was no need to rerun the experiments with Shallow-1fc, since it would conflict with our choice for time efficiency.

4.3.3.1. Experiment 1: Dataset Composition

Several Deepfake datasets are publicly available (see section 3.3), but it is up to the researcher to decide how to use them. Our experiment works with different compositions of the FaceForensics++ dataset. These datasets contain different amounts of frames from different numbers of videos. This way, we can analyse whether dataset size or dataset variety is more important.

For this experiment, only Shallow-2fc is used. The hyperparameters are set to dropout rate 0.1, batch size 100, and image input size 128 × 128. No preprocessing is used. Network training is terminated after 50 epochs.

The times elapsed for training are also reported, so we can find the relation between training time and dataset size.

4.3.3.2. Experiment 2: Preprocessing

Preprocessing can be used to create a higher variety in data, see section 3.1. This experiment executes one run with heavy preprocessing and one with light preprocessing. Heavy preprocessing means we use rotation range 40, shear range 0.2, zoom range 0.2, width shift range 0.2, height shift range 0.2, and horizontal flipping; light preprocessing means we use rotation range 25, shear range 0.2, zoom range 0.2, and horizontal flipping. This way, we can analyse what effects different types of preprocessing have on our performance criteria.

For this experiment, only Shallow-2fc is used. The dataset used is FaceForensics++. The hyperparameters are set to dropout rate 0.1, batch size 100, and image input size 128 × 128. Network training is terminated after 150 epochs[14]. If the network had not finished training at that point, it was trained again for 250 epochs.

The times elapsed for training are also reported, so we can find the relation between training time and preprocessing.

4.3.3.3. Experiment 3: Dropout

Using dropout can deter overfitting, see section 2.4.6. This experiment tries different dropout rates to analyse its effect on classification performance.

The times elapsed for training are also reported, so we can find the relation between training time and dropout.

For this experiment, only Shallow-2fc is used. The dataset used is FaceForensics++. The hyperparameters are set to batch size 100, and image input size 128 × 128. We use heavy preprocessing. Training was terminated after 250 epochs. If the network had not finished training at that point, it was trained again for 350 epochs.

4.3.3.4. Experiment 4: Batch Size

Since we use mini-batch learning, we have to set a batch size for each run. As we know from section 2.4, mini-batch learning aims to combine the advantages of batch learning and stochastic learning. This experiment aims to provide insight into the relation between batch size and our performance criteria. To this end, we execute a run with batch size 50 and a run with batch size 250.

For this experiment, only Shallow-2fc is used. The dataset used is FaceForensics++. The hyperparameters are set to dropout rate 0.1, and image input size 128 × 128. Network training was terminated after 250 epochs.

The times elapsed for training are also reported, so we can find the relation between training time and batch size.

4.3.3.5. Experiment 5: Datasets and Network Architectures

This experiment compares the performance of different network architectures on different datasets: it trains Dense-1fc, Dense-2fc, Inception-1fc, Inception-2fc, ResNet-1fc, ResNet-2fc, Shallow-1fc, Shallow-2fc, VGG16-1fc, VGG16-2fc, VGG19-1fc, VGG19-2fc, Xception-1fc, and Xception-2fc on FaceForensics++, Celeb-df, and DeeperForensics-1.0. The aim of this experiment is to provide insight into what networks are most suited for Deepfake detection on the relatively small datasets we have available. Since we use networks with large differences in i.a. depth and amount of parameters, we hope this will also provide insight in what type of network is most suited for Deepfake detection on the relatively small datasets. Training on three different datasets should provide insight into the differences in performance when using different datasets to achieve the same task.

The batch size is set to 100. Since scaling up in one dimension (in this case, depth) is not desirable (see section 3.4.2), the image size is increased to 200 × 200. This should be more suitable for deeper networks. We also use image size 200 × 200 for the shallow models, so comparison is still possible. Heavy preprocessing is

[14] Training for 50 was no longer enough for the network to finish training.

used. The 2fc-models use dropout rate 0.1; we do not use dropout for the 2fc-models, since we have already frozen their convolutional bodies. Network training is terminated after 100 epochs. If the network had not finished training by then, it is trained again for 250 epochs.

The times elapsed for training are also reported so that we can find the relation between training time, network architecture and datasets.

4.3.3.6. Experiment 6: DCT-Residuals

In this experiment, we train the network on image residuals instead of original images. The idea behind this is that the image residuals help the network focus on statistical image content instead of semantical image content. Amerini et al. [6] and Zampoglou et al. [136] also use DCT-residuals. However, whereas they use the DCT's histograms for detection of Deeepfakes, we use the DCT-residuals directly.

We computed the DCT-residuals of images by importing them in Python and using the built in `dct`-function from `scipy.fftpack`; the images are saved in PNG-format using `cv2` (version 3.4.9.33). We transformed all three datasets used in the previous experiment. The implementation of the DCT-transform in `scipy.fftpack` we use is DCT-II. Given input image x and input image y, this means the image is transformed by

$$y_k = 2\sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi k(2n+1)}{2N}\right),$$

where k is the pixel in image y that is being computed, N the amount of pixels in x[15]. In our case, this means $N = 40,000$, $0 \leq y \leq 39,999$.

For this experiment, we use the eight networks that exceeded 80% accuracy in the previous experiment: ResNet-1fc, ResNet-2fc, Shallow-1fc, Shallow-2fc, VGG16-1fc, VGG16-2fc, VGG19-1fc, and VGG19-2fc. Apart from the datasets used, this experiment is conducted identically to the previous experiment.

The times elapsed for training are not reported, although they might have been useful to clear up unexpected differences in run time in the previous experiment. However, due to time restrictions, the `t3.xlarge`-instance had to be updated to a `c5.4xlarge`-instance halfway through the experiment. Therefore, comparison of runtimes is not possible.

4.4. Generalisation and Robustness

Now that we have tested the effects of different design choices on our network performance, we have managed to exceed 80% accuracy with eight different networks. However, our goal is not to train our network to perform well on one specific dataset. Rather, we want the network to use this dataset as a means to be able to detect forged samples from any Deepfake dataset. To find out how capable our networks are of this task, we test their generalisation and robustness. To this end, we evaluate their performance on unseen samples. This validation is twofold:

- evaluation on other datasets than the network was trained on;
- evaluation on unseen frames from the dataset the network was trained on.

The remainder of this section briefly elaborates on the networks we use, then explains how we composed the unseen datasets, and lastly describes the experiment we conducted.

4.4.1. Networks

We only use networks that reached a validation accuracy of 80% or higher in experiment 5. These are eight networks: ResNet-1fc, ResNet-2fc, Shallow-1fc, Shallow-2fc, VGG16-1fc, VGG16-2fc, VGG19-1fc, VGG19-2fc, all trained on DeeperForensics-1.0. However, since the networks are frozen after a set number of epochs instead of at their maximum accuracy achieved, the referencing accuracies are not the maximum ones. Instead, they are as reported in Table 4.5.

4.4.2. Datasets

Since all networks used for testing robustness are trained on DeeperForensics-1.0, the only test dataset we need to compose is one of DeeperForensics-1.0; for Celeb-df and FaceForensics++, we can simply use the training datasets already composed, as the algorithms in question have never seen those datasets.

[15] See `https://docs.scipy.org/doc/scipy/reference/generated/scipy.fftpack.dct.html`.

Table 4.5: The accuracies of different networks after training has completed, and their maximum achieved accuracies for reference. All training was executed on DeeperForensics-1.0. All accuracies are validation accuracies.

Model	Maximum accuracy (%)	Final accuracy (%)
ResNet-1fc	81.5	81.5
ResNet-2fc	85.8	81.5
Shallow-1fc	92.5	89.7
Shallow-2fc	82.6	63.0
VGG16-1fc	86.5	80.4
VGG16-2fc	90.0	85.1
VGG19-1fc	86.1	80.1
VGG19-2fc	89.3	89.3

The testing dataset of DeeperForensics-1.0 is composed using the frames already extracted and cropped, as explained in section 4.3.2. We use the 25th frame of each video for this dataset if it is correctly cropped. If it is not correctly cropped, which is the case for 70 videos, we use the last frame that was successfully cropped. There are 44 videos for which no frame was successfully cropped; there are 4 videos for which only one frame is correctly cropped. These were not included in the testing dataset, since they are already included in the training dataset. Testing can only be done using unseen samples.

By using the last frame that is successfully cropped, we ensure as much difference between the frames in the test and training dataset as possible.

4.4.3. Experiment 7: Generalisation and Robustness

All four networks are tested on four datasets: FaceForensics++ and Celeb-df to test on unseen datasets, the DeeperForensics-1.0 training set for reference, and the DeeperForensics-1.0 test dataset for the generalisation test. We note that testing on the DeeperForensics-1.0 training dataset will result in a slightly different accuracy than the training accuracy or the validation accuracy: the training accuracy and validation accuracy are based on respectively 80% and 20% of the dataset, whereas the accuracy computed here is based on the whole dataset.

The results on the testing DeeperForensics-1.0 will provide insight into the network's generalisation ability: the frames in this dataset are from the same dataset as the frames the network was trained on. The network's performance on Celeb-df and FaceForensics++ provides insight in its robustness.

The resulting accuracies are reported, along with sensitivity, specificity, precision, fall-out, miss rate, and F_1-measure, where the label 'real' is considered negative, and the label 'fake' positive. By reporting different performance measures, we gain insight into what causes the network performance to degrade. It will show us what is most difficult for a network to detect in unseen data: 'real' or 'fake' samples, or both. Accuracy does not provide this information.

5

Results: Deepfake Detection with Convolutional Networks

This chapter presents the results of experiments conducted towards understanding the effects of design choices when building a convolutional network capable of detecting Deepfake videos. The experiments examine the effects of dataset size and variety, preprocessing, dropout rates, batch sizes, network architecture, and input types. Lastly, the networks' robustness and generalisation abilities are analysed.

5.1. Experiment 1: Dataset Composition

The first experiment conducted examines the effects of different dataset compositions. To this end, we use Shallow-2fc on three different datasets: FaceForensics++, Larger FaceForensics++, and Largest FaceForensics++. All three datasets are based on the 1.000 real and 1.000 deepfake videos contained in FaceForensics++. We constructed our dataset such that FaceForensics++ contains only one frame of 1897 videos; Larger FaceForensics++ contains all frames of 61 videos; Largest FaceForensics++ contains up to 25 frames of 1897 videos. Larger FaceForensics++ and Largest FaceForensics++ are respectively 15.0 times and 24.4 times larger than FaceForensics++. For further dataset composition details, see section 4.3.2.

The resulting loss and accuracy curves are presented in Fig. 5.1.

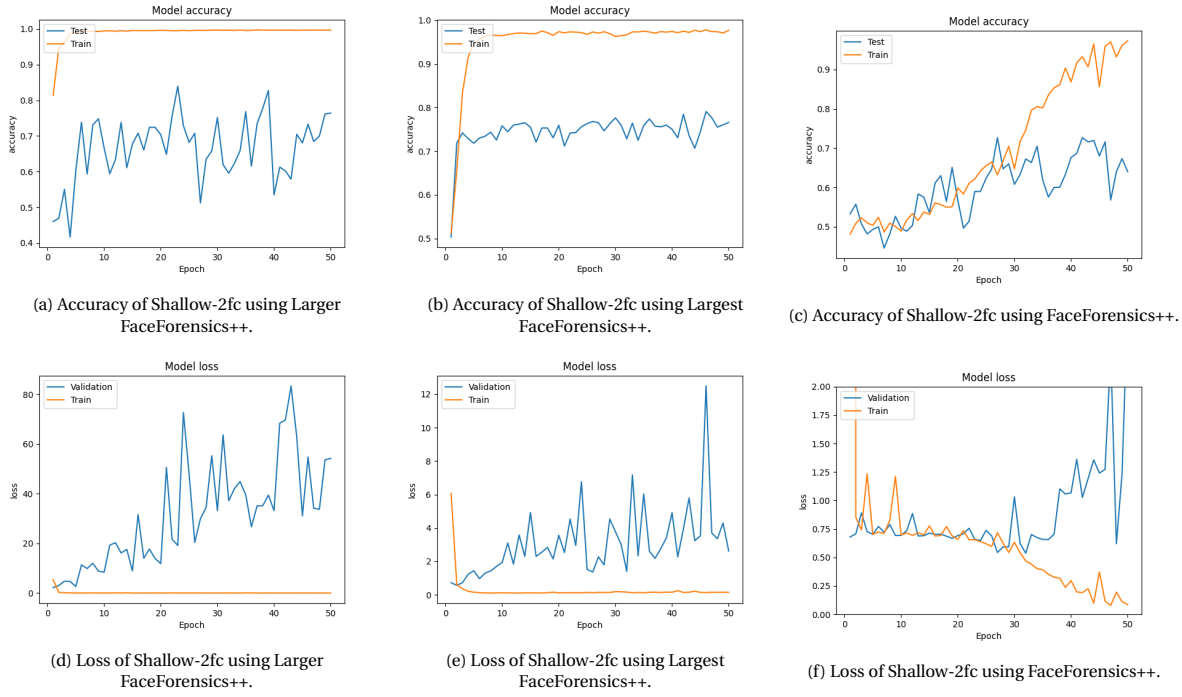
On Larger FaceForensics++ and Largest FaceForensics++, Shallow-2fc immediately overfits (see Fig. 5.1a and 5.1b respectively). However, there is a difference between the maximum losses reached on the two datasets: on Larger FaceForensics, the maximum is 83.4 (see Fig. 5.1d); on Largest FaceForensics++ the loss reaches a maximum of 12.5 (see Fig. 5.1c). Thus, on the Larger FaceForensics++ dataset, the maximum error is 6.7 times higher than on Largest FaceForensics++.

On FaceForensics++, Shallow-2fc reaches a maximum accuracy of 72.7% before overfitting sets in; Fig. 5.1f shows that its loss curve reaches a minimum of 0.54 before overfitting sets in. This means that FaceForensics++ is the only dataset on which Shallow-2fc manages to learn any distinguishing patterns. FaceForensics++ is the smallest, but also the most varied dataset. This implies that variation in a dataset can be more important than size: a network trained on a smaller, more varied dataset can outperform a network trained on a larger, less varied dataset.

Whereas literature agrees that both the amount and the variation of data are important[1], research sometimes seems to focus on size rather than variation. For example, Dordevic et al. [143] use frames from only four different videos to detect Deepfakes.

Other research does focus on variety. For example, Afchar et al. [1] imply that dataset variety is more important than size, explaining that for their dataset the amount of extracted frames "is proportional to the number of camera angle and illumination changes". Similarly, Johnston et al. [61] use a "large temporal stride [...] to limit correlation between patches from the same video sequence", since "a neural network trained with correlated dataset will be subject to overfitting". We agree that their approaches are favourable over those

[1] See, for example, Johnston and Elyan [60].



(a) Accuracy of Shallow-2fc using Larger FaceForensics++.

(b) Accuracy of Shallow-2fc using Largest FaceForensics++.

(c) Accuracy of Shallow-2fc using FaceForensics++.

(d) Loss of Shallow-2fc using Larger FaceForensics++.

(e) Loss of Shallow-2fc using Largest FaceForensics++.

(f) Loss of Shallow-2fc using FaceForensics++.

Figure 5.1: Accuracies and losses of Shallow-2fc using differently composed datasets.

focussing purely on dataset size.

The runtimes for the separate runs are reported in Table 5.1. The increase in runtime seems to be approximately linear with the increase of dataset size: on Larger FaceForensics++ the model takes approximately 13.7 times longer to train than on FaceForensics++; on Largest FaceForensics++ the model takes approximately 27.4 times longer to train than on FaceForensics++.

Table 5.1: The training times of Shallow-2fc on differently composed datasets.

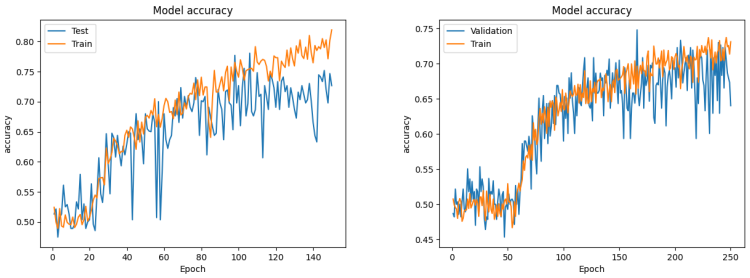
Dataset	**Epochs**	**Training time (s)**
Larger FaceForensics++	50	16223.9
Largest FaceForensics++	50	32435.1
FaceForensics++	50	1185.1

5.2. Experiment 2: Preprocessing

These experiments are conducted with Shallow-2fc on the dataset FaceForensics++. One run uses heavy preprocessing, and one run uses light preprocessing. The resulting accuracy curves are shown in Fig. 5.2. We can compare this run to the one on FaceForensics++ in section 5.1 (Fig. 5.1c and Fig. 5.1f): it uses no preprocessing, but otherwise identical settings to the runs in this experiment. The performance of Shallow-2fc with different preprocessing settings is reported in Table 5.2.

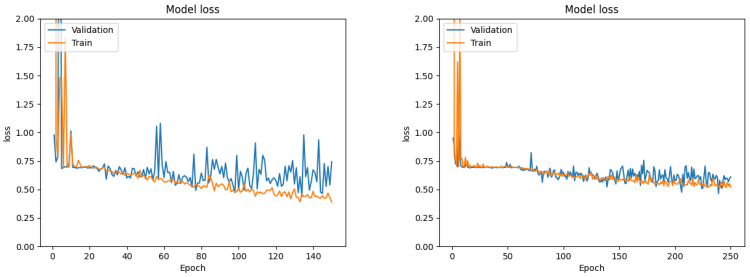
From Fig. 5.2, we notice that the heavier the preprocessing, the longer it takes the accuracy curve to start climbing. This is probably because heavier processing results in a higher variety in data, causing the network to see a considerably different subset of the data each epoch, hence taking longer to find the distinguishing global patterns.

From Table 5.2, we find that the maximum accuracy reached using heavy preprocessing is 2.1% higher than using no preprocessing, but 3.3% lower than using light preprocessing. However, heavy preprocessing's minimum loss is lower than both light preprocessing's (by 0.1) and no preprocessing's (by 0.8). Moreover, heavy preprocessing deters overfitting altogether; no preprocessing results in overfitting after epoch 29, and



(a) Accuracy of Shallow-2fc with light preprocessing.

(b) Accuracy of Shallow-2fc with heavy preprocessing.



(c) Loss of Shallow-2fc with light preprocessing.

(d) Loss of Shallow-2fc with heavy preprocessing.

Figure 5.2: Losses and accuracies of Shallow-2fc with different preprocessing settings.

light preprocessing only suffices to deter overfitting for another 81 epochs.

Table 5.2: Performance of Shallow-2fc on FaceForensics++ using different preprocessing settings. The maximum accuracy reported is the validation accuracy; the minimum loss reported is the validation loss. In case the model overfits, the reported maximum accuracy and minimum loss are those before overfitting sets in. The sixth column reports the epoch after which overfitting sets in; a hyphen indicates no overfitting takes place.

Preprocessing	**Performance**				
	Accuracy		**Loss**		**Overfitting**
	Fig.	**Max (%)**	**Fig.**	**Min**	**Epoch**
Heavy Preprocessing	5.2b	74.8	5.2d	0.46	-
Light Preprocessing	5.2a	78.1	5.2c	0.47	110
No Preprocessing	5.1c	72.7	5.1f	0.54	29

We will consider heavy preprocessing preferable to light or no preprocessing since it outperforms those on two out of three criteria; especially because 3.3% is not much given the stochastic nature of the process (see Appendix B.5).

Still, preprocessing does not seem to increase the accuracy a model can achieve on this small dataset. This is contrary to Chollet [20]'s claims, that seem to suggest any network's accuracy on a small dataset can be improved using preprocessing. It might be that the picture painted by Chollet [20] is too optimistic; it might also be that the FaceForensics++ dataset is too inconsistent for the network to find any patterns that could serve more accurate classification; for more thoughts on this, see section 5.5.5.

The time elapsed for training with different preprocessing settings are reported in Table 5.3. On average, with light preprocessing, Shallow-2fc takes 25.9 seconds per epoch; with heavy preprocessing 23.9 seconds per epoch. From Table 5.1, we gather that training Shallow-2fc without preprocessing takes an average of 23.7 seconds per epoch. This is 2.2 seconds more than with light preprocessing, and 0.2 seconds more than with heavy preprocessing.

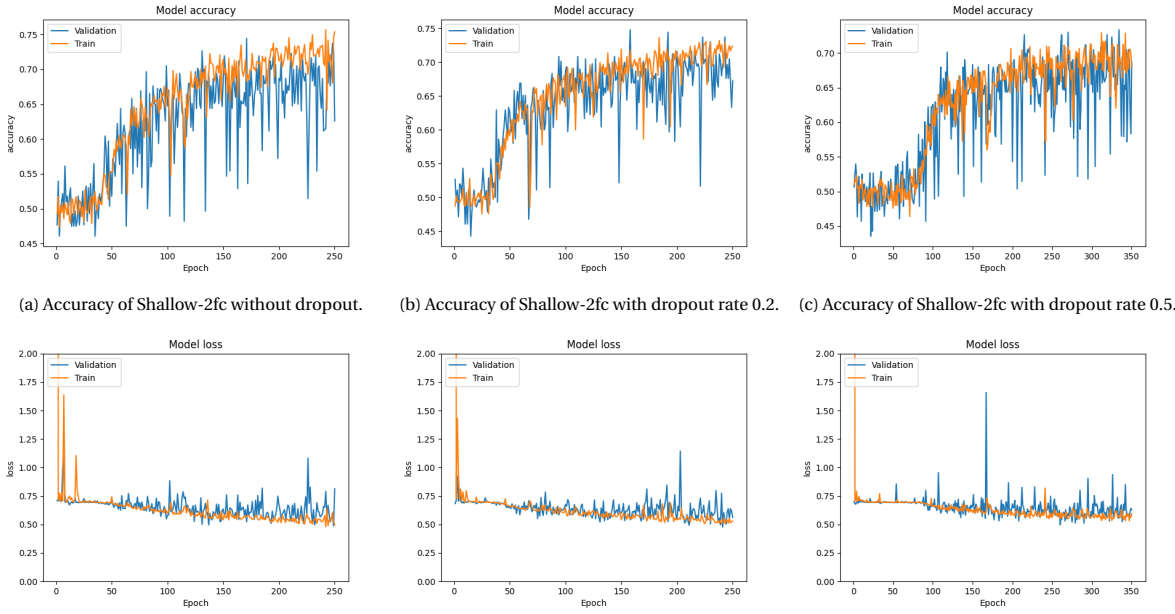
Since it would be odd for the light preprocessing to be more computationally expensive than heavy and no preprocessing, we will consider the difference in time elapsed a measurement error. Hence, since no preprocessing and heavy preprocessing differ only 0.2 seconds, we hypothesise that different preprocessing settings do not seem to have much effect on the time the model requires for training.

Table 5.3: The training times of Shallow-2fc using different preprocessing settings.

Preprocessing Settings	Epochs	Training time (s)
Light preprocessing	150	3879.3
Heavy preprocessing	250	5981.7

5.3. Experiment 3: Dropout

These experiments are conducted with Shallow-2fc on the dataset FaceForensics++. The resulting loss and accuracy curves are shown in Fig. 5.3. The performance is reported in Table 5.4. We can compare these runs to the one with heavy preprocessing in section 5.2, which uses dropout rate 0.1, but otherwise identical settings.



