AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Master's Thesis

Praca Dyplomowa Magisterska

DeepFake: Algorithms for Fake Video Detection

DeepFake: Algorytmy Wykrywania Fałszywych Wideo

Author: Andrzej PUTYRA

Field of study: COMPUTER SCIENCE

Thesis supervisor: dr. Patryk ORZECHOWSKI

Kraków, September 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście, samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Andrzej Putyra

Abstract

Advances in computer vision have given rise to a multitude of methods for the manipulation of media. Although there are many ways in which this can be used to the benefit of society, such as is the case with art and science, there are also many opportunities for misuse. The multitude of methods developed in recent times for the purpose of altering or switching faces, such as DeepFakes, has significant potential to harm individuals and communities. Given this problem, it is crucial to develop and maintain methods capable of detecting state-of-the-art manipulations. This task of image classification lends itself strongly towards machine learning, which is one of the most promising and most popular solutions used for similar problems. In order to work towards a solution, I provide an analysis of the performance of cutting edge convolutional neural network models on the task of DeepFake detection and create a novel model with an architecture based on the best performing candidates. This network is smaller and less computationally expensive than the analyzed counterparts which inspired its creation while performing very similarly to those solutions. I show that a fusion of widely respected model design concepts can be leveraged to create models with comparable performance at lower cost, and how the power of a high quality dataset with various manipulation methods can be used to develop a more widely-applicable solution to the problem of altered facial video detection, while not compromising in terms of functionality for the specified task.

Contents

Chapter 1

Introduction

Our technological society is highly dependant on the existence and usage of digital media. Images and videos are used to capture important moments in time, convey emotions, share news, create art and spread information - among hundreds of other applications. The ability to recognize digital content as genuine is very important so as not to spread fake information, which could be used to influence the public in a variety of ways and cause harm to many people. As such, rapid advances in technology used to manipulate media or create synthetic imagery have a critical impact on our society.

One of the most popular fields of visual content manipulation is that of human faces. To us, faces are a means of identification, as well as a way to express emotion and share information, whether by means of emphasis or inherently. Faces are also a popular topic of research in computer vision. A multitude of publications and tools exist on the topic of facial structure, detection and reconstruction. These factors lent to the creation of multiple face manipulation techniques.

The most important distinction in types of facial manipulations is whether the facial expression or identity is altered. In the former case, the original subject stays in the video but their face moves according to a modified pattern, typically that of someone else. This is particularly useful when adding fake audio, where the subject says something other than what originally appeared in the video. In the latter case, the face in the video is swapped with that of someone else, so the identity of the subject displayed in the video is changed. One technology which can be used for both cases is that of DeepFakes.

1.1 What is a DeepFake?

A DeepFake is a type of synthetic media in which an existing image, audio recording or video of someone's resemblance is modified or exchanged with that of someone else. Powerful techniques from the fields of computer science can be leveraged to generate this kind of content quickly and create samples which are difficult to detect. Although tools for altering media in such a manner on a smaller scale have long existed, they have typically required manual work (with tools such as Photoshop) and as such were not considered to be scalable solutions for creating large amounts of altered imagery and videos. However with this relatively new technology anyone could quickly create a vast supply of such media. This fact gives rise to a growing need for solutions capable of detecting these forgeries, especially given their growing popularity.

The exciting field of facial image and video modification offers many advantages, such as applications in various sectors of the entertainment industry and academic research. These, however, are offset by a large space for nefarious use. Examples of such uses have quickly become apparent - most notably in the form of political propaganda, altered pornography and general fraud. These clear abuses of the technology have given rise to a need for intelligent identification of facial modifications having been applied to a video. As such, most of the current academic research in this field is centered around detecting such modifications rather than improving them. It is critically important that the ability to reliably classify real and fake content exists and is consistently improved upon. Better classifiers can also be used to create more realistic fakes with a higher potential to deceive, so this may be a more permanent problem.

1.2 Detecting DeepFakes

In order to detect a facial manipulation it is important to use all the available information effectively. Some are obvious due to visibly blurry faces or large artifacts. However, modern deepfaking algorithms are almost at a point where they are indistinguishable to human observers. There is a large amount of data encoded in an image which the human eye cannot recognize.

Image processing techniques allow the detection of slight changes in skin tone, eye color, rotation, inconsistencies in the shapes of various facial landmarks or lighting and improbable combinations of actions, i.e. mouth movement and eyes closing. Face alteration algorithms also tend to make use of blending to overlay the face with the rest of the head, which can also be identified.

1.3 Thesis Statement

The hypothesis of this dissertation is that extensively exploring current solutions and conducting studies to attain a deep understanding of their results is highly beneficial in the creation of an effective new one.

The main purpose of this paper is to explore the task of identifying altered recordings of human faces and create a high performance solution to this problem. The goals are as follows:

- Choosing suitable datasets
- Cleaning and preprocessing the data
- Creating a suitable training pipeline
- Reviewing available methods and their performance against the data
- Proposing a novel model architecture inspired by the best of the explored solutions to deal with this task
- Benchmarking the new solution against state-of-the-art methods

1.4 Organization of the Thesis

This thesis explores possible solutions to the DeepFake detection problem. Chapter 2 provides a gentle introduction to the most important concepts appearing in this thesis, such as deep learning and DeepFakes. Afterwards, its organization follows that of the goals laid out in the thesis statements.

Chapter 3 details the process of acquiring data necessary for the training and evaluation of the considered methods. It describes two of what I decided were the most viable options and explains the decision making process behind choosing one to focus on for this project.

The following chapter is complementary and further explores the topic of data acquisition. Both datasets pose challenges in the form of their size, structure and format. Chapter 4 is about the preparation of data, its augmentation and the preprocessing pipeline. It goes into detail about all the steps taken with the gathered media before it can be fed fed into the actual models.

Chapter 5 provides brief descriptions of all the state-of-the-art models I chose to inspect and compares them. Then, the models are trained on the prepared data and are thoroughly inspected based on a detailed analysis of the results.

The best architectures are used as inspiration for the new, less expensive models created in Chapter 6. Here, I explain the choices involved in their designs, differences between them and how they utilize the powerful features of the previous architectures. Then, I offer an in depth overview of the training process and review the results.

In Chapter 7 I test the new models as well as the best of the existing solutions. The outcome is compared to that of another state-of-the-art model trained on the same dataset. Using a recent tool in the field of deep learning model explainability, I show that all the models were trained correctly by proving that they focus on facial features in their decision-making processes. These experimental and analytical studies prove that the combination of various design concepts can be beneficial given the right circumstances.

Finally, I offer some concluding remarks, including a summary of what was done, an overview of the achievements and insights to possible future work, in Chapter 8.

Chapter 2

AI & DeepFakes

The approach I chose to deal with the DeepFake identification problem is rooted in Deep Learning, which is also used in the creation of DeepFakes themselves. These techniques leverage a complicated combination of complex computation, cognitive capabilities, large amounts of data and applied statistics to solve specific tasks. This chapter should serve to provide an overview of the necessary concepts.

2.1 Artificial Intelligence

Artificial Intelligence is often used as a blanket term to describe the application or implementation of intelligence in machines. It is used to describe specific computer science solutions which imitate cognition, such as using some form of reasoning, learning, deduction, decision making and problem solving. These are concepts classically associated with the human mind, which AI combines with the computational capabilities of machines.

2.2 Machine Learning

This subfield of AI is focused on systems which learn through experience and improve themselves. These kinds of mathematical models can make predictions, make decisions and solve problems without being explicitly programmed to do so. In some cases, models can be used for multiple different tasks depending only on the data they are trained on. There are a few different types of learning algorithms which are used in machine learning:

- **Supervised** algorithms analyze a dataset of labeled examples to learn a function which produces predictions about the output values.
- **Unsupervised** algorithms are approaches used when the data is not labeled or classified. This kind of learning is used to infer hidden structures in the data.
- **Reinforcement** algorithms progress by interacting with the environment and learning which actions are appropriate by discovering errors and rewards. This kind of trial and error approach allows the system to find the optimal behaviors with certain contexts and is typically used in agent-based-modelling.

This project will be utilizing supervised learning with labeled datasets, which is optimal because the nature of the DeepFake identification problem is a classification task.

2.3 Neural Networks

Artificial neural network architectures [1, 2] are structures of interconnected neurons and synapses (Figure 2.1) conceptually based on the human brain [3], creating mathematical models which suite the specific task. Artificial neural networks learn the expected behavior based on sample data, which can then be extrapolated for use on different sets of data.

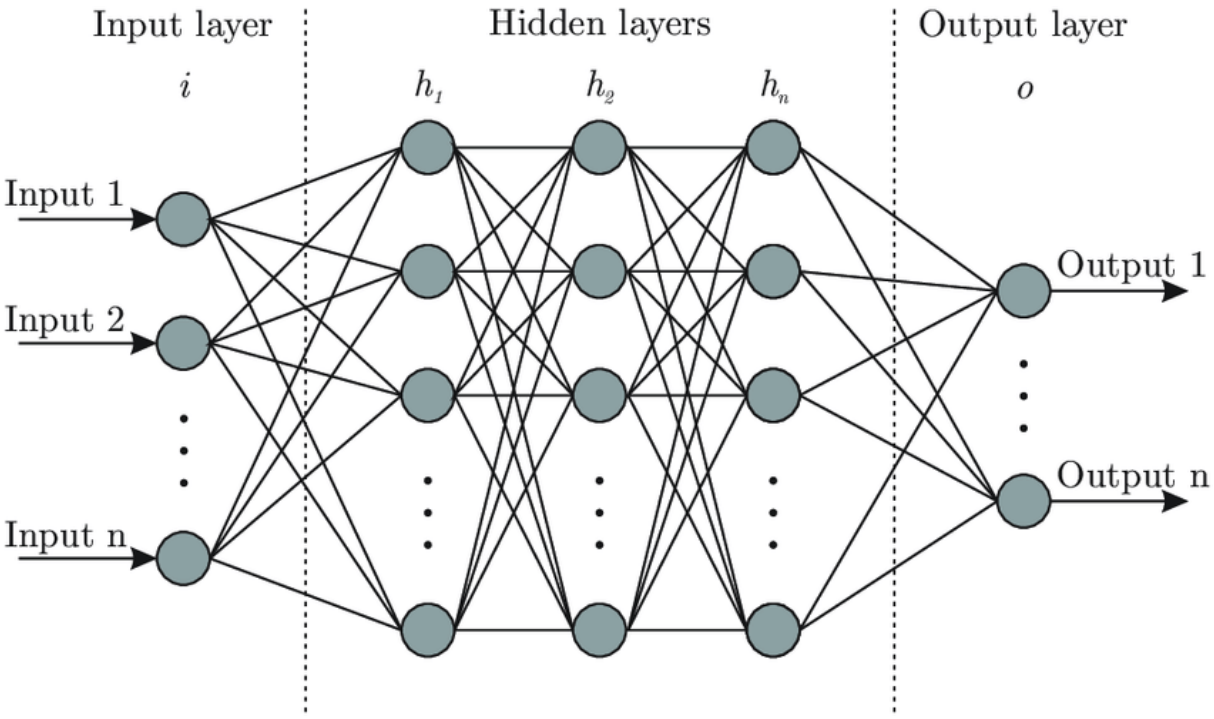

Figure 2.1: A simple neural network. Features are fed into the network as input values. Output values are returned depending on the task. The exact network architecture (# of inputs and outputs, hidden layers, connections, etc.) is dependant on the specific task. Source: [50]

2.4 Deep Learning

Deep Learning [4] is a subfield of machine learning, a natural progression developed due to the onset of big data. Classical machine learning approaches with smaller neural networks were ultimately found not to scale to datasets of enormous sizes. The adjective *deep* refers to the large number of layers utilized in these networks, so deep learning can be thought of as very large artificial neural networks. The result is a large, non-polynomial, highly nonlinear function suited to some specific tasks.

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [5, 6] are an extension of ordinary neural networks built for computer vision/image processing [7, 8]. Similarly to neural networks, CNNs take inspiration from biological processes - more specifically the visual cortex [9, 10]. They utilize a large number of filters which are moved across the image and used in convolution operations, which compute a degree of similarity with the filter in each region of the image. They are learned during training, and serve to find regions of interest in the image for further processing in following layers. These operations result in feature maps as seen on Figure 2.2 and show how much each part of the image matches the filter. One of their most common uses is image classification, which is essentially the topic of this project.

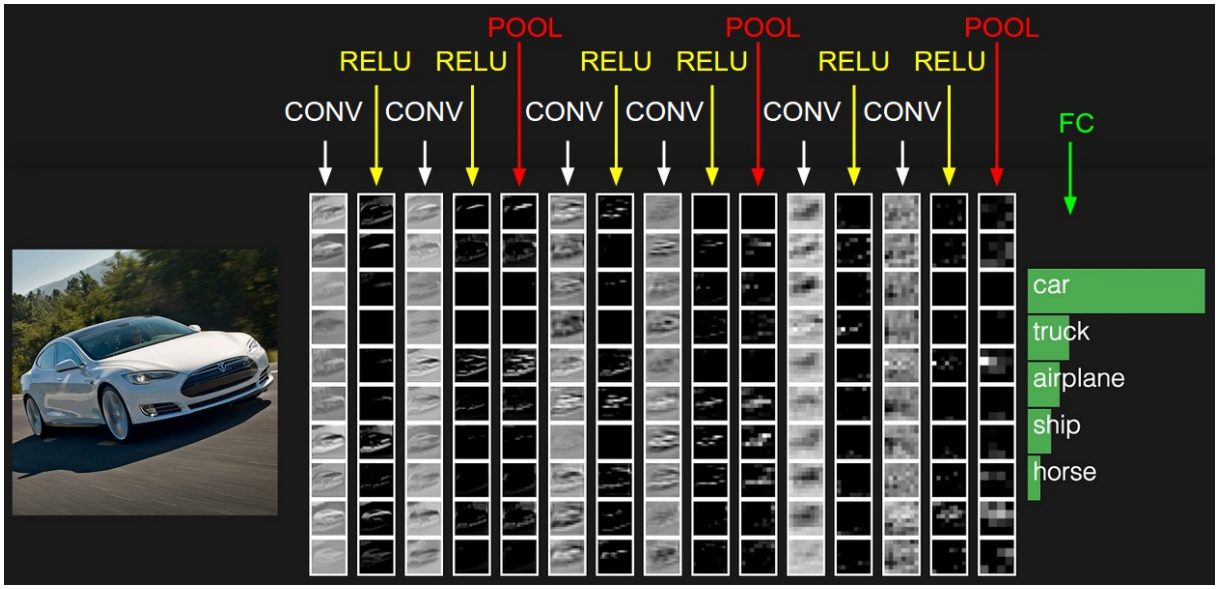

Figure 2.2: "The activations of an example CNN architecture" from Stanford University's cs231n course on convolutional neural networks (Source: [51]). The input image is on the left-hand side, the middle shows following feature maps along the processing path before arriving at the class scores on the right-hand side. In this case, the network states that the object in the image is most likely a car.

2.6 DeepFakes

As mentioned previously in the Introduction, *DeepFake* [11, 12, 13] is a term used to describe media where the displayed subject is altered in some way, either by way of their facial expressions, their identity, or the content of their speech. This is most often achieved used deep learning with generative neural networks, which serve to create something, in this case images, rather than to output some score or label. They most frequently use Generative Adversarial Networks (GANs) [14] which are a combination of two networks, where one is trying to fool the other. The generator creates fakes, while the discriminator tries to identify whether a sample is real or not. These types of architectures are difficult to train due to imbalances in the performance of the generator and the discriminator, which can arise during training. However, the resulting generator can do very powerful things, such as create DeepFakes. State-of-the-art manipulations are very realistic and dangerous in terms of their ability to deceive. Figure 2.3 has examples of both real and altered faces. It had been shown [17] that it is very hard for humans to identify these forgeries (around 50-65% accuracy depending on the level of compression).

Figure 2.3: An assortment of real and fake faces from the FaceForensics++ dataset [15]. To a human observer it can be very difficult to spot the DeepFakes.

Chapter 3

Datasets

It is no secret that neural networks require lots of data to be trained to any level of satisfactory precision. Deep neural networks are especially known as being dependant of vast quantities of data - preferably of high quality and fidelity with balanced characteristics, and relatively uniform class sizes.

In the case of detecting digital face alteration in images and video, the creation of such a dataset is particularly challenging. The first challenge is collecting data, which either requires scraping large amounts of available content in the internet for videos of faces or paying actors to play out certain scenes. In both cases, the next step is to apply video alteration techniques to swap faces in the collected videos. Finally, the dataset must be appropriately labeled. This process is detailed in Figure 3.1.

