Master's Thesis

Detecting Deepfakes and Forged Videos Using Deep Learning

Emil Johansson

Spring 2020

Supervisors: Mikael Nilsson & Kalle Åström (LTH)

LUND UNIVERSITY

LTH

FACULTY OF ENGINEERING

Abstract

Over just a few years, methods to manipulate videos have become so sophisticated that even someone without much expertise or computational resources can forge videos inseparable from pristine ones to the human eye. These methods can for instance insert a person in a video or manipulate their lip movements to make them say anything of the manipulator's liking. Though there exist harmless and constructive uses of these technologies, it is not hard to imagine the harm they could cause if put in the wrong hands.

This report presents a model to detect forged manipulated videos, more specifically those where faces have been manipulated. Four kinds of manipulation videos were taken into consideration: FaceSwap, DeepFakes, Face2Face and Neural Textures. The model proposed consists of a feature extraction CNN followed by an LSTM network. The FaceForensics++ dataset was used, as well as the associated benchmark. The model, though not competing with the state-of-the-art detectors, was able to classify videos with an accuracy higher than or close to that of several models in the benchmark.

Acknowledgements

First and foremost, I would like to thank my principal supervisor Mikael Nilsson for guidance throughout this thesis project. Thank you for all input and ideas, and for helping make the project work despite the corona pandemic. Thanks also to my supervisor Kalle Åström for listening to my ideas at the start of the project helping me turning them into a thesis project. Thank you Jonna Johansson and Jesper Lundberg for all input, as well as several well-needed distractions from working. Thank you Helena Johansson for proofreading and tips on the report. Finally, my sincerest thank you to all friends, family and teachers who have helped and supported me throughout the project, as well as my university education as a whole.

Contents

1. Introduction

Figure 1.1: Deepfake of Back to the Future, where Tom Holland and Robert Downey Jr has taken the places of Michael J Fox and Christopher Lloyd. Screenshot from [8].

In recent years, technologies to manipulate facial videos have reached the point where it might not be possible for a human to detect that the video is manipulated. One could insert a person into a video by pasting their face onto that of another person (see for instance [8]), or change a person's mouth movement and facial expressions to make them say whatever you wish (for example [17]). It is not hard to imagine what harm these kind of videos could cause if created with malicious intent; fake videos of politicians making outrageous statements, news anchors presenting fake news, business partners or family members asking for money et cetera. These could harm everyone, from individuals to society and democracy as a whole. Already there exist reports of this happening; Indian journalist Rana Ayyub discovered a fake pornographic video of her circulating on social media. In Huffington Post she describes how devastating the video had been to her, both emotionally and to her career and everyday life [4]. The video had been used with the intention of quieting her.

It should thus come to no surprise that these videos, sometimes collectively

referred to as "deepfakes", and the task to detect them has received large attention recently. Many models and algorithms have been implemented (some of them mentioned under "Related Work" below), and companies like Google and Facebook are investing large amounts of resources to the task [7, 24]. This project aspires to contribute to this.

In this report, a neural network model which is trained to classify videos as pristine or manipulated will be presented. Four different techniques to manipulate facial videos will be taken into consideration: FaceSwap, Deepfakes, Face2Face and Neural Textures. The model includes a feature extraction network, where several publicly available networks will be tested: GoogleNet, Xception and DenseNet, as well as all three networks together. The data and benchmark provided by Rössler et al. [21, 22] is used to train and evaluate the model. An introduction to neural networks is given in chapter 2, to the facial video manipulation methods in chapter 3 and the feature extraction networks in section 4.2.

1.1 Related Work

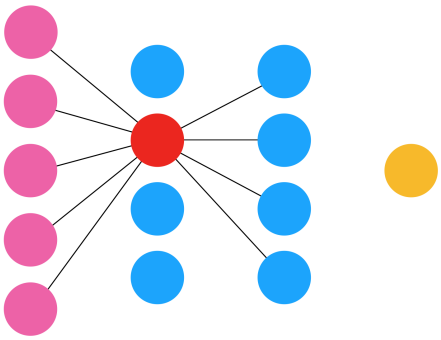
As mentioned above, the interest in both creating and detecting manipulated facial videos has largely increased, for which reason covering all recent advances in the fields here would be near impossible. Instead only a few highlights will be covered. On the generating side, the work most relevant to this work are of course the methods used to create the dataset which is used to train and evaluate the proposed model; these are covered in detail in chapter 3. Other methods have been introduced by Nirkin et al. [19], who take an RNN-based approach for both face swapping and reenactment, and Siarohin et al. [25], who animate single images (of faces in particular) using a driving video. Audio can also be used to drive facial reenactment; this is done by for instance Thies et al. [29] and Yi et al. [37]. Advances has also been made in facial image synthesis, for instance by Karras et al. [12, 13].

When it comes to detecting forged videos, most modern works use deep learning. Examples are Güera & Delp [9] who use an approach similar to that of this report, with a feature extraction network followed by an LSTM, and Afchar et al. [1], who present two network architectures, one CNN-based an one Inception-based. Both these train on datasets consisting of many different target actors. Agarwal et al. [2] instead train models on single individuals, learning solely to detect manipulated videos of that person. Some models focus on specific aspects of videos, such as blinking patterns [16] or head poses [36].

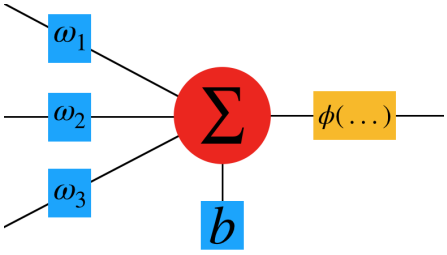
These detectors are not foolproof. Vougioukas et al [34], for instance, manage to generate facial reenactment videos with convincing blinking patterns. Meanwhile, Neekhara et al. [18] add noise to videos, invisible to the human eye, that trick detection algorithms to believe that manipulated videos are pristine. This arms race will likely continue for as long as detecting forged videos is possible.

2. Neural Networks - an Overview

Briefly put, a neural network is an attempt to mimic how the human brain solves problems. See figure 2.1a for an example of a network. The network is built up by *nodes* (circles in the figure), mimicking synapses, usually ordered in layers. Nodes send output signals (a scalar number) based on what signals they receive from other nodes; typically, input signals are received from nodes in the previous layer and output signals are sent to the next one. If every node in a layer has an input edge from every node in the previous layer, that layer is said to be *fully connected*. The first layer is referred to as the *input layer* (pink in the figure), and the last one the *output layer* (yellow). For instance, a network could take black-and-white images and try to decide if there is a human in the image; in this case, each input node would correspond to one pixel value in the image, and the output node would output one number, say "1" if there is a human present and "0" otherwise. Layers that are neither input or output layers are referred to as *hidden layers*. If a neural network has more than one hidden layer, the term *deep learning* is used.



(a) Example of a network. Only edges to and from the red node are shown.



(b) Overview of a node. The node weighs input signals with ω, sums them together, adds a bias term b and applies the activation function ϕ to receive its output signal.

Figure 2.1: Overview of neural networks.