(a) Accuracy of Shallow-2fc without dropout.

(b) Accuracy of Shallow-2fc with dropout rate 0.2.

(c) Accuracy of Shallow-2fc with dropout rate 0.5.

(d) Loss of Shallow-2fc without dropout.

(e) Loss of Shallow-2fc with dropout rate 0.2.

(f) Loss of Shallow-2fc with dropout rate 0.5.

Figure 5.3: Losses and accuracies of Shallow-2fc using different dropout rates.

From Fig. 5.3, we learn that without dropout, the accuracy curve starts climbing around epoch 40; with dropout rate 0.2, the accuracy curve starts climbing around epoch 30; with dropout rate 0.5, the accuracy curve starts climbing around epoch 80. From Fig. 5.2b, we learn that with dropout rate 0.1, the accuracy curve starts climbing around epoch 60. This implies that for large dropout rates, the model takes longer to start recognising patterns than for small dropout rates. This is probably because with high dropout rates, each epoch uses a notably different set of nodes than the previous epoch: each node is switched off with high probability. This causes the network to see a considerably different subset of the data each epoch, hence taking longer to find the distinguishing global patterns.

Fig. 5.3 also shows that dropout seems to keep the training and validation accuracies closer together: there seems to be a slightly larger gap between them when no dropout is used than with dropout rates 0.1 or 0.2. When dropout rate 0.5 is used, the two curves do not seem to separate at all.

From Table 5.4, it seems dropout does not have a large effect on network performance in terms of accuracy and loss: the maximum accuracies achieved with all four dropout rates are between 73.4% and 74.8%; the minimum losses are between 0.46 and 0.50. These differences are negligibly small. Moreover, none of the models overfit.

The small differences in accuracy and loss imply that in this particular case, dropout cannot be used to reach higher accuracy. However, this might be different for a network that is more prone to overfitting in the first place. Moreover, as section 5.2 states, the FaceForensics++ dataset might be too inconsistent for the network to find any patterns that could serve more accurate classification; dropout is not a miraculous means to fix overfitting if the problem is in the dataset (composition).

Table 5.4: Performance of Shallow-2fc on FaceForensics++ using different dropout rates. The maximum accuracy reported is the validation accuracy; the minimum loss reported is the validation loss.

Dropout Rate	Performance			
	Accuracy		Loss	
	Fig.	Max (%)	Fig.	Min
0	5.3a	74.5	5.3d	0.49
0.1	5.2b	74.8	5.2d	0.46
0.2	5.3b	74.8	5.3e	0.48
0.5	5.3c	73.4	5.3f	0.50

Table 5.5: The training times of Shallow-2fc using different dropout rates.

Dropout Rates	Epochs	Training time (s)
0	250	6388.1
0.2	250	6426.3
0.5	350	9006.7

The time elapsed for training with different dropout rates is reported in Table 5.5. On average, training without dropout elapsed 25.6 seconds per epoch, training with dropout rate 0.2 elapsed 25.7 seconds per epoch, and training with dropout rate 0.5 elapsed 25.7 seconds per epoch. Training with dropout rate 0.1 elapsed 23.9 seconds per epoch. Compared to training without dropout, training with dropout rate 0.1 takes 1.7 seconds shorter per epoch, training with dropout rate 0.2 or 0.5 takes 0.1 seconds longer per epoch. We consider these discrepancies to be negligibly small.

For the remainder of this research, we will use dropout rate 0.1, to reduce the gap between the training and validation curves. Since the difference between dropout rates 0.1 and 0.2 seems small, we choose dropout rate 0.1: it is the dropout rate used in previous experiments and allows us to compare upcoming experiments to previous ones.

5.4. Experiment 4: Batch Size

These experiments are conducted on the dataset FaceForensics++. Different batch sizes are used. The resulting loss and accuracy curves are shown in Fig. 5.4. The performance is reported in Table 5.6. We can compare these results to the run with heavy preprocessing in section 5.2, which uses batch size 100, but otherwise identical settings.

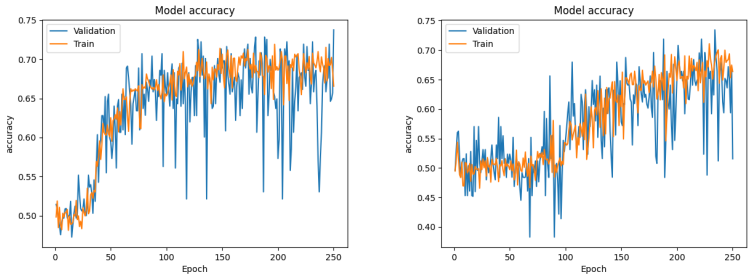
Fig. 5.4a shows that with batch size 50, the accuracy curve has its steepest slope between epochs 20-59; Fig. 5.4b shows that batch size 250, the accuracy curve climbs gradually without any significantly steep slopes. Fig. 5.2b shows that with batch size 100, the accuracy curve has its steepest slope between epochs 60-70.

We see that the larger the batch size, the longer it takes for the accuracy curves to start climbing. This means that for larger batch sizes, it takes the algorithm longer to learn to distinguish between two classes. This is probably because the gradient descent algorithm updates the parameters after it has seen a whole batch of images. Seeing 250 images instead of 50 might make it more difficult to find initial patterns to work with.

Table 5.6: Performance of Shallow-2fc on FaceForensics++ using different batch sizes. The maximum accuracy reported is the validation accuracy; the minimum loss reported is the validation loss.

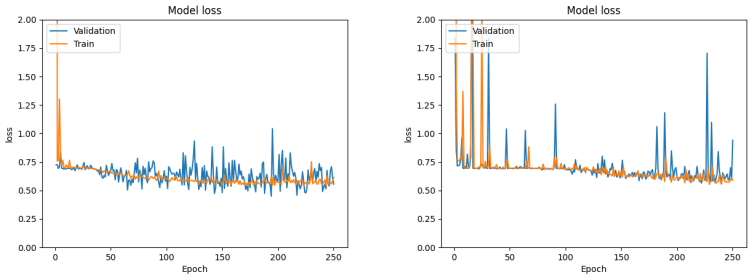
Batch Size	Performance			
	Accuracy		Loss	
	Fig.	Max (%)	Fig.	Min
50	5.4a	73.8	5.4c	0.45
100	5.2b	74.8	5.2d	0.46
250	5.4b	73.4	5.4d	0.57

From Table 5.6, we learn that the differences in achieved accuracies are small. The difference in loss is



(a) Accuracy of Shallow-2fc with batch size 50.

(b) Accuracy of Shallow-2fc with batch size 250.



(c) Loss of Shallow-2fc with batch size 50.

(d) Loss of Shallow-2fc with batch size 250.

Figure 5.4: Losses and accuracies of Shallow-2fc using different batch sizes.

considerably larger: batch size 250 results in higher loss than batch sizes 50 and 100. This might be related to optimisation being more difficult when seeing many samples before updating parameters.

The time elapsed for training with different batch sizes is reported in Table 5.7. The time elapsed during training was, on average, 26.2 seconds with batch size 50, and 22.8 seconds for batch size 250. For batch size 100, it was 23.9 seconds per epoch. It seems time required for training decreases with batch size. This is likely due to backpropagation being applied less frequently. The differences being relatively small might imply that although backpropagation does affect the time elapsed for training, it is not the main component.

Table 5.7: The training times of Shallow-2fc using different batch sizes.

Batch Size	Epochs	Training time (s)
50	250	6547.8
250	250	5691.0

For the remainder of this research, we will use batch size 100. This allows us to compare the results of upcoming experiments with those of previous experiments, without influencing the accuracies. Although one could argue batch size 200 preferable due to the reduced training time per epoch, we argue this reduction is nullified by the additional epochs needed for the algorithm to start learning.

5.5. Experiment 5: Different Architectures and Different Deepfake Qualities

In this section, fourteen different models (Shallow-2fc, VGG19-2fc, Inception-2fc, ResNet-2fc, VGG16-2fc, Xception-2fc, DenseNet-2fc, Shallow-1fc, VGG19-1fc, Inception-1fc, ResNet-1fc, VGG16-1fc, Xception-1fc, DenseNet-1fc) are tested on three different datasets (FaceForensics++, Celeb-df, and DeeperForensics-1.0).

The performance in terms of loss, accuracy, and overfitting is summarised in Table 5.8. The relevant accuracies for the 2fc-models are shown in Fig. 5.5; the corresponding losses are shown in Fig. 5.6. The relevant accuracies for the 1fc-models are shown in Fig. 5.7; the corresponding losses are shown in Fig. 5.8. Except for DenseNet-2fc on FaceForensics++, DenseNet-2fc is not featured in this table or these figures; nor are DenseNet-1fc, Inception-2fc, Inception-1fc, Xception-2fc, Xception-1fc. The reason for this is that all these networks immediately overfit on all three datasets. This means they do not learn any patterns useful for

distinguishing between real and fake samples; they merely learn to memorise the training data. Hence, these networks are not suited for distinguishing between the 'real' and 'fake' images in the used datasets, with the used settings.

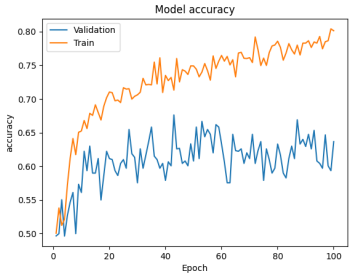
Table 5.8: Performance of different models on different datasets. The maximum accuracy reported is the validation accuracy; the minimum loss reported is the validation loss. In case the model overfits, the reported maximum accuracy and minimum loss are those before overfitting sets in. The seventh column reports the epoch after which overfitting sets in; a hyphen indicates no overfitting takes place. The highest accuracy and lowest loss per dataset are presented in bold.

Model	**Dataset**	**Performance**				
		Accuracy		**Loss**		**Overfitting**
		Fig.	**Max (%)**	**Fig.**	**Min**	**Epoch**
ResNet-2fc	FaceForensics++	5.5a	67.6	5.6a	0.62	-
ResNet-2fc	Celeb-df	5.5b	68.7	5.6b	0.69	4
ResNet-2fc	DeeperForensics-1.0	5.5c	85.8	5.6c	0.30	-
Shallow-2fc	FaceForensics++	5.5d	**76.3**	5.6d	**0.45**	-
Shallow-2fc	Celeb-df	5.5e	**68.9**	5.6e	0.55	63
Shallow-2fc	DeeperForensics-1.0	5.5f	82.6	5.6f	0.34	-
VGG16-2fc	FaceForensics++	5.5g	70.5	5.6g	0.55	-
VGG16-2fc	Celeb-df	5.5h	66.4	5.6h	**0.48**	4
VGG16-2fc	DeeperForensics-1.0	5.5i	90.0	5.6i	**0.14**	-
VGG19-2fc	FaceForensics++	5.5j	68.3	5.6j	0.59	-
VGG19-2fc	Celeb-df	5.5k	67.2	5.6k	0.72	3
VGG19-2fc	DeeperForensics-1.0	5.5l	89.3	5.6l	0.21	-
DenseNet-2fc	FaceForensics++	5.5m	55.4	5.6m	0.69	-
ResNet-1fc	FaceForensics++	5.7a	65.5	5.8a	0.59	-
ResNet-1fc	Celeb-df	5.7b	71.4	5.8b	0.53	2
ResNet-1fc	DeeperForensics-1.0	5.7c	81.5	5.8c	0.36	-
Shallow-1fc	FaceForensics++	5.7d	71.9	5.8d	0.52	-
Shallow-1fc	Celeb-df	5.7e	63.9	5.8e	0.62	29
Shallow-1fc	DeeperForensics-1.0	5.7f	**92.5**	5.8f	0.19	-
VGG16-1fc	FaceForensics++	5.7g	66.5	5.8g	0.59	-
VGG16-1fc	Celeb-df	5.7h	52.1	5.8h	0.50	3
VGG16-1fc	DeeperForensics-1.0	5.7i	86.5	5.8i	0.25	-
VGG19-1fc	FaceForensics++	5.7j	65.0	5.8j	0.56	-
VGG19-1fc	Celeb-df	5.7k	57.6	5.8k	0.91	2
VGG19-1fc	DeeperForensics-1.0	5.7l	86.1	5.8l	0.25	-

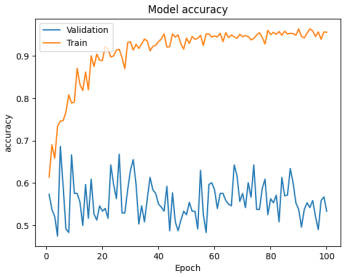
This section explores numerous aspects of network architecture, datasets and their effects at once. To keep things organised, we will divide this section into subsections dedicated to input size, performance on datasets, performance of convolutional bodies, and performance of classification networks. There is also a section with a disclaimer about unstable training of Shallow-1fc that is not included in the other sections, but is relevant for understanding how to interpret the other results.

5.5.1. Input Size

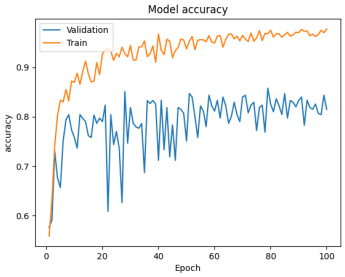
We can compare Shallow-2fc on FaceForensics++ from this section to Shallow-2fc on FaceForensics++ with heavy preprocessing in section 5.2; the only difference between these two runs is the input size. With input size 200×200, the maximum achieved accuracy is 1.5% higher than with input size 128×128; the maximum achieved loss is 0.01 lower with input size 200×200. Hence, the difference in accuracy and loss between different input sizes is negligible when using Shallow-2fc. We can conclude that Shallow-2fc's capacity is equally suited for the problem with input sizes 128×128 and 200×200. This could mean that this difference in image size has little impact on the complexity of the task, meaning both image sizes are much smaller than desired; it is also possible that the optimal input size for Shallow-2fc is somewhere in between 128×128 and 200×200 (e.g. plotting performance against image size could be a parabola, with its maximum exactly in the middle of the two input sizes we tested on). However, since literature obtains better results using a larger image size, we favour the first hypothesis.



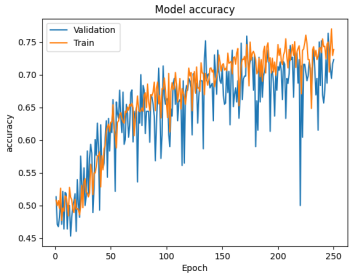
(a) Accuracy of ResNet-2fc on FaceForensics++.



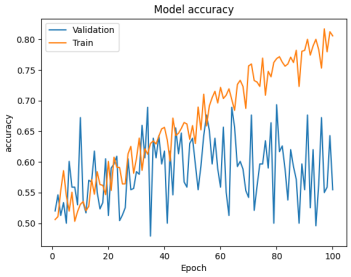
(b) Accuracy of ResNet-2fc on Celeb-df.



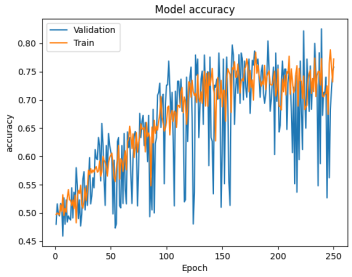
(c) Accuracy of ResNet-2fc on DeeperForensics-1.0.



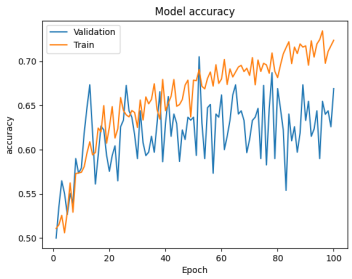
(d) Accuracy of Shallow-2fc on FaceForensics++.



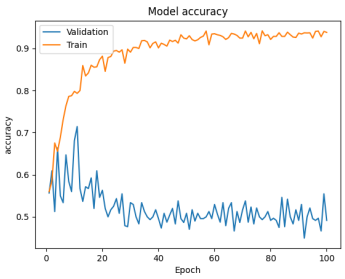
(e) Accuracy of Shallow-2fc on Celeb-df.



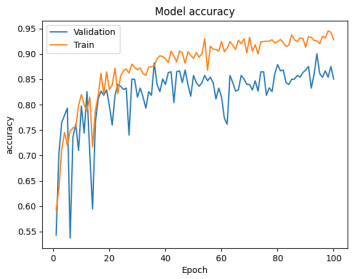
(f) Accuracy of Shallow-2fc on DeeperForensics-1.0.



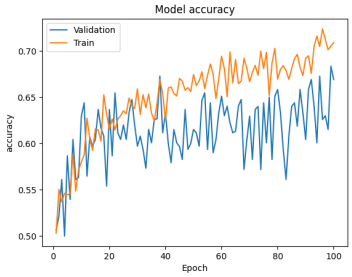
(g) Accuracy of VGG16-2fc on FaceForensics++.



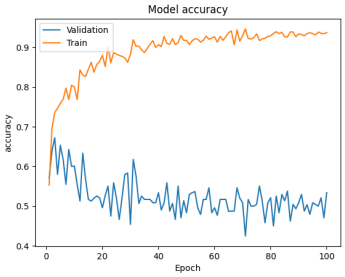
(h) Accuracy of VGG16-2fc on Celeb-df.



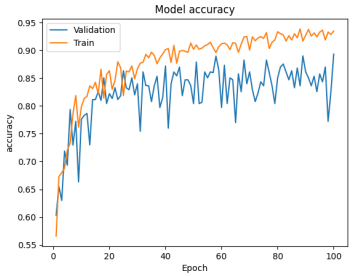
(i) Accuracy of VGG16-2fc on DeeperForensics-1.0.



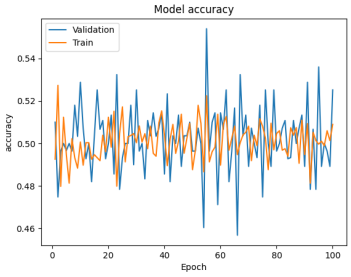
(j) Accuracy of VGG19-2fc on FaceForensics++.



(k) Accuracy of VGG19-2fc on Celeb-df.

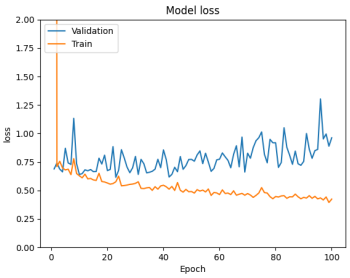


(l) Accuracy of VGG19-2fc on DeeperForensics-1.0.

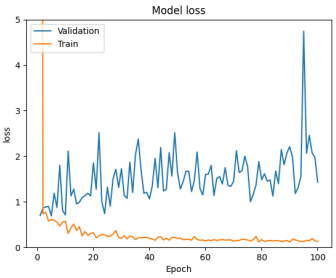


(m) Accuracy of DenseNet-2fc on FaceForensics++.

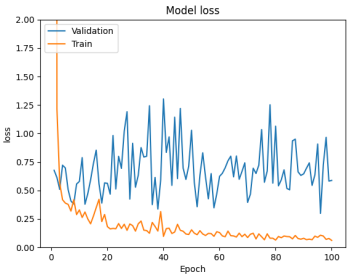
Figure 5.5: Accuracy curves of 2fc-models trained on different datasets.



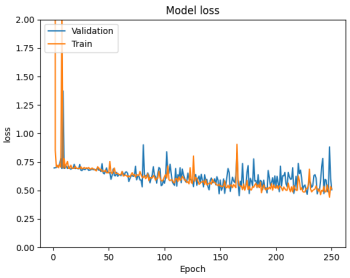
(a) Loss of ResNet-2fc on FaceForensics++.



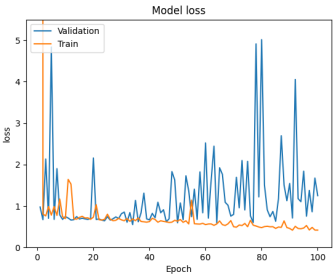
(b) Loss of ResNet-2fc on Celeb-df.



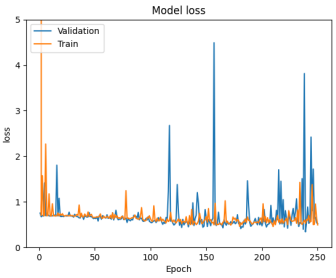
(c) Loss of ResNet-2fc on DeeperForensics-1.0.



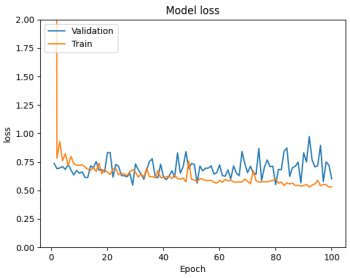
(d) Loss of Shallow-2fc on FaceForensics++.



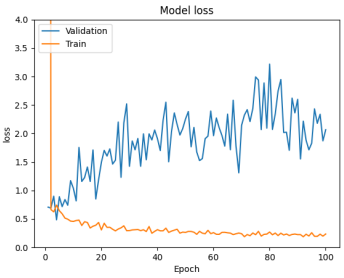
(e) Loss of Shallow-2fc on Celeb-df.



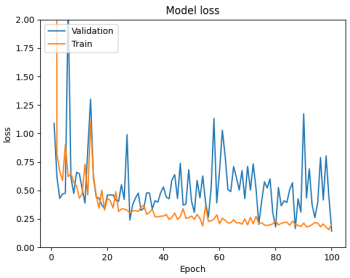
(f) Loss of Shallow-2fc on DeeperForensics-1.0.



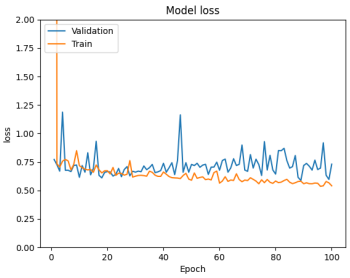
(g) Loss of VGG16-2fc on FaceForensics++.



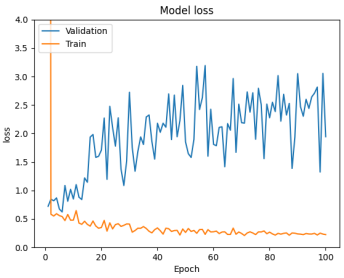
(h) Loss of VGG16-2fc on Celeb-df.



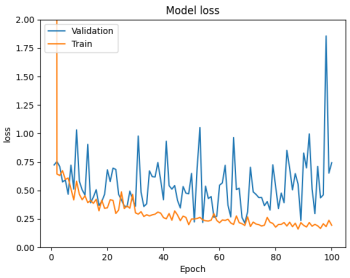
(i) Loss of VGG16-2fc on DeeperForensics-1.0.



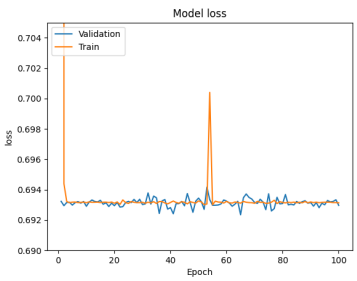
(j) Loss of VGG19-2fc on FaceForensics++.



(k) Loss of VGG19-2fc on Celeb-df.