Figure 3.1: DeepFake detection dataset creation pipeline

There are multiple choices which can be made when creating such a dataset. These differences are visible in the two datasets I chose to work with for the purpose of this project.

3.1 FaceForensics++

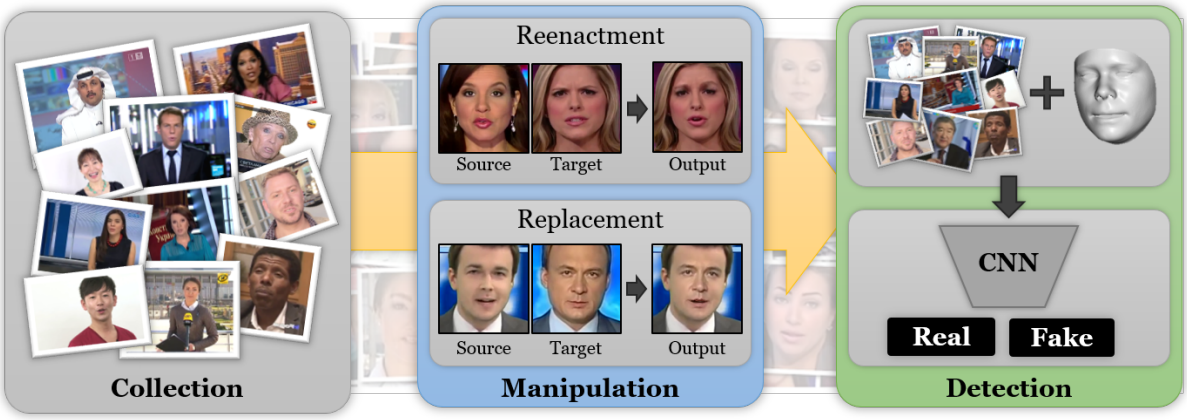

Figure 3.2: "*FaceForensics++* is a dataset of facial forgeries that enables researchers to train deep-learning-based approaches in a supervised fashion." Source: [18]

This dataset [15, 17, 18] was created for the general purpose of facial forensics. It consists of 1000 original video sequences sourced almost entirely from YouTube videos, which are manipulated in pairs into corresponding fake samples using four different face manipulation methods.

It also features the DeepFakeDetection dataset prepared by Google & Jigsaw [22], which is composed of over 3000 manipulated video sequences generated from originals featuring 28 individual actors in various scenes with different backgrounds. Because the dataset is hosted by the FaceForensics team and integrated into the FaceForensics benchmark I consider it to be part of the FaceForensics++ dataset, despite not being in the source [18].

3.1.1 Organization

FaceForensics++ Dataset
- Original Sequences
- Manipulated Sequences
  - DeepFakeDetection
  - Deepfakes
  - Face2Face
  - FaceSwap
  - NeuralTextures

Figure 3.3: The FaceForensics++ dataset directory tree structure

The assortment of videos is prepared in such a way that it can be downloaded according to the users specification, i.e. with a chosen level of compression. This is one of raw, c23 or c40, where c23 and c40 are H.264 standard compressed videos using a compression rate factor of 23 and 40 respectively. I chose c23 as a compromise between disk space and the characteristic by which a higher level of compression generally leads to increased difficulty in classification tasks. It is also possible to download images rather than videos, as well as to download only selected parts of the dataset. Labelling is provided by means of the directory tree shown in Figure 3.3, wherein real samples are part of the original_sequences folder, while fake samples are found in the manipulated_sequences directory.

Training, validation and test splits are provided [16] by the FaceForensics team for the data gathered by them with 72% of data being intended for use in training while 14% is used for validation and testing respectfully. Unfortunately, there are no splits for the additional dataset provided by Google & Jigsaw. As such, a script was prepared to generate metadata files listing all the videos in a folder with their respective split, as well as the name of the original video for each of the manipulated sequences. The data was split randomly according to the same ratios (0.72, 0.14, 0.14).

3.1.2 Characteristics

A big advantage of the dataset is the variety of manipulation methods it features:

- DeepFake [11]
- Face2Face [23]
- FaceSwap [24]
- NeuralTextures [25]

The original FaceForensics dataset featured the first three methods, while manipulations using neural textures were added in FaceForensics++. Each of them have equal representation (1000 manipulated video sequences based on 500 pairs of unique videos from the original 1000 videos) with the exception of DeepFakes, which are additionally augmented by the Deep Fake Detection dataset prepared by Google & Jigsaw [22] (around 3000 videos).

Although the purpose of this project is to detect deepfakes, exposure to different methods will allow the model to better generalise and look for a broader array of features when making decisions. It is also useful in allowing the existence of a variety of manipulated samples based on a single original video.

3.2 Kaggle DFDC

Figure 3.4: The Deep Fake Detection Challenge hosted by Kaggle. Source: [20]

The Kaggle Deep Fake Detection Challenge [20] was assembled with the intention of spurring researchers to invent innovative new technologies and techniques for detecting deepfakes and other forms of altered media. The challenge featured a $1,000,000 prize pool to be split among top contestants with the caveat that their solutions were to be made public. The challenge was sponsored by AWS, Facebook, Microsoft, the Partnership on AI's Media Integrity Steering Committee, and various academics.

With close to 100,000 videos spanning almost 450GB of data, it is fair to say that the dataset is massive.

3.2.1 Organization

The dataset is downloadable either in it's entirety or in parts (50 archives of around 10GB each). Of those parts, each has around 2,000 10-second video clips with randomised names. Each of these directories has a metadata file in the JSON format which specifies

whether the videos are real or fake as well as a reference to the original in the case of fakes. The metadata files also have a 'split' key for each video, however it is set to 'train' for all videos. I decided to assign splits randomly to the real videos with a ratio of 80%, 10%, 10% to training, validation and test sets respectively. Each corresponding fake video had its split changed to that of its original.

3.2.2 Characteristics

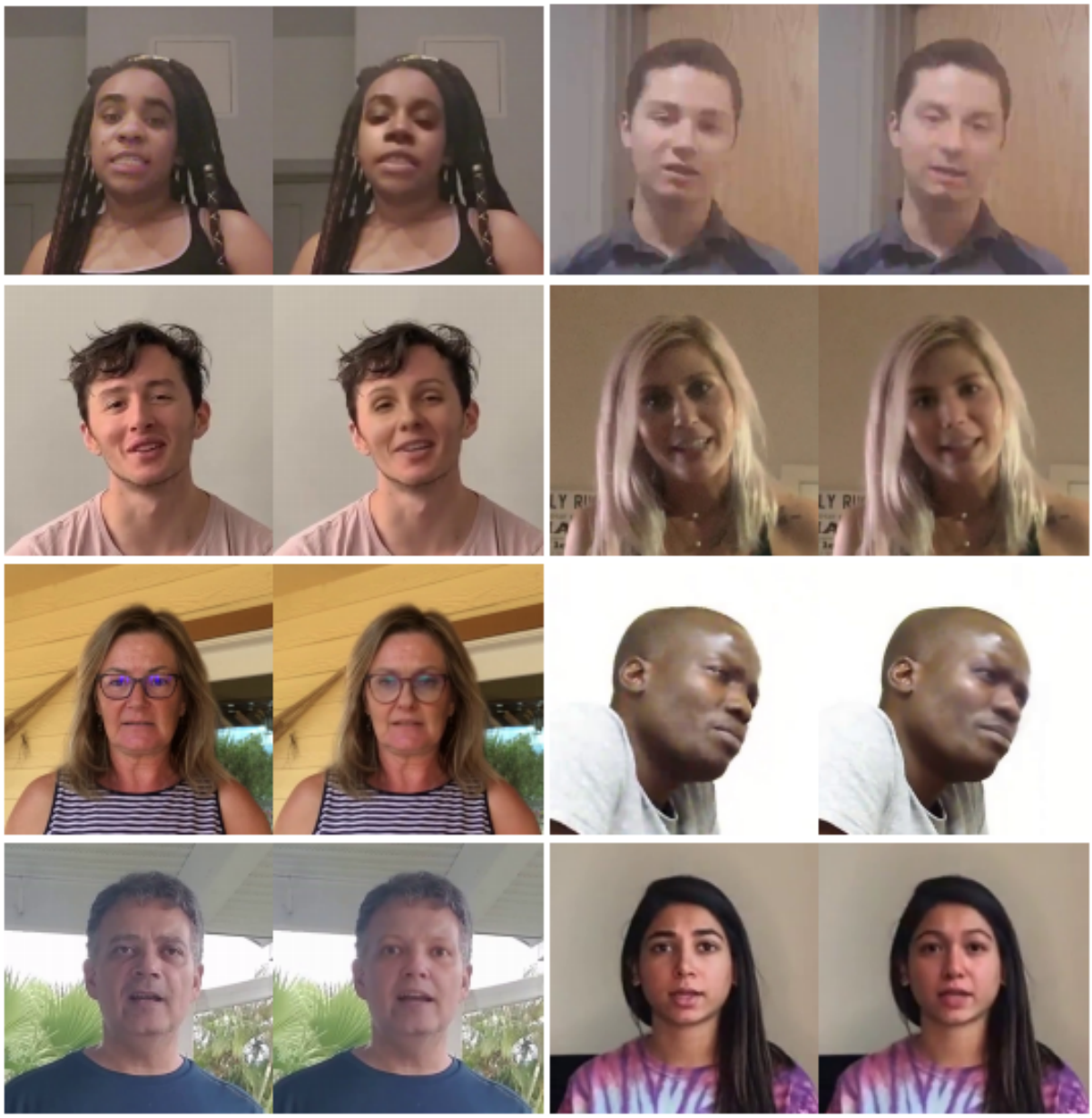

Figure 3.5: Some example face swaps from the Kaggle DFDC preview dataset. Source: [19].

The individuals appearing in the videos are crowdsourced actors. Each participant submitted a set of videos with varying backgrounds, lighting conditions and head poses. In addition, they answered an assortment of pre-specified questions which allowed a wide

range of facial expressions while speaking. These videos were trimmed from the start and the end (many subjects spent the first and last few seconds getting into or out of frame) and cut into 10 second clips. Facial manipulations were performed using various undisclosed methods on pairs with inferred similar characteristics, such as skin tone, facial hair and the presence of glasses. Some examples are shown on Figure 3.5.

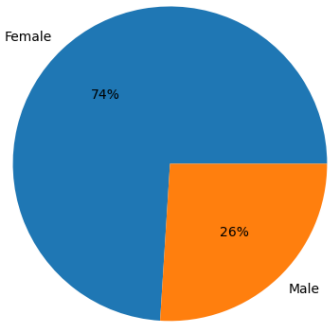

Figure 3.6: Approximate gender distribution in the Kaggle DFDC preview dataset.

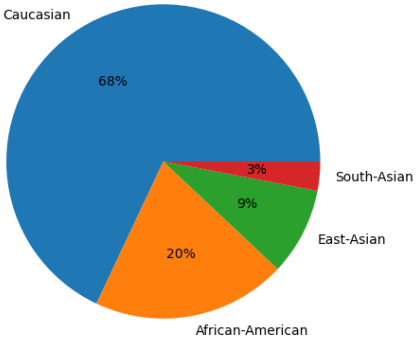

Figure 3.7: Approximate race distribution in the Kaggle DFDC preview dataset.

During the preparation of the Kaggle DFDC challenge and dataset a paper was released on a smaller subset of the accumulated data. The Deep Fake Detection Challenge Preview Dataset [19] is a dataset of 5000 videos (which also appear in the full version) and the publication offers some insights as to the data collection process, the collected data and the process of creating manipulations. An approximate overview of gender and race distributions in this partition of the dataset is shown on Figures 3.6 & 3.7, showing a skew towards females in the gender axis and caucasians in the race axis.

The biggest advantage of this dataset is its size. To the best of my knowledge, no other face manipulation dataset comes close to the size of Kaggle DFDC. With modern deep learning models requiring more and more data this is a big advancement to the field of research. However, developing such a large dataset is difficult and comes at the price of some issues. Examining the dataset reveals that quite a lot of samples have facial manipulations of very low quality. Another, less frequent issue is the existence of glitchy samples.

One problem I noticed occurs in videos with multiple faces. The manipulation is often not applied to one single face over the course of the entire video, but instead changes between available faces. An example of this behavior is shown on Figure 3.8. I believe the

algorithm used in these videos applies the facial manipulation to the object recognised as a face with the highest probability, which would cause it to sometimes switch the target.

Figure 3.8: An example from the Kaggle DFDC dataset where the target face being manipulated is changing between different faces in the video.

Figure 3.9 presents a similar problem appearing in some videos with a single actor. On some frames no facial manipulation is applied to the subject, i.e. it is flickering between the real and fake faces. This is obviously very problematic for training, as there is no way to tell on which frames this will occur. It is worth noting that in the case of this particular video there is a painting of a female face in the background and in one of the 6 frames shown in Figure 3.9, a deep fake is applied to it. Therefore there is reason to believe that what is happening might be the same issue as in videos with multiple human actors.

Figure 3.9: An example from the Kaggle DFDC dataset where the manipulation is flickering. The top shows consecutive frames from the original video, while the bottom shows consecutive frames from the fake.

3.3 Conclusions

Due to the mentioned problems which are apparent in the Kaggle dataset I chose to make use of the FaceForensics++ dataset in this project. The large size of Kaggle coupled with many examples of low-fidelity manipulations and large amounts of videos with multiple faces make it difficult and resource-intensive to work with. The overall smaller size of FaceForensics++ also makes it possible for me to use further optimizations such as those described towards the end of chapter 4.1.4, which are dependant on hardware limitations. The usage of highly-varied manipulation methods in FaceForensics++ is also a strong benefit. Even with this inherent variety, it has two categories of DeepFake videos, one of which is based on original sequences which are completely separate from those used in the creation of all the other manipulations. This will be very helpful given that DeepFake videos are the main topic of this project. Some tests were done with the Kaggle DFDC dataset, however due to its scale it wasn't feasible to work with long-term.

Chapter 4

Preprocessing

Before data can be used in the networks it has to undergo multiple preprocessing steps. This chapter is about the transformations used to make the dataset usable in my training and classification pipeline.

4.1 Challenges

The primary problem to tackle is extracting the relevant faces from the provided videos. This task poses multiple challenges.

4.1.1 Face Location & Cropping

The first challenge is finding the location of the faces in each frame. This can be accomplished using one of many available models or libraries to find bounding boxes. Research and testing lead to finding that a mobilenet [27] solution worked best [31, 32, 33], largely due to time and memory constraints posed by the hardware available to me. A mobilenet is a type of deep neural network developed specifically for mobile and embedded vision applications with a more streamlined architecture and more efficient convolution operations. The size and speed of the mobilenet comes with a trade-off in performance, especially when objects of interest are further away [33], however given that the dataset guarantees that subjects are always in focus and relatively close to the camera this is not a concern. The particular model I chose [29] is a single shot multibox detector (SSD) [28] implemented in TensorFlow and trained on the WIDER FACE dataset [30]. SSDs generate scores for the presence of objects in a number of default boxes and improve the bounding boxes

afterwards. As such, they can find multiple objects and encapsulate the entire process in a single network. The network is relatively fast and small while working with satisfactory precision, providing a list of bounding boxes with corresponding probabilities of being a face as an output. It does not provide additional information, such as coordinates for certain facial landmarks, which some other solutions include. Using such information would partially tie the model and its performance to the face detection model, which I chose against. It is also relevant to note that finding these landmarks is less reliable when dealing with manipulated faces, some of which are quite distorted.

The bounding boxes returned by the model are the coordinates of corners forming a rectangle around the detected face. This rectangle only contains a part of the face ending approximately on certain facial landmarks. In order to obtain a full view of the face it is necessary to use an additional margin to scale up the bounding box. The margin I chose to use in this project is 0.3, i.e. the actual bounding box is 1.3 times larger than indicated by the coordinates returned from the face detector. This number is subjective, as different face detectors will return slightly different regions of the face. Convolutional neural networks use square-sized inputs, so it is also worth increasing the smaller dimension to that of the larger one so as to form a square. The effects of my face detection and extraction method are exemplified in Figure 4.1.

Figure 4.1: Detecting and extracting faces from the FaceForensics++ dataset. Real samples are on the left while fake samples are on the right.

4.1.2 Multiple Faces

The next challenge is dealing with videos in which multiple faces are detected (such as those displayed in Figure 4.2). While this is acceptable when using the model to detect

manipulations where each face can be considered separately, it is problematic for training. In videos with multiple faces it is unknown how many of the faces are altered and which ones are altered. My solution is two thresholds, *mobilenet_score_threshold* for face detection probability and *multiple_face_threshold* for the amount of times a second face is detected in the video. A video is considered to have multiple faces if it detects a probable second face in a number of frames (scaled to the particular video's length) exceeding the *multiple_face_threshold*, after which it is discarded from the pool of samples used during training.