Figure 2.1b shows a zoom-in on one node. First, each input signal to the node are weighted by multiplying them with a number $(\omega_1, \omega_2, ...)$. *Training* a network means finding values for all these weights so that the network solves

its task as well as possible. This is done by feeding in input data into the network and telling it what output is expected. By using a method called *back propagation* [3], the weights can then be tweaked iteratively.

Once the input signals have been weighted, they are summed together and a *bias* is added. This is another scalar number which is trained alongside the weights. The received value is fed into an *activation function*, which could be nearly any function. Today, the RELU function is the most commonly used function, which simply passes through the input if it is positive and outputs zero if the input is negative. This function value is then what is sent as output from the node. This calculation is done for every node in the network until the output layer is reached.

See Appendix B for any equations. A slightly longer, yet easy to understand, overview of neural networks can be found at [35], while a more mathematical introduction can be found at [38].

2.1 Recurrent Neural Networks

A recurrent neural network (RNN) is a network tailored for sequential data, such as stock exchange data, letters in a text or frames in a video. It takes into account not only data from the current element, but also that from previous elements in the sequence. One kind of RNN is the *long short-term memory* (LSTM) network. The details of how these networks work are fairly technical, for which reason they are not covered here. See for instance [26] for an introduction.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNN's) are most commonly used when the input consists of images or videos. These are usually fairly large; for instance, if the input were $300 \cdot 300$ pixel color images, the network would require $300 \cdot 300 \cdot 3 = 270000$ input nodes. If one was to use a network consisting of fully connected layers, the number of weights in the network would explode. Moreover, these networks tend to struggle with translation in images; simply moving around objects in the image could render completely different results.

CNN's solve both these problems. They use so-called *filters* to analyse the image: windows, usually only a few pixels wide, that are moved across the image. For all positions the window is moved into, the pixel values are multiplied with learned parameters to generate an output value, all of which together form a 2D image. Using several filters in a layer results in one 2D image per filter, together forming a 3D "data cube". The layer creating one of these cubes from an image or another cube is called a *convolutional layer*, named after the mathematical operation used to get the output value from the filter (*convolutions*). A CNN is a network built up by these layers, often with other layers in between these (*max-pooling layers*, for instance, are popular). For more information on CNN's, see for instance [27], or [39] for a more mathematical introduction.

3. Facial Video Manipulation - an Overview

There are two main types of facial video manipulations: *identity modification* means to replace a person's face with someone else's, while *expression modification* (or *facial reenactment*) means changing a person's facial expressions and lip movements. For the proposed model, four methods for facial video manipulation will be taken into account. FaceSwap and DeepFake are both examples of methods for identity modification, while Face2Face and NeuralTextures are examples of expression modification methods. All methods take videos of two persons as input: the "original" video, referred to as the target video, and the video of the person whose identity or facial expressions one wants to transfer into the other video, referred to as the source video. Below are the four manipulation methods explained. Unless otherwise specified, all facial images in the figures are screenshots from [22].

3.1 FaceSwap

Faceswap is usually a very broad term, used to denote any instance of the face of a person in an image or video is replaced by that of someone else. Here, however, it denotes a specific method for this, similar to the one implemented by Kowalski [15]. See figure 3.1 for an overview.

The method works for each pair of frames for the target and source video, until one of them ends. In both images, facial *landmarks* are detected; these could be the contour of the face, eyes, mouth, nose or more obscure features that a face detector recognises as part of a face. Using the landmarks from the source image, a 3D model of the source actor's face is created. This is then transformed to match the facial landmarks of the target actor, and blended into the target image. After this is done for each frame, a video with the target actor's face swapped with that of the source actor is retrieved.

The algorithm does not require any training or having seen either actor's faces before, and doesn't require much computational power (depending on face detection algorithm). However, it struggles when there are large differences in lighting in the two videos, and is very reliant on a good face detection algorithm.

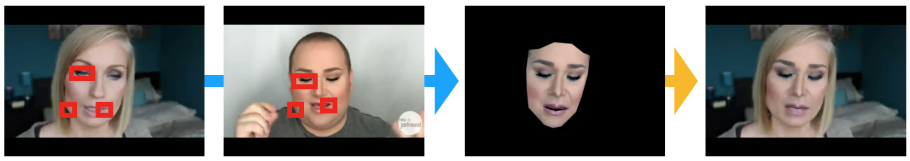


Figure 3.1: The FaceSwap pipeline. A 3D model of the source actor's face is created using facial landmarks, and then transformed to match those of the target actor. Finally, the 3D face is blended into the target video.

3.2 DeepFake

Just like the word faceswap can be used very broadly, deepfake is sometimes used as a synonym for any video manipulated using a neural network, but was originally the name of a specific manipulation method. This method is derived from a kind of neural network known as an *autoencoder*.

The goal of an autoencoder is to encode a specific kind of input data (for example images of a person) into a smaller, more compact representation and then be able to retrieve the original data from this representation. The model can be split into two parts; the *encoder* responsible for compressing the original data, and the *decoder* responsible for trying to reconstruct the original image. Both these parts are deep neural networks, and need to be trained on data similar to the input one wishes to use the autoencoder on. See figure 3.2 for an image of this.

The basic architecture of a deepfake model can be found in figure 3.3. Here, two autoencoders are trained on videos of a specific person each, the two people whose identity one wishes to swap. These autoencoders are forced to share the same encoder, while the decoders are allowed to differ. Ideally, this means that the encoder focuses on video-specific information like facial expressions and lighting while ignoring the traits of the two persons; these can be restored by the decoders. Most often some face detection algorithm is used beforehand so that only the faces of the persons are fed into the autoencoders.

Figure 3.4 shows how the deepfake videos are generated. Once the autoen-

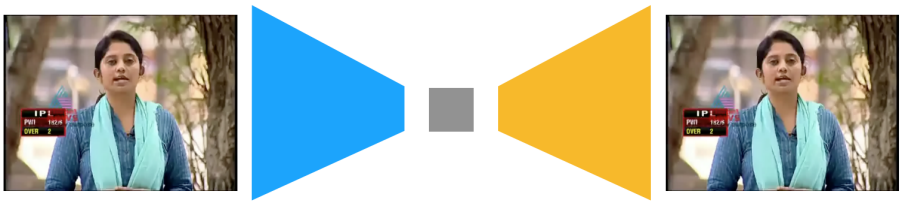


Figure 3.2: An autoencoder. An encoder (blue) takes the input image and returns a compact representation of this image (gray), while the decoder (yellow) tries to reconstruct the original image from this representation.