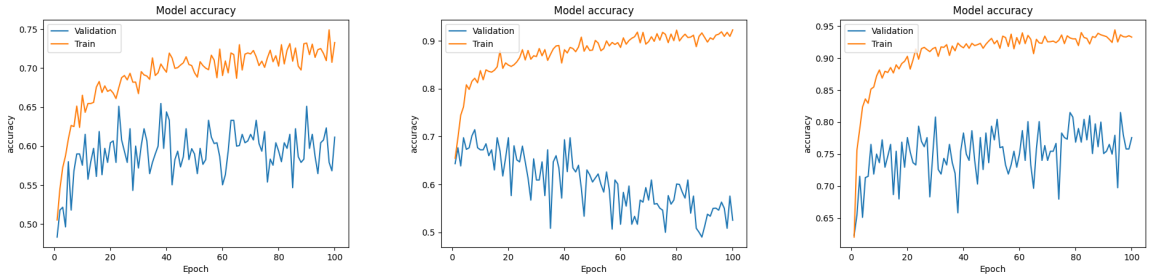


(l) Loss of VGG19-2fc on DeeperForensics-1.0.



(m) Loss of DenseNet-2fc on FaceForensics++.

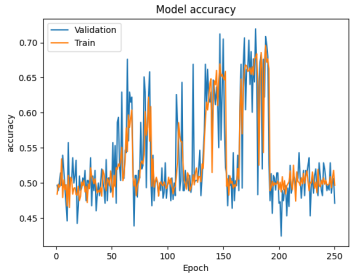
Figure 5.6: Relevant losses of convolutional networks on the different datasets.



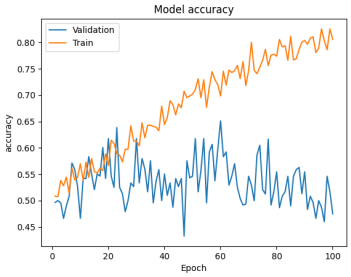
(a) Accuracy of ResNet-1fc on FaceForensics++.

(b) Accuracy of ResNet-1fc on Celeb-df.

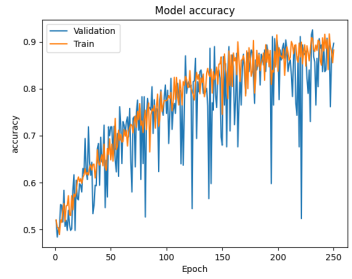
(c) Accuracy of ResNet-1fc on DeeperForensics-1.0.



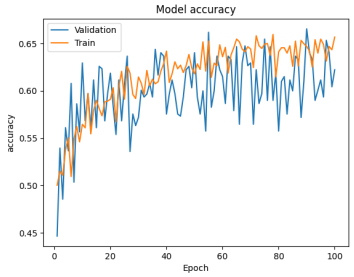
(d) Accuracy of Shallow-1fc on FaceForensics++.



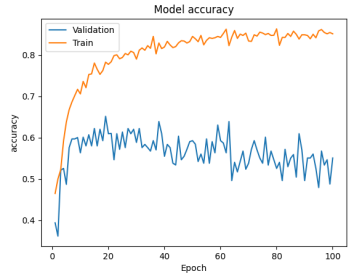
(e) Accuracy of Shallow-1fc on Celeb-df.



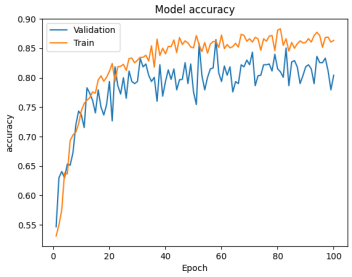
(f) Accuracy of Shallow-1fc on DeeperForensics-1.0.



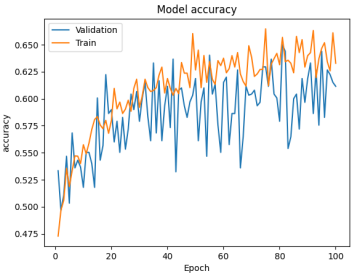
(g) Accuracy of VGG16-1fc on FaceForensics++.



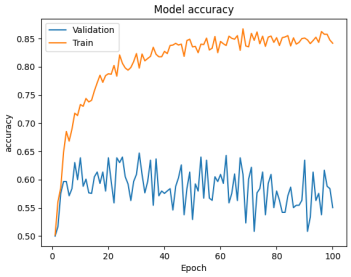
(h) Accuracy of VGG16-1fc on Celeb-df.



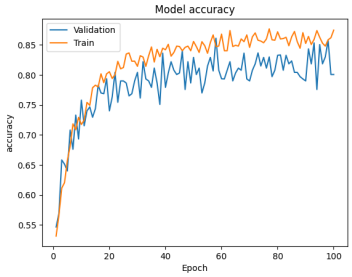
(i) Accuracy of VGG16-1fc on DeeperForensics-1.0.



(j) Accuracy of VGG19-1fc on FaceForensics++.



(k) Accuracy of VGG19-1fc on Celeb-df.



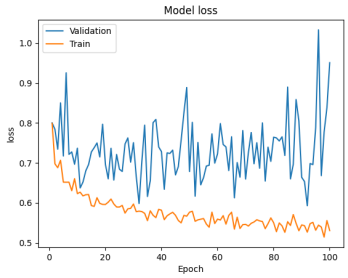
(l) Accuracy of VGG19-1fc on DeeperForensics-1.0.

Figure 5.7: Accuracy of Shallow-1fc, VGG16-1fc, and VGG19-1fc on the different datasets.

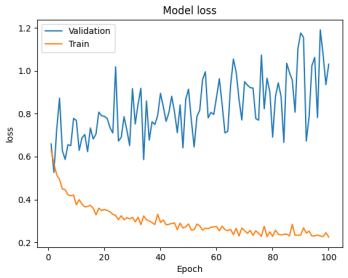
It is possible that the overfitting networks would perform better with a larger input size. For example, Raghavendra et al. [101] also use VGG19 for detection, although with input size 227×227. Their training set contains 783 training samples. They do not use accuracy for a performance measure, but their network reaches an EER-score[2] of 12.47%, meaning their VGG19 also manages to learn from a relatively small dataset. Zhou et al. [140] and Zhang et al. [137] both use Inception-v3 pretrained on ImageNet; Zhou et al. [140] manage to train it to an AUROC[3] of 0.854 with a training set containing 2105 samples; Zhang et al. [137] manage to train it to a validation accuracy of 98% using a training set with 1000 samples. Both networks use image resolutions of 299×299; it might be that Inception-v3 overfits in our experiments because the image size is too small. Zhou et al. [139] manage to use ResNet-101 (so a smaller ResNet than ours) for

[2] See appendix B.1.
[3] See appendix B.1.

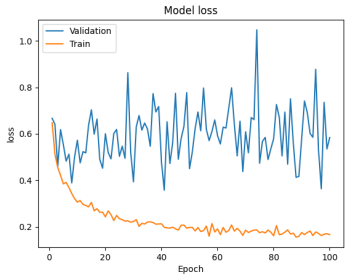
Deepfake detection to an AUROC score of 0.857; however, with 42 thousand samples, their training dataset is significantly larger than ours. However, due to time restrictions, we are not pursuing this hypothesis any further.



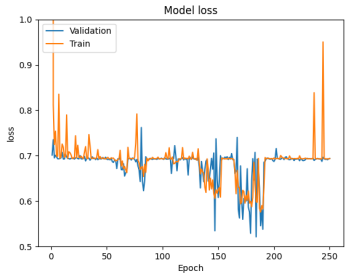
(a) Loss of ResNet-1fc on FaceForensics++.



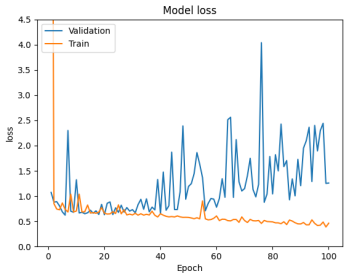
(b) Loss of ResNet-1fc on Celeb-df.



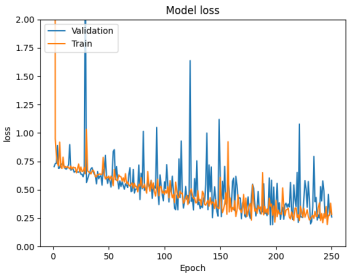
(c) Loss of ResNet-1fc on DeeperForensics-1.0.



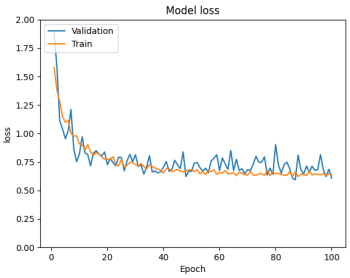
(d) Loss of Shallow-1fc on FaceForensics++.



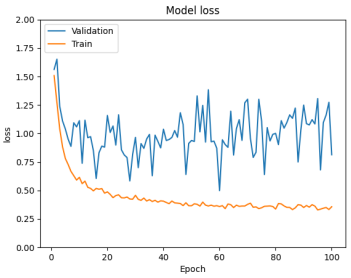
(e) Loss of Shallow-1fc on Celeb-df.



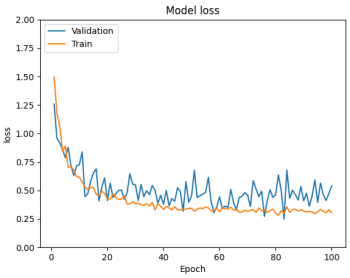
(f) Loss of Shallow-1fc on DeeperForensics-1.0.



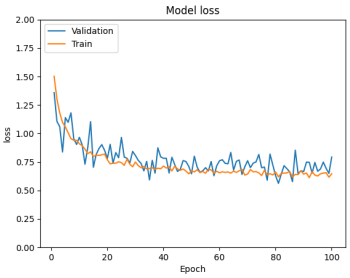
(g) Loss of VGG16-1fc on FaceForensics++.



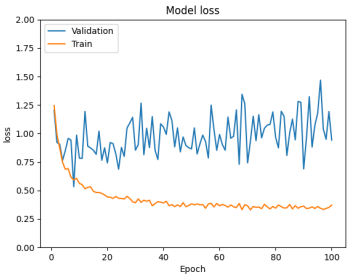
(h) Loss of VGG16-1fc on Celeb-df.



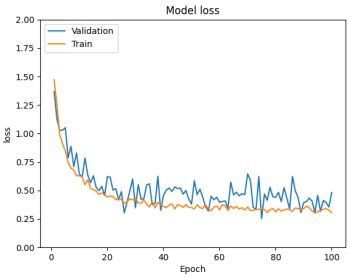
(i) Loss of VGG16-1fc on DeeperForensics-1.0.



(j) Loss of VGG19-1fc on FaceForensics++.



(k) Loss of VGG19-1fc on Celeb-df.



(l) Loss of VGG19-1fc on DeeperForensics-1.0.

Figure 5.8: Relevant losses of convolutional networks on the different datasets.

5.5.2. Performance on Different Datasets

All models reach their highest accuracies and lowest losses on DeeperForensics-1.0. ResNet-1fc and ResNet-2fc both reach their lowest accuracies on FaceForensics++; all other models reach their lowest accuracies on Celeb-df. VGG16-2fc and VGG16-1fc reach their highest losses on FaceForensics++; all other models reach their highest losses on Celeb-df.

The highest accuracy on FaceForensics++ is 76.3%, achieved by Shallow-2fc. The lowest loss on FaceForensics++ is 0.45, also achieved by Shallow-2fc. The highest accuracy on Celeb-df is 68.9%, achieved by Shallow-2fc. The lowest loss on Celeb-df is 0.48, achieved by VGG16-2fc. The highest accuracy on Deeper-

Forensics-1.0 is 92.5%, achieved by Shallow-1fc. The lowest loss on DeeperForensics-1.0 is 0.14, achieved by VGG16-2fc. All models overfit on Celeb-df. Except for Shallow-2fc and Shallow-1fc, all overfitting sets in within five epochs. DeeperForensics-1.0 is the only dataset on which 80% accuracy is exceeded: by ResNet-2fc, Shallow-2fc, VGG16-2fc, VGG19-2fc, ResNet-1fc, Shallow-1fc, VGG16-1fc, and VGG19-1fc.

The highest accuracy achieved on DeeperForensics-1.0 exceeds the highest accuracy achieved on FaceForensics++ by 16.2%; the minimum loss on DeeperForensics++ is 0.31 lower than the minimum loss on FaceForensics++. These large differences might be due to the variety of artefacts in FaceForensics++. When studying Fig. B.4, different types of artefacts show: some faces have not merged properly with the heads they have been pasted into; some faces show two pairs of eyebrows; some faces are exceptionally blurry; some faces look flat rather than three dimensional; one face is not a face at all but a mesh of colours. For a convolutional network, these artefacts would all fall in different categories. Hence, the network might not be able to find telltale signs of Deepfakes that appear in all images in the dataset. DeeperForensics-1.0 shows a lower variety of artefacts: its images do not suffer double eyebrows, faces are merged more naturally into the scenes, faces are less blurry. Hence, the artefacts in the frames from DeeperForensics-1.0 might be closer related and therefore easier to use for solving this binary classification problem.

The differences in performance between FaceForensics++ and Celeb-df are smaller: the difference in maximum achieved accuracy is 8.4%, the difference in minimum loss 0.03. A small survey among people who have never worked with Deepfake videos showed that between Fig. B.4, Fig. B.7, Fig. B.10, the 10 test subject unanimously considered Celeb-df the most realistic dataset. It seems natural that the algorithm has more difficulty in distinguishing between real and fake samples when the fake samples are more sophisticated. However, it is hopeful that we have so far reached almost 70% accuracy. Further research will surely be able to exceed this accuracy.

5.5.3. Performance of Convolutional Bodies

Three of the seven convolutional bodies (DenseNet, Inception, Xception) immediately overfit on all three datasets (except DenseNet-2fc on FaceForensics++, which underfits). This implies that the capacity of DenseNet-2fc is too small for FaceForensics++, and the other networks' capacities are too high for classifying the relatively small datasets we used.

The other four convolutional bodies do manage to learn global patterns in the training data. Out of those four convolutional bodies, three were 'simple' networks without additional connections: shallow, VGG16, and VGG19. Between those three, in all experiments presented, the best performance was achieved either by one of the VGG16 models or by one of the shallow models. The VGG19 models' performance stays close to the VGG16's, albeit slightly worse. This suggests the optimal capacity for the task at hand may be somewhere between the shallow networks' and VGG16's capacity. We experimented with networks with 5 or 6 convolutional layers, but did not include these results in this research for the differences were only slight. However, it is entirely possible that the optimal capacity for the classification task at hand can be achieved by a network containing between 4 and 12 convolutional layers (which are the amounts of convolutional layers the shallow models and VGG16 have respectively). It is also possible that the shallow networks perform so well because they were the only ones whose convolutional bodies were trained on a Deepfake dataset, rather than only on ImageNet. It is also possible that the optimal dataset composition, preprocessing settings, dropout, and batch size were only optimal for Shallow-2fc; it might be that carrying out those experiments with the other networks indicates different optima for each network.

Our experiments seem to indicate that capacity does not merely depend on the amount of parameters a network has: VGG16-1fc (15 million parameters) and VGG19-1fc (20 million parameters) have more parameters than Dense-1fc (7.0 million parameters), yet their capacity seems closer to Shallow-1fc's (241 thousand parameters) and Shallow-2fc's (6.8 million parameters). Inception-v3 (22 million parameters) and Xception-1fc (21 million parameters) are approximately the same size as VGG19-1fc; DenseNet-2fc (26 million parameters) is smaller than VGG19-2fc (29 million parameters). Of the models mentioned above, only the shallow and VGG models learn any useful features from any of the datasets. This indicates that the overfitting cannot be blamed solely on the amount of parameters the networks have.

Since the shallow network, VGG16, and VGG19 do not include inception modules, it could be argued that these architectural tricks increase the other models' capacities such that they overfit on the small datasets. ResNet-1fc (109 million parameters) and ResNet-2fc (58 million parameters) have more parameters and depth than all other models, but they do not overfit on our data. This implies that residual connections do not influence model capacity (or at least, not as much as inception modules do).

However, there are some issues with this hypothesis. The first is that He et al. [46] did not design residual

connections to deter overfitting, whereas Szegedy et al. [119] did design inception modules for that purpose. Similarly, Huang et al. [53] designed dense connections to deter overfitting. Still, we could argue that their research focusses on larger datasets. We can support this argument with proof from the literature. For example, Tariq et al. [123] manage to train Xception to 79.32% on 64×64 images; but their dataset contains 160 thousand training samples[4]. What is more, with such a large dataset, the capacity favour seems to tip towards Xception rather than VGG19, which only reaches 56.69% accuracy on the same dataset. It is important to note that although the scales seem to have tipped, a larger input image size still increases VGG19's performance. The examples provided in section 5.5.1 also support our hypothesis, because larger input size also increases the required network capacity: the larger the image, the more capacity is required to capture all relevant patterns it contains.

We also need to point out that on FaceForensics++, DenseNet-2fc's accuracy oscillates between 55.4% and 45.5%; and that the corresponding loss oscillates between 0.692 and 0.694. This means DenseNet-2fc does not learn anything on FaceForensics++, implying its capacity is too low. This is odd because all other DenseNet models used immediately overfit on all datasets.

5.5.4. Performance of Classification Networks

This section explores the differences between the two types of classification network used on top of the convolutional bodies: the 1fc-models and the 2fc-models.

ResNet-2fc outperforms ResNet-1fc with 4.3% accuracy and 0.06 loss on FaceForensics++; and with 2.1% accuracy on DeeperForensics-1.0, although ResNet-1fc's loss is 0.03 lower on DeeperForensics-1.0. ResNet-1fc outperforms ResNet-2fc on Celeb-df with 4.4% and 0.16 loss. Except for the loss on Celeb-df, all these differences are small; and since on Celeb-df, ResNet-1fc overfits after two epochs, it is possible the low loss achieved is a coincidence. To pursue this hypothesis, more research would have to be carried out; we will leave this for future research. For now, we cannot conclude what classification network fits better on top of ResNet-152.

Shallow-2fc outperforms Shallow-1fc with 4.4% accuracy and 0.07 loss on FaceForensics++, and with 5.0% accuracy and 0.07 loss on Celeb-df. Shallow-1fc outperforms Shallow-1fc with 9.9% accuracy and 0.15 loss on DeeperForensics-1.0. The differences on DeeperForensics-1.0 are considerable, so on that dataset, it seems Shallow-1fc is better suited. The differences on FaceForensics++ are small in terms of accuracy and loss, but Shallow-1fc seems unstable on FaceForensics++ (we will go deeper into this in section 5.5.5). Fig. 5.7d shows that its accuracy oscillates between 42.4% and 71.9%; Fig. 5.8d shows the corresponding loss oscillates around 0.69, with a maximum of 0.76 at and a minimum of 0.52. Hence, on FaceForensics++, Shallow-2fc seems better suited. On Celeb-df, Shallow-2fc starts overfitting considerably later. These results suggest that the preferable combination of convolutional body and classification network might differ per dataset (and hence per task at hand).

VGG16-2fc outperforms VGG16-1fc by 4.0% accuracy and 0.04 loss on FaceForensics++; with 14.3% accuracy and 0.02 loss on Celeb-df; and with 3.5% accuracy and 0.11 loss on DeeperForensics-1.0. Although some of these differences are small, VGG16-2fc seems to be a better combination of convolutional body and classification network than VGG16-2fc.

VGG19-2fc outperforms VGG19-1fc with 9.6% accuracy and 0.19 loss on Celeb-df; VGG19-2fc outperforms VGG19-1fc with 3.2% accuracy and 0.04 loss on DeeperForensics-1.0. VGG19-2fc outperforms VGG19-1fc with 3.3% accuracy on FaceForensics++, but VGG19-1fc outperforms VGG-2fc with 0.03 loss. Although some of these differences are small, VGG19-2fc seems to be a better combination of convolutional body and classification network than VGG19-2fc.

We also notice that ResNet-2fc on DeeperForensics-1.0 keeps the training and validation accuracy curves closer together than ResNet-1fc; on FaceForensics++, it is the other way around. For the VGG-models on FaceForensics++ and DeeperForensics-1.0, it also holds that the 1fc-models keep the training and validation accuracy curves closer together than the 2fc-models. This speaks in favour of the 1fc-models. However, evidence is still not conclusive enough to appoint a preferable classification network for any convolutional body.

5.5.5. Disclaimer: Unstable Training of Shallow-1fc on FaceForensics++

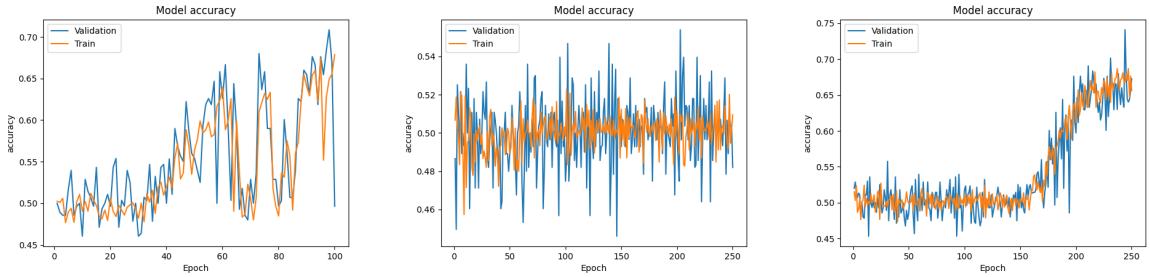
Shallow-1fc is not stable when training on FaceForensics++. We ran the model four times: once for 100 epochs, and three times for 250 epochs. The results not shown in section 5.5 are shown in Fig. 5.9. Fig.

[4] This dataset consists of generated images, not of video frames; hence their access to a varied dataset this size.

5.9a shows the same unstable behaviour as Fig. 5.7d; Fig. 5.9b shows the model underfitting; Fig. 5.9c shows the model starts learning only around epoch 160 and has not finished learning at epoch 250. However, we decided that it was not desirable to keep trying to produce a stable accuracy curve due to several reasons:

- all other runs were run at most twice; and that only when the training had not finished at epoch 100, and a run of 250 epochs was necessary to make sure the model finished training, or when it underfitted during the first run, making it an unfair comparison to run this particular run more often;
- since literature agrees networks cannot be frozen forever, but need to be retrained regularly to keep up with new types of manipulations, it is not desirable to work with an unstable network.

We chose to include Fig. 5.7d in section 5.5 instead of the figures shown in Fig. 5.9 because it best reflects the instability of this particular training process.



(a) Run 1. (b) Run 2. (c) Run 3.

Figure 5.9: Unstable training results of Shallow-1fc on FaceForensics++.

5.5.6. Time Elapsed

Table 5.9 shows large discrepancies between the training times of the different versions of some of the networks. There is no large difference in time elapsed for training between DenseNet-1fc and DenseNet-2fc, between Xception-1fc and Xception-2fc or between Shallow-1fc and Shallow-2fc. On FaceForensics++, VGG16-1fc and VGG16-2fc require the same time for training; on Celeb-df and DeeperForensics-1.0, Shallow-1fc takes almost twice as long to train. ResNet-1fc takes 1,5-2× longer to train than ResNet-2fc on all three datasets; Inception-1fc takes approximately twice as long to train on all datasets than Inception-2fc; VGG19-1fc takes approximately 2,5× longer to train on all three datasets than VGG19-2fc. This means that the models with less (learnable) parameters take at least as long to train as the corresponding models with more (learnable) parameters.