Figure 4.2: Samples of frames from the Kaggle DFDC dataset with multiple faces.

4.1.3 No Faces

Finally, there exists the possibility of no faces being detected in a video, i.e. none of the choices have a satisfactory probability of being a face. This can be due to compression, artifacts from manipulation obscuring the facial features, obstruction, or any number of other possible problems. Thankfully, there is a guarantee during training that each video in the dataset has at least one face. As such, the object detected as a face with the highest probability can always be considered to be a valid face, regardless of the probability threshold for other faces.

4.1.4 Dataset Preparation

With these problems solved, bounding boxes were extracted for the entirety of the FaceForensics++ and Kaggle DFDC datasets. Fortunately, the fake videos in the latter have one-to-one correspondence with their real counterparts. As such, the bounding boxes detected for the real videos can also be used for all the fakes generated from them. After this operation they were saved in the JSON [34] format as coordinates for each detected face in every frame of each particular video. Doing this allows for large improvements to training time, as videos are used multiple times during training.

After testing it was found that this was insufficient. To increase speed even further, some of the videos had their faces extracted as PNG images according to those bounding boxes so those videos don't need to be processed during runtime, which is quite computationally expensive. More specifically, all real videos and only some fake videos were preprocessed in this way. The inability to pre-extract manipulated faces from fake videos is due to physical memory constraints (extracted images occupy much more HDD space than videos) and there being much more of them, while real videos are used much more often in training due to reasons explained in the subchapter 4.3 on batch creation.

4.2 Data Augmentation

Data augmentation is well known to significantly improve performance with CNN tasks. Although the used dataset already presents a wide variety of features, such as race, gender, facial structure, environment and camera quality, it will still be very useful to apply certain transformations to the input images during training to improve model performance. The effects of my data augmentation process can be seen on Figure 4.3.

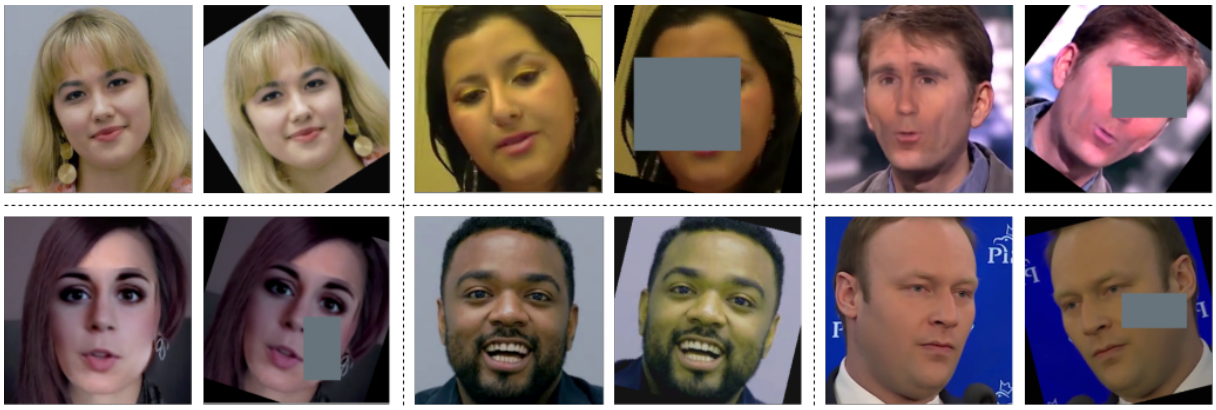

Figure 4.3: Input images from the FaceForensics++ dataset before and after data augmentation.

As mentioned previously, the bounding box of detected faces is expanded 1.3x to encapsulate a larger feature and capture some of the background. This gives the networks an increased amount of information (such as about skin tone, hair color and lighting). For example, given a smaller frame for the face of the woman in the bottom-left image, you might miss how the eye and eyebrow cover some of her hair in the image, which is obviously impossible. Most of the augmentations are applied with some probability which is why they do not appear in all cases, such as is the case for the gray rectangle (random erasing). All the techniques used for data augmentation are as follows:

- **Dropout**

  The usage of dropout layers to reduce generalization error and combat overfitting was introduced in "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"[36]. The idea is to randomly disable some neurons in the network during each forward pass while training a model, which means a part of the network isn't considered each time and the importance of the rest of the network is increased for that sample. In this project, dropout is applied just before the classifier.

- **Random Horizontal Flip**

  A simple transformation which horizontally flips a given image with some probability (default of 0.5). It allows the model to learn certain asymmetries in the faces as well as being exposed to them from different perspectives. Additionally, it balances the dataset in terms of the direction the recorded subjects are facing.

- **Random Rotation**

  Rotates the input image by some angle. By default, this transformation does not resample or expand the image after rotation and uses the center of the image as the center of rotation. These were the settings used, along with the degrees parameter being a random value in the range of (-40, 40).

- **Color Jitter**

  Allows augmentations to the training data color palette by modifying the brightness, contrast, saturation and hue of an image. Each of these has a respective range of modification. Applying color jitter during training makes the model more robust to traits related to color such as skin tone, hair and eye colors, as well as environmental features like room lighting.

- **Random Erasing**

  This technique selects a random rectangular area in the image and erases its pixels with certain values. It offers improvements in model performance by making it more robust to occlusion. In practice it is similar to applying dropout at the image level, which also helps to decrease dependence on specific features. It was introduced in "Random Erasing Data Augmentation"[35] and can be observed on Figure 4.4.

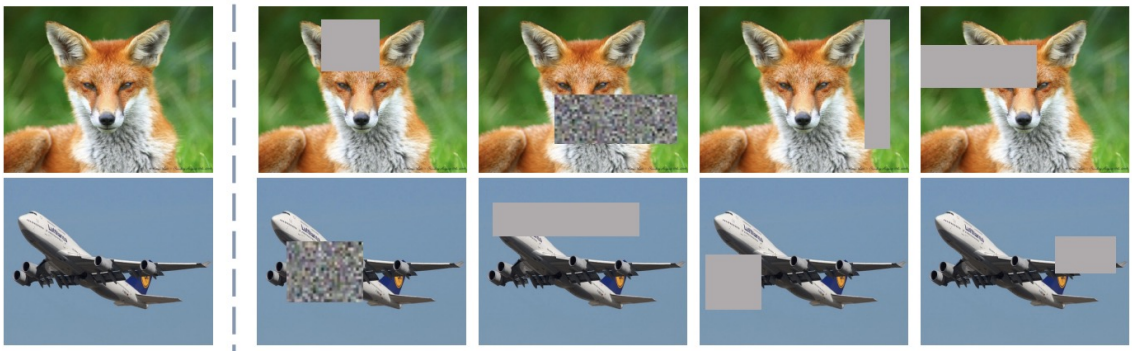

Figure 4.4: An example of random erasing on two input images. Source: [35].

4.3 Batch Creation

The data being used in this project does not have a simple structure such as a single folder of images. Rather, there are many folders of images and videos which must be matched in complex pairs by verifying their metadata. In addition, not all samples are homogeneous: while some are stored as folders of consecutive face images, others exist in the form of videos with accompanying bounding box files and must have the faces extracted at runtime. Because of the nature of the data, as well as the complexity of the task, it is necessary for me to create custom methods for batch creation.

4.3.1 Batch Types

During training, pairs of fake and real videos are gathered together in a dictionary, along with information about their lengths, whether they are stored as images or videos with bounding boxes, and the absolute paths to that data. Each real video has various fakes generated from it, so they appear in multiple pairings. Afterwards, they are processed based on the chosen type of batch. In the mentioned formulas *epoch* is the current epoch, *end* signifies the last frame of the video, *video_length* is total number of frames in the video, *total_epochs* is the total number of epochs to be run during training, *batch_size* is the batch size (which is an even number) and n is a chosen parameter (in this case 8).

- **Batch of Frames from a Pair of Videos**

  The first type of batch contains frames from a single matching pair of videos, i.e. a manipulated sequence and the original it is based on. In this case, each batch processes a few different, equally spaced frames per video during each epoch, allowing the model to gain more specific information about the videos. The numbers of the specific frames chosen from each video are calculated according to the following formulas:

$$frame\_numbers = [epoch : frame\_step : end]$$

$$frame\_step = \left\lfloor \frac{video\_length - total\_epochs}{0.5 * batch\_size} \right\rfloor$$

- **Batch of Frames from Various Pairs of Videos**

  The other type of batch contains pairs of frames from various matching videos, i.e. single frames from manipulated sequences and the originals they are based on. This means the model gains a smaller amount of information about a larger variety of videos in each batch. The number of the specific frame chosen from each of the videos is calculated for it based on the following formula:

$$frame\_number = (n * epoch) \bmod (video\_length - total\_epochs) + \frac{n * epoch}{video\_length}$$

4.4 Proposed Solution

4.4.1 Finding Bounding Boxes

In order to prepare the dataset for usage during training it is imperative to extract the bounding boxes with the locations of borders of the detected faces for each frame. First, I initialize the mobilenet SSD face detector, prepare a list of paths to all the real and fake videos and create a directory for the bounding box data JSON files. Then, for each video I find the bounding boxes and save them as below:

1. Create an empty JSON file which will be used to store a dictionary with the bounding box data for each frame and a multiple faces flag. The name of the file matches that of its source video.

2. Open the video at frame 0. Create an empty dictionary.

3. Retrieve an array with a number of frames equal to the specified batch size.

4. Input the array of frames into the mobilenet SSD face detector. It will return a list of bounding boxes and probabilities of there being a face in them sorted by those face recognition scores.

5. Keep only the highest scoring face and any others if they are are above the threshold I specified (45%). We can always keep the highest scoring face because the dataset guarantees there are always people in frame in the videos.

6. For each of those faces, expand the size of the bounding box by a factor of 1.3x. If the resulting box is not a perfect square, extend the length of smaller dimension to that of the larger one.

7. Add each face from each frame to the dictionary. Their locations are stored as top, bottom, left and right (the borders of the bounding box).

8. Repeat steps 3-7 until there are no more frames in the video.

9. Count the number of frames where there is more than one face location recorded in the dictionary. If this value exceeds my specified threshold (10%), set the multiple_faces flag to true. Videos with multiple faces are removed from the pool of samples.

10. Close the video and write the bounding box data to the JSON file.

4.4.2 Extracting Faces

If there is physical disk space available, it would be even better to save the faces as images, so as to cut down on processing overhead during training. For videos where this is not done, it becomes necessary to read the bounding box data from a file, open the video, read the requested frames and crop them according to the borders specified by the bounding box data each time the video is used, which is either once per epoch or multiple times in the case of real videos. The following details my full extraction process, which I used to preprocess as many videos as possible.

First, create a list of the relevant folders of videos. Then, while processing each folder create a list of all the videos in that folder. Then, for each video:

1. Create a directory for the face images extracted from that video.
2. Find its bounding box data JSON file based on its name.
3. Open the file and decode the dictionary.
4. Open the video at frame 0.
5. Retrieve a frame. For each box at that frame specified in the dictionary crop the specified area of the image. Save those crops.
6. Repeat step 5 until there are no more frames in the video.
7. Close the video.

4.4.3 Creating Batches

I created a batch creation method which prepares batches of faces to be processed by the models. There are two types of batches as was explained in 4.3, however the process is similar. Before calling the method a data dictionary is created based on the type of batch. This contains the numbers of the frames which should be considered as well as the video path.

The first part of the method checks whether the faces from the video have already been extracted and saved locally. If so, they are read from their respective files and assembled in an array. Otherwise, the faces are retrieved from the video based on its bounding box file in a process like that detailed in the previous subsection and assembled in an array.

Next, the faces are resized according to the input size of the used model. Then they get cast to tensors, normalized and are allocated memory on the Graphics Processing Unit (GPU).

Finally, data augmentation is applied if the model is currently being trained. The full pipeline of the batch creation process can be seen on Figure 4.5.

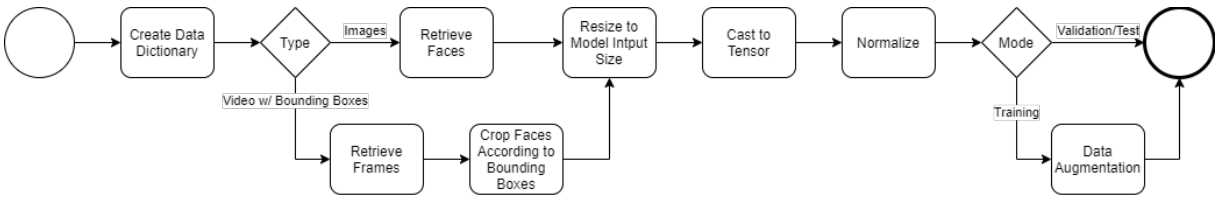

Figure 4.5: The batch creation process

Chapter 5

Transfer Learning - A Review of State of the Art Models

A common method for solving neural network tasks is to use available, state-of-the-art models with pretrained weights and train them using transfer learning. For the purpose of this project it will also be beneficial to see how various techniques perform on the task in order to later choose an appropriate model architecture. Finally, this will serve to gain a baseline for comparison.

5.1 Model Overview

5.1.1 Xception

Inspired by Inception modules, which can be considered a middle ground between the regular convolution and depthwise seperable convolution operations, the Xception model [37] makes use of a modularized architecture which replaces those with depthwise seperable convolutions. This operation consists of a depthwise convolution followed by a pointwise convolution (Figure 5.1) and can be understood as an Inception module with a maximum number of towers, i.e. one spatial convolution for each output channel of the 1x1 convolution (Figure 5.2), with the only difference being the order. As such, this model was named Xception for "Extreme Inception".

5.1.2 InceptionV3

It is well known that increased model size and computational cost often lead to better performance. However, there are ways to improve network behavior more intelligently. The original Inception network [38] put a larger focus on computational efficiency and a low parameter count by introducing the Inception module (Figure 5.3), while the follow up paper on the Inception-v2 and Inception-v3 models [39] introduced improvements. One example is replacing larger, more expensive convolutions with mini-networks, as shown on Figure 5.4.

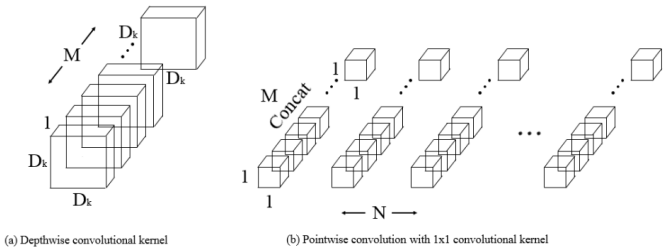

Figure 5.1: Depthwise seperable convolution. Illustration by Xiong Lin.

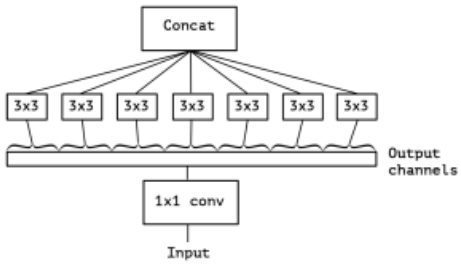

Figure 5.2: "An "extreme" version of the Inception module, with one spatial convolution per output channel of the 1x1 convolution." Source: [37]

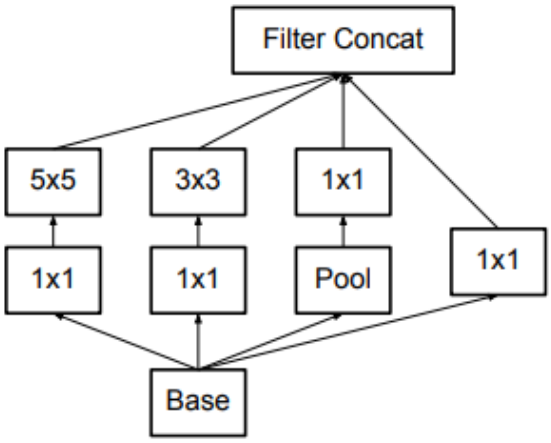

Figure 5.3: The original Inception module

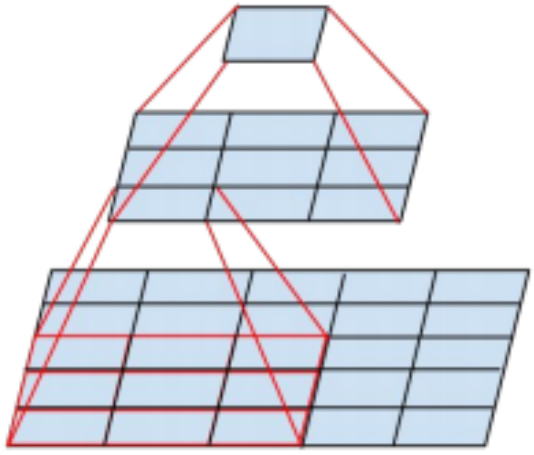

Figure 5.4: A 5x5 convolution can be replaced by two stacked 3x3 convolutions. Source: [39]

5.1.3 EfficientNetB5

EfficientNets [41] are a family of models created with a high focus on scalability. By carefully studying model scaling it was discovered that the relationships of network depth, width and resolution are key factors in improving performance. A new method was created for uniformly scaling these parameters via a *compound coefficient*, as shown on Figure 5.5. The paper boasts significantly improved results on the ImageNet task while using fewer parameters. For the purpose of this project I chose the B5 version due to having a relatively similar parameter count to the Xception and Inception V3 networks.