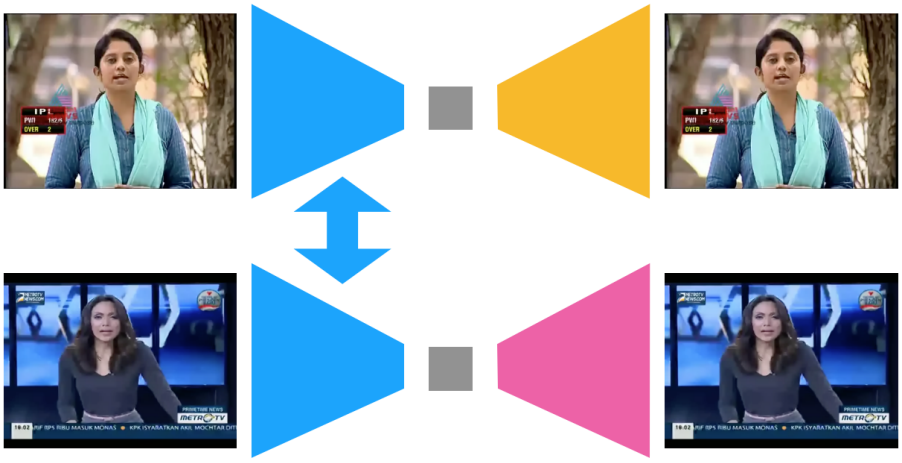


Figure 3.3: The deepfake architecture. Two autoencoders are trained with the same encoder (blue) but different decoders (yellow and pink).

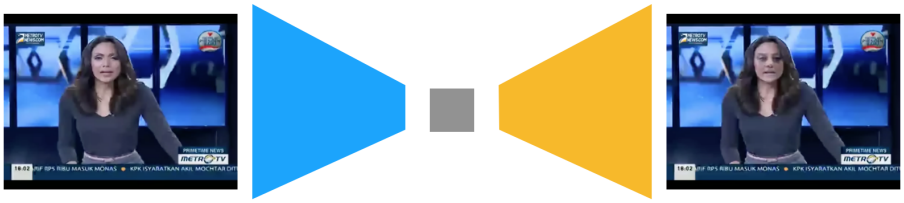


Figure 3.4: The "wrong" video is fed into one of the autoencoders, resulting in a video with the faces of the two persons swapped.

coders are trained, this is straightforward. The target video is fed into the encoder, from witch a video is retrieved using the decoder trained on the source actor instead of the target actor. This results in a video where the face of the target actor is replaced by that of the source actor, while facial expressions, lip movements et cetera remain unchanged.

Several of the most sophisticated examples of identity modification has been created using the Deepfake algorithm. However, the algorithm require large quantities of training videos of the target and source actors, and the autoencoders take a long time to train. Too little data or training time translates to poor quality of the forged video.

3.3 Face2Face

Introduced by Thies et al. [31], Face2Face is one of the more well-known method for facial reenactment. The core idea behind the algorithm is to create 3D models of the faces of the source and target actors, then track deformations in

the model of the source actor and transfer these into that of the target actor. This is not done by learning; instead, Principal Component Analysis (PCA) is used to track deformations and parametrize the faces. These parameters are plugged into an objective function, thus reducing the task to an optimisation problem, which is solved using Iteratively Reweighted Least Squares (IRLS). When the expressions have been transferred, a mouth interior is retrieved by searching the video of the target actor and warping the best match to the current frame. For this reason, the video of the target actor has to be known beforehand, but the source video can be used to manipulate the video in real-time. See figure 3.5 for an overview of the algorithm.

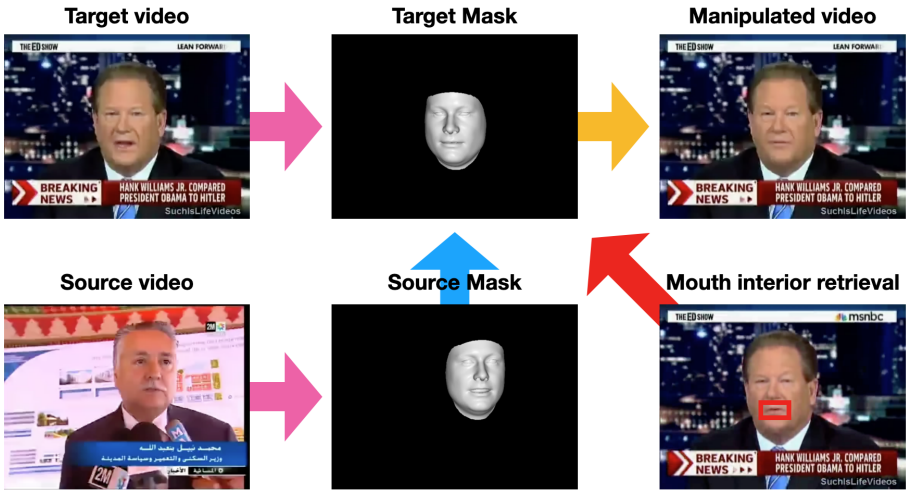


Figure 3.5: Face2Face. 3D models of the source and target actors' faces are created (pink arrows). Then, for each frame, the model of the target actor's face is deformed to match the expressions of the source actor (blue), whereafter the mouth interior that best matches the new facial expression is retrieved from the target video (red). Finally, the modified 3D face is blended back into the target video (yellow).

3.4 NeuralTextures

In computer graphics, a *texture map* is a map that holds information of a 3D object, about for instance color or texture of the object, its albedo or small displacements on the surface. Introduced by Thies et al. [30], *neural textures* replace these hand-crafted features of the object with learned ones, using images of the object as input. With this neural texture, a viewpoint and a *uv-map* that links elements in the neural texture map to points in the object, a viewpoint-specific texture can be generated. If this is plugged into a *deferred neural renderer*, trained alongside the neural texture, it can generate an image

of the object from the given viewpoint. Figure 3.6 visualises the pipeline.

Neural textures can thus be used for more than facial video manipulation, but Thies et al. mention this as one of its applications. This is shown in figure 3.7. Here, just as in the Face2Face method, training videos are used to create a 3D model of the target actor's face. On top of that, however, a person-specific neural texture and a person-specific deferred neural renderer is created for the target actor. By using the same expression transfer technique as Face2Face, a uv-map is created from the target video and is then altered to match the expressions of the source actor. Using this uv-map, the neural texture and the neural renderer, one receives the manipulated video.

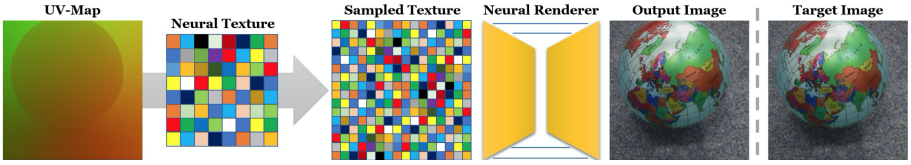


Figure 3.6: The process of generating an image of an object from a novel viewpoint using neural textures. The neural texture of the object, along with an UV-map generated using the desired viewpoint, is used to sample a viewpoint-specific texture. Feeding this texture into the deferred neural renderer returns an image of the object from the desired viewpoint. Image from [30].