The differences in times elapsed per epoch between the two versions of the different models are counterintuitive: one would expect that a network with less (learnable) parameters would train faster, rather than slower, since it would have fewer parameters to update. For example, ResNet-1fc has 0.00004% of ResNet-2fc's learnable parameters, yet its training lasts up to two times longer per epoch.

Table 5.9: The time elapsed for training different models on different datasets. The used abbreviations are: FF for FaceForensics++, C-DF for Celeb-df, and DF for DeeperForensics-1.0. All run times represent a run of 100 epochs, except Shallow-1fc and Shallow-2fc on FaceForensics++ and DeeperForensics-1.0 (which ran for 250 epochs).

Model	**Training Time (s)**					
	Two fc-layers			**One fc-layer**		
	FF	**C-DF**	**DF**	**FF**	**C-DF**	**DF**
DenseNet-121	33,540.7	29,511.8	33,984.7	32,790.2	29,192.6	32,632.3
Inception-v3	7,124.0	6,015.0	6,726.8	15,529.0	13,829.7	15,675.5
ResNet-152	48,140.2	39,487.9	29,561.6	67,250.0	59,382.4	67,186.9
Shallow	13,851.0	4,871.9	14,860.8	14,796.9	5.322,5	15,248.9
VGG16	20,951.0	18,503.8	26,365.1	21,054.7	33,729.8	49,719.5
VGG19	24,577.1	21,758.9	25,317.7	60,376.8	53,299.8	60,583.6
Xception	43,091.4	38,605.5	43,445.6	42,086.6	37,240.8	41,848.2

All runs were executed on an AWS `t3.xlarge` instance. Since the runs elapsed several hours, it seems unlikely that the AWS instance ran so many/such long background processes to disturb the time elapsed enough to cause these discrepancies.

Another hypothesis is inspired by Szegedy et al. [118]'s finding that a too large network suffered from dying nodes, meaning all nodes in the network soon turned to zero, and the network could no longer learn anything. Now, for the networks with one fully connected layer, this layer can be interpreted as deciding how important the activation of every filter in the last convolutional layer is for the final classification result. In the networks with two fully connected layers, there is a fully connected layer in between the convolutional body and the layer that decides what information is important for classification. It could be argued that this leads to nodes in the first fully connected layer dying. This would then make the backpropagation algorithm cheaper: once a coefficient has become zero, the chain rule no longer has to be computed, saving computational costs.

In order to test this hypothesis, we looked into VGG19-1fc's and VGG19-2fc's weights. Between the convolutional body and the first fully connected layer, VGG19-2fc has 9,437,184 weights. Of these weights, 5,138,380 are smaller than 0.0000001. This is 54.4%. On the other hand, 1,791,519 of these weights are larger than 0.01. This is 19.0%. Between the first and second convolutional layer, VGG19-2fc has 512 weights. Of these weights, 266 are smaller than 0.0000001. This is 52.0%. On the other hand, 211 are larger than 0.01. This is 41.2%.

VGG19-1fc has 512 weights between its convolutional body and classification layer. Of these weights, 263 are smaller than 0.0000001. This is 51.4%. On the other hand, 160 weights are larger than 0.01; this is 31.3%. This shows that indeed, many of VGG19-2fc's weights are small; however, in the last layer, VGG19-1fc has fewer large (>0.01) weights than VGG19-2fc does. It might still be that the location of the small weights manages to cancel large part of the chain rule, saving computational costs. However, the evidence is far from overwhelming.

The last hypothesis is that the differences in time elapsed during training are due to AWS `t3`-instances being burstable[5]. This means their performance is not as predictable as that of regular desktop computer's. For now, we consider this hypothesis the most likely one.

To form a clearer idea of what causes the discrepancies in training times, the training runs should be executed again to gather more data. However, since this would take approximately two weeks to execute on the `t3.xlarge` instance, we deem this too expensive for how much impact the discrepancies have on this research.

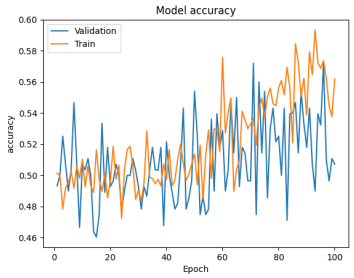
5.6. Experiment 6: Using Discrete Cosine Transform of Images

These experiments are conducted on three different datasets (FaceForensics++, Celeb-df, DeeperForensics-1.0), and with eight different models (ResNet-2fc, ResNet-2fc, Shallow-2fc, Shallow-1fc, VGG16-2fc, VGG16-1fc, VGG19-2fc, VGG19-1fc). The preprocessing settings are rotation range 40, shear range 0.2, zoom range 0.2, width shift range 0.2, height shift range 0.2, and horizontal flipping. The batch size 100, the input size 200×200. Fig. 5.10 shows the accuracies of the different models on the different datasets; Fig. 5.11 shows the losses of the relevant models on the different datasets.

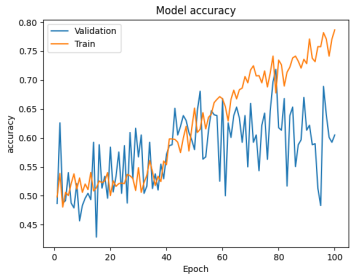
Whereas on original images ResNet-1fc and ResNet-2fc managed to learn something for a few epochs before overfitting set in, training on DCT-residuals of images resulted in immediate overfitting. The same goes for VGG16-1fc , VGG16-2fc, VGG19-1fc, and VGG19-2fc on Celeb-df. This means the capacity of these networks is not suited for detecting Deepfakes with the used datasets. It seems that the required capacity for this task is lower than for the task on original images: more networks overfit on DCT-residuals. Moreover, on Celeb-df and DeeperForensics-1.0, the shallow networks seem to have more advantage on DCT-residuals than on original images. Except for the highest accuracy on FaceForensics++, all top scores were achieved by Shallow-2fc and Shallow-1fc; and the differences in accuracy between all models on FaceForensics++ are negligible.

There are two networks trained on DCT-residuals that outperform their counterparts trained on original images, although the differences in accuracy are negligible: 1.4% for both Shallow-2fc on DeeperForensics-1.0, and Shallow-1fc on Celeb-df. The difference in loss is also negligible in the latter case: 0.01. The difference in loss for Shallow-2fc on DeeperForensics-1.0 is more considerable: 0.06, in the advantage of the model trained on DCT-residuals. On both DCT-residuals and original Celeb-df images, Shallow-1fc starts overfitting around epoch 30. Although we cannot conclude that these two models trained on DCT-residuals are better suited for the task of Deepfake detection due to the small differences, we can at least conclude they are equally suited as their counterparts trained on original images.

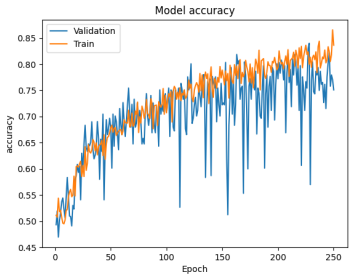
[5] See `https://aws.amazon.com/ec2/instance-types/t3/`.



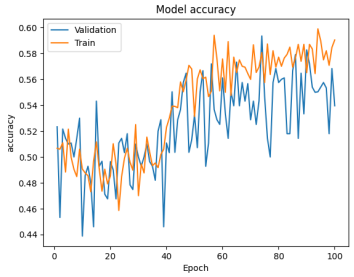
(a) Accuracy of Shallow-2fc on DCT-residuals of FaceForensics++.



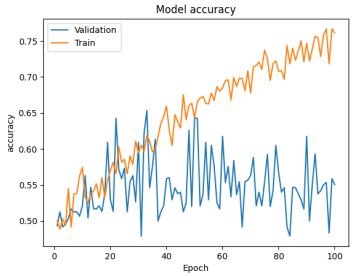
(b) Accuracy of Shallow-2fc on DCT-residuals of Celeb-df.



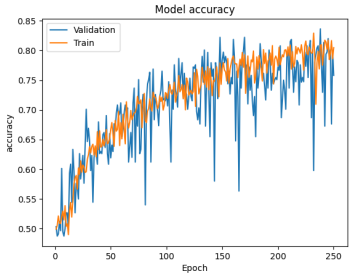
(c) Accuracy of Shallow-2fc on DCT-residuals of DeeperForensics-1.0.



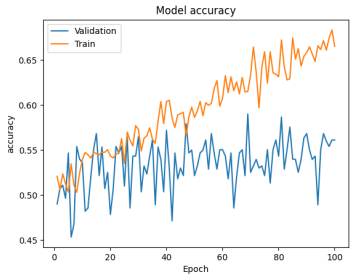
(d) Accuracy of Shallow-1fc on DCT-residuals of FaceForensics++.



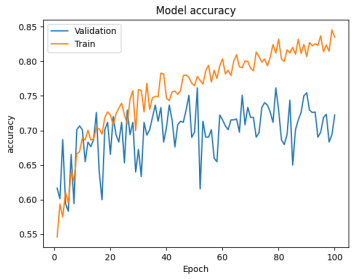
(e) Accuracy of Shallow-1fc on DCT-residuals of Celeb-df.



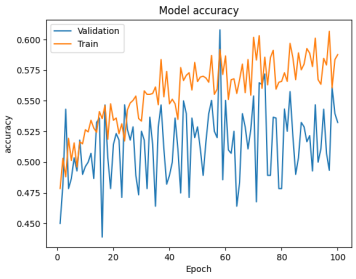
(f) Accuracy of Shallow-1fc on DCT-residuals of DeeperForensics-1.0.



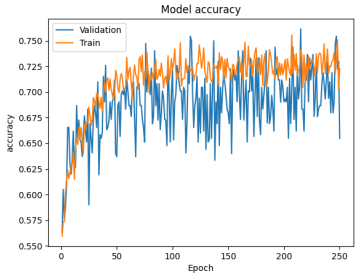
(g) Accuracy of VGG16-2fc on DCT-residuals of FaceForensics++.



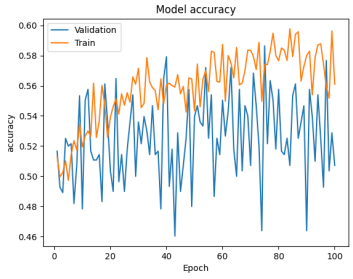
(h) Accuracy of VGG16-2fc on DCT-residuals of DeeperForensics-1.0.



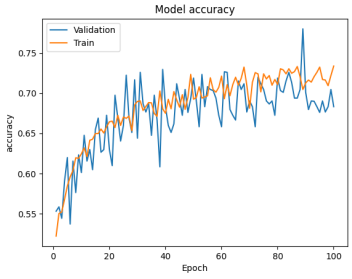
(i) Accuracy of VGG16-1fc on DCT-residuals of FaceForensics++.



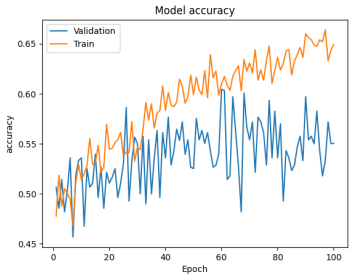
(j) Accuracy of VGG16-1fc on DCT-residuals of DeeperForensics-1.0.



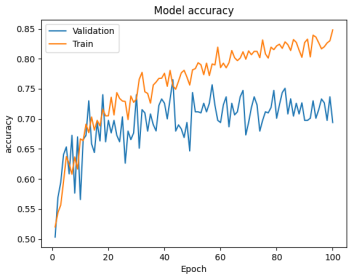
(k) Accuracy of VGG19-1fc on DCT-residuals of FaceForensics++.



(l) Accuracy of VGG19-1fc on DCT-residuals of DeeperForensics-1.0.

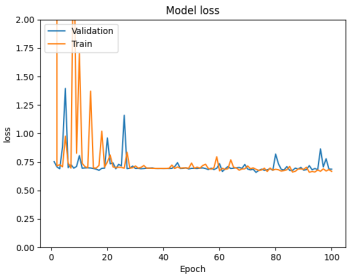


(m) Accuracy of VGG19-2fc on DCT-residuals of FaceForensics++.

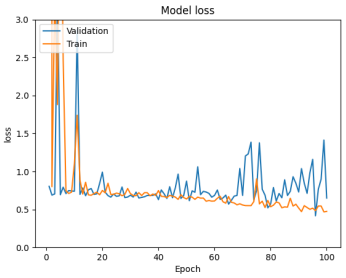


(n) Accuracy of VGG19-2fc on DCT-residuals of DeeperForensics-1.0.

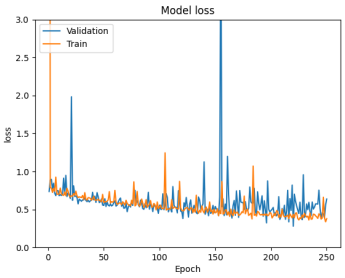
Figure 5.10: Accuracies of Shallow-1fc, Shallow-2fc, VGG16-1fc, VGG16-2fc, VGG19-1fc, and VGG19-2fc on the DCT-redisuals of different datasets.



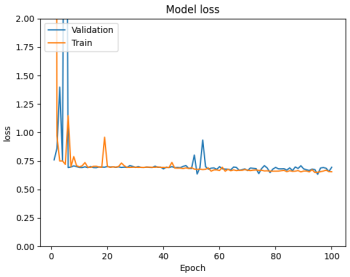
(a) Loss of Shallow-2fc on DCT-residuals of FaceForensics++.



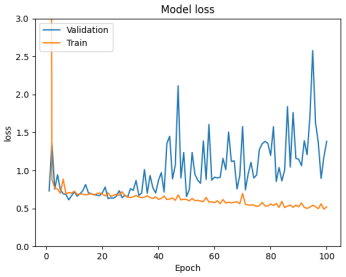
(b) Loss of Shallow-2fc on DCT-residuals of Celeb-df.



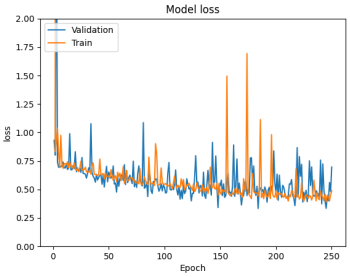
(c) Loss of Shallow-2fc on DCT-residuals of DeeperForensics-1.0.



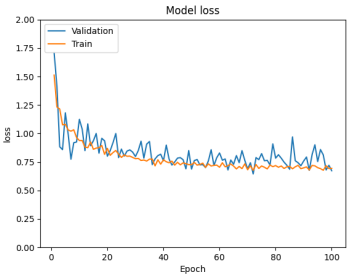
(d) Loss of Shallow-1fc on DCT-residuals of FaceForensics++.



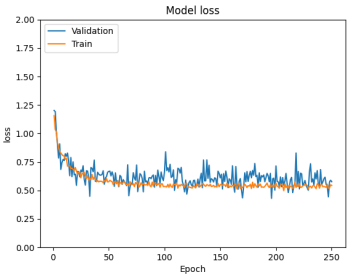
(e) Loss of Shallow-1fc on DCT-residuals of Celeb-df.



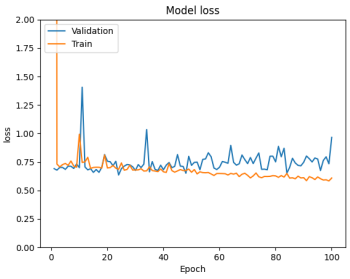
(f) Loss of Shallow-1fc on DCT-residuals of DeeperForensics-1.0.



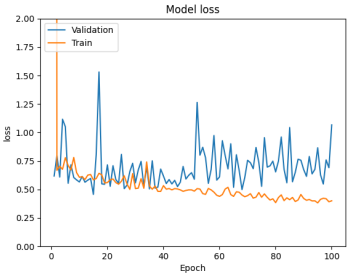
(g) Loss of VGG16-1fc on DCT-residuals of FaceForensics++.



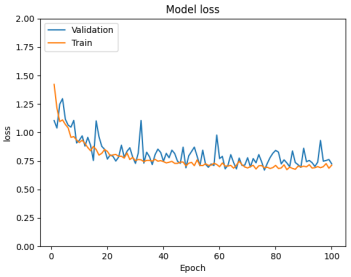
(h) Loss of VGG16-1fc on DCT-residuals of DeeperForensics-1.0.



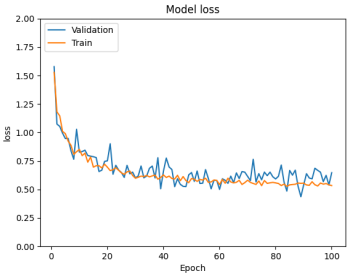
(i) Loss of VGG16-2fc on DCT-residuals of FaceForensics++.



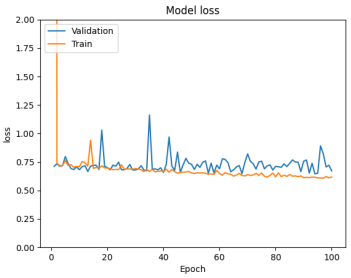
(j) Loss of VGG16-2fc on DCT-residuals of DeeperForensics-1.0.



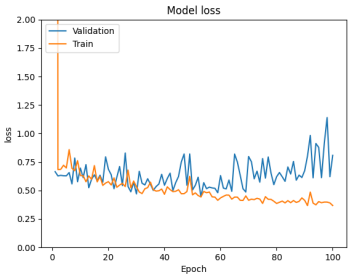
(k) Loss of VGG19-1fc on DCT-residuals of FaceForensics++.



(l) Loss of VGG19-1fc on DCT-residuals DeeperForensics-1.0.



(m) Loss of VGG19-2fc on DCT-residuals of FaceForensics++.



(n) Loss of VGG19-2fc on DCT-residuals DeeperForensics-1.0.

Figure 5.11: Losses of Shallow-1fc, Shallow-2fc, VGG16-1fc, VGG16-2fc, VGG19-1fc, and VGG19-2fc on the DCT-residuals of different datasets.

Table 5.10: Performance of different models on different datasets. The maximum accuracy reported is the validation accuracy; the minimum loss reported is the validation loss. In case the model overfits, the reported maximum accuracy and minimum loss are those before overfitting sets in. The seventh column reports the epoch after which overfitting sets in; a hyphen indicates no overfitting takes place. The highest accuracy and lowest loss per dataset are presented in bold.

Model	Dataset	Performance				
		Accuracy		Loss		Overfitting
		Fig.	Max (%)	Fig.	Min	Epoch
Shallow-1fc	FaceForensics++	5.10d	59.4	5.11d	**0.63**	-
Shallow-1fc	Celeb-df	5.10e	65.3	5.11e	0.61	32
Shallow-1fc	DeeperForensics-1.0	5.10f	83.6	5.11f	0.33	-
Shallow-2fc	FaceForensics++	5.10a	57.2	5.11a	0.66	-
Shallow-2fc	Celeb-df	5.10b	**68.1**	5.11b	**0.57**	64
Shallow-2fc	DeeperForensics-1.0	5.10c	**84.0**	5.11c	**0.28**	-
VGG16-1fc	FaceForensics++	5.10i	**60.8**	5.11g	0.65	-
VGG16-1fc	DeeperForensics-1.0	5.10j	75.1	5.11h	0.45	-
VGG16-2fc	FaceForensics++	5.10g	59.0	5.11i	0.64	39
VGG16-2fc	DeeperForensics-1.0	5.10h	76.2	5.11j	0.43	-
VGG19-1fc	FaceForensics++	5.10k	58.7	5.11k	0.67	-
VGG19-1fc	DeeperForensics-1.0	5.10l	78.0	5.11l	0.44	-
VGG19-2fc	FaceForensics++	5.10m	60.4	5.11m	0.65	60
VGG19-2fc	DeeperForensics-1.0	5.10n	76.5	5.11n	0.44	42

Although all other models trained on DCT-residuals reach lower accuracy and higher loss than their counterparts trained on original images, the differences are negligible in the case of Shallow-2fc on Celeb-df: 0.8% accuracy and 0.02 loss. For both models, overfitting sets in around epoch 65. This means that for Shallow-2fc, DCT-residuals of Celeb-df might also be equally suited for learning Deepfake detection as original Celeb-df images.

All other models are outperformed by their counterparts trained on original images, both in terms of loss and accuracy. Moreover, ten of the model-dataset combinations used overfit earlier on DCT-residuals than they did on original images, if they overfitted on the latter at all. This means that networks trained on DCT-residuals do not generally outperform their counterparts trained on original images. We conclude that training our models on DCT-residuals does not have the desired effect of improved performance.

Although the training on DCT-residuals has not increased performance, we do note it has changed the relations between the datasets. Although most networks overfit on Celeb-df, we note that the ones that do not overfit (Shallow-1fc and Shallow-2fc) reach higher accuracies on Celeb-df than on FaceForensics++. On the original images, this relation was reversed.

Since DCT stresses edges, we hoped that either the edges around swapped faces would stand out clearer, or many features would be diminished due to heavy smoothing. The first does not seem to happen at all; the latter happens, for example, in the frame from 316_W006 depicted in Fig. B.11. However, it does not happen systematically. Several other images in Fig. B.9, Fig. B.6 , and Fig. B.11 even show that many of the DCT-residuals of fake frames have clear eyes. We expected them to be less clear, since e.g. Matern et al. [88] discuss that features such as eyes are difficult to model clearly.

It is clear that not just any residual filter suffices to improve performance. Since literature (e.g., Zhou et al [140]) does report improvents when training on image residuals, we suppose this means a more suitable filter has to be sought. This can either be another type of DCT (since we used DCT-II) or another filter altogether. Moreover, it is possible that DCT-residuals are not suited for Deepfake detection as such, but only their histograms, which are used by Amerini et al. [6] and Zampoglou et al. [136].

5.7. Experiment 7: Generalisation and Robustness

Table 5.11 reports the accuracies, sensitivities, specificities, precisions, fall-outs, miss rates, and F_1-measures of ResNet-1fc, ResNet-2fc, Shallow-1fc, Shallow-2fc, VGG16-1fc, VGG16-2fc, VGG19-1fc, and VGG19-2fc trained on DeeperForensics-1.0. The measures are reported on four different datasets: FaceForensics++, Celeb-df, DeeperForensics-1.0 test, DeeperForensics-1.0 train.

Table 5.11: Robustness of different models trained on DeeperForensics-1.0. The robustness is measured by accuracy (ACC), sensitifity (SEN), specificity (SP), precision (PR), fall-out (FO), miss rate (MR), and F_1 on three different datasets: FaceForensics++, Celeb-df, and the DeeperForensics-1.0 test dataset. All measures are reported as percentages.