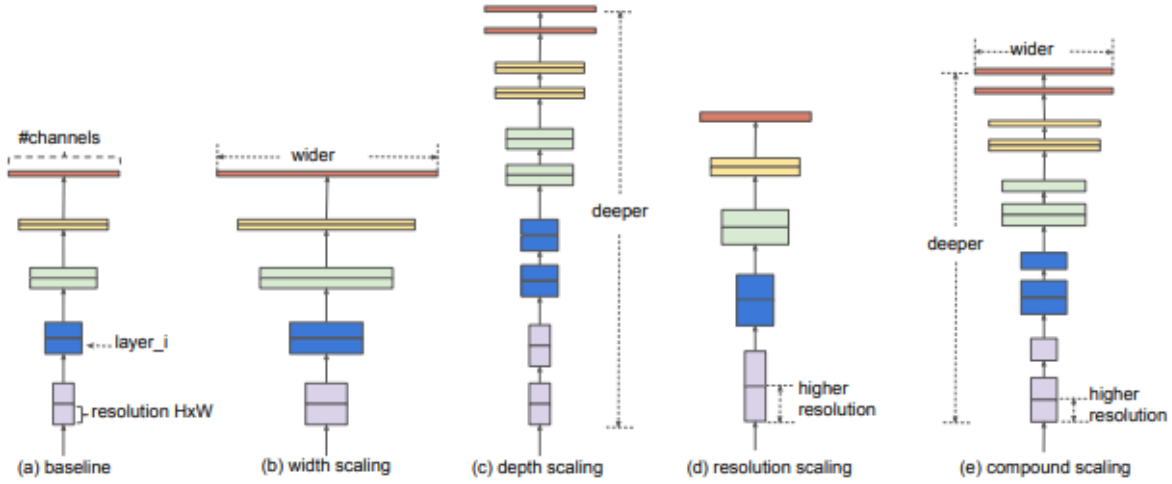

Figure 5.5: Compound scaling for control of network depth, width and resolution. Source: [41]

5.1.4 ResNet152

Deep neural network architectures are very difficult to train. Increasing network width or adding layers can often lead to a degradation in performance. The residual learning framework improves the training process of deep networks and allows them to safely gain accuracy while considerably increasing depth. This is accomplished by using residuals in the building blocks, which is actually just the input passed through and added to the output of such blocks. ResNet152 is the 152 layer variant, is the largest of the models introduced in the publication [42], and was chosen as an example of a large network.

5.1.5 ResNeXt101_32x8d

The ResNeXt architecture [43] is a highly modularized extension of the ResNet concept, resulting in a multi-branch architecture with few hyperparameters to set and a new dimension (in addition to depth and width) in the form of *cardinality*.

These models perform better on benchmarks than their ResNet equivalents. A comparison between the ResNet and ResNeXt blocks is shown on Figure 5.6. The ResNeXt101_32x8d version features a cardinality of 32 (grouped convolutions or branches), with 8 dimensional filters. It is the largest of the state-of-the-art models chosen for testing.

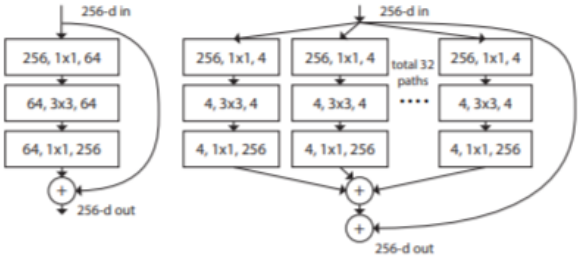

Figure 5.6: "**Left**: A block of ResNet. **Right**: A block of ResNeXt with cardinality 32, with roughly the same complexity. A layer is shown as (input channels, filter size, output channels)." Source: [43]

5.2 Model Comparison

The models were chosen in such a way so as to test a variety of state-of-the-art deep convolution network technology. Table 5.1 showcases some basic differences in terms of model sizes and performance.

Model	**Input Size**	**Params**	**Size**	**top-1 err (%)**	**top-5 err (%)**
Xception [37] [45]	299x299	22.9 M	89 M	21.1	5.7
InceptionV3 [39] [44]	299x299	23.83 M	91 M	22.55	6.44
EfficientNetB5 [41] [46]	224x244	28 M	111 M	16.3	3.3
ResNet-152 [42] [44]	224x244	60.19 M	230 M	21.69	5.94
ResNeXt-101 [43] [44]	224x244	88 M	347 M	20.69	5.47

Table 5.1: A comparison of the chosen models in terms of parameter count, input size, model size and top-1/top-5 performance on the ImageNet task. Model input sizes and parameter counts are obtained from their respective model sources. Size is model's size in memory (saved weights), while the accuracy scores are taken from the locations of their pretrained models. Top-1 and top-5 errors mean how often the correct answer wasn't in the top 1 or 5 predictions of the classifier. Image Each model is listed with its source and the source of the pretrained model that was considered.

Some similarities can be drawn in between the networks based on their architectures. One group is Xception and InceptionV3, which make use of modular structures with large amounts of seperate depthwise and spatial aggregation operations while having a similar size. The other group is comprised of models focused on residual learning - the ResNet152 and ResNeXt101 models. This technique allows building deep neural networks with many layers, which is reflected in the size of these two models. The final case is the EfficientNetB5, which uses different concepts and was made with a rather different goal.

Interestingly, an XceptionNet was used by the FaceForensics++ [18] team for their best benchmark on the dataset. Based on Table 5.1, the chosen EfficientNet seems likely to be the best model. Introduced in 2019, it is the newest of the models mentioned here and seems to significantly outperform the others in image classification on the ImageNet task, despite being on the lower side in terms of model size. Figure 5.7 compares the EfficientNet family to other popular models.

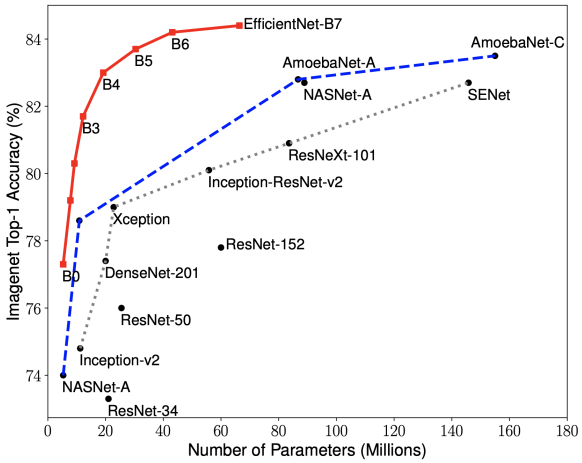

Figure 5.7: Graph comparing top-1 accuracy on ImageNet vs model size for various models. All models mentioned in this section are included with the exception of InceptionV3. Source: [41]

5.3 Results - Model Performance

In order to conduct a fair comparison of the models it was important to establish a relatively homogeneous testing environment. Due to hardware and time constraints it was simply not possible to train full networks from scratch. As such, I tried to unfreeze similar portions of the networks.

Due to a high variance in network structures and total sizes it was hard to find a fair way to compare them while transfer learning. I ultimately decided to unfreeze roughly equivalent portions of the networks in terms of their filter count (around 90% for each model). A breakdown of the unfrozen layers can be seen in Table 5.2.

	xception	**inception_v3**	**efficientnet_b5**	**resnet152**	**resneXt101**
frozen	stem (2xConv2D)	stem (5xConv2D)	stem (1xConv2D)	stem (1xConv2D)	stem (1xConv2D)
	blocks 1:3		blocks 1:9	blocks 1:2	blocks 1:2
unfrozen	blocks 4:12	inception modules	blocks 10:37	blocks 3:4	blocks 3:4
	2xSeperableConv2D				
	fc layer	fc layer	fc layer	fc layer	fc layer

Table 5.2: Overview of unfrozen parts of models. The table showcases some basic information about the network's building blocks/modules and convolution blocks. Other features such as kernel size or batch normalization and activation layers are not included for the simplicity of the comparison.

The networks were trained for 30 epochs using batches of frames from various pairs of videos from the FaceForensics++ dataset with a stable set of hyperparameters for each network. Each model was trained with two different optimizers - SGD with a momentum of 0.9 and RAdam, a relatively new adaptive learning rate optimization algorithm which focuses on the learning rate warmup heuristic. RAdam purports to decrease variance in the early stages of learning, stabilize training and accelerate convergence. In an environment with very limited resources these are very desirable traits. In both cases the initial learning rate was set to 1e-3 (for all unfrozen parameters), which is the default for RAdam. Additionally, the learning rate was adjusted by means of the ReduceLROnPlateau scheduler set to reduce the learning rate by a factor of 0.5 if an improvement wasn't observed in the last 2 epochs (patience = 2). Due to the large size of the ResNeXt101 network (even while only partially unfrozen), the maximal batch size which could fit in GPU memory was 12. As such, each of the models was trained with a batch size of 12. Validation uses 7 equally spaced frames from each video. All models have their final fully connected layer connected to a single output neuron and use binary cross-entropy as their loss functions.

5.3.1 Comparison of Model Training Performance

Figure 5.8 displays an overview of the training process for the state-of-the-art models. All the values collected and used in the graphs are averages across training and validation epochs respectively. The threshold for classification is set to 0.5, and videos are classified based on the average of outputs for all frames considered from that video for the purpose of the balanced accuracy metric.

Generally, the models are seen to improve over time and approach their plateaus. The only outlier is the EfficientNet_b5, which plateaus much earlier and at a much higher loss value than the others. Moreover, the slope of the loss graph does not indicate further improvement is possible, while all the other models are still on a downward slope. The Inception V3 model seems to be the most stable during training. The most import metric is the balanced accuracy, because the main goal of the models is to correctly classify videos as being real or fake. In this case, performance is quite good because all models achieve >95% accuracy. There doesn't seem to be much of a difference in results between the optimizers.

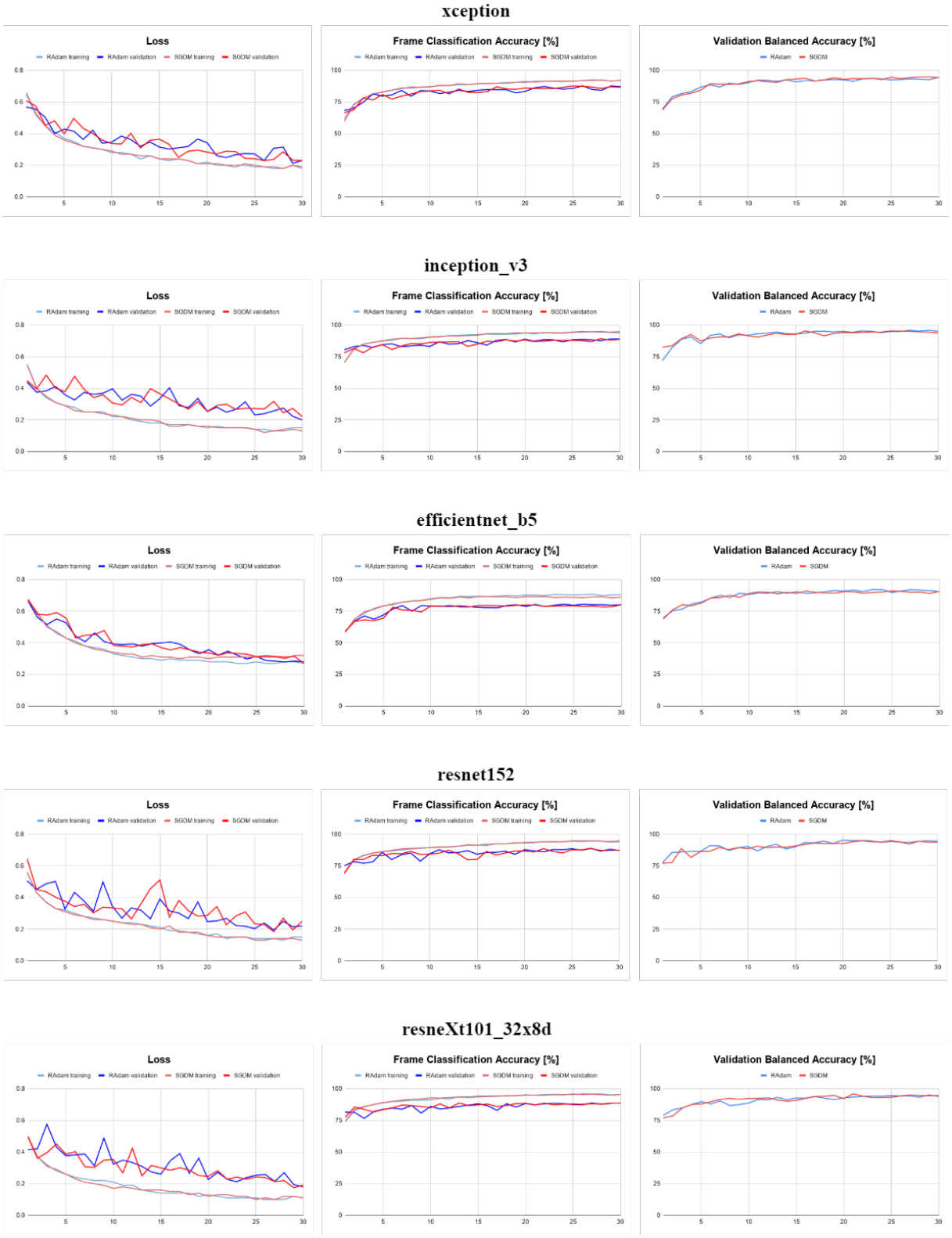

Figure 5.8: An overview of training with some state-of-the-art models on the FaceForensics++ dataset. Each graph compares data for training with the RAdam and SGDM optimizers. Loss graphs show training and validation loss values, frame classification accuracy shows the percentage of frames that were correctly classified, and the final graph shows the balanced accuracy metric.

5.3.2 Breakdown of Results Across Different Categories

While these graphs are a good indicator of the models actually learning, it is worth doing a more in depth analysis of their behavior, such as whether certain models are better suited to some manipulation categories than others, and how their performance varies depending on the optimizer. For this reason I compiled Tables 5.3 & 5.4, which showcase a comparison of model performance on specific manipulation methods, as well as on the original sequences and the full set (overall). This data is based on an analysis of the output values for all videos in the validation split after the last epoch of training.

Average	xception		inception-v3		efficientnet-b5		resnet152		resneXt101	
Loss	RAdam	SGDM	RAdam	SGDM	RAdam	SGDM	RAdam	SGDM	RAdam	SGDM
DeepFakeDetection	0.07747	0.07964	0.08581	0.11864	0.12919	0.12060	0.07280	0.09557	0.07551	**0.06792**
Deepfakes	0.23878	0.24799	0.17022	0.14986	0.22424	0.20906	0.16950	0.20432	**0.08597**	0.11129
Face2Face	0.18101	0.15777	0.13532	0.12317	0.28388	0.24273	0.15604	0.16302	**0.07295**	0.09403
FaceSwap	0.16892	0.18374	0.12309	0.13770	0.34345	0.33468	0.14518	0.20475	**0.09978**	0.14496
NeuralTextures	0.58468	0.56878	0.51568	0.49554	0.61345	0.50669	0.60115	0.66151	**0.37453**	0.44791
original_sequences	0.24783	**0.19683**	0.26317	0.23706	0.22583	0.30639	0.23444	0.27800	0.41117	0.32583
Overall	0.21857	0.20868	0.19393	0.19478	0.26890	0.25743	0.20165	0.23712	**0.17519**	0.17992

Table 5.3: Average loss across different categories of videos from the FaceForensics++ dataset during validation for various models with RAdam and SGDM optimizers. The best results for each category are marked in bold.