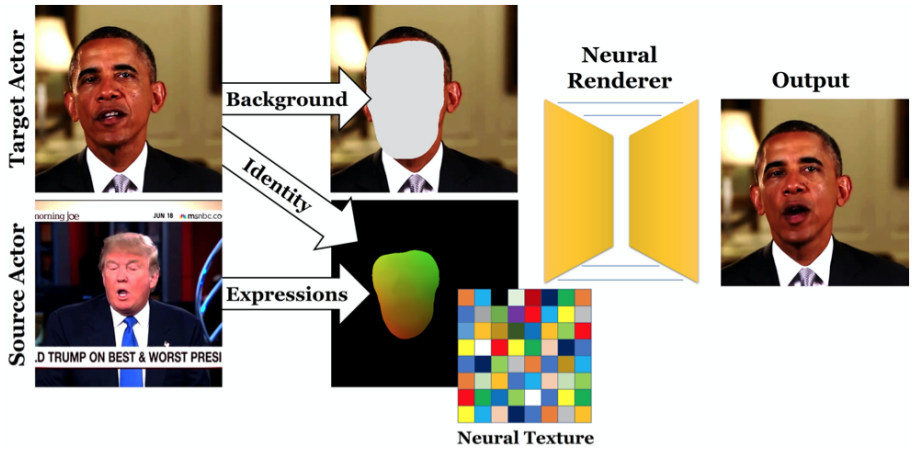


Figure 3.7: Expression modification using neural textures. A person-specific neural texture and neural renderer is trained on the target actor. Then, a UV-map matching the target actor's face but the source actor's facial expressions is calculated. By sampling a texture from this and the neural texture, and feeding this into the neural renderer, the manipulated video is received. Image from [30].

4. The Classifier

This chapter is where the proposed model is explained in detail. This consist of one or several feature extraction networks followed by an LSTM layer and lastly a classification output layer, see section 4.3 for details. MATLAB was used for all implementations.

4.1 Data

The dataset used for training and evaluating the network is the FaceForensics++ dataset, created by Rössler et al. [21, 22]. This dataset consists of 1000 sequences of videos downloaded from Youtube, each around a few hundred frames long, 30 frames per second, focused on a human face. The person in the video is always close to front-facing, and there's always only one face in the video. All videos have also been modified using the four manipulation methods described above, the entire dataset thus consisting of 5000 videos. In the Neural Texture videos, only the mouth of the target actor was modified. All videos were compressed using the H.264 codec, with a constant rate quantisation parameter of 40, resulting in fairly low-quality videos. Higher quality videos as well as the uncompressed ones are also available in the dataset, but wasn't used in this report due to memory limitations. Examples of video screenshots can be found in figure 3.1-3.5.

Masks were also supplied for each manipulation method, of which only the masks for the Deepfake videos were used. These are videos with a white square in the area that the corresponding deepfake video has been manipulated, and are black everywhere else. Since the manipulated area always was the face area in the video, these could be used to cut out the this area and input this into the network instead of the entire video. The reason the masks were used instead of a face detector was so that the input videos would look as similar as possible as the ones used for the benchmarks [21]. Finally, after this was done, the videos were rescaled to the input size of the used feature extraction network.

4.2 Feature Extraction

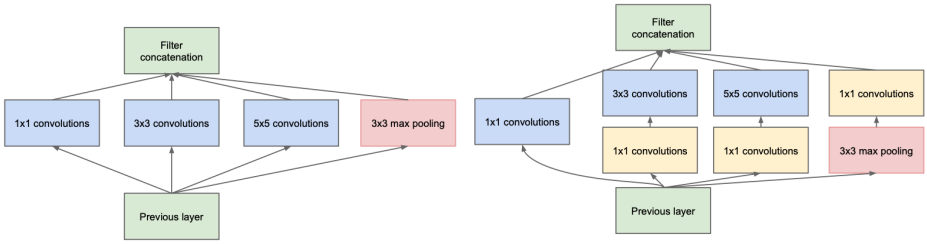
After a video is preprocessed, it is either fed into a feature extraction network. Four different options were tested here: *GoogleNet*, *Xception*, *DenseNet* (all

introduced below) and a combination of all three. In this last case. the video was simply fed into all three networks, and all output was sent to the LSTM layer. The three networks are all trained on the ImageNet dataset [23], made for training image classification networks, and were not retrained due to this taking too much time. The layer before the classification layer of these networks was used as output layer.

4.2.1 GoogLeNet

GoogLeNet is an incarnation of the *Inception* network architecture (first version), and the two were introduced in the same paper [28]. Inception networks are essentially CNN's, but were designed to tackle a few problems that conventional CNN's struggle with. Firstly, image classification can often be challenging, with high variation within classes and sometimes small variations between them. For this reason, very deep networks have been used for these tasks, but these use a lot of computational resources to train and are prone to overfitting. Secondly, while conventional CNN's have no trouble with translations in images, they struggle when objects vary in size. If all objects of a class in the training data take up roughly the same amount of space in their images, the classifier generally will not recognise objects of different sizes.

Inception networks solve these problems by going wide rather than deep. They are built up by Inception modules, which consist of several filters of different sizes applied to the input. See figure 4.1a for this. Having filters of different sizes means it is easier to detect objects of different sizes, and since an Inception module can extract more information from its input than a conventional CNN layer, the network doesn't have to be as deep. The modules are still computationally expensive, however.



(a) A naïve Inception module.

(b) An Inception (v1) module using 1x1 convolutions for dimensionality reductions.

Figure 4.1: Images from [28].

To tackle this, 1x1 convolutions were introduced before each set of filters was applied (see figure 4.1b for this). These simply reduce the size of the "data cube" received from each set of filters. For instance, applying D 3x3 filters to an image of size HxW, one would get a data cube of size HxWxD (this assumes we've done some form of *padding*, dealing with what happens when the filter

partly moves outside of the image). A 1x1 convolution is a filter of size 1x1xD that "pushes" the cube into a flat HxW image. By choosing a number N how many if these we use, where N $<$ D, we've reduced the data to the size HxWxN. See [28] for more details.

4.2.2 Xception

Xception, or *Extreme Inception*, was introduced by Chollet [5] and builds on the Inception architecture. An overview of an Xception module can be found in figure 4.2. First, an 1x1 convolution is used on the input. Then, unlike in Inception networks, each slice of the data cube is filtered *separately* with unique filters. The hypothesis that this works, that each slice can be handled independently, is the "extreme" part of Extreme Inception.

The Xception architecture also features *residual connections*. First introduced by He et al. [10] as part of the ResNet architecture, these are "shortcuts" in the network, where a layer sends its output not only to the next layers but also further ahead, typically without passing it through any activation function. This helps with the *vanishing gradient problem* which makes deep networks hard to train, and was shown experimentally to improve the network's performance significantly.

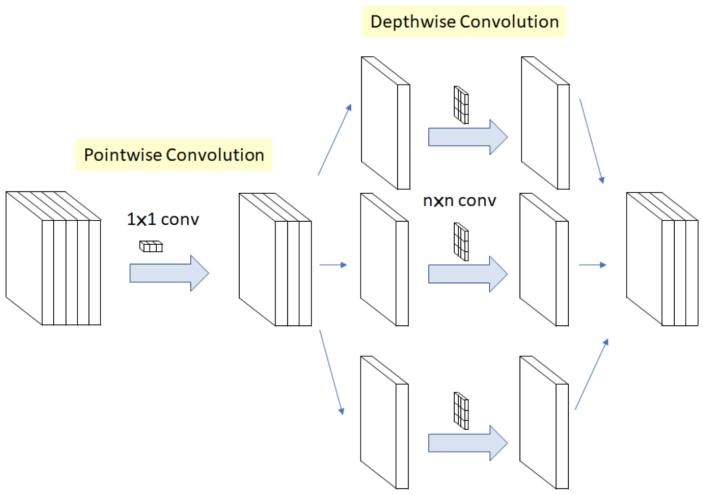


Figure 4.2: Overview of an Xception module. Image from [32].