FaceForensics++							
	ACC (%)	**SEN (%)**	**SP (%)**	**PR (%)**	**FO (%)**	**MR (%)**	**F_1 (%)**
ResNet-1fc	48.6	11.9	84.9	43.8	15.1	88.1	18.7
ResNet-2fc	50.3	12.1	88.1	50.2	11.9	87.9	19.5
Shallow-1fc	50.1	10.0	89.9	49.5	10.1	90.0	16.6
Shallow-2fc	51.2	12.5	89.6	54.4	10.4	87.5	20.3
VGG16-1fc	49.3	10.0	88.4	45.9	11.6	90.0	16.4
VGG16-2fc	**51.4**	11.9	90.7	55.7	9.3	88.1	19.6
VGG19-1fc	47.9	**17.2**	78.3	43.9	21.7	**82.8**	**24.7**
VGG19-2fc	51.2	4.6	**97.4**	**63.2**	**2.6**	95.4	8.5
Celeb-df							
	ACC (%)	**SEN (%)**	**SP (%)**	**PR (%)**	**FO (%)**	**MR (%)**	**F_1 (%)**
ResNet-1fc	43.8	49.7	37.8	44.7	62.2	50.3	47.1
ResNet-2fc	44.5	51.2	37.8	45.5	62.2	48.8	48.2
Shallow-1fc	44.3	15.0	74.0	36.9	26.0	85.0	21.3
Shallow-2fc	45.5	15.7	75.7	39.5	24.3	84.3	22.5
VGG16-1fc	**51.7**	42.6	60.9	**52.5**	39.1	57.4	47.0
VGG16-2fc	50.7	**54.0**	47.3	50.9	52.7	**46.0**	**52.4**
VGG19-1fc	45.5	35.6	55.5	44.8	44.5	64.4	39.7
VGG19-2fc	47.1	15.4	**79.3**	43.0	**20.7**	84.6	22.6
DeeperForensics-1.0 (test)							
	ACC (%)	**SEN (%)**	**SP (%)**	**PR (%)**	**FO (%)**	**MR (%)**	**F_1 (%)**
ResNet-1fc	81.2	85.2	77.1	78.8	22.9	14.8	81.9
ResNet-2fc	84.3	89.2	79.4	81.3	20.6	10.8	85.0
Shallow-1fc	**92.4**	95.6	89.2	89.8	10.8	4.4	**92.6**
Shallow-2fc	81.6	74.1	89.2	87.3	10.8	25.9	80.1
VGG16-1fc	84.0	86.4	81.6	82.5	18.4	13.6	84.4
VGG16-2fc	88.1	**97.5**	78.8	82.1	21.2	**2.5**	89.2
VGG19-1fc	82.0	91.6	72.3	76.8	27.7	8.4	83.5
VGG19-2fc	91.2	89.5	**92.9**	**92.6**	**7.1**	10.5	91.0
DeeperForensics-1.0 (train)							
	ACC (%)	**SEN (%)**	**SP (%)**	**PR (%)**	**FO (%)**	**MR (%)**	**F_1 (%)**
ResNet-1fc	84.0	83.2	84.9	84.7	15.1	16.8	83.9
ResNet-2fc	87.9	87.8	88.1	88.1	11.9	12.3	87.9
Shallow-1fc	93.6	97.3	89.9	90.6	10.0	2.7	93.8
Shallow-2fc	82.9	76.3	89.6	88.0	10.4	23.7	81.7
VGG16-1fc	85.9	83.5	88.4	87.8	11.6	16.5	85.6
VGG16-2fc	**94.1**	**97.5**	90.7	91.3	93.4	**2.5**	**94.3**
VGG19-1fc	84.5	90.8	78.3	80.7	21.7	9.2	85.5
VGG19-2fc	93.7	90.1	**97.4**	**97.2**	**2.6**	9.9	93.5

5.7.1. Generalisation

By comparing performance on the DeeperForensics-1.0 testing and training datasets, we can analyse networks' generalisation capabilities. The differences in accuracy are mostly small. VGG16-2fc seems to be an outlier with 6.0% accuracy degradation. Otherwise, all accuracy degradation is between 3.6% (ResNet-2fc) and 1.2% (Shallow-1fc). This means all eight networks seem to generalise at least reasonably. However, since the frames from the test dataset are all at most 25 frames apart from the frames in the training dataset, the networks might have more difficulties classifying the test dataset if the frames are further apart.

ResNet-1fc's accuracy degrades 2.2% less than ResNet-2fc's; Shallow-1fc's degrades 0.1% less than Shallow-2fc's; VGG16-1fc's degrades 4.1% more than VGG16-2fc's; VGG19-1fc and VGG19-2fc's accuracy both degrade 2.5%. This might indicate that the 1fc-models generalise better, but since the differences are small, more data would have to be collected to confirm or deny this hypothesis. However, due to time restrictions, we will not

pursue this approach.

Again, if we consider VGG16-2fc an outlier, the Shallow models generalise best, followed by the VGG16s, the VGG19s, and lastly the ResNets. However, it is interesting that VGG16-2fc should be considered an outlier: it is the best performing network in terms of accuracy, sensitivity, and F_1-measure on the training data. This might imply that VGG16-2fc started overfitting on the data right before training was terminated.

ResNet-1fc, ResNet-2fc, VGG16-1fc and VGG19-1fc have higher sensitivity scores on the testing data than on the training data. VGG16-2fc has scored the same on training sensitivity as on testing sensitivity. This means that these networks classified more (or as many) fake samples as fake in the test dataset than in the training dataset. The other networks' sensitivity is between 0.6% (VGG19-2fc) and 2.2% (Shallow-2fc) lower.

The differences in specificity are larger than the differences in the other performance measures. In terms of specificity, Shallow-1fc and Shallow-2fc clearly show the least degradation (10.7% and 0.4% respectively). VGG16-2fc's specificity degrades the most (11.9%); VGG16-2fc's specificity degrades slightly less (6.8%). VGG19-1fc and VGG16-2fc are in between the Shallow and the VGG16 models (6.0% and 4.5% respectively). ResNet-1fc and ResNet-2fc's specificities degrade more (7.8% and 8.7% respectively). These considerable degradations mean that the networks have more difficulties classifying a real sample as real in the testing than in the training dataset.

Lastly, when comparing the precision, Shallow-1fc and Shallow-2fc again show the least degradation (0.8% and 0.7% respectively), followed by VGG19-1fc and VGG19-2fc (3.9% and 4.6% respectively). VGG16-1fc, VGG16-2fc, ResNet-1fc, and ResNet-2fc show more degradation (5.3%, 9.2%, 15.9%, and 6.8% respectively). This means that on the testing data, considerably fewer samples classified as fake are indeed fake.

It seems that although the differences in terms of accuracy are relatively small, the shallow models' specificities and precisions show considerably less degeneration than the other models' specificities. Since the Shallow models also show the least degradation in their accuracies, we can conclude they generalise best.

5.7.2. Robustness

By comparing the performance on the DeeperForensics-1.0 test dataset to the Celeb-df and FaceForensics++ datasets, we can analyse the networks' robustness. It is clear that none of the models is robust. On Celeb-df, only two out of eight networks perform better than random in terms of accuracy: VGG16-1fc and VGG16-2fc (51.7% and 50.7% respectively).

On FaceForensics++, the specificity is the same as for the DeeperForensics-1.0 training dataset. This is because we used the real samples from FaceForensics++ as the real data counterpart of the DeeperForensics-1.0 training dataset. Therefore, we can only draw conclusions on FaceForensics++'s sensitivity. The highest sensitivity is reached by VGG19-1fc with 17.2%; the lowest by VGG19-2fc with 4.6%. This means that out of all fake samples, all networks classify only between 4.5% and 17.2% as fake. To be able to compare more than only sensitivity, we would have to construct a dataset with different real images. However, given the drastically bad sensitivity, we feel that would be a waste of time.

On Celeb-df, two networks manage a sensitivity better than random: ResNet-1fc and VGG16-2fc (51.2% and 54.0%). This means these two networks manage to classify more than half the fake samples as fake. Three networks manage a specificity above 70%: Shallow-1fc, Shallow-2fc, and VGG19-2fc (74.0%, 75.7%, and 79.3% respectively). This means these three networks manage to classify approximately 75% of real samples as real. Two networks manage a precision better than random: VGG16-1fc and VGG16-2fc (52.5% and 50.9% respectively). This means that for these two networks, at least half of the samples classified as fake is indeed fake.

What is interesting is that section 5.5 shows that Celeb-df is the most difficult dataset to learn Deepfake detection on. FaceForensics++ and DeeperForensics-1.0 resulted in considerably higher accuracies, lower losses, and less overfitting. However, this experiment shows that our networks trained on DeeperForensics-1.0 have more difficulties making sense of the FaceForensics++ dataset than of the Celeb-df dataset. This supports our earlier hypothesis that FaceForensics++ is too inconsistent to use for training networks for Deepfake detection (see section 5.3).

This lack of robustness is especially troubling because most research into Deepfake detection focusses on VAE- and GAN-generated imagery. However, section 3.2 shows that there are other types of generation algorithms being developed. If our models are not robust to other datasets generated by similar (although not identical) generation algorithms, we can hardly expect them to be robust to other datasets generated by different generation algorithms.

6

Conclusions

This research aimed to provide insight into the effects that hyperparameters, dataset composition, and network architecture have on the performance of convolutional neural networks for Deepfake detection. This research also compared models' performance on unseen data to judge their robustness.

We found that dataset variety influences performance more than dataset size. A dataset with 25 frames per FaceForensics++ video resulted in immediate overfitting. In contrast, a dataset with one frame per FaceForensics++ video (containing only 1897 frames divided over two categories) helped the network reach 72.7% accuracy before overfitting set in. Increases in runtime seem to be approximately linear with increases of dataset size. We carried out the remainder of our research with the dataset counting 1897 frames.

Preprocessing helps curtail overfitting. However, in our case, it does not help the model achieve higher accuracy. We expect this is due to inconsistent artefacts in the dataset rather than to lack of potential in preprocessing. Preprocessing does not seem to influence time elapsed for training.

Similarly, higher dropout rates manage to keep training and validation learning curves closer together but do not help the model achieve higher accuracy. The accuracy not increasing is not necessarily caused by a lack of potential in dropout: it can also be due to inconsistent artefacts in the dataset. Higher dropout rates do not seem to influence time elapsed for training.

Batch size also does not seem to affect the maximum accuracy a model can achieve. However, it does seem to affect how many epochs it takes the model to start learning: the smaller the batch size, the sooner the algorithm starts learning. The additional epochs needed for the network with larger batch size to start training is probably due to the gradient descent algorithm having to average its update over more samples, resulting in less steep gradients. Lastly, runtimes for training seem to decrease with batch size. This decrease is likely due to the backpropagation algorithm running more frequently with smaller batch sizes.

DenseNet-121, Inception-v3, and Xception immediately overfitted on FaceForensics++, Celeb-df, and DeeperForensics-1.0. We suspect their capacity is too great for the small (though highly varied) datasets we composed (although all datasets except the shallow one were pretrained on ImageNet). ResNet-152, VGG16, VGG19, and our shallow network all managed to learn something from all three datasets, although they too overfitted on Celeb-df after several epochs. The specific amount of epochs at which overfitting set in differs between 2 (ResNet-152 and VGG19) and 63 (shallow network).

We also tested different classification networks on top of the convolutional bodies: one with a global average pooling layer followed by a single fully connected classification layer, and one with two fully connected layers. Whereas the latter results in many more trainable parameters, in most experiments it overfitted later than the former.

There is not one convolutional body that scored best on all three datasets, nor a single classification network that performed best. The highest accuracies achieved per dataset were

- 76.3% on FaceForensics++, by the shallow network with two fully connected layers on top,
- 68.9% on Celeb-df, by the shallow network with two fully connected layers on top,
- 92.5% on DeeperForensics-1.0, by VGG16 with one fully connected layer on top.

It was clear that Celeb-df was the most challenging dataset to learn Deepfake detection on. We expect this is because it is the most sophisticated dataset. It is also clear that DeeperForensics-1.0 was the easiest dataset

to learn Deepfake detection on. We expect this is due to the high variety of artefacts in the less sophisticated FaceForensics++ making it difficult for the network to find global signs of forgery. These findings imply that to be able to detect Deepfake videos, we first need a large, sophisticated dataset to train networks on.

Using the DCT-residuals of images for network training was supposed to suppress semantical image content, helping the network to focus on statistical image content. However, our networks trained on DCT-residuals do not outperform their counterparts trained on original images. The shallow network combined with both classification networks on DeeperForensics-1.0 and the shallow network with two fully connected layers on Celeb-df trained on DCT-residuals perform similarly to their counterparts trained on original images. The other networks are outperformed by their counterparts trained on original images. Hence, DCT-residuals turn out not to be useful for detecting Deepfake images with our small datasets.

Lastly, we tested the networks' generalisation and robustness capabilities. For robustness, we tested the network on unseen frames from the dataset on which it was trained. The results showed that degradation in accuracy was small. Most networks showed larger degradations in specificity and precision. Since the shallow networks showed little degradation in these two performance measures as well, we conclude they generalise best. None of the networks was robust: testing them on frames from other datasets than they were trained on resulted in approximately random accuracy.

6.1. Future Research

This section provides angles for future research.

6.1.1. Architectural Improvements

Our experiments suggest that with the small datasets we have at our disposal, networks with relatively small capacity might be favourable. However, there is a large gap between the best performing networks, shallow and VGG16, which have four convolutional layers and 13 convolutional layers respectively. Further research could investigate whether there is some optimal convolutional body with five to 12 convolutional layers.

Moreover, further research could perform experiments with other VGGs, DenseNets, and ResNets, to try to understand why the DenseNet-121 overfitting on all datasets, while ResNet-152 did not. Experimenting with more of these networks of different depths but with similar architectures could provide insight in this.

Lastly, it is likely that improvements in general deep learning architecture and convolutional network architecture specifically will be applicable in the field of Deepfake detection. For example, research into better activation functions seems promising. We expect improved versions of ReLU, such as Leaky-ReLU, can have a positive effect on our network performance.

6.1.2. Datasets

Although our research suggests that dataset variety is more important than dataset size, ideally a dataset will be both large and varied. Therefore, there is an urgent need for larger datasets with a higher variety of target individuals, source individuals, and more videos. For framewise detection, training on several frames from one video only seems desirable when the video is long enough to contain several frames with little likeness. Further research could also try to define this 'little likeness' criterium. Although videos in currently available Deepfake datasets usually are around 10 seconds long, it is possible that using one frame per second of video would still yield enough variation in data to be useful.

Until datasets with more videos are available, it might be possible to combine different datasets to obtain a higher variety of videos. However, a problem here is that if a dataset becomes imbalanced, a different approach is needed[1]; since two of the three datasets we used are based on the same real videos, a combined dataset would be imbalanced.

Another improvement could be to use a network pretrained not on a general dataset such as ImageNet, which contains no human shapes, but on a dataset that is focussed on faces. Alternatively, it seems promising to use ImageNet as pretraining dataset if we decide not to freeze the latter layers of the convolutional bodies. This would allow the network to tweak its most complicated filters to specifically fit the Deepfake detection task.

Furthermore, if a Deepfake detection tool is to be implemented to work on a specific type of data, it is desirable that it has been (re)trained on a dataset made of this specific datatype. For example, most datasets are based on videos in which the person on the footage is not the person shooting the footage. This results in, for example, more stable backgrounds than would occur in videos made with selfie cameras. However, many

[1] See for example Du and Swamy [32].

videos on the internet are made with selfie cameras. Therefore, it is ways important to finetune a detection tool on the specific type of data that is to be detected.

6.1.3. Interframe Detection

When watching Deepfake videos, the signs of manipulation are often temporal. Therefore, it is likely that network performance will improve with interframe detection. This is a promising next step in the detection of Deepfake videos.

6.1.4. Residual Filters

We still believe that a suitable residual filter can improve performance. Since manipulators will focus on the semantical content of images, because in the first place a Deepfake is meant to convince a human public of its integrity, it is likely easier to find discrepancies in statistical image content. Residual filters can help detection algorithms focus on statistical image content, which should make discrepancies easier to find.

Further research into different residual filters, their properties, and their effects on the performance of detection algorithms is required.

6.1.5. Active Detection

Although at the time of writing this report, active detection is not yet applicable in real-world scenarios, it seems to be an interesting solution for the long term. If in the future all mobile phones will be equipped with software enabling active detection, it might even outperform passive detection. This would mean that passive detection is only needed to bridge the gap in time until active detection becomes available. Active detection might be able to end the cat-and-mouse game that is going on between image manipulation and its detection.

Bibliography

[1] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen. Mesonet: a compact facial video forgery detection network. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7, 2018.

[2] S. Agarwal, H. Farid, Y. Gu, M. He, K. Nagano, and H. Li. Protecting world leaders against deep fakes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019.

[3] B. Ahmed, T. A. Gulliver, and S. alZahir. Image splicing detection using mask-rcnn. *Signal, Image and Video Processing*:1863–1711, 2020.

[4] Z. Akhtar and D. Dasgupta. A comparative evaluation of local feature descriptors for deepfakes detection. In *2019 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–5, 2019.

[5] M. Albright and S. McCloskey. Source generator attribution via inversion. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019.

[6] I. Amerini, T. Uricchio, L. Ballan, and R. Caldelli. Localization of jpeg double compression through multi-domain convolutional neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1865–1871, 2017.

[7] I. Amerini, L. Galteri, R. Caldelli, and A. Del Bimbo. Deepfake video detection through optical flow based cnn. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, 2019.

[8] S. Anwar, M. Milanova, M. Anwer, and A. Banihirwe. Perceptual judgments to detect computer generated forged faces in social media. In F. Schwenker and S. Scherer, editors, *Multimodal Pattern Recognition of Social Signals in Human-Computer-Interaction*, pages 38–48, Cham. Springer International Publishing, 2019.

[9] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 214–223, Sydney, NSW, Australia. JMLR.org, 2017.

[10] A. Bansal, S. Ma, D. Ramanan, and Y. Sheikh. Recycle-gan: unsupervised video retargeting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[11] J. H. Bappy, A. K. Roy-Chowdhury, J. Bunk, L. Nataraj, and B. S. Manjunath. Exploiting spatial structure for localizing manipulated image regions. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[12] B. Bayar and M. C. Stamm. A deep learning approach to universal image manipulation detection using a new convolutional layer. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '16, 5–10, Vigo, Galicia, Spain. Association for Computing Machinery, 2016.

[13] Y. Bengio. Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[14] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen. *Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis*. Springer International Publishing, Cham, 2017.

[15] L. Bondi, S. Lameri, D. Güera, P. Bestagini, E. J. Delp, and S. Tubaro. Tampering detection and localization through clustering of camera-based cnn features. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1855–1864, 2017.

[16] P. Borwarnginn, K. Thongkanchorn, S. Kanchanapreechakorn, and W. Kusakunniran. Breakthrough conventional based approach for dog breed classification using cnn with transfer learning. In *2019 11th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–5, 2019.

[17] K. Boyd, K. H. Eng, and C. D. Page. Area under the precision-recall curve: point estimates and confidence intervals. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 451–466, Berlin, Heidelberg. Springer Berlin Heidelberg, 2013.

[18] A. L. Caterini and D. E. Chang. *Deep Neural Networks in a Mathematical Framework*. Springer International Publishing, Cham, 2018.

[19] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo. Stargan: unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[20] F. Chollet. *Deep Learning with Python*. Manning Publications Company, Shelter Island, New York, 2017.

[21] F. Chollet. Xception: deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[22] C. Chui, X. Li, and H. N. Mhaskar. Neural networks for localized approximation. *Mathematics of Computation*, 63(208):607–623, 1994.

[23] D. Cozzolino and L. Verdoliva. Noiseprint: a cnn-based camera model fingerprint. *IEEE Transactions on Information Forensics and Security*, 15:144–159, 2020.

[24] D. Cozzolino, G. Poggi, and L. Verdoliva. Recasting residual-based local descriptors as convolutional neural networks: an application to image forgery detection. In *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '17, 159–164, Philadelphia, Pennsylvania, USA. Association for Computing Machinery, 2017.

[25] D. Cozzolino, G. Poggi, and L. Verdoliva. Extracting camera-based fingerprints for video forensics. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019.

[26] D. Cozzolino, J. Thies, A. Rössler, M. Nießner, and L. Verdoliva. Spoc: spoofing camera fingerprints, 2019. arXiv: 1911.12069 [cs.CV].

[27] D. Cozzolino, J. Thies, A. Rössler, C. Riess, M. Nießner, and L. Verdoliva. Forensictransfer: weakly-supervised domain adaptation for forgery detection, 2018. arXiv: 1812.02510 [cs.CV].

[28] D. D'Avino, D. Cozzolino, G. Poggi, and L. Verdoliva. Autoencoder with recurrent neural networks for video forgery detection. *arXiv e-prints*:arXiv:1708.08754, arXiv:1708.08754, Aug. 2017. arXiv: 1708.08754 [cs.CV].

[29] L. Dinh, D. Krueger, and Y. Bengio. NICE: non-linear independent components estimation. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[30] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[31] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[32] K.-L. Du and M. N. S. Swamy. *Neural Networks and Statistical Learning*. Springer London, London, 2019.

[33] R. A. Dunne and N. A. Campbell. On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proceedings of the 8th Australian Conference on the Neural Networks*, volume 181, pages 181–185, Melbourne, 1997.

[34] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 –874, 2006. ROC Analysis in Pattern Recognition.

[35] S. Fernandes, S. Raj, E. Ortiz, I. Vintila, M. Salter, G. Urosevic, and S. Jha. Predicting heart rate variations of deepfake videos using neural ode. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, 2019.

[36] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, 189–194 vol.3, 2000.

[37] F. Gers. Learning to forget: continual prediction with lstm. English. *IET Conference Proceedings*:850–855(5), 1999.

[38] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber. Learning precise timing with lstm recurrent networks. *J. Mach. Learn. Res.*, 3(null):115–143, Mar. 2003.

[39] N Girish and C Nandini. A review on digital video forgery detection techniques in cyber forensics. *Science, Technology and Development*, 8(9), 2019.

[40] A. Gokhale, P. Mulay, D. Pramod, and R. Kulkarni. A bibliometric analysis of digital image forensics. *Science & Technology Libraries*, 39(1):96–113, 2020. eprint: https://doi.org/10.1080/0194262X.2020.1714529.

[41] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, Cambridge (MA), 2016.

[42] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[43] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: a search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

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

[45] D. Güera, S. Baireddy, P. Bestagini, S. Tubaro, and E. J. Delp. We need no pixels: video manipulation detection using stream descriptors, 2019. arXiv: 1906.08743 [cs.LG].

[46] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[47] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel. Flow++: improving flow-based generative models with variational dequantization and architecture design. In K. Chaudhuri and R. Salakhutdinov, editors, volume 97 of *Proceedings of Machine Learning Research*, pages 2722–2730, Long Beach, California, USA. PMLR, 2019.

[48] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. eprint: https://doi.org/10.1162/neco.1997.9.8.1735.

[49] J. Horvath, D. Guera, S. Kalyan Yarlagadda, P. Bestagini, F. Maggie Zhu, S. Tubaro, and E. J. Delp. Anomaly-based manipulation detection in satellite images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019.

[50] X. Hou, L. Shen, K. Sun, and G. Qiu. Deep feature consistent variational autoencoder. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1133–1141, 2017.

[51] M. J. Howard, A. S. Williamson, and N. Norouzi. Video manipulation detection via recurrent residual feature learning networks. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1–5. IEEE, 2019.

[52] C.-C. Hsu, Y.-X. Zhuang, and C.-Y. Lee. Deep fake image detection based on pairwise learning. *Applied Sciences*, 10(1):370, 2020.

[53] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[54] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 646–661, Cham. Springer International Publishing, 2016.

[55] M. Huh, A. Liu, A. Owens, and A. A. Efros. Fighting fake news: image splice detection via learned self-consistency. In *The European Conference on Computer Vision (ECCV)*, 2018.

[56] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[57] A. K. Jain and A. Ross. *Handbook of Biometrics*. A. K. Jain, P. Flynn, and A. A. Ross, editors. Springer US, Boston, MA, 2008.

[58] H. Jeon, Y. Bang, and S. S. Woo. Faketalkerdetect: effective and practical realistic neural talking head detection with a highly unbalanced dataset. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, 2019.