Classification	xception		inception-v3		efficientnet-b5		resnet152		resneXt101	
Accuracy	RAdam	SGDM	RAdam	SGDM	RAdam	SGDM	RAdam	SGDM	RAdam	SGDM
DeepFakeDetection	99.70%	99.70%	99.70%	99.10%	97.89%	98.80%	99.70%	99.10%	**100.00%**	**100.00%**
Deepfakes	92.09%	92.09%	96.40%	97.84%	93.53%	93.53%	97.12%	97.12%	97.12%	**98.56%**
Face2Face	96.40%	97.12%	97.12%	97.84%	92.09%	94.96%	96.40%	97.12%	97.84%	**98.56%**
FaceSwap	97.12%	94.24%	97.84%	98.56%	89.21%	90.65%	98.56%	97.12%	98.56%	**99.28%**
NeuralTextures	79.86%	81.29%	84.89%	86.33%	76.98%	79.14%	84.17%	85.61%	91.37%	**92.09%**
original_sequences	96.11%	**97.78%**	95.56%	95.56%	95.56%	92.78%	95.00%	96.11%	91.11%	92.22%
Overall	94.76%	94.94%	96.07%	96.44%	92.32%	92.98%	95.97%	96.07%	96.54%	**97.19%**

Table 5.4: Video classification accuracy across different categories of videos from the FaceForensics++ dataset during validation for various models with RAdam and SGDM optimizers. The best results for each category are marked in bold.

These tables give quite a few insights to the task and the dataset. First of all, the EfficientNet is indeed much worse than all the other approaches. Of all the models, it has the worst statistics by far and, as was mentioned earlier, seemingly little room for improvement. As such, it can safely be discarded as a candidate moving forward.

The second finding is about the complexity of detecting the NeuralTextures manipulation method. It has the highest loss and lowest accuracy across all models. On average, the loss values for this method are around double or triple that of the other categories. As such, learning to detect fakes created via NeuralTextures is key for succeeding on this task. The FaceForensics team also reported [18] that NeuralTextures was the most difficult.

Next, there are some things to notice in regards to optimizer behavior. For all models, the difference in overall performance is nominally different (within about 1-2% in terms of video classification accuracy). There are no clear favorites in terms of categories either, because there is no manipulation method for which one of the optimizers completely outperformed the other. There are a few cases where one performed a lot better than the other, such as in the xception case for the original sequences, where the SGDM network did better on loss by about 0.05, and was the lowest loss value for any category. In conclusion, it cannot be said that either optimizer did significantly better.

Finally, it can be observed that the ResNeXt101 model was the best performer, followed closely by Inception V3, which is quite surprising as it is one of the smallest models. ResNeXt101 has the best results across almost all categories, including the very important NeuralTextures method, as well as the best overall result with RAdam. It is, however, worth noting that it was by far the largest model. In summary there is a lot of merit to residual connections on this task, but inception modules were also very powerful.

5.3.3 Hardest Samples for Classification

xception					
RAdam			SGDM		
method	filename	loss	method	filename	loss
NeuralTextures	538_493.mp4	5.74	NeuralTextures	548_632.mp4	6.38
NeuralTextures	548_632.mp4	5.73	NeuralTextures	538_493.mp4	5.19
Face2Face	567_606.mp4	3.49	NeuralTextures	115_939.mp4	3.85
Deepfakes	834_852.mp4	3.43	Face2Face	115_939.mp4	3.57
NeuralTextures	115_939.mp4	3.4	original_sequences	939.mp4	3.46
original_sequences	939.mp4	3.26	NeuralTextures	223_586.mp4	3.43
NeuralTextures	223_586.mp4	3.23	NeuralTextures	419_824.mp4	3.13
Face2Face	115_939.mp4	3.1	Deepfakes	834_852.mp4	3.02
NeuralTextures	794_779.mp4	2.75	NeuralTextures	794_779.mp4	2.61
NeuralTextures	419_824.mp4	2.7	NeuralTextures	468_470.mp4	2.54

Table 5.5: 10 samples with worst loss values for the **xception** model with RAdam and SGDM optimizers. Samples which appear on the list for both models are highlighted.

inception-v3					
RAdam			SGDM		
method	filename	loss	method	filename	loss
NeuralTextures	548_632.mp4	6.84	NeuralTextures	223_586.mp4	7.24
NeuralTextures	538_493.mp4	6.03	NeuralTextures	538_493.mp4	6.85
NeuralTextures	223_586.mp4	5.22	NeuralTextures	548_632.mp4	5.75
NeuralTextures	584_823.mp4	4.36	original_sequences	939.mp4	4.38
Face2Face	584_823.mp4	4.09	DeepFakeDetection	04_02_(...)	4.36
NeuralTextures	794_779.mp4	3.61	Face2Face	584_823.mp4	3.94
NeuralTextures	004_982.mp4	3.22	NeuralTextures	584_823.mp4	3.77
original_sequences	586.mp4	3.22	NeuralTextures	126_104.mp4	3.17
NeuralTextures	126_104.mp4	3.01	NeuralTextures	115_939.mp4	3.09
Face2Face	548_632.mp4	2.8	DeepFakeDetection	04_15_(...)	2.87

Table 5.6: 10 samples with worst loss values for the **inception-v3** model with RAdam and SGDM optimizers. Samples which appear on the list for both models are highlighted.

efficientnet-b5					
RAdam			SGDM		
method	filename	loss	method	filename	loss
NeuralTextures	548_632.mp4	3.86	NeuralTextures	126_104.mp4	2.86
NeuralTextures	538_493.mp4	3.33	Face2Face	817_827.mp4	2.46
NeuralTextures	126_104.mp4	2.94	NeuralTextures	538_493.mp4	2.45
NeuralTextures	443_514.mp4	2.73	FaceSwap	818_820.mp4	2.35
NeuralTextures	362_354.mp4	2.51	NeuralTextures	548_632.mp4	2.35
NeuralTextures	223_586.mp4	2.45	FaceSwap	817_827.mp4	2.25
FaceSwap	818_820.mp4	2.45	NeuralTextures	775_742.mp4	2.23
NeuralTextures	775_742.mp4	2.39	NeuralTextures	817_827.mp4	2.21
NeuralTextures	004_982.mp4	2.33	NeuralTextures	362_354.mp4	2.17
NeuralTextures	104_126.mp4	2	Deepfakes	817_827.mp4	2.14

Table 5.7: 10 samples with worst loss values for the **efficientnet-b5** model with RAdam and SGDM optimizers. Samples which appear on the list for both models are highlighted.

resnet152					
RAdam			SGDM		
method	filename	loss	method	filename	loss
NeuralTextures	223_586.mp4	10.09	NeuralTextures	538_493.mp4	12.17
NeuralTextures	538_493.mp4	5.96	NeuralTextures	223_586.mp4	9.21
NeuralTextures	126_104.mp4	5.84	NeuralTextures	794_779.mp4	6.08
NeuralTextures	419_824.mp4	4.76	original_sequences	939.mp4	5.31
NeuralTextures	584_823.mp4	4.72	NeuralTextures	419_824.mp4	4.95
DeepFakeDetection	04_15_(...)	4.04	FaceSwap	091_116.mp4	4.84
NeuralTextures	775_742.mp4	3.8	NeuralTextures	115_939.mp4	4.69
NeuralTextures	548_632.mp4	3.37	NeuralTextures	762_832.mp4	4.5
NeuralTextures	115_939.mp4	3.2	Deepfakes	606_567.mp4	4.36
Deepfakes	492_325.mp4	3.19	NeuralTextures	126_104.mp4	4.14

Table 5.8: 10 samples with worst loss values for the **resnet152** model with RAdam and SGDM optimizers. Samples which appear on the list for both models are highlighted.

resneXt101					
RAdam			SGDM		
method	filename	loss	method	filename	loss
NeuralTextures	538_493.mp4	6.75	NeuralTextures	223_586.mp4	7.81
NeuralTextures	223_586.mp4	6.25	NeuralTextures	538_493.mp4	7.57
original_sequences	939.mp4	5.01	NeuralTextures	794_779.mp4	5.81
original_sequences	933.mp4	4.72	NeuralTextures	126_104.mp4	4.11
original_sequences	768.mp4	4.69	original_sequences	939.mp4	3.96
NeuralTextures	419_824.mp4	3.84	NeuralTextures	775_742.mp4	3.76
original_sequences	416.mp4	3.56	NeuralTextures	419_824.mp4	3.75
NeuralTextures	126_104.mp4	3.49	original_sequences	768.mp4	3.64
NeuralTextures	115_939.mp4	3.24	Face2Face	115_939.mp4	3.44
NeuralTextures	794_779.mp4	3.18	NeuralTextures	115_939.mp4	3.26

Table 5.9: 10 samples with worst loss values for the **resneXt101** model with RAdam and SGDM optimizers. Samples which appear on the list for both models are highlighted.

Tables 5.5 to 5.9 list the 10 worst results for all models in their final validation run, with duplicate results for both optimizers highlighted. It is immediately noticeable that the most dominant method in these tables is NeuralTextures, which is the expected result given information from Tables 5.3 & 5.4, and further confirms that it is the most difficult type of manipulation to detect. Some NeuralTexture videos are frequently classified as being absolutely real with a score of 1 leading to high loss values (>5).

It is also worth noting that there is a lot of overlap between optimizers for both models, as about half of the worst samples are the same for both of each model's networks. This overlap isn't necessarily limited locally to models, because certain samples appear in multiple tables. Some videos can be seen in almost all of the lists, which makes them worth looking into. A few examples are shown on Figure 5.9, on which there are 5 equally spaced frames from chosen videos which appear frequently in these tables.

The NeuralTexture examples are very realistic and it is easy to see that they may be hard samples to classify. The last example is from the original sequences, so it is a real video which is being misclassified as fake. This is likely due to the rather poor, blurry quality of the frames. The subject of the video is a news anchor, so it is also likely that makeup has been applied to the face, making it look somewhat blended.

Figure 5.9: Random frames from some of the videos which appeared frequently in the tables of worst results. They can be considered some of the hardest samples for the models to classify.

5.4 Conclusions

In conclusion, the best performing model of those that were chosen was ResNeXt101. It is hard to say how much of this was a contribution of the model's size, however it has been documented [43] that it performs better on tasks such as ImageNet [26] than the other models. Nevertheless, the relatively good performance of the regular ResNet152 lends some additional merit to the power of residual connection based models in the field of face manipulation detection. The performance of the inception model, which had the second best results, was also very impressive, especially given the fact that it is much smaller than the residual models.

The NeuralTexture manipulation method generates the most realistic looking fakes and is generally hard to identify. The quality of this method is so high that it creates tension between itself and real samples. Models with better scores on NeuralTextures most often have worse scores on the original sequences.

These issues will be important for the designing of a novel model architecture and generally for further work on the task.

Chapter 6

The Reseption Models

One of the main reasons for testing various models was to see how certain deep learning architectures performed on the task in order to better construct a model for this problem. The best of those tested under the specified test conditions was ResNeXt101. However, it is worth noting that the InceptionV3 model also did quite well, which lends to the idea that residual connections combined with inception modules could be beneficial for this task. Intuitively, it makes sense that the features which need to be observed in order for the model to accurately assess the legitimacy of a face are complex and require many layers of filters. Given that, preserving some of the original information by utilizing an identity connection will do much to improve the network. It is also well known that residual connections are helpful for training, which is important due to resources being limited and the models being trained for a relatively low number of epochs.

The combination of inception and resnet models has previously been explored with the Inception-ResNet models [40]. The main finding was that the introduction of residual connections significantly improved training speed when compared to models without them of a similar computational cost. However, in the case of the proposed Inception-ResNet-v1 and Inception-ResNet-v2 models the building blocks are more similar to those used in resnets than inception modules. Due to my findings in the previous chapter, my model will focus on the modules used in Inception-V3 to minimize the model size, while slightly downsizing them and adding residual connections for efficiency and training speed.

6.1 Model Architecture

Residual connection based models tend to make use of a large number of 1×1 convolutions. ResNet, ResNeXt and Inception-ResNet blocks use initial 1×1 convolutions in each branch to split the input into lower-dimensional embeddings, followed by specialized filters of a larger size to collect spatial information and form feature maps. Afterwards, they use more 1×1 convolutions, either preceeded or proceeded by branch concatenation, as filter expansion layers, serving to return to a higher-dimensional embedding. These convolutions are computationally efficient and very useful, however I believe the rich assortment of feature maps created by the various branches of inception modules can also be leveraged to create a powerful model architecture.

In order to test this theory, I propose two models - Reseption V1 & V2. Inspired by Inception-V3 [39] and Inception-ResNet [40], the name reflects the combination of the inception network methodology with residual connections. Both networks remove the auxiliary branches of Inception V3, which have been found not to improve convergence during early training steps [40]. They both use almost identical stems to Inception V3, and the branches found in modules are also similar with a smaller number of filter banks. Reseption V1 uses a larger number of filter banks within these branches compared to Reseption V2 and avoids the use of 1×1 convolutions to map the outputs to higher-dimensional embeddings, instead using simple filter concatenation before being combined with the residuals. On the other hand, Reseption V2 does make use of more significant dimension reduction prior to the larger filters and employs remapping via these 1×1 convolutions prior to summation with the input identity. Additionally, the models make use of the Leaky ReLU activation function with a negative slope of 0.2.

6.2 Building Blocks

The most basic building block of these models is shown on Figure 6.1. The stem can be seen on Figure 6.2, while Figure 6.3 shows a diagram of the entire network. The schema for both models can be observed on these Figures. The main Reseption blocks are displayed on Figures 6.4 to 6.9. The reduction blocks on Figures 6.10 & 6.11 are largely the same as those used in Inception-V3, however the filter bank sizes are modified so as to suit the Reseption specifications.

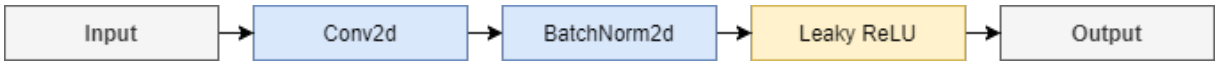

Figure 6.1: Basic convolution block (BasicConv2d) as defined for Inception [39] and used in the Reseption models. All "Conv" blocks in model schematics are instances of this BasicConv2d structure with configurable parameters.

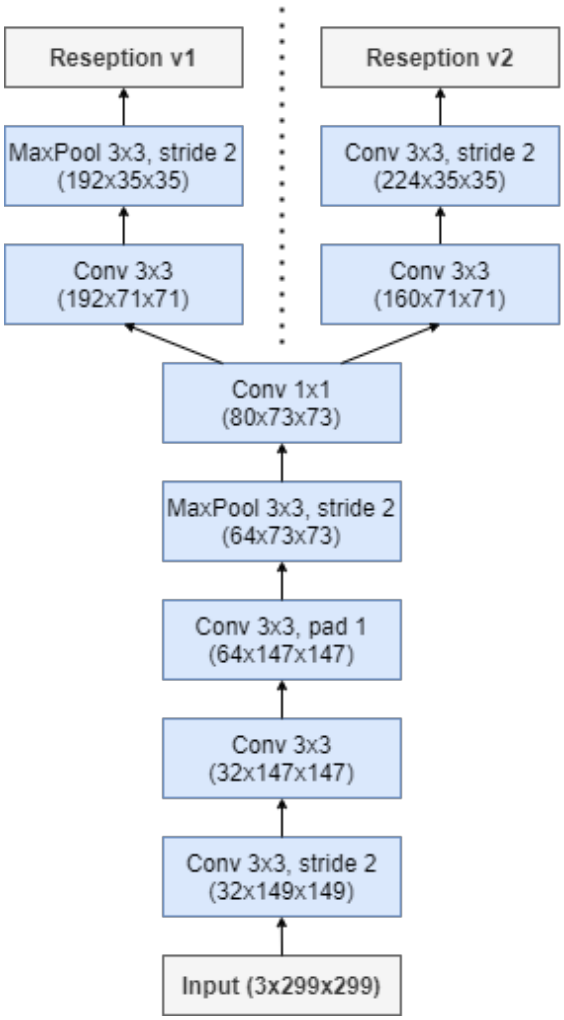

Figure 6.2: Reseption network stem. The only difference between the Reseption V1 and V2 stems is near the output of the stem.