4.2.3 DenseNet

DenseNet blocks, or Dense Convolutional Network blocks, take the residual connections discussed above to the extreme. Here, *every* layer is connected to each other; layers get inputs from all previous layers and sends its output to all layers ahead. See figure 4.3 for an illustration of this. DenseNets are created by

putting together one or more of these blocks with transition layers in-between, consisting of a 1x1 convolution as well as a 2x2 *average pooling* layer. This looks at every set of four neighbouring pixels in the input and returns the average pixel value for each of these.

The core argument behind connecting all layers is that it preserves information. In a traditional CNN, all information that is not passed on from one layer to the next is "forgotten" by the network. This means that these networks typically have to be fairly wide to preserve as much information as possible. By remembering the output of all previous layers, each layer can be much smaller and only add a smaller amount of "knowledge" to the network. Thus, slightly counter-intuitively, even though DenseNets seem to add more connections to the CNN architecture, they actually require fewer parameters.

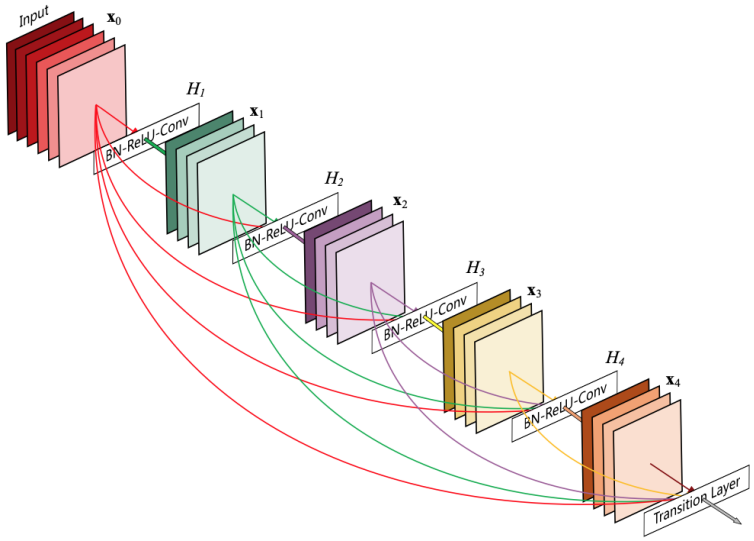


Figure 4.3: An example of a DenseNet block. Image from [11].

4.3 The Proposed Architecture

Figure 4.4 shows an overview of the proposed model. As explained in section 4.1, the model takes a video as input and starts by extracting the facial region. This video is fed into one of the three feature extraction networks discussed above, or alternatively all of them. The output of these is fed into a small LSTM network, consisting of an LSTM layer with 75 nodes, followed by a fully connected layer and finally a classification layer, outputting either "1" for pristine or "0" for manipulated.

To evaluate the model, the LSTM network was trained both on all manipulation methods individually and on all methods simultaneously. In both cases,

half of the videos used for training were pristine and the other half were the corresponding manipulated videos. Training was done for 15 epochs using the ADAM optimiser, binary cross-entropy loss, a dropout of 0.5 and a learning rate of 0.001.

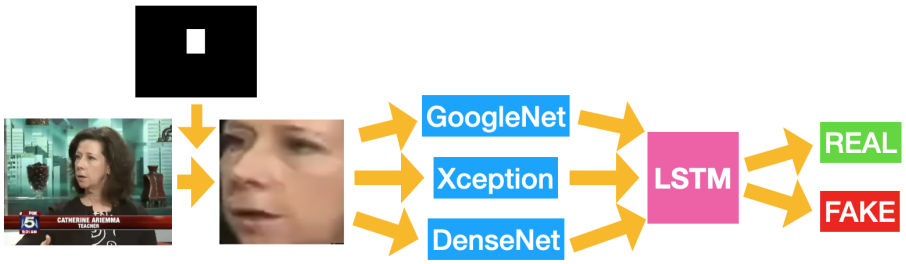
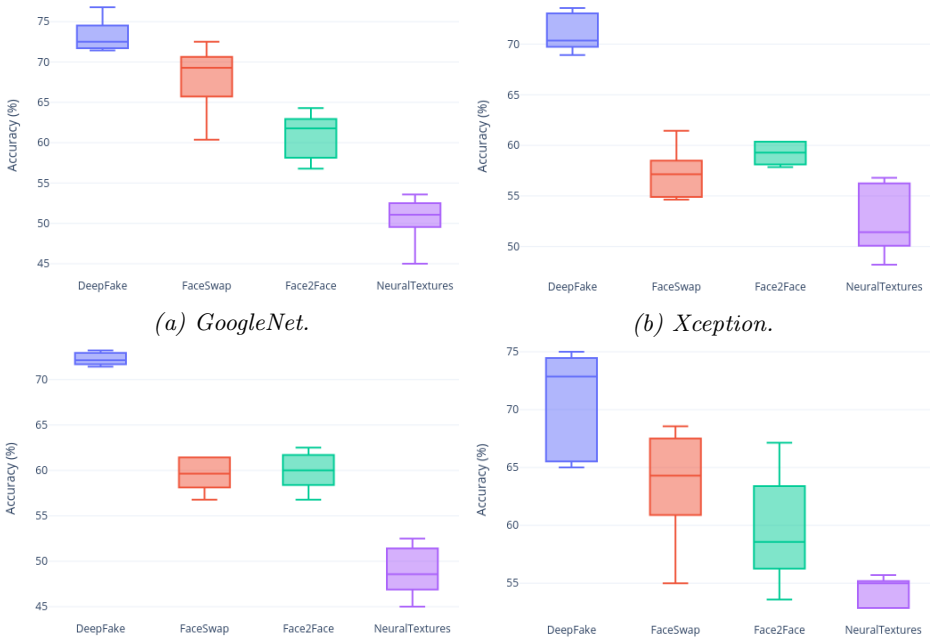


Figure 4.4: An overview of the proposed network. The facial area of the input video is extracted and rescaled, and then sent to one (or all) of the feature extraction network. The output from these are fed into an LSTM network, classifying the video as either real or fake.

5. Results



(a) GoogleNet. *(b) Xception.*

(c) DenseNet. *(d) Combined.*

Figure 5.1: The results from training the networks on each manipulation method individually, for each feature extraction network and the combination of all of them.

Figure 5.1 shows the results from training the network five times on each manipulation method individually, and then evaluating it by calculating the classification accuracy on the validation data. The results from training on all methods simultaneously can be found in figure 5.2. To analyse biases in the latter model, such as it potentially classifying videos as manipulated much more often than pristine, the accuracy of using the model on only pristine videos was plotted against that of only manipulated videos. One of these plots can be found in figure 5.3, the rest can be found in Appendix A along with raw data.