[59] L. Jiang, R. Li, W. Wu, C. Qian, and C. C. Loy. Deeperforensics-1.0: a large-scale dataset for real-world face forgery detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2886–2895, 2020.

[60] P. Johnston and E. Elyan. A review of digital video tampering: from simple editing to full synthesis. *Digital Investigation*, 29:67 –81, 2019.

[61] P. Johnston, E. Elyan, and C. Jayne. Video tampering localisation using features learned from authentic content. *Neural Computing and Applications*, 2019.

[62] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[63] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 1857–1865, Sydney, NSW, Australia. JMLR.org, 2017.

[64] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[65] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[66] D. P. Kingma and M. Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[67] D. P. Kingma and P. Dhariwal. Glow: generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10215–10224. Curran Associates, Inc., 2018.

[68] A. Krizhevsky, I. Sutskever, and G. 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.

[69] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. Kingma. Videoflow: A conditional flow-based model for stochastic video generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[70] P. Kumar, M. Vatsa, and R. Singh. Detecting face2face facial reenactment in videos. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2020.

[71] T. Kurutach, A. Tamar, G. Yang, S. J. Russell, and P. Abbeel. Learning plannable representations with causal infogan. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8733–8744. Curran Associates, Inc., 2018.

[72] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In G. Gordon, D. Dunson, and M. Dudík, editors, volume 15 of *Proceedings of Machine Learning Research*, pages 29–37, Fort Lauderdale, FL, USA. JMLR Workshop and Conference Proceedings, 2011.

[73] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: ultra-deep neural networks without residuals. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[74] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. eprint: https://doi.org/10.1162/neco.1989.1.4.541.

[75] H. Li, B. Li, S. Tan, and J. Huang. Detection of deep network generated images using disparities in color components, 2018. arXiv: 1808.07276 [cs.MM].

[76] Y. Li, M. Chang, and S. Lyu. In ictu oculi: exposing ai created fake videos by detecting eye blinking. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7, 2018.

[77] Y. Li and S. Lyu. Exposing deepfake videos by detecting face warping artifacts. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019.

[78] Y. Li, X. Yang, P. Sun, H. Qi, and S. Lyu. Celeb-df: a large-scale challenging dataset for deepfake forensics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[79] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: common objects in context, 2015. arXiv: 1405.0312 [cs.CV].

[80] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 700–708. Curran Associates, Inc., 2017.

[81] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng. Reactnet: towards precise binary neural network with generalized activation functions. *arXiv preprint arXiv:2003.03488*, 2020.

[82] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[83] S. Luo, A. Peng, H. Zeng, X. Kang, and L. Liu. Deep residual learning using data augmentation for median filtering forensics of digital images. *IEEE Access*, 7:80614–80621, 2019.

[84] P. Majumdar, A. Agarwal, R. Singh, and M. Vatsa. Evading face recognition via partial tampering of faces. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019.

[85] F. Marra, D. Gragnaniello, D. Cozzolino, and L. Verdoliva. Detection of gan-generated fake images over social networks. In *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 384–389, 2018.

[86] F. Marra, D. Gragnaniello, L. Verdoliva, and G. Poggi. Do gans leave artificial fingerprints? In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 506–511, 2019.

[87] F. Marra, C. Saltori, G. Boato, and L. Verdoliva. Incremental learning for the detection and classification of gan-generated images. In *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2019.

[88] F. Matern, C. Riess, and M. Stamminger. Exploiting visual artifacts to expose deepfakes and face manipulations. In *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pages 83–92, 2019.

[89] S. McCloskey and M. Albright. Detecting gan-generated imagery using saturation cues. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 4584–4588, 2019.

[90] H. Mo, B. Chen, and W. Luo. Fake faces identification via convolutional neural network. In *Proceedings of the 6th ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '18, 43–47, Innsbruck, Austria. Association for Computing Machinery, 2018.

[91] V. Naidu, A. Narayanan, and M. Mohanty. Using amino acids of images for identifying pornographic images. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 9–14, 2019.

[92] L. Nataraj, T. M. Mohammed, B. Manjunath, S. Chandrasekaran, A. Flenner, J. H. Bappy, and A. K. Roy-Chowdhury. Detecting gan generated fake images using co-occurrence matrices. *Electronic Imaging*, 2019(5):532–1–532–6, Media Watermarking, Security, and Forensics, 2019.

[93] H. H. Nguyen, J. Yamagishi, and I. Echizen. Capsule-forensics: using capsule networks to detect forged images and videos. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[94] H. H. Nguyen, F. Fang, J. Yamagishi, and I. Echizen. Multi-task learning for detecting and segmenting manipulated facial images and videos, 2019. arXiv: 1906.06876 [cs.CV].

[95] H. H. Nguyen, T. N.-D. Tieu, H.-Q. Nguyen-Son, V. Nozick, J. Yamagishi, and I. Echizen. Modular convolutional neural network for discriminating between computer-generated images and photographic images. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, Hamburg, Germany. Association for Computing Machinery, 2018.

[96] H. H. Nguyen, J. Yamagishi, and I. Echizen. Use of a capsule network to detect fake images and videos, 2019. arXiv: 1910.12467 [cs.CV].

[97] L. Nixon, D. Fischl, and A. Scharl. *Video Verification in the Fake News Era*. V. Mezaris, L. Nixon, S. Papadopoulos, and D. Teyssou, editors. Springer International Publishing, Cham, 2019.

[98] B. Ozenne, F. Subtil, and D. Maucort-Boulch. The precision–recall curve overcame the optimism of the receiver operating characteristic curve in rare diseases. *Journal of Clinical Epidemiology*, 68(8):855 – 859, 2015.

[99] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran. Image transformer. In J. Dy and A. Krause, editors, volume 80 of *Proceedings of Machine Learning Research*, pages 4055–4064, Stockholmsmässan, Stockholm Sweden. PMLR, 2018.

[100] J. Patterson and A. Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly Media, 2017.

[101] R. Raghavendra, K. B. Raja, S. Venkatesh, and C. Busch. Transferable deep-cnn features for detecting digital and print-scanned morphed face images. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1822–1830, 2017.

[102] M. A. Rahman and Y. Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, F. Porikli, S. Skaff, A. Entezari, J. Min, D. Iwai, A. Sadagic, C. Scheidegger, and T. Isenberg, editors, *Advances in Visual Computing*, pages 234–244, Cham. Springer International Publishing, 2016.

[103] N. Rahmouni, V. Nozick, J. Yamagishi, and I. Echizen. Distinguishing computer graphics from natural images using convolution neural networks. In *2017 IEEE Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2017.

[104] A. Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Niessner. Faceforensics++: learning to detect manipulated facial images. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[105] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533 –536, 1986.

[106] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211 –252, 2015.

[107] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner. Faceforensics: a large-scale video dataset for forgery detection in human faces, 2018. arXiv: 1803.09179 [cs.CV].

[108] E. Sabir, J. Cheng, A. Jaiswal, W. AbdAlmageed, I. Masi, and P. Natarajan. Recurrent convolutional strategies for face manipulation detection in videos. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019.

[109] M. Saito, E. Matsumoto, and S. Saito. Temporal generative adversarial nets with singular value clipping. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[110] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[111] M. E. Schuckers. *Computational Methods in Biometric Authentication: Statistical Methods for Performance Evaluation*. Springer London, London, 2010.

[112] B. Shi and S. S. Iyengar. *Mathematical Theories of Machine Learning - Theory and Applications*. Springer International Publishing, Cham, 2020, pages 3–11.

[113] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[114] S. J. Sohrawardi, A. Chintha, B. Thai, S. Seng, A. Hickerson, R. Ptucha, and M. Wright. Poster: towards robust open-world detection of deepfakes. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, 2613–2615, London, United Kingdom. Association for Computing Machinery, 2019.

[115] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[116] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2377–2385. Curran Associates, Inc., 2015.

[117] H. H. Sultan, N. M. Salem, and W. Al-Atabany. Multi-classification of brain tumor images using deep neural network. *IEEE Access*, 7:69215–69225, 2019.

[118] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, 4278–4284, San Francisco, California, USA. AAAI Press, 2017.

[119] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[120] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[121] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[122] M. Tan and Q. Le. EfficientNet: rethinking model scaling for convolutional neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA, 2019.

[123] S. Tariq, S. Lee, H. Kim, Y. Shin, and S. S. Woo. Detecting both machine and human created fake face images in the wild. In *Proceedings of the 2nd International Workshop on Multimedia Privacy and Security*, MPS '18, 81–87, Toronto, Canada. Association for Computing Machinery, 2018.

[124] S. Tariq, S. Lee, H. Kim, Y. Shin, and S. S. Woo. Gan is a friend or foe? a framework to detect various fake face images. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, 1296–1303, Limassol, Cyprus. Association for Computing Machinery, 2019.

[125] J. Thies, M. Zollhofer, M. Stamminger, C. Theobalt, and M. Niessner. Face2face: real-time face capture and reenactment of rgb videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[126] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz. Mocogan: decomposing motion and content for video generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[127] A. van den Oord, N. Kalchbrenner, L. Espeholt, k. kavukcuoglu, O. Vinyals, and A. Graves. Conditional image generation with pixelcnn decoders. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4790–4798. Curran Associates, Inc., 2016.

[128] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, 1747–1756, New York, NY, USA. JMLR.org, 2016.

[129] L. Verdoliva. Media forensics and deepfakes: an overview, 2020. arXiv: 2001.06564 [cs.CV].

[130] B. V. Wee and D. Banister. How to write a literature review paper? *Transport Reviews*, 36(2):278–288, 2016. eprint: https://doi.org/10.1080/01441647.2015.1065456.

[131] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

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

[133] X. Yang, Y. Li, H. Qi, and S. Lyu. Exposing gan-synthesized faces using landmark locations. In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec'19, 113–118, Paris, France. Association for Computing Machinery, 2019.

[134] N. Yu, L. S. Davis, and M. Fritz. Attributing fake images to gans: learning and analyzing gan fingerprints. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[135] S. Zagoruyko and N. Komodakis. Wide residual networks. In E. R. H. Richard C. Wilson and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, 2016.

[136] M. Zampoglou, F. Markatopoulou, G. Mercier, D. Touska, E. Apostolidis, S. Papadopoulos, R. Cozien, I. Patras, V. Mezaris, and I. Kompatsiaris. Detecting tampered videos with multimedia forensics and deep learning. In I. Kompatsiaris, B. Huet, V. Mezaris, C. Gurrin, W.-H. Cheng, and S. Vrochidis, editors, *MultiMedia Modeling*, pages 374–386, Cham. Springer International Publishing, 2019.

[137] Z. Zhang and Q. Liu. Detect video forgery by performing transfer learning on deep neural network. In Y. Liu, L. Wang, L. Zhao, and Z. Yu, editors, *Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery*, pages 415–422, Cham. Springer International Publishing, 2020.

[138] Y. Zheng, Y. Cao, and C. Chang. A puf-based data-device hash for tampered image detection and source camera identification. *IEEE Transactions on Information Forensics and Security*, 15:620–634, 2020.

[139] P. Zhou, X. Han, V. I. Morariu, and L. S. Davis. Two-stream neural networks for tampered face detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1831–1839, 2017.

[140] P. Zhou, X. Han, V. I. Morariu, and L. S. Davis. Learning rich features for image manipulation detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1053–1061, 2018.

[141] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[142] F. Zou, L. Shen, Z. Jie, W. Zhang, and W. Liu. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[143] M. Đorđević, M. Milivojević, and A. Gavrovska. Deepfake video analysis using sift features. In *2019 27th Telecommunications Forum (TELFOR)*, pages 1–4, 2019.

Index

A

Amazon Web Services

This research uses AWS (Amazon Web Services) EC2-instances. To choose the optimal trade-off between time and money, the model was tested on several instances to calculate which would be most desirable. The first two runs were carried out using part of the FaceForensics++ dataset: 14,148 real images (13 GB), and 15,844 fake images (12 GB). The fake images are frames from 30 different videos. All frames are of compression level c23. The runs were one epoch with batch size 100; the results are listed in Table A.1. No preprocessing was used.

Table A.1: Run times for two different instances using original FaceForensics++ data.

Instance	**Time per epoch (seconds)**	**Price (dollars per hour)**	**GPU (NVIDIA)**	**RAM (GB)**	**EBS (Gbps)**
g4dn.xlarge	787	0.615	T4 (1)	16	3.5
g4dn.12xlarge	638	4.574	T4 (4)	192	7

Since the difference in price did not weigh up to the difference in run time, all images were cropped using `autocrop`. The size of the cropped images was 400×400. Frames `autocrop` failed to crop were stored in a separate folder and not used for training. This left 13,128 (1.9GB) real images, and 15,327 (2.2GB) fake images.

The cropped versions of the first frames of each fake video can be seen in B.4. The runs were one epoch with batch size 100; the results are listed in Table A.2.

Table A.2: Run times for two different instances using cropped FaceForensics++ data.

Instance	**Time per epoch (seconds)**	**Price (dollars per hour)**	**GPU (NVIDIA)**	**RAM (GB)**	**EBS (Gbps)**
g4dn.xlarge	334	0.615	T4 (1)	16	3.5
g4dn.12xlarge	204	4.574	T4 (4)	192	7
p3.2xlarge	323	3.06	Tesla V100 (1)	61	1.5
t3.xlarge	411	0.1888	-	16	5

Since the price did still not weigh up to the difference in run time, we chose to work without GPU. However, it is possible that GPUs would have been preferable to use with deeper networks; however, we did not change instances for them, so we were able to compare run times.

B

Deep Learning: Additional Theory and Presentation of Data

B.1. Performance Measures

B.1.1. ROC-curve

The ROC-curve (Receiver Operating Characteristics) plots the sensitivity on the vertical axis against fall-out on the horizontal axis [32]. Some sources also calculate the ROC by plotting $1 - \text{miss rate}$ on the vertical axis against fall-out on the horizontal axis [111]; indeed, this results in the same curve (see Appendix C.2). The score associated with these plots is the AUC (Area Under Curve); the higher the AUC is, the better the algorithm's performance [32].

A discrete classification algorithm, which returns only a binary label for a sample, results in a single point on a ROC-curve [34]. A perfect ROC-score would be (0,1): no fall-out, and maximum sensitivity. An ROC-score of (0,0) indicates a classifier labelling all samples as negative; an ROC-score of (1,1) indicates a classifier labelling all samples as positive [34]. Any score on the line segment $x = y$ corresponds to a random classifier: one that has no information about the data it is classifying [34]. Any score below the line segment $x = y$ indicates a classifier worse than random; any score above the line segment $x = y$ indicates a classifier better than random [34]. Scores in a neighbourhood of (0,1) indicate a classifier that requires high certainty to grant positive labels, resulting in little false positives, but in many false negatives; scores in a neighbourhood of (1,1) require little certainty to grant positive labels, resulting in little false negatives, but many false positives [34].

The reason the ROC-curve of a Deepfake detection algorithm contains more than one point is that the ROC-score is computed for different *thresholds* [34]. This means there are many points on the curve, corresponding to different levels of required certainty for labelling a sample positive. If false negatives are much worse than false positives, we might be content with the algorithm labelling a sample positive if it is at least 40% sure the sample is real. Similarly, if false positives are much worse than false negatives, we might require the algorithm to be at least 60% sure a sample is real before granting a positive label. For all these different percentages, a point is added to the plot; together, they make up the ROC-curve.

B.1.2. PR-curve

The Precision-Recall curve (PR-curve) plots precision on the vertical axis against recall on the horizontal axis [41]. It can be used when the performance measurement should not be influenced by true negatives [98], for example, when a dataset is imbalanced [17]. Drawing a PR-curve happens the same way as drawing an ROC-curve does [17]. Similarly to ROCs, the PR-score can be measured by the AUC[98]. The PR-curve of a model without any knowledge about the data will, contrary to the AUC of an ROC, not be 0.5; it depends on the data distribution [98]. If there are little true positives in the data, the AUC of a random PR-curve will be near 0 [98]. Contrary to ROC-curves, PR-curves are not necessarily increasing over the threshold[98].

The area under the PR-curve is also called the *average precision*[1].

[1] See https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.

B.1.3. Equal Error Rate

The Equal Error Rate (EER) is the point on the ROC-curve where the fall-out is equal to the miss rate [57]. Consequently, the lower the EER, the better the algorithm's performance [57]. In Raghavendra et al. [101], to specify the neural network solves a detection problem, the term D-EER (Detection Equal Error Rate) is used.

B.1.4. FRR@FAR10%

The FRR@FAR10% stands for False Rejection Rate at False Acceptance Rate 10%, as can be deducted from Table 2.2. Like the ROC-curve, this value can be found by changing the threshold. FRR@FAR10% is the false rejection rate when the threshold is chosen such that the false acceptance rate is 10% [57]. Similarly, FRR@FAR1% is the false rejection rate when the threshold is chosen such that the false acceptance rate is 1% [57].

B.1.5. GAR@FAR1%

The Genuine Acceptance Rate at False Acceptance Rate 1% is the Genuine Acceptance Rate at the threshold for which the false acceptance rate is 1% [84]; similarly for other percentages. The higher the value of this measure, the better the algorithm performs.

B.1.6. Half Total Error Rate

The Half Total Error Rate (HTER) is defined as $\frac{FRR+FAR}{2}$ [93]. It is the average of the false rejection rate and the false acceptance rate, or, as the title suggests, half of the total amount of misclassified samples. Hence, the lower the value is, the better the algorithm's performance is.

B.1.7. Intersection over Union

The Intersection over Union (IoU or IOU) is used to measure the performance of object category segmentation [102]. Hence, it can be used, e.g., for splicing detection. The IoU then expresses the degree to which the detected area agrees with the actual spliced area: $IoU = \frac{\text{intersection of detected and actual area}}{\text{union of detected and actual area}}$ [102]. The higher the IoU, the better the algorithm's performance: an IoU of 0 means the detected area has zero pixels in common with the actual area; an IoU of 1 means the detected area is exactly the same as the actual area.

In Huh et al. [55], class-balanced IOU (cIOU) is used. This measure uses an IoU per individual class [55].

B.1.8. Mean Average Precision

Mean Average Precision (mAP or MAP) is used by several sources in this research. Depending on the source, there are different definitions for mAP (and AP)[2]. Originally, the AP is calculated by setting the IoU-threshold to 0.5 and averaging the IoUs over all images in the dataset (per object class). For COCO (Common Objects in Context) [79], an object recognition challenge, AP is defined as the average of IoUs for thresholds increasing from 0.5 to 0.95 (with stepsize 0.05).

Similarly, traditionally mAP is the average of APs over all categories. However, COCO has eliminated this distinction and simply calls this measure AP instead of mAP.

In [136], MP@20 is reported; this is the Mean Precision for the 20 best-classified samples.

B.1.9. Matthews Correlation Coefficient

Matthews Correlation Coefficient (MCC) is defined as $MCC = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$ [55]. As the name suggests, it returns the correlation between the predicted and actual labels[3]. Its values range from -1 for a classifier that is always wrong, to 1 for a classifier that is always right; an MCC-score of 0.5 implies a random classifier. As can be seen from the formula, the MCC-measure weighs the positive as heavily as the negative class.

B.2. Robustness against compression

An example of a claim in Deepfake detection literature that seems generally accepted, but not very much confirmed, is about generalisation against robustness. This claim states that deep networks are more robust

[2] See cocodataset.org/#detection-eval.

[3] See https://towardsdatascience.com/the-best-classification-metric-youve-never-heard-of-the-matthews-correlation-coefficient-3

Table B.1: Depths of different backbone CNNs. Depths are taken to be the amount of weighted layers the network contains. The fourth column provides details on the type of layers the networks contain.

Reference	**NetName (optional)**	**Layers (learnable)**	**Layers (details)**
Afchar [1]	Meso-4	6	4 convolutional, 2 fully connected, 1 sigmoid
	MesoInception-4	7	2 inception modules (4 stacked convolutional layers, 1 concatenation), 2 convolutional, 2 fully connected, 1 sigmoid
Bayar [12]		6	3 convolutional, 2 max-pool, 3 fully connected
Chollet [21]	Xception	36	36 convolutional, (optional: fully connected layers), 1 logistic regression
Cozzolino [24]		3	2 convolutional, 1 softmax, 1 avg-pool, 1 fully connected
He [46]	ResNet-18	18	17 convolutional, 1 max-pooling, 1 avg-pool, 1 fully connected, 1 sofmax
	ResNet-34	34	33 convolutional, 1 max-pooling, 1 avg-pool, 1 fully connected, 1 softmax
	ResNet-50	50	49 convolutional, 1 max-pooling, 1 avg-pool, 1 fully connected, 1 softmax
	ResNet-101	101	100 convolutional, 1 max-pooling, 1 avg-pool, 1 fully connected, 1 softmax
	ResNet-152	152	151 convolutional, 1 max-pooling, 1 avg-pool, 1 fully connected, 1 softmax
Huang [53]	DenseNet-121	121	120 convolutional, 4 avg-pooling, 1 max-pooling, 1 fully connected, 1 softmax
	DenseNet-161	161	160 convolutional, 4 avg-pooling, 1 max-pooling, 1 fully connected, 1 softmax
	DenseNet-169	169	168 convolutional, 4 avg-pooling, 1 max-pooling, 1 fully connected, 1 softmax
	DenseNet-201	201	200 convolutional, 4 avg-pooling, 1 max-pooling, 1 fully connected, 1 softmax
Krizhevsky [68]	AlexNet	9	6 convolutional, 3 max-pooling, 3 fully connected, 1 softmax
Nguyen [93]	Capsule-Forensics	11	6 convolutional, 3 max-pooling 3 primary capsules, 1 output capsule
Nguyen [95]		10	6 convolutional, 2 fully connected, 1 softmax, 3 modules
Raghavendra [101]			Uses AlexNet and VGG19 as twin networks
Rahmouni [103]	Stats-1	2	1 convolutional, 1 pooling, 1 fully connected, 1 softmax
	Stats-2L	3	2 convolutional, 1 pooling, 1 fully connected, 1 softmax
	Stats-2S	3	2 convolutional, 1 pooling, 1 fully connected, 1 softmax
	Stats-3	4	3 convolutional, 1 pooling, 1 fully connected, 1 softmax
Simonyan [113]	VGG11	11	8 convolutional, 5 max-pooling, 3 fully connected, 1 soft-max
	VGG13	13	10 convolutional, 5 max-pooling, 3 fully connected, 1 soft-max
	VGG16	16	13 convolutional, 5 max-pooling, 3 fully connected, 1 soft-max
	VGG19	19	16 convolutional, 5 max-pooling, 3 fully connected, 1 soft-max
Szegedy [120]	Inception-v3	57	6 convolutional, 2 pooling, 10 inception modules, 1 fully connected, 1 softmax
Zhou [139]			Two-stream: Inception-v3 and a 2-layer fully connected network

against compression than shallow networks. In this appendix, we will collect what literature says about these claims and what evidence they provide to support it. Before we can do this, we need to count the depths of the networks under consideration.

However, different papers count layers differently. For example, Afchar et al. [1] call one of their networks Meso-4, which has four convolutional layers, but seven layers in total (an additional two fully connected layers, and one sigmoid layer); He et al. [46] call one of their networks ResNet-50, which has fifty weighted (convolutional/fully connected) layers, but also two pooling layers; Bayar et al. [12] say their network contains eight layers, which is the total of weighted and unweighted layers together. Table B.1 states the number of weighted/learnable layers, since this way of counting is most consistent with the names networks have, and therefore seems the least confusing approach. Since this way of counting does not agree with all of literature, the types of layers contained in each network are also listed, to elaborate why there might be discrepancies.