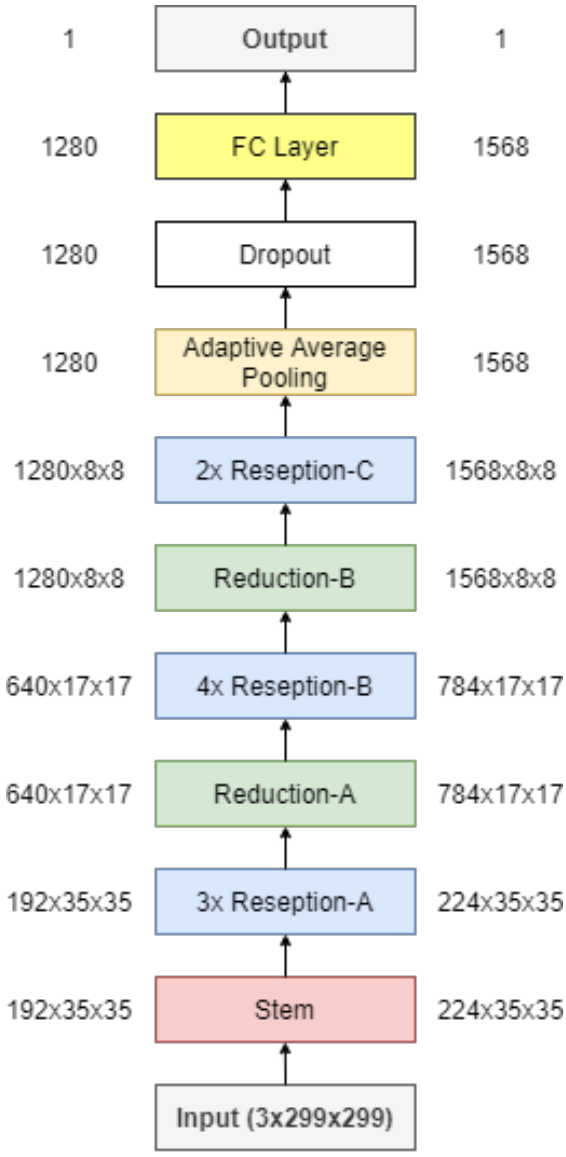

Figure 6.3: Reseption network structure. Reseption v1 block output sizes are listed on the left side, while Reseption v2 block output sizes are listed on the right.

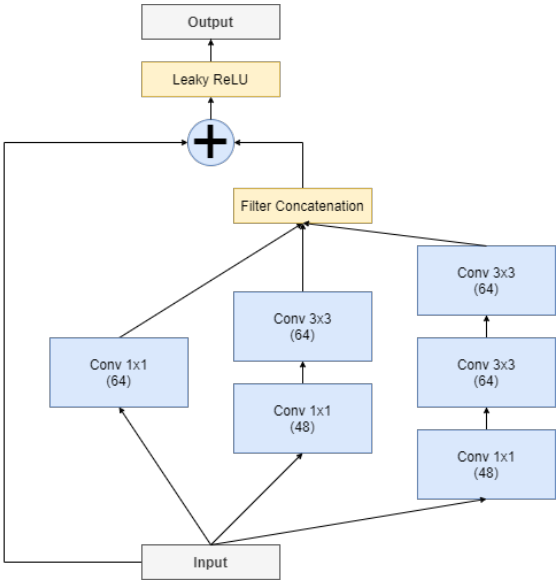

Figure 6.4: Reseption-V1 block A for an input grid of size 35x35.

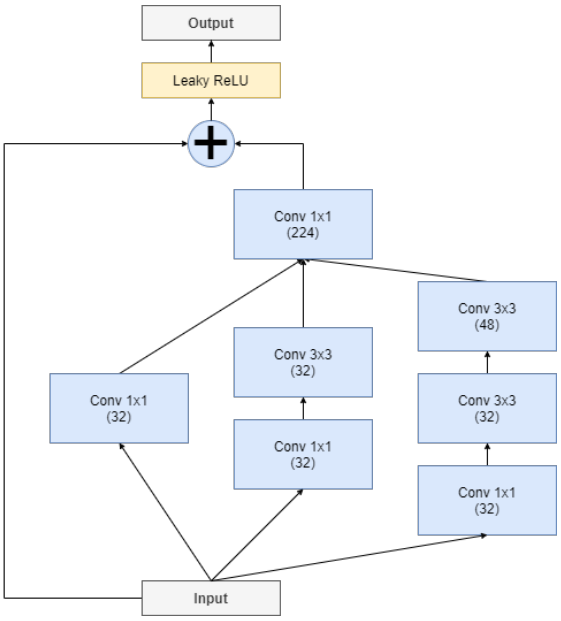

Figure 6.5: Reseption-V2 block A for an input grid of size 35x35.

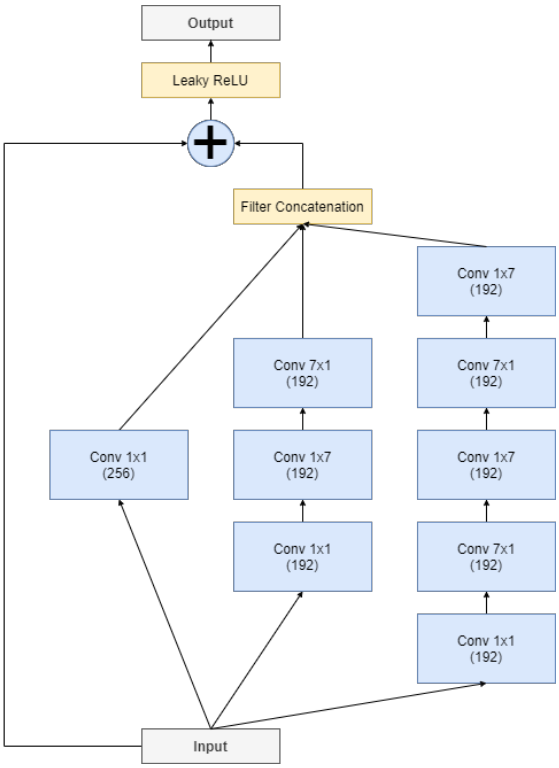

Figure 6.6: Reseption-V1 block B for an input grid of size 17x17.

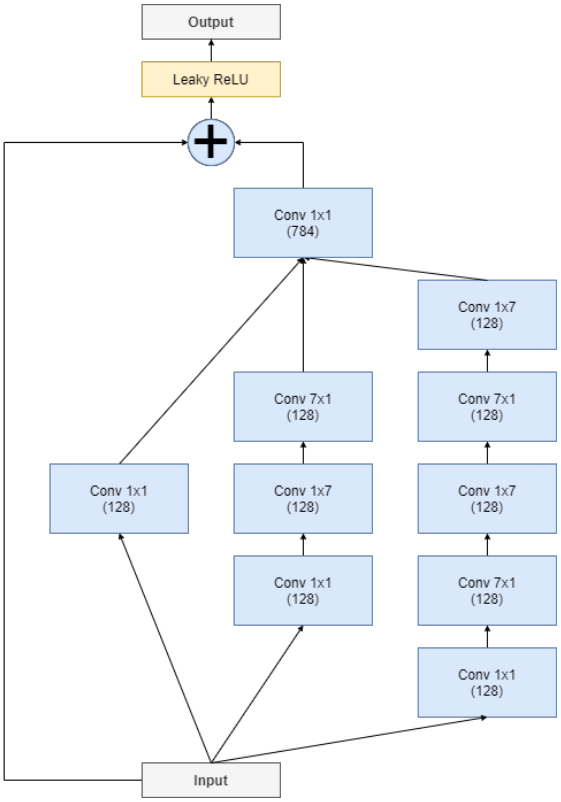

Figure 6.7: Reseption-V2 block B for an input grid of size 17x17.

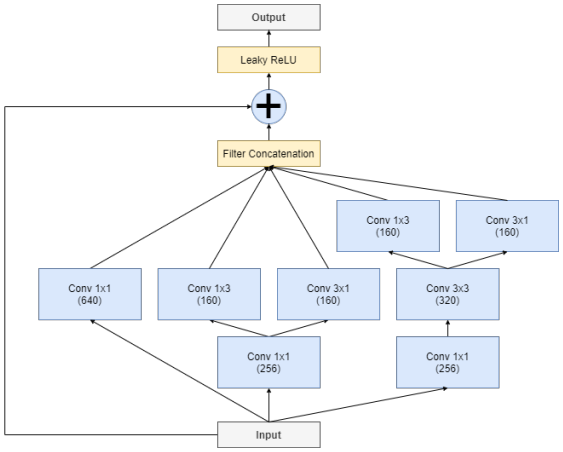

Figure 6.8: Reseption-V1 block C for an input grid of size 8x8.

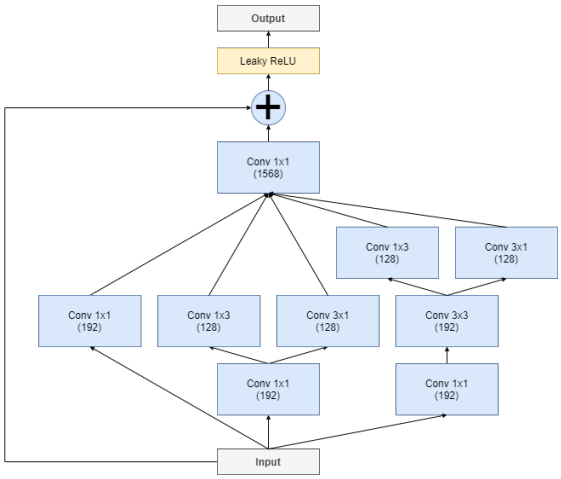

Figure 6.9: Reseption-V2 block C for an input grid of size 8x8.

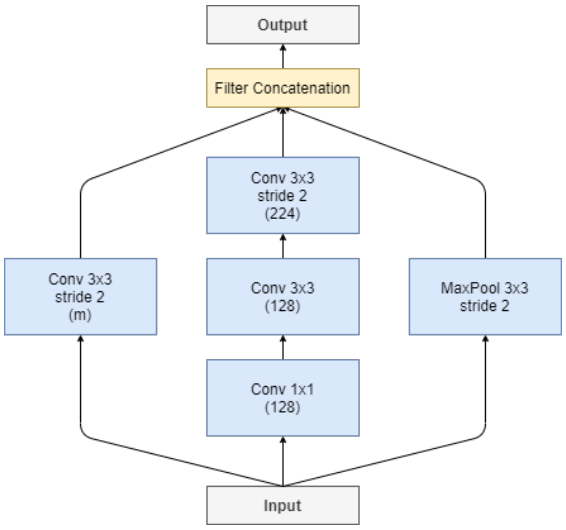

Figure 6.10: Reseption reduction block A, which has an identical structure for both models with the exception of filter bank size (m). Reduces maps of size 35x35 to 17x17. (m) is equal to 224 for Reseption-V1 and 336 for Reseption-V2.

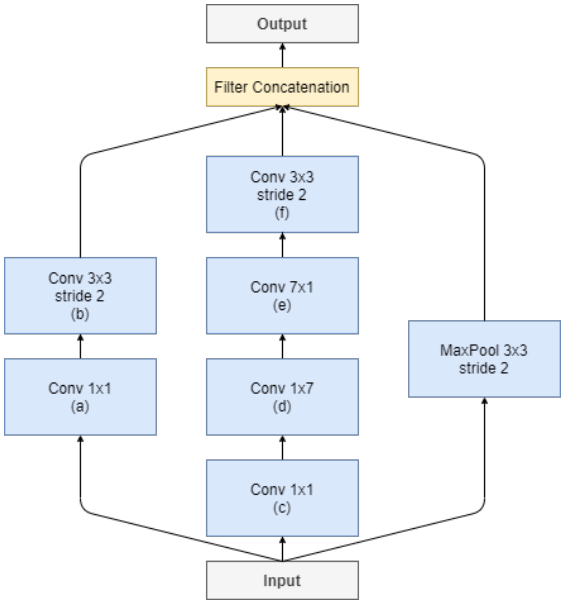

Figure 6.11: Reseption reduction block B, which has an identical structure for both models with the exception of filter bank sizes (a, b, c, d, e, f). Reduces maps of size 17x17 to 8x8. (a, b, c, d, e, f) is equal to (192, 320, 192, 192, 256, 320) for Reseption-V1 and (256, 392, 256, 256, 320, 392) for Reseption-V2.

Chapter 7

Results

The purpose of this project was to develop a model for the task of DeepFake detection. By comparing multiple state-of-the-art models used in the field of computer vision I decided on a suitable architecture, the Reseption models. They are fusions of the concepts behind ResNeXt and Inception, which were the two best performers of those tested.

Table 7.1 shows how they compare to the those two models. Both versions are significantly smaller than Inception V3, especially in terms of the parameter counts of the models. The ensemble is much larger, however nothing compares to the size and computational complexity of the ResNeXt101_32x8d model. With 30.51G MACs and 86.74 million parameters, it is easy to see why it outperforms the other state-of-the-art solutions. The Reseption models are about 5-6 times smaller in both categories.

Model Comparison	**Input Size**	**MACs (G)**	**Params (M)**	**Size in Memory (MB)**
Reseption V1	299x299	5.15	16.89	66.2
Reseption V2	299x299	4.79	15.75	61.8
Reseption Ensemble	299x299	9.94	34	133.7
Inception V3*	299x299	5.73	21.79	85.4
ResNeXt101_32x8d*	224x224	30.51	86.74	339.7

Table 7.1: A comparison of various characteristics of the Reseption models and the two best performing SotA solutions. The MACs and parameter counts were gathered using the THOP (PyTorch-OpCounter) module [52]. Size in memory was checked locally. The Inception V3 and ResNeXt101 models mentioned here have a single output neuron.

7.1 Training the Reseption Models

7.1.1 Weight Initialization

The proper initialization of weights is key when training a deep, multi-layer neural network. In the case of initial weights being too small or too large, layer activation outputs may vanish or be driven into saturation. In both cases, this leads to a degradation of performance, and may even lead to the destabilization of the entire network. One large advancement in this field was the introduction of xavier initialization [49], which attempts to maintain the variance of activations and gradients during forward and backward propagation. In the case of using this method from a normal distribution, weight values are sampled from $N(0, \sigma^2)$, where the standard deviation is based on the number of input and output neurons to the layer. This is the method of initialization I chose for the Reseption models, and in PyTorch it is implemented as follows:

$$\sigma = G \times \sqrt{\frac{2}{n_i + n_o}}$$

Here, σ is the standard deviation, n_i & n_o are the numbers of input and output neurons to the layer, and G is the gain. The gain is a scaling factor, which for the case of Leaky ReLU activation function is defined as $G = \sqrt{\frac{2}{1+negative\_slope^2}}$.

7.1.2 Gradient Flow

One problem that was observed during my initial tests of training the models were those of very small gradients. Due to this issue, training was quite slow, and some layers were barely seeing any updates. It quickly became apparent, that a solution needed to be found. One change which helped was changing the activation function from ReLU to Leaky ReLU, which has a small slope for negative values rather than setting them to 0, helping to fight the vanishing gradient problem. Two other, lesser used methods were also tested.

- **Gradient Noise Addition**

  Proposed by Neelakantan et al. in 2015 [48], the idea of this method was to add some noise sampled from a random Gaussian distribution to each gradient. The noise is added to each gradient g at each training step t. It is chosen from a normal distribution, where the variance is decayed over time:

$$g_t \leftarrow g_t + N(0, \sigma_t^2)$$

$$\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$$

The second equation describes the annealing function used to increase gradient values during the early steps of training while decaying to lower values later on. In the source, η is a scaling factor chosen from the range 0.01, 1.0 and γ is set to 0.55. Although this method was tested with various hyperparameters, including different values of γ and using different definitions of the time step (as each epoch or as each batch), it ultimately proved too difficult to use stably. This is most likely because of a combination of the network complexity, long epochs and small number of epochs.

- **Gradient Scaling**

  During backward propagation, it may occur that certain gradient values are too small to be represented and thus will underflow, or be flushed to zero. To prevent this, the network's losses can be scaled by some scale factor, resulting in a larger magnitude for gradient values. This scale factor is optimized each iteration, so as to be as large as possible while not underflowing or overflowing gradient values. Since this method works across all gradients, in many situations it can significantly improve training speed. This method was found to work very well for the Reseption models.

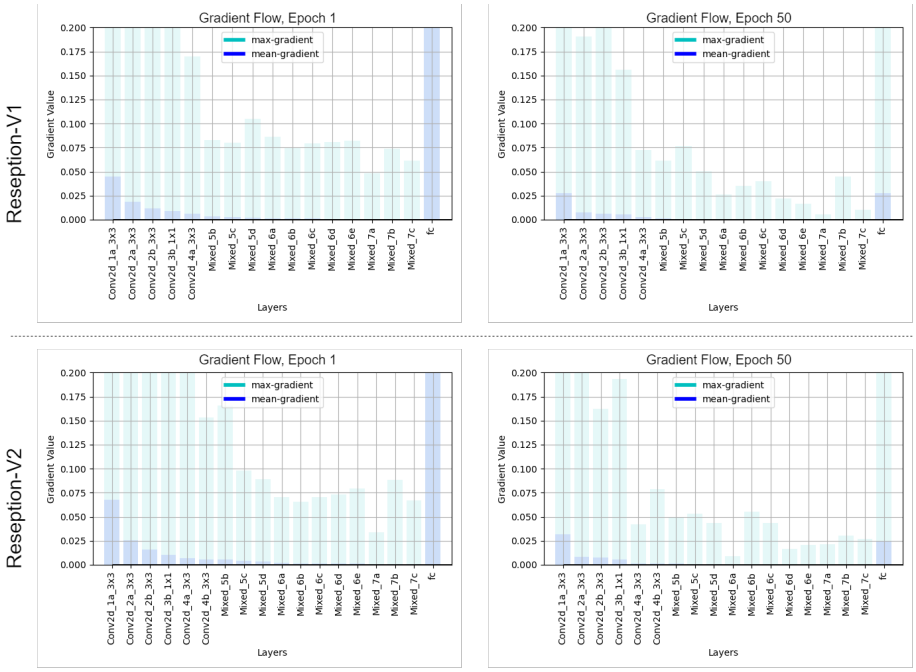

Figure 7.1: Gradient flow for both Reseption models in early and late epochs.