(a) GoogleNet. *(b) Xception.*

(c) DenseNet. *(d) Combined.*

Figure 5.2: The results from training the networks on all manipulation method simultaneously, for each feature extraction network and the combination of all of them.

Figure 5.3: Accuracy when classifying pristine versus manipulated videos using the DenseNet network and having trained on all methods simultaneously.

6. Discussion and Conclusion

6.1 The Results

Although the results vary with manipulation method and feature extraction network, looking at figure 5.1 and 5.2, it is apparent that the model currently is not commercially useful. Barely ever achieving an accuracy above 75% would cause far too many videos to be incorrectly classified. This does not mean that the results should be seen as a failure, however. The model with GoogleNet for feature extraction when training on forgery methods individually and the one using all networks when training on all methods are determined to be the best models among the tested ones. Comparing the results from these with the benchmark of Rössler et al. [21], it can be noted that their results are better or equal to that of several methods presented, including the networks proposed by Cozzolino et al. [6] and Rahmouni et al. [20]. Given the scope of the project, this is considered a success.

There are still improvements to be made to the model, though. For one, there is always more room for more polishing, adjusting hyperparameters and longer training. The improvement believed to have the largest positive effect, however, is retraining the feature extraction network; only the LSTM part of the network was trained. This is likely the reason that one of the models in the benchmark, using the Xception network, performs significantly better than those proposed here. This network was initialised with the ImageNet weights, but was retrained for 15 epochs. It is easy to imagine that this would make a large difference; the task the ImageNet dataset is designed for is to classify objects in videos. The network is thus more likely to pick up features that helps identify the faces as faces and less likely to notice pixel-level details that help classifying the video as pristine or manipulated. The training would have taken far too long time to be a feasible part of this project however.

It is also worth noting that all the videos used for training and evaluation were heavily compressed, more so than videos encountered on social media and Youtube. It is evident in the benchmarks that these lower-quality videos are far more hard to classify correctly; when using raw, uncompressed videos, all evaluated architectures were close to 100% accuracy [21]. It is therefore likely that the proposed architecture would perform better on less compressed data, with more realistic compression factors, if trained on it. The retraining is crucial, though; no conclusions can be drawn without any experiments made how well

the network would perform if it was not retrained on these videos.

Some manipulation methods were easier to detect than others; looking at 5.1, deepfakes were clearly the easiest to detect, while videos creating using neural textures were the hardest. This goes for the benchmark as well [21]. Why this is the case could be analysed further; it is likely a combination of some methods being superior to others in terms of being hard to detect, and the quality of the videos in the dataset. One factor that could, at least partly, explain why the neural texture videos were harder to detect is that only the mouth region were modified in these videos, while the entire face was modified in the rest. A smaller modified region could mean fewer artefacts left for the classifier to find.

Comparing figures 5.1 and 5.2, a higher accuracy overall was achieved when training one network per manipulation method than when training on all methods simultaneously. This is to be expected; learning to identify one kind of manipulation method is easier than keeping track of four of them at the same time. In the case of neural textures, however, the accuracy seemed to increase when training on all manipulation methods. The reason behind this is unknown, but one hypothesis could be that neural textures leave artefacts similar to those of some other method, but because of how small the modified area is, the network isn't able to pick up on these as signifiers. When there are "easier" videos in the training dataset with the same artefacts present, the network learns to look for these and will then find them in the neural texture videos as well.

One observation can be made in figure 5.3, which compares true positives and true negatives of one of the models. Although the accuracy of the network was fairly constant during the five times it was retrained and reevaluated, the true-positive-to-true-negative-ratio varied greatly. The same phenomenon could be seen when using for any feature extraction network (see Appendix A). The models thus tended to either classify too many videos as pristine or as forged, but which of them was the case changed when retraining the model. Whether many false positives or many false negatives are to be preferred is debatable (both cause serious problems, see below), but it is nevertheless important to know of biases in the model to be able to correctly interpret its results.

6.2 The Role of a Forged Video Detector

As mentioned, the network presented in this work does not perform well enough to be usable in its current state, but even if it would achieve a much higher accuracy, it should still be used with caution. Any forged video detector can arguably never be expected to perform with perfect accuracy, and expecting such of one would be nearly as naïve as ignoring the threat of forged videos altogether. Falsely assuming the authenticity of videos could result to everything from wrongful convictions to authenticating fake news, while a too high ratio of false positives would damage online content creation if videos were filtered away too harshly. Add to this the rapidity at which forgery methods are improving, and it becomes apparent that the rulings of forgery detectors should be taken with a grain of salt.

Furthermore, we might reach a point where detecting forged videos is no longer possible. Hao Li, coauthor of [2], is one of the people who warn about this, saying that "videos are just pixels, ultimately" [33]. The higher rate at which detection algorithms improve, the sooner that point is reached; just like detectors are trained on videos manipulated by current forgery algorithms, these are often trained against current state-of-the-art detectors.

All this is not to say that developing forged video detectors is pointless. These are still very much useful, if for instance used for showing warning messages when a video is thought to be manipulated, and the more detection algorithms and models, the more trustworthy are their rulings if many of them say the same. Thus, while they cannot make up the entire line of defence against manipulated videos, they still play an important role in it. Research is being made on other ways the authenticity of videos can be guaranteed; for instance, Korus & Memon [14] has created a way to digitally "watermark" pristine videos. Most important, however, is to spread the knowledge that videos can be convincingly forged, and as with all media and information, advocate source criticism and common sense.

6.3 Future Studies

There are several ways to improve on the proposed model, some of which have already been mentioned. Training the feature extraction network along with some polishing of hyperparameters would most likely increase the accuracy of the model drastically. To increase utility of the model, more manipulation methods could be taken into consideration, for instance those proposed by Nirkin et al [19] and/or Siarohin et al. [25]. They could either be trained on alongside the other forgery methods (as in 5.2), or individual models could be created for each method (as in 5.1). Videos could also be taken from more sources than the FaceForensics++ dataset. On top of completely automatic manipulation methods, one could also try training the model on videos that has been manipulated by hand, or a with a combination of automatic and manual manipulation.

On a broader scope, while there are many forgery detection methods that are to be explored more thoroughly (such as using audio as well as video when classifying), the most important task right now might be to find alternative methods to tackle forged videos. Watermark methods such as the one presented in [14] could potentially guarantee the authenticity of videos without having to detect forged ones, as could blockchain approaches. Still, the challenges forged videos entails can't be solved by computer scientists alone. For instance, a juridical framework needs to be created for how to treat these videos, as well as who bears the responsibility if a forged video is discovered. Although the threat of forged videos cannot be eliminated, it is the author's hope that as vigorous effort as possible is made to diminish the harm they can cause, and that videos will still be a trustworthy source of information in the future.