Five of the considered articles provided data on the robustness networks have against compression. Of these five, Kumar et al. [70], Marra et al. [85], Rössler et al. [104], and Rössler et al. [107] conclude that deep networks are more robust than shallow networks; Nguyen et al. [93] does not make any claims on a relation between network depth and robustness against compression. For each article, this section provides a table with calculated accuracy drops. Tables B.2, B.3, B.4, and B.5 contain one row with the difference between accuracy for uncompressed data and accuracy for easy compressed data (compression ratio 23); and one row with the difference between accuracy for uncompressed data and accuracy for hard compressed data (compression ratio 40). Table B.6 contains the difference between accuracy for uncompressed data and 'twitter-compressed' data. Table B.1 reports the depths of all considered convolutional networks. This section is concluded by our analysis of these data.

Kumar et al. [70] compare MesoNet [1], Bayar [12], Zhou [139], Raghavendra [101], and Xception [21]. They refer to MesoNet and Bayar as shallow; to Xception and Zhou as deep. They do not categorise Raghavendra. The accuracy drop calculated from Kumar's data can be seen in Table B.2.

Nguyen et al. [93] compare Cozzolino [24], Bayar [12], Stats-2L [103], Raghavendra [101], Zhou [139], and Xception [21] (up until this point with exactly the same results as [107]), MesoNet [1], MesoInception [1],

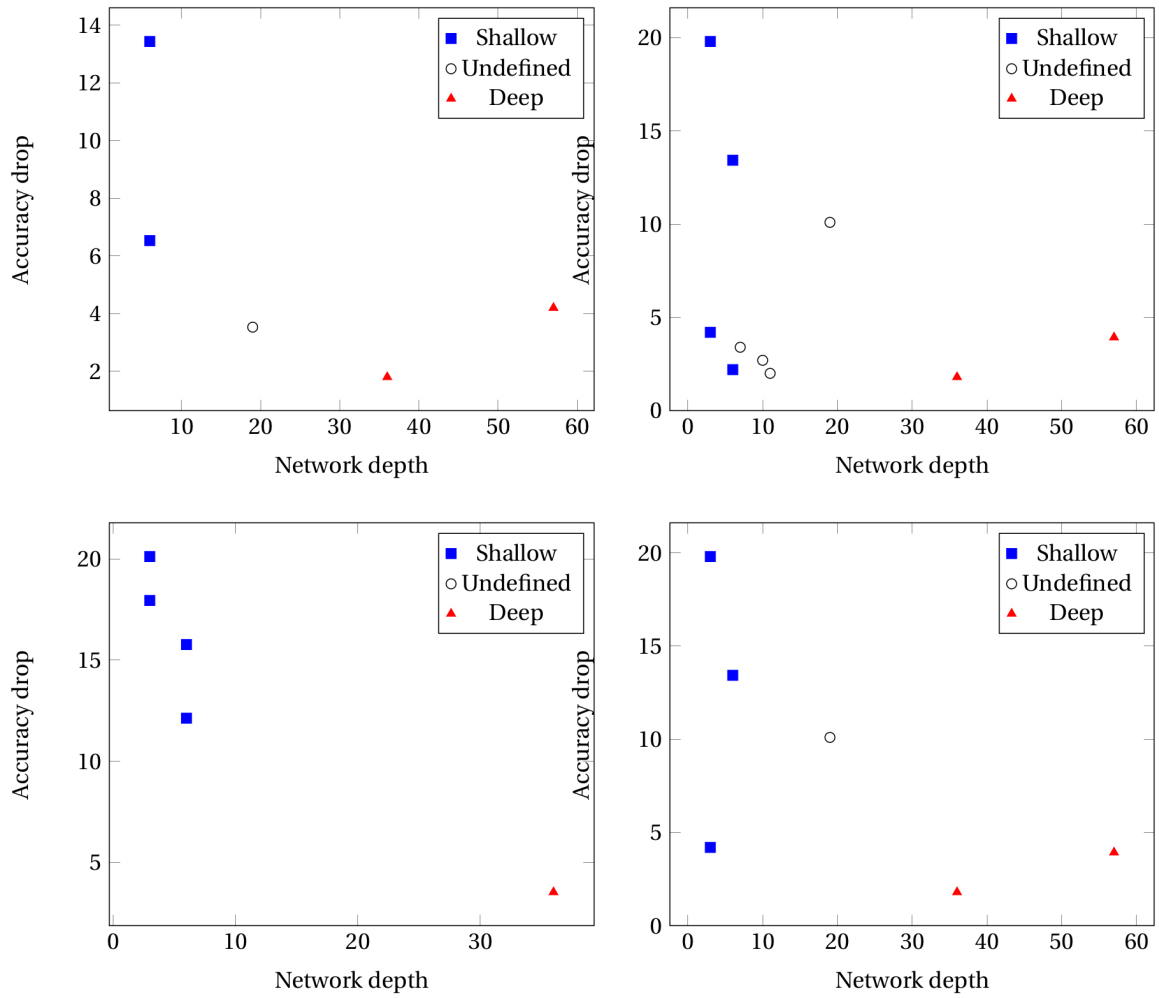


Figure B.1: Plot of network depths versus the drop in performance with easy compression as reported by different papers. Starting at the top left, going clockwise, the results are reported by Kumar et al. [70], Nguyen et al. [93], Rössler et al. [104], and Rössler et al. [107].

Nguyen [93], and Nguyen [95]. The accuracy drop calculated from Nguyen's data can be seen in Table B.3.

Rössler et al. [104] perform a similar test with a different dataset. They do not use the two-stream or twin networks. The accuracy drop calculated from Rössler's data can be seen in Table B.4.

Rössler et al. [107] compare Cozzolino [24], Bayar [12], Stats-2L [103], Raghavendra [101], Zhou [139], and Xception [21]. They refer to Bayar and Cozzolino as shallow; to Xception and Zhou as deep. The accuracy drop calculated from Rössler's data can be seen in Table B.5.

Fig. B.1 and Fig. B.2 visualise the accuracy loss versus the network depths for easy and hard compression respectively.

Marra et al. [85] compare three shallow networks (Cozzolino [24], Bayar [12], Stats-2L [103]) and three deep networks (DenseNet [53], Inception-v3 [120], Xception [21]). The accuracy drop calculated from Marra's data can be seen in Table B.6; the accuracy loss versus the network depths are plotted in Fig. B.3.

There are a few problems we have with the conclusion the four articles draw based on these data:

- Kumar's data for easy compression show that one of the shallow datapoints (Bayar) has a relatively large accuracy drop, whereas the other shallow network (MesoNet, which has exactly the same amount of weighted layers) has less than half its accuracy drop. The network with the least accuracy drop is a deep one (Xception), but a network with twice its depth (Zhou) has twice the accuracy drop.

- Only Rössler et al. [104]'s data seem to suggest a monotonically decreasing relation between network depth and accuracy drop (although they only consider one deep network). Since the other articles provide no more than ten data points, it is difficult to say what the relation between accuracy and

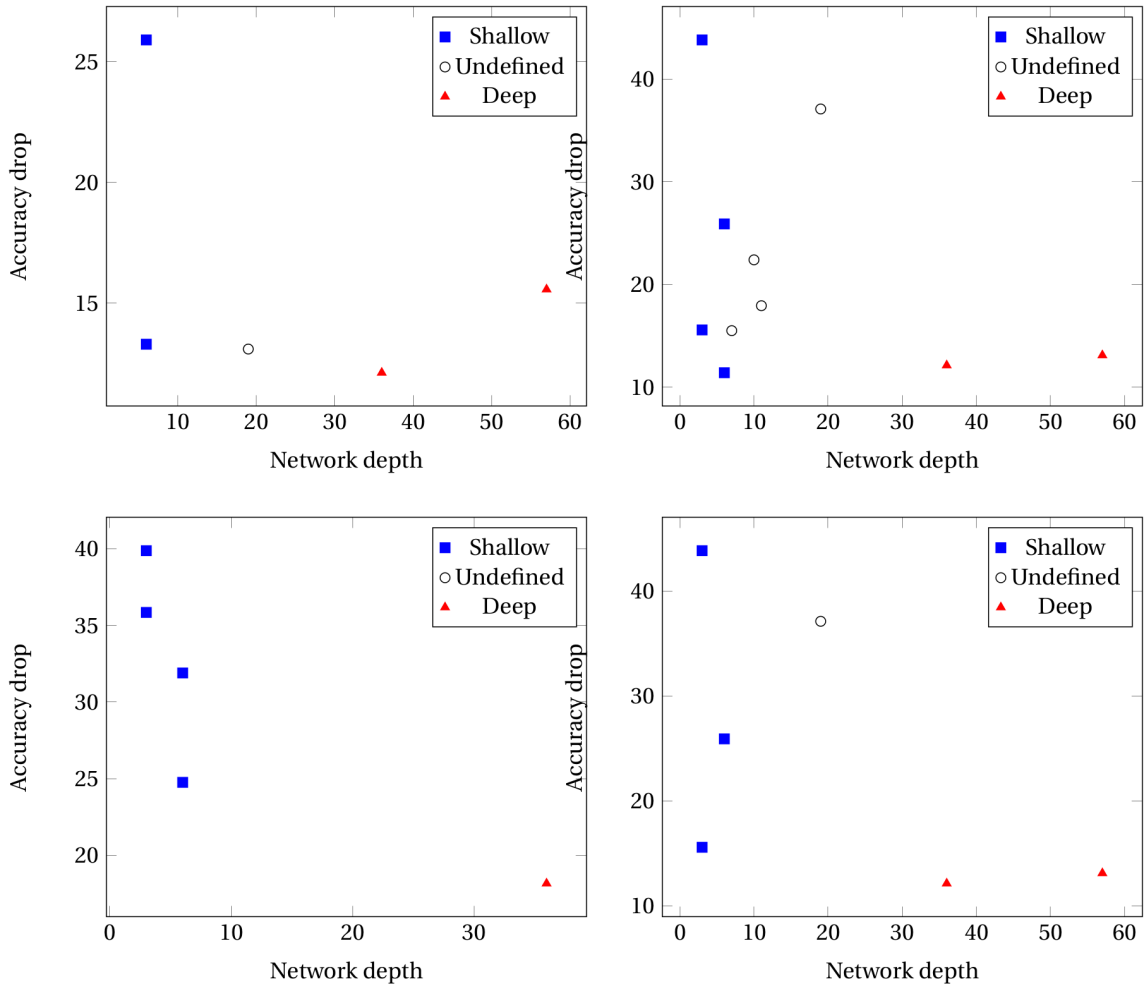


Figure B.2: Plot of network depths versus the drop in performance with easy compression as reported by different papers. Starting at the top left, going clockwise, the results are reported by Kumar et al. [70], Nguyen et al. [93], Rössler et al. [104], and Rössler et al. [107].

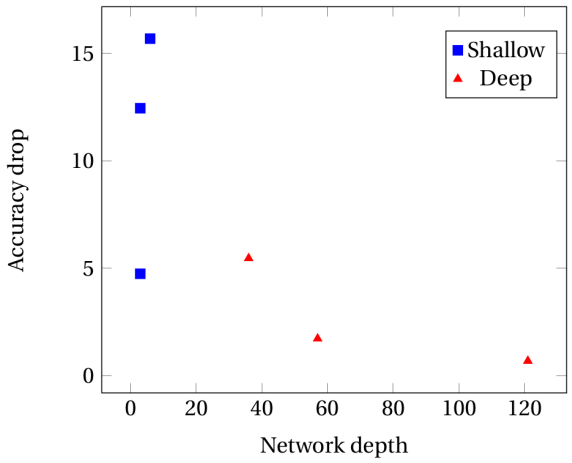


Figure B.3: Plot of network depths versus the drop in performance with easy compression as reported by Marra et al. [85].

network depth is (if there is any), because it is not possible to determine what data points should be considered proper measurements and which measurement errors.

- In Nguyen's data for easy compression, Stats-2L, Zhou, MesoNet, MesoInception, Nguyen [93, 95] all

Table B.2: The performance drop reported by Kumar [70]. The reported values are the differences between the accuracy (accuracy measured in percentages) with different compression levels.

CNN	**Accuracy$_{\text{no compression}}$ − accuracy$_{\text{easy compression}}$**	**Accuracy$_{\text{no compression}}$ − accuracy$_{\text{high compression}}$**
MesoNet [1]	6.53	13.3
Bayar [12]	13.43	25.9
Zhou [139]	3.53	13.1
Raghavendra [101]	4.2	15.57
Xception [21]	1.8	12.12

Table B.3: The performance drop reported by Nguyen et al. [93]. The reported values are the differences between the accuracy (accuracy measured in percentages) with different compression levels.

CNN	**Accuracy$_{\text{no compression}}$ − accuracy$_{\text{easy compression}}$**	**Accuracy$_{\text{no compression}}$ − accuracy$_{\text{high compression}}$**
Cozzolino [24]	19.8	43.83
Bayar [12]	13.43	25.9
Raghavendra [101]	10.1	37.1
Rahmouni [103]	4.2	15.57
Zhou [139]	3.93	13.1
Meso-4 [1]	2.20	11.40
MesoInception-4 [1]	3.40	15.50
Nguyen [95]	2.70	22.40
Nguyen [93]	2.00	17.93

have an accuracy loss of less than 4.5%, although their depths range from 3 to 57 weighted layers. Nor does the shallowest network have the highest accuracy drop, or the deepest network the lowest accuracy drop.

- Although in Marra's data, the highest accuracy drops occur for shallow networks (Bayar, Cozzolino) and the lowest accuracy drops appear for deep networks (DenseNet, Inception-v3), the 36-layered Xception has higher accuracy drop than the 3-layered Stats-2L.

- Not all networks are simply built up of convolutional layers, where the output of layer $n-1$ is the (sole) input to layer n. Some networks have capsules or modules in them; some networks even have two streams alongside each other and combine the output of those streams to come to a final decision on an image's integrity. Since Rössler et al. [104] are the only article whose results imply that deep networks always outperform shallow networks, and Rössler et al. [104] are also the only one that does not consider two-stream or twin networks, these differences in architecture can be argued to influence results.

- Rössler et al. [104, 107]'s data for hard compression compare four overlapping networks: Cozzolino, Bayar, Stats-2L, and Xception. However, there is a considerable difference in accuracy drops between the two. If we take Rössler et al. [107] as the baseline, then the differences in accuracy drops for hard compression are -3.94, 6.00, 20.28, 6.04 for Cozzolino (3 layers), Bayar (6 layers), Stats-2L (3 layers),

Table B.4: The performance drop reported by Rössler et al. [104]. The reported values are the differences between the accuracy (accuracy measured in percentages) with different compression levels.

CNN	**Accuracy$_{\text{no compression}}$ − accuracy$_{\text{easy compression}}$**	**Accuracy$_{\text{no compression}}$ − accuracy$_{\text{high compression}}$**
Cozzolino [24]	20.12	39.88
Bayar [12]	15.77	31.90
Stats-2L [103]	17.95	35.85
MesoNet [1]	12.13	24.76
Xception [21]	3.53	18.16

Table B.5: The performance drop reported by Rössler et al. [107]. The reported values are the differences between the accuracy (accuracy measured in percentages) with different compression levels.

CNN	$\textbf{Accuracy}_{\textbf{no compression}}$ − $\textbf{accuracy}_{\textbf{no compression}}$	$\textbf{Accuracy}_{\textbf{easy compression}}$ − $\textbf{accuracy}_{\textbf{high compression}}$
Cozzolino [24]	19.8	43.83
Bayar [12]	13.43	25.9
Raghavendra [101]	10.1	37.1
Rahmouni [103]	4.2	15.57
Zhou [139]	3.93	13.1
Xception [21]	1.8	12.12

Table B.6: The performance drop reported by Marra et al. [85]. The reported values are the differences between the accuracy (accuracy measured in percentages) with different compression levels.

CNN	$\textbf{Accuracy}_{\textbf{no compression}}$-$\textbf{Accuracy}_{\textbf{twitter compression}}$
Cozzolino [24]	12.45
Bayar [12]	15.69
Stats-2L [103]	4.74
DenseNet [53]	0.68
Inception-v3 [120]	1.72
Xception [21]	5.46

and Xception (36 layers) respectively. Both the largest and the smallest difference are for the shallowest networks; the difference for the 6- and 36-layered networks is almost the same. The claimed relation between accuracy drop and network depth seems unlikely to be the (sole) explanation for these differences.

B.3. No Free Lunch Theorem

An import theorem for machine learning is the No Free Lunch Theorem. It was first introduced by Wolpert et al. [131].

Theorem 1 (No Free Lunch) *Suppose $\mathscr{F}$ is a set of problems and a_1, a_2 are two optimisation algorithms. Suppose a_1 outperforms a_2 on the problems $\mathscr{F}_1 \subseteq \mathscr{F}$; a_2 outperforms a_1 on the problems $\mathscr{F}_2 \subseteq \mathscr{F}$. Let m denote the amount of distinct function evaluations an algorithm has performed, and d_m^y the ordered set of all cost values y from function evaluations $1, ..., m$. Then*

$$\sum_{f \in \mathscr{F}} P(d_m^y | f, m, a_1) = \sum_{f \in \mathscr{F}} P(d_m^y | f, m, a_1).$$

Proof 1 *See [131].*

The consequences of this theorem are that, given a performance measure $\Phi(d_m^y)$, the average of $P(\Phi(d_m^y)|f, m, a)$ over all $f \in \mathscr{F}$ does not depend on a [131]. This implies that if an algorithm a increases its performance on some subset of $\mathscr{F}$, it simultaneously decreases its performance on the complement of this subset [131]. In other words, there is no universal learning algorithm that can outperform another algorithm on all problems [41]; any type of problem needs its own specialised learning algorithm.

B.4. Data

This section provides example frames of the datasets we used.

B.5. Stochastic Nature of Training

Training neural networks is a stochastic process. This means that training a network twice not result in identical accuracy and loss curves. However, in our research, we run every algorithm only once. This appendix will defend this choices, as well as explore its consequences.

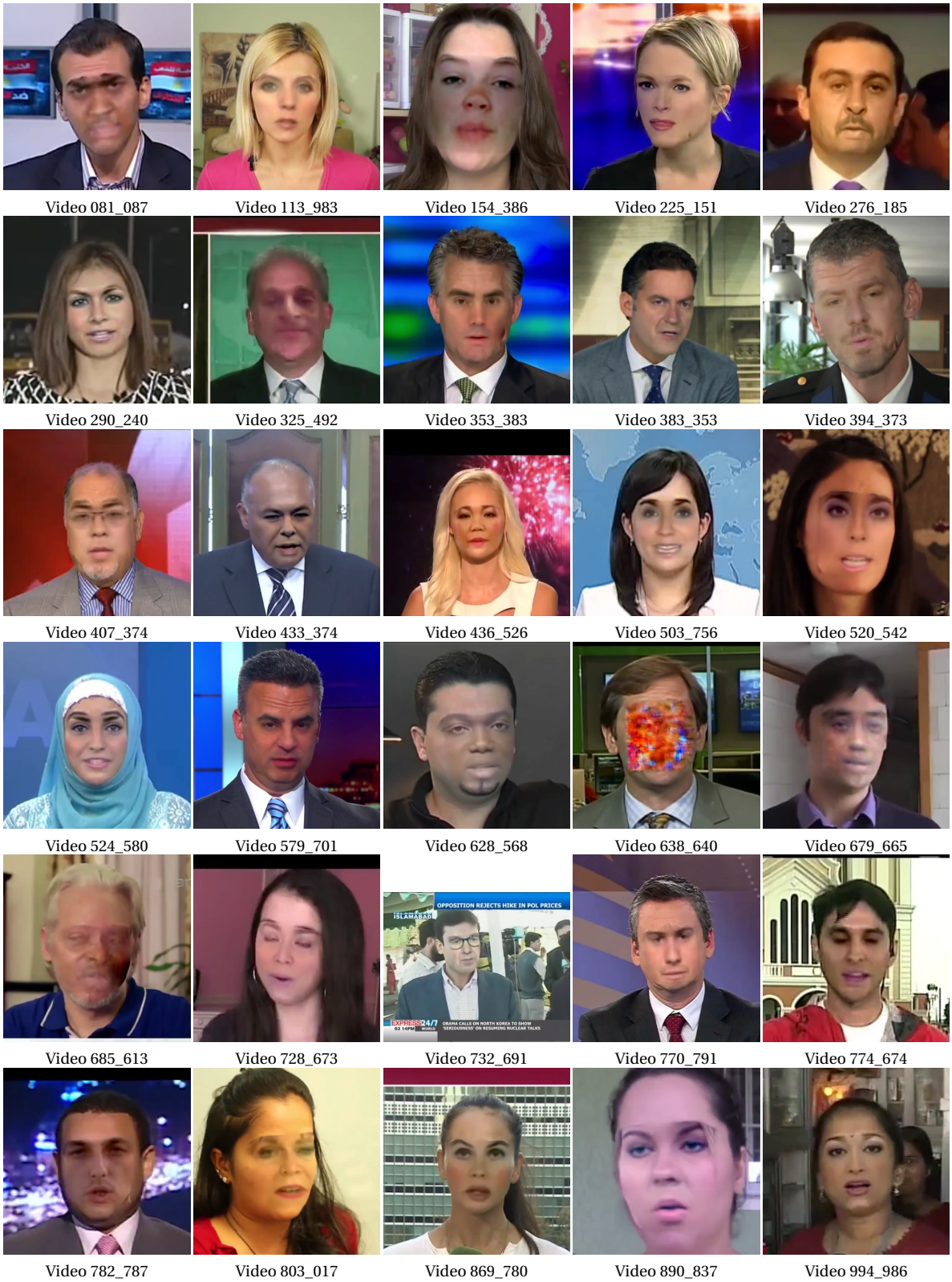


Figure B.4: The first frames of all cropped fake videos in Larger FaceForensics++.