After the implementation of the leaky ReLU and gradient scaling, the gradient flow of the network is significantly improved. Figure 7.1 displays how the maximum and mean gradient values behave for each layer in the first and last epochs of training. It can be observed that the reseption and reduction blocks (labeled as various 'Mixed' layers) have quite low values, likely due to the existence of residual connections allowing the skipping of these layers.

The ability of gradient scaling to improve training speed is visible on Figure 7.2, where SGDM+GS consistently outperforms the SGDM variant for both models and also looks to be more stable (a 'cleaner' line).

7.1.3 Optimizer

During the previous chapter both RAdam and SGDM optimizers were used for testing. It was found that both generally tended towards good results, with each performing better than the other for some model/manipulation method combinations. It is worth noting, however, that in that case the models began with weights which were pretrained on ImageNet [26], and as such training required considerably less effort and time to reach convergence. The Reseption models will be trained from scratch. The main benefit of RAdam for this project is its ability to outperform other optimizers in early training [47], which is of keen importance here.

This theory was tested on the Reseption models, and the results can be observed on Fig. 7.2. RAdam with gradient scaling outperforms the SGDM alternatives on both Reseption V1 and Reseption V2 by a significant margin of about 0.15-0.2 in terms of training loss. On top of that, it does not seem that SGDM will catch up at a point in the near future of those training cycles. The models will only be trained for 50 epochs and as such, RAdam will be the optimizer chosen for this task.

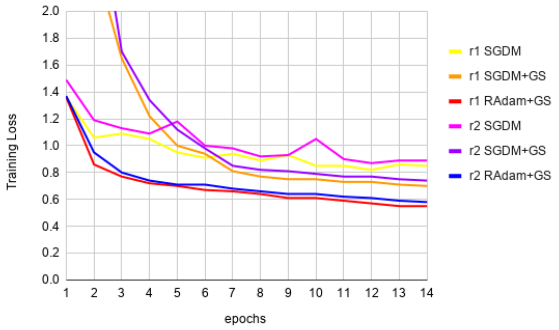

Figure 7.2: Loss during the first 14 epochs of training for both Reseption models (r1 and r2). A comparison of SGDM and RAdam performance during early epochs and the value of gradient scaling (GS). Both optimizers were initialized with their default learning rates (1e-2 for SGDM and 1e-3 for RAdam), which was decayed by 0.94 every 2 epochs.

7.1.4 Batch Type

In this project there are two types of batches which can be used to train the network, either containing frames from various pairs of videos or multiple frames from a single pair. Training with this 'various' type batch has the advantage of being faster, while 'dual' type batches allow a more thorough use of the dataset during each epoch. While training with the former converges quicker, the former would most likely eventually result in a better performance. 'Various' type batches were used in the previous chapter to attain a large spread of results much faster. The usage of single pairs per iteration was briefly tested and training was slightly more stable, however it was ultimately decided to use batches composed of various videos due to significant time and resource constraints.

7.1.5 Training Methodology

The two models were trained for 50 epochs using batches of frames from various pairs of videos from the FaceForensics++ dataset with a stable set of hyperparameters for both. The results were achieved using RAdam with its default initial learning rate of 1e-3, decayed every second epoch using an exponential rate of 0.94 (as was done with all inception models [38],[39],[40]) and adjusted by means of the ReduceLROnPlateau scheduler set to reduce the learning rate by a factor of 0.5 with patience equal 2. The batch size for training was set to 24, while the validation batches used 7 frames per video. The models are capped by a single output neuron after the fully-connected layer and use binary cross-entropy as their loss functions. In cases where the network output for a frame is considered, a sigmoid activation function is used. Training each model while running validation after every epoch takes about 2 days on a GeForce GTX 980M.

7.1.6 Model Performance

Training was conducted once for both models. Because RAdam is an adaptive learning rate algorithm there is not as much of a benefit to training multiple networks. The plots displayed on Figures 7.3 to 7.6 showcase the average loss and classification accuracy parameters for alternating runs of training and validation. One interesting characteristic is the spiky nature of the validation loss curve. Although this was also somewhat noticeable in the plots of Figure 5.8, this behavior is more apparent during the training of these models, though the Reseption V2 model has a smoother curve.

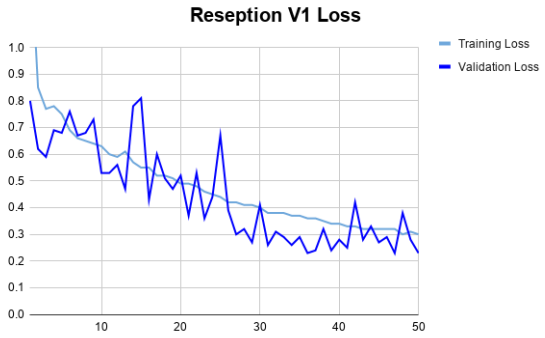

Figure 7.3: Reseption-V1 loss values over 50 epochs of training for two different training cycles a & b.

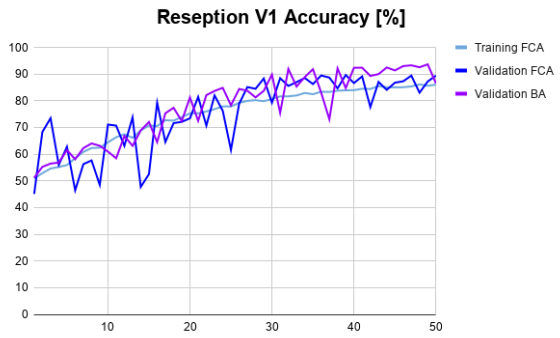

Figure 7.4: Reseption-V1 accuracy values over 50 epochs of training for two different training cycles a & b. FCA stands for frame classification accuracy, while BA is balanced accuracy with regards to video classification.

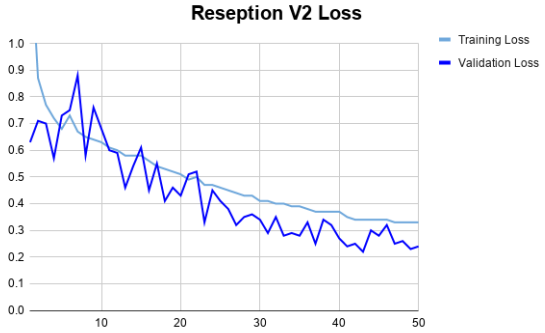

Figure 7.5: Reseption-V2 loss values over 50 epochs of training for two different training cycles a & b.

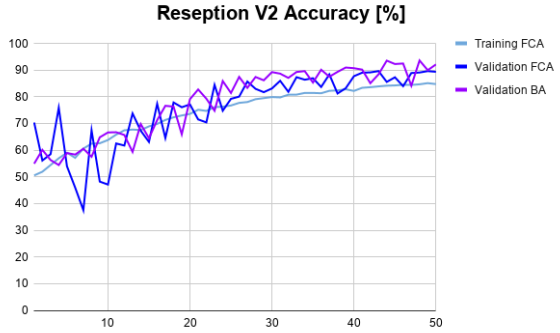

Figure 7.6: Reseption-V2 accuracy values over 50 epochs of training for two different training cycles a & b. FCA stands for frame classification accuracy, while BA is balanced accuracy with regards to video classification.

The plots for both models show steady improvement across training and validation cycles. Validation loss is significantly lower than training loss, which possibly indicates very strong data augmentation. Another topic of interest is the spiky nature of validation losses. Attempts to rectify this by means of altering the learning rate, either directly or by lowering the decay step from two epochs to one, were unsuccessful. The smoother nature of the curves for the state-of-the-art models could be a byproduct of them being pretrained, leading to nicer behavior during training. If it is related to the characteristics of the models and their initial weights, this could explain the ReseptionV2 model having a slightly smoother curve. It relies more heavily on residual connections, which have been shown [42] to improve network behavior during training. Another possibility is that it is an issue related to training focusing alternately on fake and real samples. Because

of the imbalanced number of fake/real samples in the validation runs, but a balanced amount during training, overfocusing on real samples during a training run could cause a jump in the validation loss of the next cycle. Frame classification accuracy scores start to plateau in the high 80 range, while balanced accuracy scores for video classification achieve values in the low 90 range. These values are quite similar to those shown during the testing of state-of-the-art models on Fig. 5.8, however they do not exceed those of the best performers.

Figures 7.3 to 7.6 indicate that Reseption V1 outperforms Reseption V2 based on the top values being achieved in the cases of loss and accuracy. This is further proven by Tables 7.2 and 7.3, which showcase the best results achieved by both models (based on the epoch in which they had the highest balanced accuracy scores for video classification). The first model, based more heavily on the inception architecture, outperforms the second model in most categories. More specifically, it outperforms the second model in all categories other than the original sequences, in which it loses by a fairly small margin. Performance on the DeepFake categories, which are most relevant to this project, and NeuralTextures, the hardest category, is noticeably better.

When compared to the average loss and classification accuracy for the SotA models listed in Tables 5.3 & 5.4, Reseption V1 outperforms every model except ResNeXt101 in the NeuralTextures category, while being fairly outperformed in terms of identifying the real samples. Its performance on the DeepFake categories and Face2Face is also quite comparable to the best of those models. The only manipulation method it seems to struggle with compared to other solutions is FaceSwap, for which only the efficientnet-b5 model is worse. I believe the models could still benefit from a few more epochs of training and achieve slightly better results. It is also worth noting that both Reseption models are smaller than all the solutions explored in the previous chapter, which will be explored further in the next chapter.

Comparing the worst-performing videos from the validation set (Fig. 7.4) shows that the models struggle with a lot of the same samples. These samples are consistent with those which were problematic in Tables 5.5 to 5.9, however it is interesting to note that the highest loss values are significantly lower than those seen previously. Unsurprisingly, videos manipluated with NeuralTextures appear the most often, which is in accordance with it scoring as the hardest method to detect.

Average Loss	reseption_v1 epoch 47	reseption_v2 epoch 48
DeepFakeDetection	**0.08265**	0.11636
Deepfakes	**0.19784**	0.23353
Face2Face	**0.14640**	0.19849
FaceSwap	**0.25906**	0.32281
NeuralTextures	**0.46122**	0.52871
original_sequences	0.36150	**0.33200**
Overall	**0.22517**	0.25918

Table 7.2: Average loss across different categories of videos from the FaceForensics++ dataset during validation for both of the Reseption models. Values are displayed from the epochs with the best results (highest video classification balanced accuracy). The best results for each category are marked in bold.

Classification Accuracy	reseption_v1 epoch 47	reseption_v2 epoch 48
DeepFakeDetection	**99.40%**	99.70%
Deepfakes	**93.53%**	90.65%
Face2Face	**95.68%**	95.68%
FaceSwap	**93.53%**	93.53%
NeuralTextures	**79.86%**	78.42%
original_sequences	92.78%	**93.89%**
Overall	**93.73%**	93.45%

Table 7.3: Video classification accuracy across different categories of videos from the FaceForensics++ dataset during validation for both of the Reseption models. Values are displayed from the epochs with the best results (highest video classification balanced accuracy). The best results for each category are marked in bold.

reseption_v1			reseption_v2		
method	**filename**	**loss**	**method**	**filename**	**loss**
NeuralTextures	548_632.mp4	3.53	NeuralTextures	548_632.mp4	3.6
NeuralTextures	538_493.mp4	3.29	NeuralTextures	538_493.mp4	3.32
NeuralTextures	419_824.mp4	3.25	NeuralTextures	223_586.mp4	2.93
Deepfakes	638_640.mp4	2.58	NeuralTextures	419_824.mp4	2.92
NeuralTextures	223_586.mp4	2.33	NeuralTextures	126_104.mp4	2.32
NeuralTextures	775_742.mp4	2.08	Face2Face	923_023.mp4	2.22
NeuralTextures	794_779.mp4	2.03	FaceSwap	820_818.mp4	2.21
NeuralTextures	482_465.mp4	1.78	FaceSwap	923_023.mp4	2.21
DeepFakeDetection	04_15__outside_talking_pan_laughing__7NW4Z6WZ.mp4	1.6	NeuralTextures	923_023.mp4	2.19
NeuralTextures	126_104.mp4	1.56	Deepfakes	923_023.mp4	2.17

Table 7.4: 10 samples with worst loss values for best validation epochs during training of the **Reseption** models. Samples which appear on the list for both networks are highlighted.

7.1.7 Ensemble

The only advantage the Reseption V2 network had over Reseption V1 was in the original_sequences category. Additionally, they had many of the same samples in Table 7.4. Nevertheless, the two networks may observe separate features in which case their union would be beneficial. As such I decided to create an ensemble using the two pretrained Reseption models.

The FC layers/classifiers were cut out and replaced so each network could contribute 512 features to the final classifier, as shown on Figure 7.7. This should lead to a more balanced performance across all manipulated methods and in the recognition of real samples.

The ensemble was trained using the same methodology as the Reception models, which was explained in section 7.1.5. Only the parameters of the fully-connected layers and the classifier were unfrozen and trained, so the main bodies of the 2 networks were preserved.

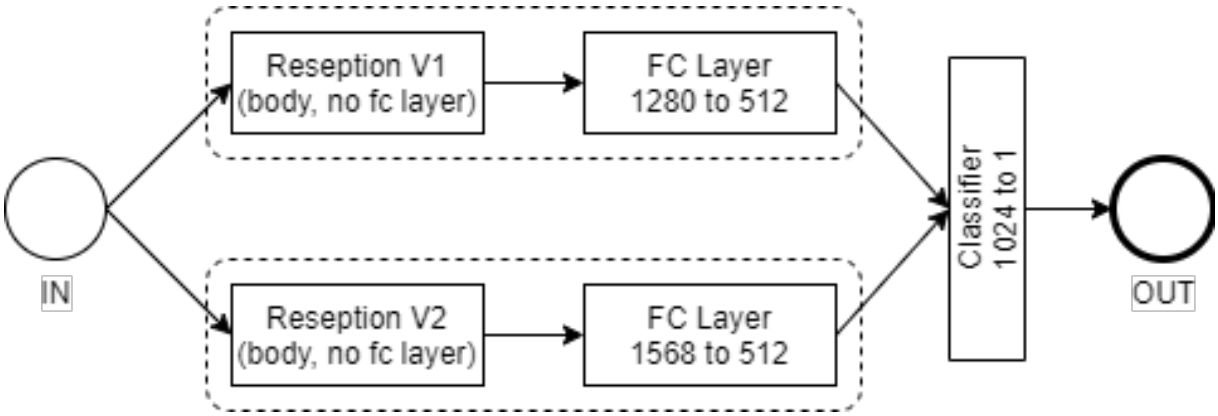

Figure 7.7: Schematic of the ensemble of Reception networks. The ensemble accepts an input of size 299x299x3 and returns a single score.

The ensemble was quick to achieve convergence. After about 10 epochs, the loss and accuracy scores did not improve much further, as can be seen on Figures 7.8 & 7.9. The best validation performance was achieved in epoch 19, the specifics of which are displayed in Table 7.5. With an overall average validation loss of about 0.21, the network achieved very good results, outperforming both Reception V1 and Reception V2 in multiple categories.

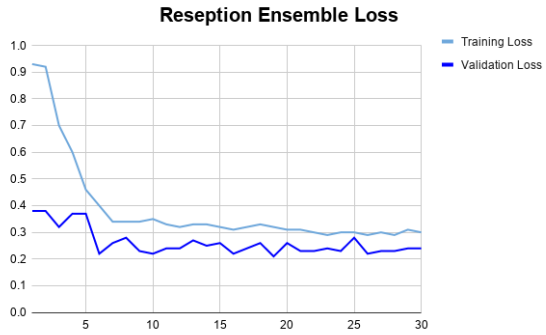

Figure 7.8: Reception ensemble loss values over 30 epochs of training.