Bibliography

[1] Afchar, D., Nozick, V., Isao Echizen, J.Y. (2018). *MesoNet: a Compact Facial Video Forgery Detection Network*. 2018 IEEE International Workshop on Information Forensics and Security (WIFS). https://arxiv.org/pdf/1809.00888.pdf

[2] Agarwal, S., Farid, H., Gu, Y., He, M., Nagano, K., Li, H. (2019, June). *Protecting World Leaders Against Deep Fakes*. IEEE Conference on Computer Vision and Pattern Recognition. http://openaccess.thecvf.com/content_CVPRW_2019/papers/Media%20Forensics/Agarwal_Protecting_World_Leaders_Against_Deep_Fakes_CVPRW_2019_paper.pdf

[3] Al-Masri, A. (2019, January 30). *How Does Back-Propagation in Artificial Neural Networks Work?* Towards Data Science. https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7 (Retrieved 2020, May 12).

[4] Ayyub, R., (2018, November 21). *I Was The Victim Of A Deepfake Porn Plot Intended To Silence Me*. Huffington Post UK. https://www.huffingtonpost.co.uk/entry/deepfake-porn_uk_5bf2c126e4b0f32bd58ba316?

[5] Chollet, F. (2017). *Xception: Deep Learning with Depthwise Separable Convolutions*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://arxiv.org/pdf/1610.02357.pdf.

[6] Cozzolino, D., Poggi, G., Verdoliva, L. (2017). *Recasting residual-based local descriptors as convolutional neural networks: an application to image forgery detection*. ACM Workshop on Information Hiding and Multimedia Security. https://arxiv.org/pdf/1703.04615.pdf

[7] Dufour, N., Gully, A. (2019, September 24). *Contributing Data to Deepfake Detection Research*. Google AI. https://ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html

[8] [EZRyderX47]. (2020, February 14). *Robert Downey Jr and Tom Holland in Back to the future - This is heavy! [deepfake]*. Retrieved from: https://www.youtube.com/watch?v=8OJnkJqkyio

[9] Güera, D., Delp, E. J. (2018, November). *Deepfake Video Detection Using Recurrent Neural Networks*. 2018 15th IEEE International Conference

on Advanced Video and Signal Based Surveillance. https://ieeexplore.ieee.org/abstract/document/8639163

[10] He, K., Zhang, X., Ren, S., Sun, J. (2015). *Deep Residual Learning for Image Recognition*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://arxiv.org/pdf/1512.03385.pdf

[11] Huang, G., Liu, Z., van der Maaten, L., Weinberger, K. (2017). *Densely Connected Convolutional Networks*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://arxiv.org/pdf/1608.06993.pdf

[12] Karras, T., Aila, T., Laine, S., Lehtinen, J. (2018). *Progressiv Growing of GANs for Improved Quality, Stability and Variation*. Arxiv. https://arxiv.org/pdf/1710.10196.pdf

[13] Karras, T., Laine, S., Aila, T. (2019). *A Style-Based Generator Architecture for Generative Adversarial Networks*. Arxiv. https://arxiv.org/pdf/1812.04948.pdf

[14] Korus, P., Memon, N. (2019). *Neural Imaging Pipelines - the Scourge or Hope of Forensics?*. Arxiv. https://arxiv.org/pdf/1902.10707.pdf

[15] Kowalski, M. FaceSwap. Github. https://github.com/MarekKowalski/FaceSwap/

[16] Li, Y., Chang, M. C., Lyu, S. (2018). *In ictu oculi: Exposing ai created fake videos by detecting eye blinking*. 2018 IEEE International Workshop on Information Forensics and Security (WIFS). https://arxiv.org/pdf/1806.02877.pdf

[17] Mack, D. (2018, April 17). *This PSA About Fake News From Barack Obama Is Not What It Appears*. BuzzFeed News. Retrieved from: https://www.buzzfeednews.com/article/davidmack/obama-fake-news-jordan-peele-psa-video-buzzfeed#.ugOXGqvAn3

[18] Neekhara, P., Hussain, S., Jere, M., Koushanfar, F., McAuley, J. (2020). *Adversarial Deepfakes: Evaluating Vulnerability of Deepfake Detectors to Adversarial Examples*. Arxiv. https://arxiv.org/pdf/2002.12749.pdf

[19] Nirkin, Y., Keller, Y., Hassner, T. (2019, August). *FSGAN: Subject Agnostic Face Swapping and Reenactment*. International Conference on Computer Vision. https://arxiv.org/pdf/1908.05932.pdf

[20] Rahmouni, N., Nozick, V., Yamagishi, J., Echizen, I. (2017).*Distinguishing computer graphics from nat- ural images using convolution neural networks*. IEEE Workshop on Information Forensics and Security 2017. http://www-igm.univ-mlv.fr/~vnozick/publications/Rahmouni_WIFS_2017/Rahmouni_WIFS_2017.pdf

[21] Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., Nießner, M. (2019, August 26). *FaceForensics++: Learning to Detect Manipulated Facial Images*. International Conference on Computer Vision. https://arxiv.org/pdf/1901.08971.pdf

[22] Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., Nießner, M. *FaceForensics++: Learning to Detect Manipulated Facial Images* [Data set]. (Updated 2019, Agust 30) Retrieved from https://github.com/ondyari/FaceForensics

[23] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, A., Berg, A. C., Fei-Fei, L. (2014). *Imagenet large scale visual recognition challenge*. International Journal of Computer Vision, 2015. https://arxiv.org/pdf/1409.0575.pdf

[24] Schroepfer, M. (2019, September 5). *Creating a data set and a challenge for deepfakes*. Facebook. https://ai.facebook.com/blog/deepfake-detection-challenge/

[25] Siarohin, A., Lathuilière, S., Tulyakov, S., Ricci, E., Sebe, N. (2019). *First Order Motion Model for Image Animation*. Advances in Neural Information Processing Systems 32. http://papers.nips.cc/paper/8935-first-order-motion-model-for-image-animation.pdf

[26] Srivastava, P. (2017, December 10). *Essentials of Deep Learning: Introduction to Long Short Term Memory*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/ (Retrieved 2020, March 25).

[27] Stewart, M. (2019, Februart 27). *Simple Introduction to Convolutional Neural Networks*. Towards Data Science. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac (Retrieved 2020, April 2).

[28] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. (2015). *Going Deeper with Convolutions*. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://arxiv.org/pdf/1409.4842.pdf

[29] Thies, J., Elgharib, M., Tewari, A., Theobalt, C., Nießner, M. (2019, December). *Neural Voice Puppetry: Audio-driven Facial Reenactment*. https://arxiv.org/pdf/1912.05566.pdf

[30] Thies, J., Zollhöfer, M., Nießner, M. (2019, July). *Deferred Neural Rendering: Image Synthesis using Neural Textures*. ACM Transactions on Graphics. https://arxiv.org/pdf/1904.12356.pdf

[31] Thies, J., Zollhöfer, M., Stamminger, M., Theobalt, C., Nießner, M. (2016, November). *Face2Face: Real-Time Face Capture and Reenactment of RGB*

Videos. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://niessnerlab.org/papers/2016/1facetoface/thies2016face.pdf

[32] Tsang, S. (2018, September 25). *Review: Xception - With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification)*. Towards Data Science. https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568 (Retrieved 2020, April 10).

[33] Vincent, J. (2019, June 27). *Deepfake detection algorithms will never be enough*. The Verge. https://www.theverge.com/2019/6/27/18715235/deepfake-detection-ai-algorithms-accuracy-will-they-ever-work (Retrieved 2020, April 23).