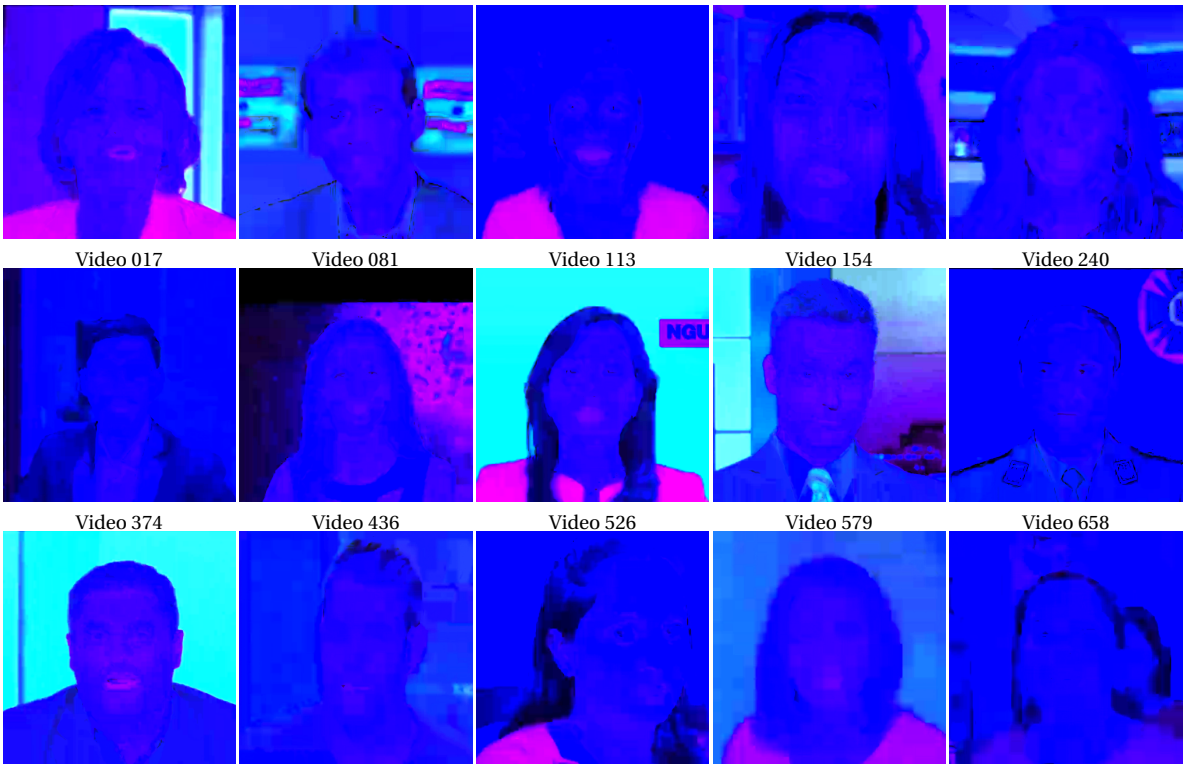


Figure B.5: Some DCT-residuals of real images in the FaceForensics++ dataset.

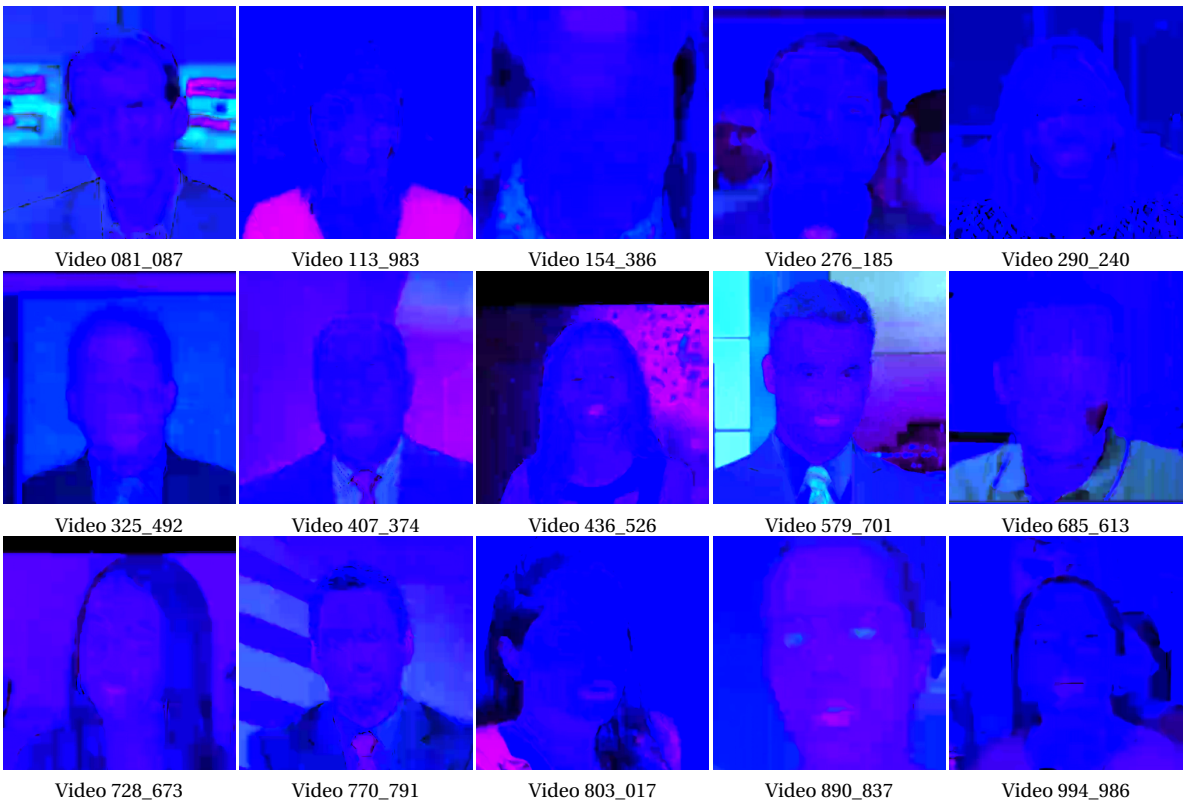


Figure B.6: Some DCT-residuals of fake images in the FaceForensics++ dataset.

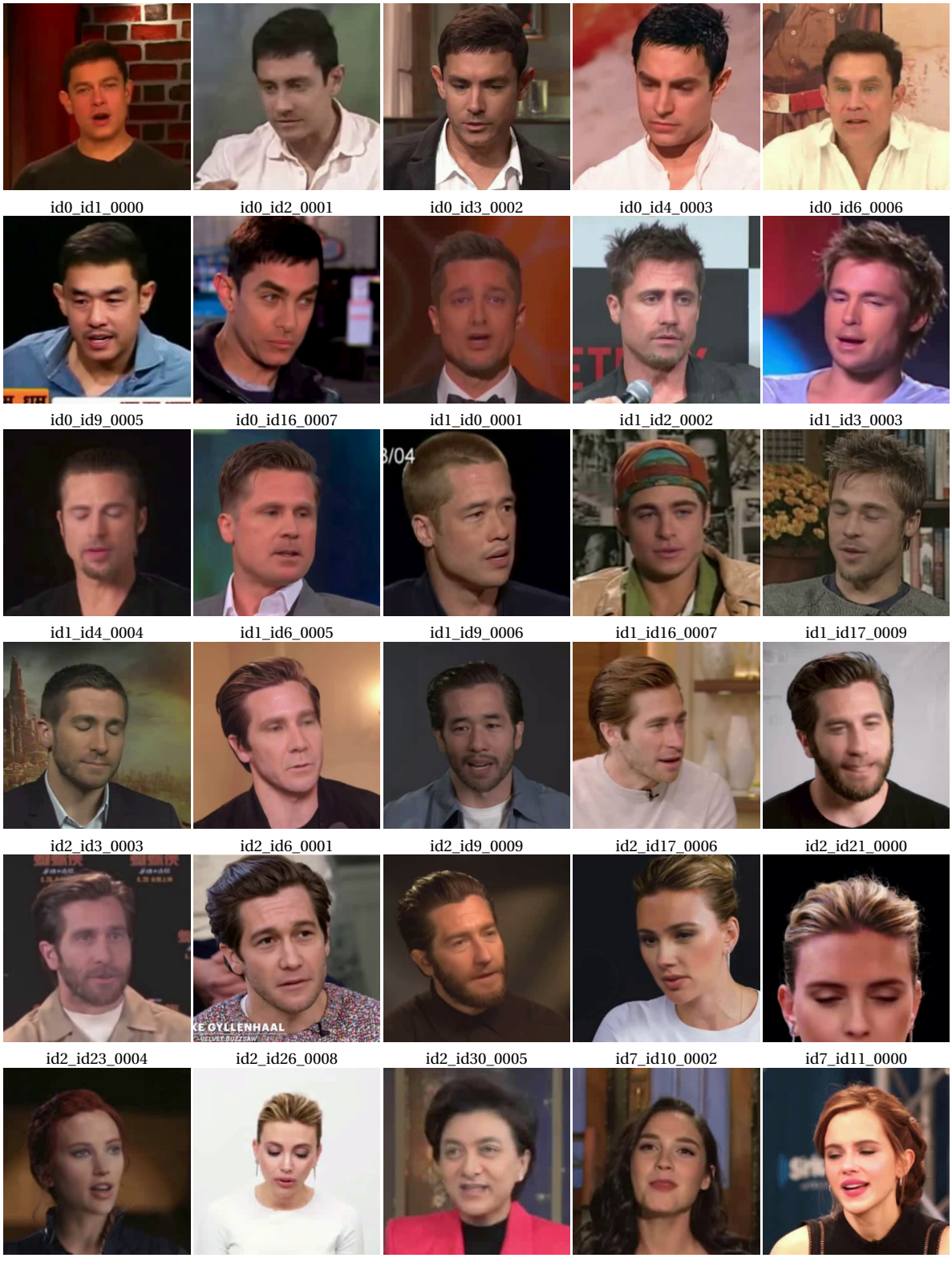


Figure B.7: Some fake frames from Celeb-df.

Figure B.8: Some real DCT-residual frames from Celeb-df.

Figure B.9: Some fake DCT-residual frames from Celeb-df.

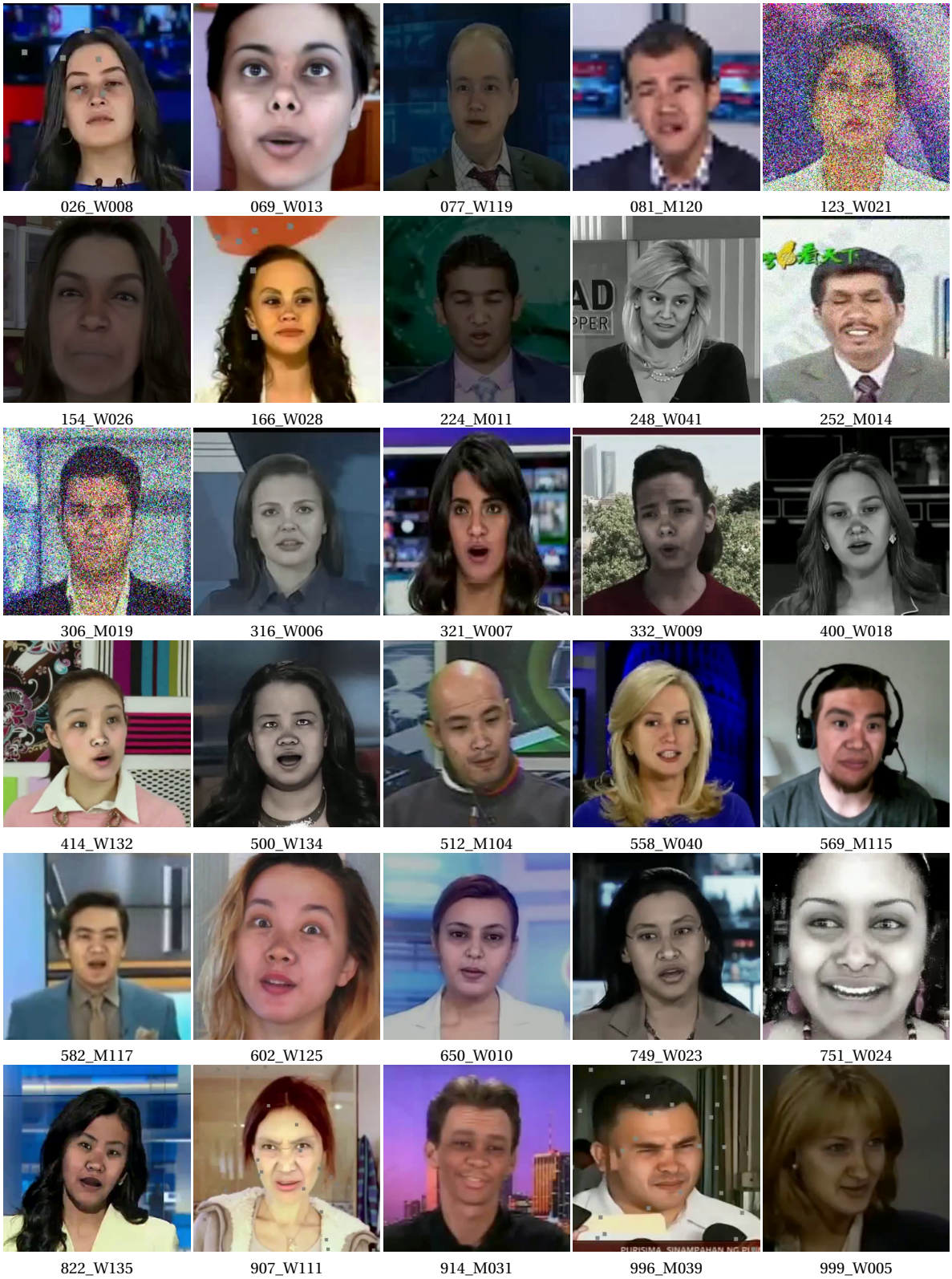


Figure B.10: Some fake frames from DeeperForensics-1.0.

Figure B.11: Some fake DCT-residual frames from DeeperForensics-1.0.

Before building our shallow networks for Deepfake detection, we built the classic introductory neural network that distinguishes between cats and dogs. During this orientational research, we ran every run three times. Due to smaller input size and fewer convolutional layers, this was feasible time-wise. However, the differences in accuracy between those three runs were never more than a few per cent. Therefore, due to time restrictions with the larger images and larger networks in our research, we decided to base our research on one run for each experiment, unless the results of that run indicated a need to be rerun. This section explores what the differences between different runs of the same algorithm might be, so we know what conclusions we can (and cannot) draw based on a single run. To this end, we trained Shallow-2fc seven times on DeeperForensics-1.0. The exact settings of the experiment are described in section 4.3.3.5. The resulting accuracy curves are shown in Fig. B.12; the loss curves in Fig. B.13; and the performance in Table B.7.

We see that it is possible for a network to underfit once, as run 4 is the only one during which the model does not manage to learn anything. This means that any run showing underfitting has to be run again, to see whether underfitting occurs structurally or incidentally. For the remainder of this appendix, we will consider run 4 an outlier and not take it into account.

Comparing the accuracy curves in Fig. B.12, we see that the steepness of all curves is similar at first. However, whereas some curves merely flatten somewhere between epoch 100-150 (run 1, 3, and 5), others seem to degrade slightly (run 2 and 7), or even severely (run 6). However, when we look at Table B.7, we see that Runs 1, 2, 3, and 7 reach their maximum accuracies between epochs 237-247. The only outliers are runs 5 and 6, reaching their maximum accuracies at epochs 204 and 155 respectively. Therefore, we will not draw conclusions on the degradation of the accuracy curves after having reached the maximum accuracy. We restrict this decision to cases where the validation and training accuracy degrade together; overfitting does not count for this decision.

The accuracies reached are all between 80.1-84.3%. This implies that differences up to 4.2% can be caused by the stochastic nature of the training process, rather than actual differences in settings, datasets or network architecture. Since we run 'only' seven runs, we will take this result with a margin, and decide that differences up to 5.0% between runs can not be used to draw conclusions.

Comparing the loss curves in Fig. B.13, we see that there are large differences in the maximum losses

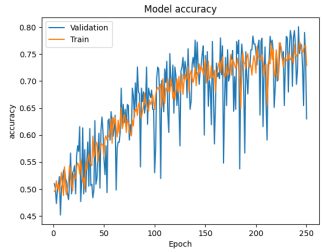
Table B.7: Performance of seven different training runs of Shallow-2fc on DeeperForensics-1.0. The maximum accuracy reported is the validation accuracy; the minimum loss reported is the validation loss. The 'Epoch'-colums report at what epochs the maximum accuracy and minimum loss were reached.

Run	Performance					
	Accuracy			Loss		
	Fig.	Max (%)	Epoch	Fig.	Min	Epoch
1	B.12a	80.1	241	B.13a	0.40	237
2	B.12b	82.6	238	B.13b	0.34	238
3	B.12c	84.3	237	B.13c	0.36	234
4	B.12d	58.7	7	B.13d	0.69	9
5	B.12e	83.0	204	B.13e	0.41	242
6	B.12f	81.9	155	B.13f	0.41	155
7	B.12g	82.9	247	B.13g	0.39	215

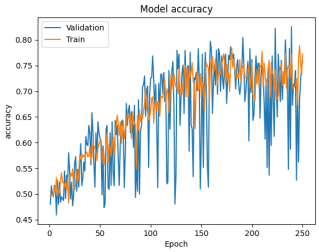
reached. Runs 1, 2, and 6 do not exceed 5; run 3, 5, and 7 reach 7.5, 12.3, and 26.4 respectively. However, these maxima are incidents. Overall, the loss curves all reduce well. Only run 6 shows that its loss starts increasing again after epoch 155. The other runs reach their minimum losses between epochs 215 and 242. The minimum losses themselves are all between 0.34 and 0.41. This means that differences in losses under 0.05 are not necessarily due to differences in settings, data, or network architecture. Otherwise, these findings support our earlier hypotheses.

We can summarise our findings as follows. When only one run has been executed, no conclusions can be drawn on account of

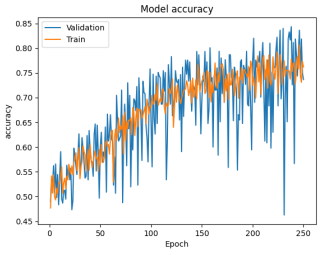
- underfitting,
- differences in accuracy smaller than 5%,
- differences in loss smaller than 0.05,
- degradation of the accuracy or loss curves after reaching the maximum accuracy or minimum loss, only if the training and validation curves degrade together.



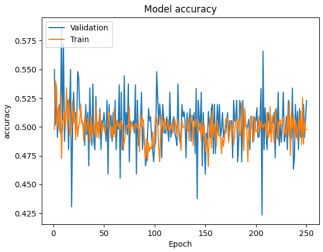
(a) Run 1.



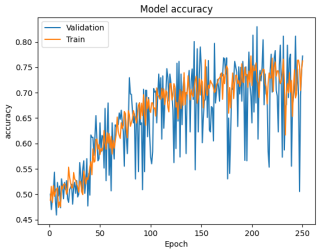
(b) Run 2.



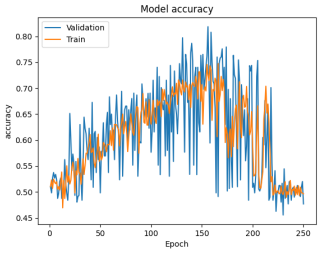
(c) Run 3.



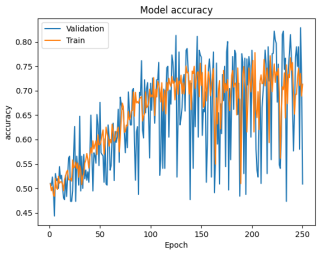
(d) Run 4.



(e) Run 5.

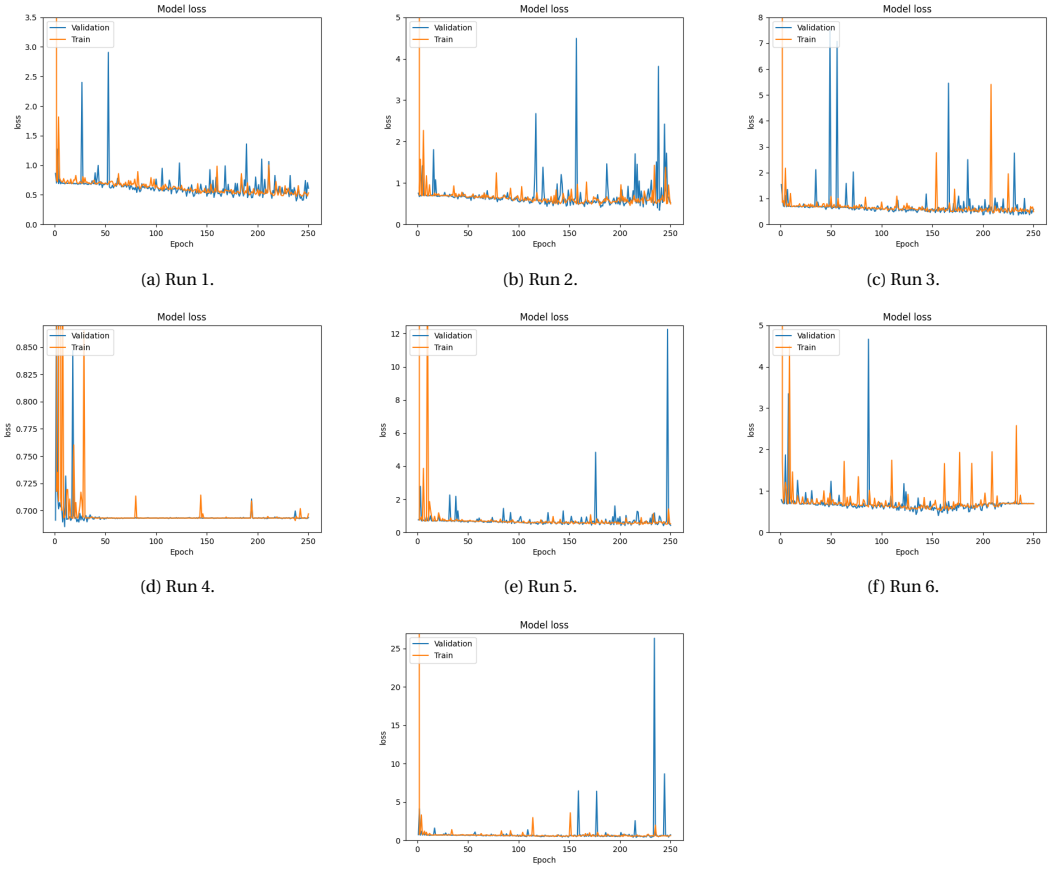


(f) Run 6.



(g) Run 7.

Figure B.12: Accuracies of seven different training runs of Shallow-1fc on DeeperForensics-1.0.



(a) Run 1.

(b) Run 2.

(c) Run 3.

(d) Run 4.

(e) Run 5.

(f) Run 6.

(g) Run 7.

Figure B.13: Losses of seven different training runs of Shallow-1fc on DeeperForensics-1.0.

C

Additional Mathematics

C.1. Norms

Given a vector $x \in \mathbb{R}^n$, the L^p norm ($p \in \mathbb{N}$), denoted $\|\cdot\|_p$, is defined as

$$\|x\|_p = \left(\sum_{i=1}^{n} |u_i|^p \right)^{\frac{1}{p}}.$$

For $p = \infty$, it is defined as

$$\|x\|_\infty = \max_{1 \le i \le n} |u_i|.$$

Given a matrix $A \in \mathbb{R}^{n \times m}$, the Frobenius norm is defined as

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}.$$

C.2. Calculations

This section provides calculations.

C.2.1. F_1-measure

This section proves that the F_1-measure is also a rate.

$$\begin{aligned}
F_1 &= \frac{2 \cdot \text{Precision} \cdot \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}} \\
&= \frac{2 \frac{TP}{TP+FP} \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}} \\
&= \frac{2 \frac{TP}{(TP+FP)(TP+FN)}}{\frac{1}{TP+FP} + \frac{1}{TP+FN}} \\
&= \frac{2 \frac{TP}{(TP+FP)(TP+FN)}}{\frac{TP+FN}{(TP+FN)(TP+FP)} + \frac{TP+FP}{(TP+FP)(TP+FN)}} \\
&= \frac{2TP}{2TP+FN+FP}
\end{aligned}$$

C.2.2. ROC-curve

This section proves that sensitivity is equal to 1 – miss rate.

$$\begin{aligned} \text{sensitivity} &= \frac{TP}{TP+FN} \\ &= \frac{TP+FN-FN}{TP+FN} \\ &= 1-\frac{FN}{TP+FN} \\ &= 1-\text{miss rate} \end{aligned}$$