[34] Vougioukas, K., Petridis, S., Pantic, M. (2019). *Realistic Speech-Driven Facial Animation with GANs*. International Journal of Computer Vision. https://arxiv.org/pdf/1906.06337.pdf

[35] Woodford, C. (2019, April 4). *Neural Networks*. https://www.explainthatstuff.com/introduction-to-neural-networks.html (Retrieved 2020, March 12).

[36] Yang, X., Li, Y., Lyu, S. (2019). *Exposing Deep Fakes Using Inconsistent Head Poses*. ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). https://arxiv.org/pdf/1811.00661.pdf

[37] Yi, R., Ye, Z., Zhang, J., Bao, H., Lie, Y. (2020, March). *Audio-driven Talking Face Video Generation with Learning-based Personalized Head Pose*. Arxiv. https://arxiv.org/pdf/2002.10137.pdf

[38] Zhou, V. (2019, July 24). *Machine Learning for Beginners: An Introduction to Neural Networks* https://victorzhou.com/blog/intro-to-neural-networks/ (Retrieved 2020, March 25).

[39] Zhou, V. (2019, August 8). *CNNs, Part 1: An Introduction to Convolutional Neural Networks*. https://victorzhou.com/blog/intro-to-cnns-part-1/ (Retrieved 2020, March 26).

Appendix A: Data

	DeepFake						FaceSwap				
GoogleNet	71.43	76.79	71.79	73.79	72.50		72.50	70.00	60.36	67.50	69.29
Xception	73.57	68.93	70.36	70.00	72.86		57.14	61.43	57.50	55.00	54.64
DenseNet	73.21	71.79	71.43	72.14	72.86		61.43	58.57	59.64	56.73	61.43
Combined	75.00	65.71	65.00	72.86	74.29		64.29	67.14	62.86	55.00	68.57

	Face2Face						NeuralTextures				
GoogleNet	56.79	62.50	58.57	64.29	61.79		52.14	53.57	51.07	45.00	51.07
Xception	57.86	59.29	60.36	60.36	58.21		51.43	56.79	50.71	56.07	48.21
DenseNet	62.50	60.00	61.43	58.93	56.79		52.50	45.00	48.57	47.50	51.07
Combined	57.14	62.14	53.59	58.57	67.14		55.71	55.00	52.86	55.00	52.86

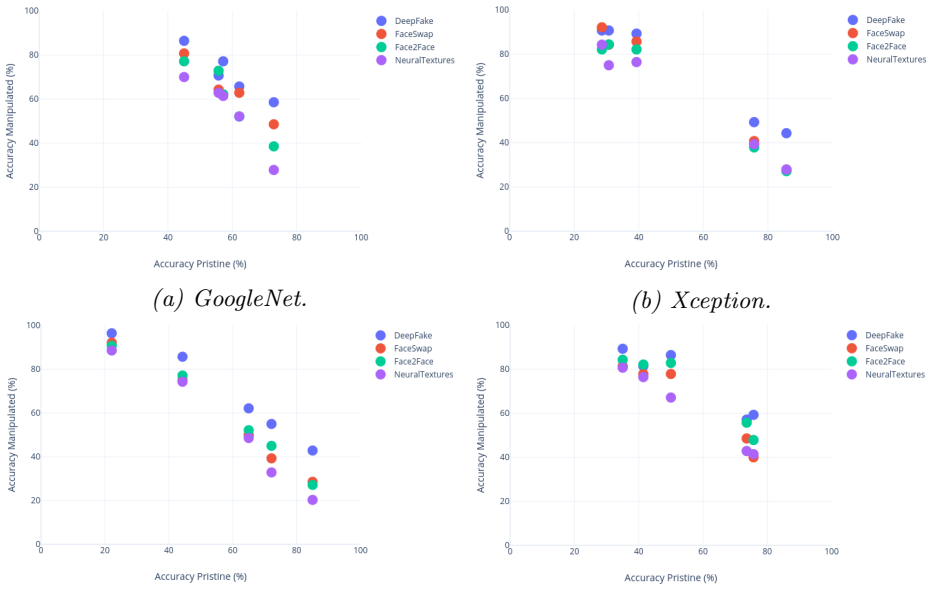
Table A.1: Validation results from training the network five times for each combination of manipulation method and feature extraction network, when training on the manipulation methods separately.

	DeepFake						FaceSwap				
GoogleNet	58.57	86.43	77.14	65.71	70.71		48.57	80.71	62.14	62.86	64.29
Xception	44.29	89.29	90.71	49.29	90.71		22.86	85.71	92.14	40.71	84.29
DenseNet	55.00	62.14	96.43	85.71	42.86		39.29	50.00	92.14	75.00	28.57
Combined	81.43	59.29	57.14	86.43	89.29		77.86	40.00	48.57	77.86	81.43

	Face2Face						NeuralTextures				
GoogleNet	38.57	77.14	62.14	52.14	72.86		27.86	70.00	61.43	52.14	62.86
Xception	27.14	82.14	82.14	37.86	84.29		27.86	76.43	84.29	39.29	75.00
DenseNet	45.00	52.14	90.71	77.14	27.14		32.86	48.57	88.57	74.29	20.71
Combined	82.14	47.86	55.71	82.86	84.29		76.43	41.43	42.86	67.14	80.71

	Pristine				
GoogleNet	72.86	45.00	57.14	62.14	55.71
Xception	85.71	39.29	28.57	75.71	30.71
DenseNet	72.14	65.00	22.14	44.29	85.00
Combined	41.43	75.71	73.57	50.00	35.00

Table A.2: Validation results from training the network five times for each feature extraction network, when training on all manipulation methods simultaneously.



(a) GoogleNet. *(b) Xception.*

(c) DenseNet. *(d) Combined.*

Figure A.1: Accuracy when classifying pristine versus manipulated videos using the different feature extraction networks, as well as all of them.

Appendix B: Equations

Given input $x = (x_1, x_2, \ldots, x_k)$, weights $\omega = (\omega_1, \omega_2, \ldots, \omega_k)$, bias b and an activation function ϕ, the output y from a node is given by:

$$y = \phi(\sum_{i=1}^{k} \omega_i x_i + b). \tag{B.1}$$

The RELU function:

$$\phi_{RELU}(a) \ = \begin{cases} a & \text{if } a \geq 0, \\ 0 & \text{if } a < 0. \end{cases} \tag{B.2}$$

Mean square error loss (typically for regression):

$$MSE = \frac{1}{n} \sum_{i=i}^{n} (y_i - y_i^*)^2, \tag{B.3}$$

where y is the true values or labels of what is being predicted, n is the number of these and y^* is the predicted output.
Binary cross-entropy loss (for binary classification):

$$BCE = -\frac{1}{n} \sum_{i=1}^{n} (y_i \cdot log(y_i^*) + (1 - y_i) log(i - y_i^*)), \tag{B.4}$$

with the same notations as above.

Filtering the image $I(i, j)$ with the filter $F(m, n)$ giving the output $S(i, j)$:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) F(m, n). \tag{B.5}$$