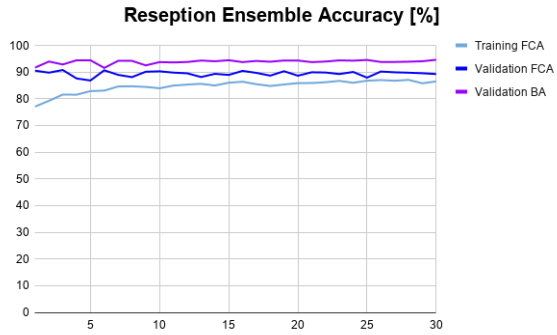

Figure 7.9: Reception ensemble accuracy values over 30 epochs of training. FCA stands for frame classification accuracy, while BA is balanced accuracy with regards to video classification.

With frame classification accuracy and balanced accuracy scores of about 90% and 95% respectively, the ensemble does very well on the task of classifying whether a face has been manipulated or not. The ensemble could be further improved by adding more trained versions of the Reseption models, more epochs of training for the Reseption networks themselves, or by upscaling the individual models. I believe that with these steps the ensemble could contend with the best scores of the SotA models.

The scores for Deepfakes, Face2Face, FaceSwap and NeuralTextures in Table 7.5 are better than those in Figures 7.8 & 7.9, while those for DeepFakeDetection and original_sequences are between the two achieved by the Reseption models. As the score on DeepFakeDetection manipulations is almost identical, this means an overall improvement on the most relevant tasks, i.e. the DeepFakes and the category boasting the highest level of difficulty. This is surprising given that it didn't seem like the two models could contribute much to each other. In terms of overall performance on the FaceForensics++ task, the Reseption ensemble is the best of the newly proposed networks. However, the ResNeXt101 and InceptionV3 networks still perform better.

Ensemble epoch 19	**Average Loss**	**Classification Accuracy**
DeepFakeDetection	0.08431	99.70%
Deepfakes	0.17906	94.96%
Face2Face	0.12662	96.40%
FaceSwap	0.22338	94.96%
NeuralTextures	0.43410	82.73%
original_sequences	0.36028	93.89%
Overall	0.21228	94.85%

Table 7.5: Table of average loss and frame classification accuracy across all manipulation methods for the best validation run (epoch 19) during training of the Reseption ensemble.

7.2 Testing

The models were tested against the test set of FaceForensics++ according to the official splits listed by the FaceForensics team, with the addition of the DeepFakeDetection category. The models classify frames based on faces extracted according to the process detailed in Chapter 4 on preprocessing. The test was conducted using 12 equally-spaced frames from each video. The loss is calculated as the average across all frames, while they are classified based on the average output from all considered frames. The results are documented in Table 7.6.

Model Testing	reseption-v1		reseption-v2		reseption ensemble		inception-v3		resneXt101		XceptionNet*
	Loss	Acc	Loss	Acc	Loss	Acc	Loss	Acc	Loss	Acc	Acc
DeepFakeDetection	0.10025	98.91%	0.12719	98.63%	0.11087	98.63%	0.14934	99.45%	0.15169	98.91%	-
Deepfakes	0.18550	94.29%	0.24100	91.43%	0.18036	92.86%	0.21071	95.00%	0.14436	96.43%	97.49%
Face2Face	0.19000	95.00%	0.25671	92.86%	0.17371	94.29%	0.21664	92.86%	0.17407	96.43%	97.69%
FaceSwap	0.35364	88.57%	0.38886	87.14%	0.29329	90.71%	0.20536	97.14%	0.22064	96.43%	96.79%
NeuralTextures	0.44721	83.57%	0.50829	79.29%	0.42179	82.14%	0.46536	89.29%	0.39014	92.14%	92.19%
original_sequences	0.43222	89.44%	0.38800	90.00%	0.40906	92.22%	0.32233	92.78%	0.39711	91.67%	95.41%
Overall	0.25242	93.04%	0.28180	91.68%	0.23860	93.22%	0.24088	95.39%	0.23245	95.93%	95.73%

Table 7.6: Average loss and video classification accuracy scores for various models on the test split of the FaceForensics++ dataset. Results are seperated into different categories based on manipulation methods. The XceptionNet data in the right-most column is taken from FaceForensics++ [18], and showcases the results of their best model's performance on the dataset. It is an Xception network with two outputs, as opposed to the singular output of all the models I used. It was trained on all manipulation methods at once, however at that point DeepFakeDetection was not a part of the dataset.

Interestingly, the scores are worse across almost all categories for all the considered models compared to those observed in the validation runs analyzed in the previous chapter. This may be due to the fact that the best performers were chosen based on their balanced accuracy scores for video classification on the validation set, which doesn't necessarily generalize perfectly and might not account for some variance seen in the test set. I included the results of the FaceForensics team's best performing model on the same level of compression, XceptionNet. It boasts results which are better in all categories it was trained on. The model was trained in a similar environment, however it was trained without the use of DeepFakeDetection, which is in general data of a higher resolution and of a larger volume than the other 4 manipulation categories. To the best of my knowledge, it is also not known what level of data augmentation was utilized. Although imperfect, it offers an outside comparison of model performance.

Here, the Reseption models are a lot more competetive with the others. Their performance on DeepFakeDetection is better in all cases than that of InceptionV3 or ResNeXt101. Reseption V1 also beats Inception V3 on DeepFakes, Face2Face and NeuralTextures. Reseption V2 beats ResNeXt101 in detecting real videos. They are significantly worse on videos manipulated

	Balanced Acc
reseption-v1	91.81%
reseption-v2	90.79%
reseption ensemble	93.04%
inception-v3	94.78%
resneXt101	94.21%

Table 7.7: Balanced accuracy scores for real/fake video classification on the test split for the FaceForensics++ dataset.

with FaceSwap, but the scores are very close overall. The balanced accuracy scores attained during testing can be seen in Table 7.7 on the right-hand side. These are mildly lower than those attained on the validation set, which is consistent with the above-mentioned theory. InceptionV3 had the best score, however it was still beaten by ResNeXt in terms of average loss and general video classification accuracy. As previously, Reseption V1 is the better of the two newly proposed models and the ensemble offers a mild improvement to overall performance. This is consistent with the data in Table 7.6. Given the advantage that the two best models have in terms of their scale, the performance of the Reseption models is impressive.

7.3 Model Explainability

One of the most important problems when dealing with machine learning problems is whether a model can be trusted. While it may seem to work on a given task using some specific data, it may actually be coming to conclusions in a highly flawed way. For example, a model tasked with classifying dogs and cats may notice that all pictures of dogs have grass in the background while pictures of cats are taken indoors and then use the surroundings, rather than the animals, for its decisions. As such, it is necessary to check what a model is actually doing.

LIME [53, 54] is a tool created for the purpose of explaining individual predictions made by some supported types of classifiers, one of which is images. The name stands for local interpretable model-agnostic explanations. LIME works by exploring the neighborhood of a predication for some individual sample, and then provides an explanation for why the model, which is treated as a black box, came to some decision in a manner which can be understood by humans. In the case of this project and the models being used, it will show which areas of images fed to the network have the largest impact on classification.

In order to verify the decision-making process of the Reseption models I tested them with LIME against some random samples from the each of the manipulation methods used in FaceForensics++, as well as against a real image. In order to have a better idea of how the results should look, I also tested the two best state-of-the-art models, due to their performance. Additionally, they have been pretrained on ImageNet, so object segmentation should not be an issue for them. The images are frames chosen from videos in the test split. The output for each case can be seen in Figure 7.10.

The first thing to notice is that in all cases the areas of interest mostly overlap with the face. This is very important, because it means the models are actually considering facial features in their decisions. The next characteristic is that segmentation works quite well. In many cases is is visible that the features end at the edge of the face or hair. This is shown very well in the cases of images from the Face2Face and Original Sequences categories. There are also some examples where the feature focuses on a specific part of the face, such as the nose or mouth.

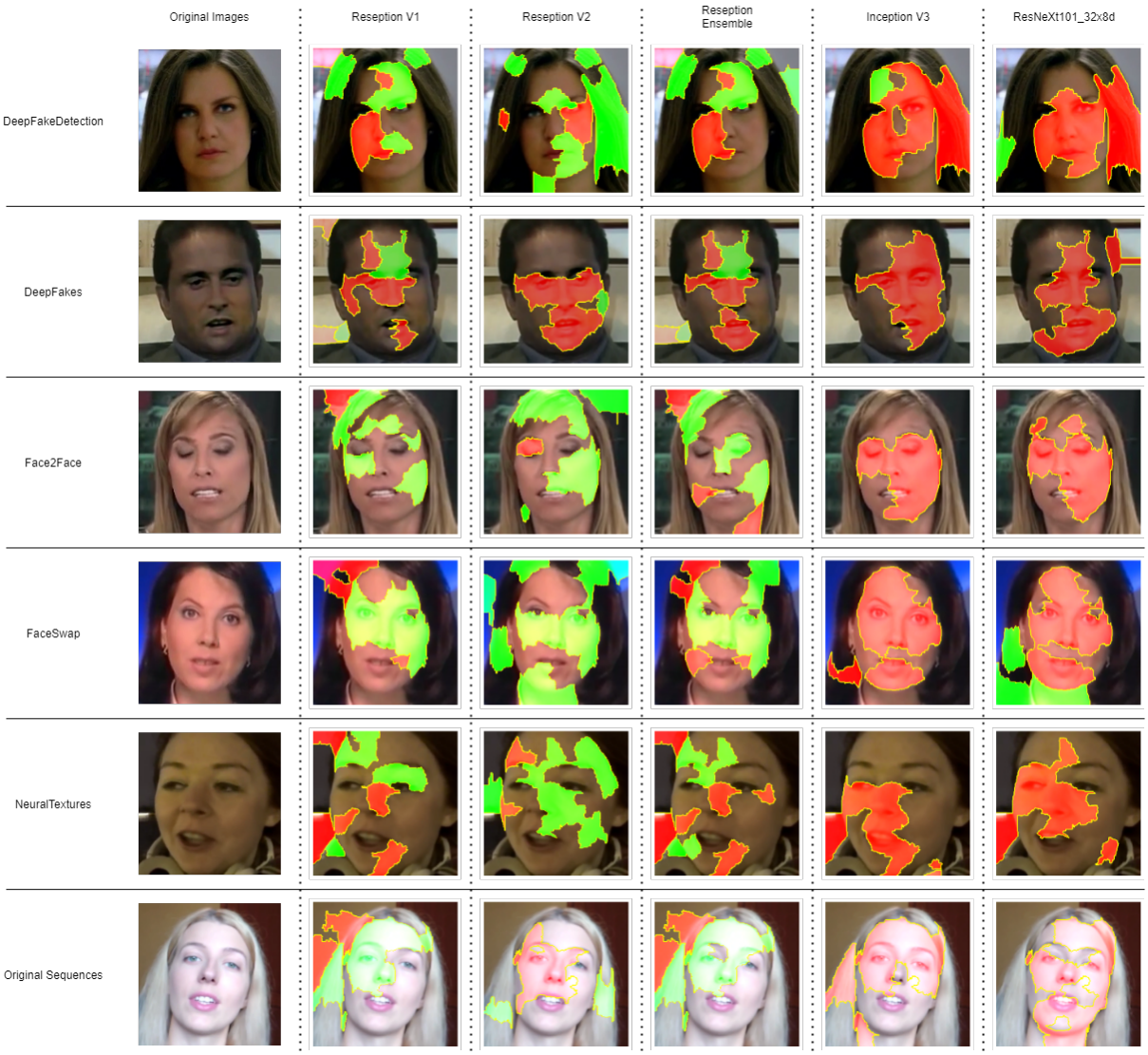

Figure 7.10: Results of LIME testing for certain models on random samples from each of the categories in FaceForensics++. Features marked in red have a high impact on the model predicting the image to be fake, while features marked in green have a high impact on the model predicting the image to be real.

An interesting thing to note is how the features used in the decisions of the Reseption models overlap in the ensemble. The most notable areas in the case of the ensemble explanation are a combination of those used in the two networks it is composed of.

There are also some problem areas. For example, the Reseption models have many green areas, which indicate that the highlighted part of the image is strongly contributing to a 'real' classification, in images which are fake. On the other hand, Inception V3 and ResNeXt101 mark almost the entirety of the actual real sample in red. Table 7.8 shows how the models actually classified each of the images.

There are quite a few classifications here. While many of them are close, being around the critical area where a prediction is close to going the other way (<65%), many classifiers are very confident in their mistaken prediction. Surprisingly, Reseption V1 performed the best here, as it only classified the real sample incorrectly. Both SotA models misclassified the Face2Face sample with a high probability (around 90%), despite the face being almost entirely red in the LIME test. ResNeXt also classified the real image as fake with almost absolute certainty (97.22%). Most of the videos these frames are from were classified correctly, showing how much easier it is to classify a video which has a larger sample size of images to analyze.

Fortunately, all models classified the DeepFake examples correctly. This is a very good note, as that is the main topic of this project.

Model Predictions	**reseption-v1**	**reseption-v2**	**reseption ensemble**	**inception-v3**	**resneXt101**
DeepFakeDetection	65.77%	51.34%	50.94%	91.25%	81.67%
Deepfakes	94.07%	51.75%	74.58%	93.48%	63.41%
Face2Face	82.72%	65.12%	55.67%	89.38%	96.37%
FaceSwap	62.06%	66.75%	56.07%	90.64%	58.07%
NeuralTextures	74.97%	58.16%	51.30%	90.29%	54.68%
original_sequences	64.14%	87.87%	68.14%	96.06%	97.22%

Table 7.8: Predictions made by the respective models on samples from various categories of the FaceForensics++ dataset. These predictions correspond to the samples displayed in figure 7.10. Predictions marked in red indicate the model decided the image was fake with the documented probability, while green samples indicate a real image prediction. The 10 most important features are shown.

Chapter 8

Conclusions

Identifying facial manipulations in images and video is a very difficult task. After tackling the problem of DeepFake detection I feel very strongly that this is a field which requires a lot more research and will be a hot topic in academia for a long time.

In this thesis, I analyzed two datasets and chose FaceForensics++ due to its smaller size and higher quality. In addition to various samples of the same faces with different, high quality manipulations performed upon them it was easier to work with in terms of sorting out samples with multiple faces and provided more interesting test data due to the various categories. This choice allowed me to broaden the task and explore it in greater depth.

In dealing with the challenge of preparing the data for use with deep learning models I designed a batch creation process which was able to overcome the inherent class imbalance of the training data. Each sample of the four manipulation methods used in FaceForensics++ [18] comes from the same source videos, so there are four fakes to each real video. The DeepFakeDetection dataset contributed by Google & Jigsaw is even more imbalanced. I was able to overcome this issue thanks to my method of always pairing samples from fake videos with samples from their real counterparts, achieving a balanced accuracy for video classification of in the mid-90 range for all tested models.

In this project I set out to explore the power of machine learning and its ability to solve this task. To do so I tested some of the most widely-used and best performing state-of-the-art image classification models of recent years. Of those, ResNeXt101_32x8d and Inception V3 showed the best results. An depth analysis of data gathered during training revealed important dataset characteristics, highlighted data-specific challenges, and showed how certain models may complement each other. It is also interesting to note that these models have very different structures.

Due to all these factors, I decided to make use of both and created two models which combine features responsible for those two models' success - residual connections and inception modules. I named the novel architectures Reseption after that union. The two models I proposed are Reseption V1 and Reseption V2. Reseption V1 focused more heavily on inception modules, while Reseption V2 focused more on residual connections and the large amount of 1x1 convolutions used in resnets for remapping feature spaces to a lower or higher dimensionality.

These models are smaller and less computationally intensive than those they are based on. Thanks to the power of residual connections and a modular schema, they are highly scalable. As such, larger versions could be explored in the future. The current versions proved capable of competing with the other, larger and more demanding models. Although they still lag slightly behind in performance, they frequently beat those more powerful models in identifying certain types of manipulations. Of the two proposed models, Reseption V1 proved to be much better, likely because it focused more on inception modules. The Inception architecture performed almost as well as the ResNeXt architecture, while being about 4 times smaller, which lead me to believe that the concepts behind it would be more apt in a small model. This ultimately proved to be true. It is still possible that Reseption V2 could be stronger if they were both tested with increased network depth, since of the two it focuses more heavily on residual connections, which are well known to be highly scalable.

While training the Reseption models the validation loss curves were consistently below the training loss curves. I believe this is due to fairly strong data augmentation which was used in this project. Especially given that these models were trained from scratch, it is possible that they could benefit from weaker data augmentation.

The models used in this project were tested against a separate split from that used in training and validation. The findings concluded the results to be consistent, and model explainability analysis proved that the models do actually utilize facial features in their decisions. Considering the models boasted a >90% overall accuracy on the video classification task, I consider them to be successful. In addition, all models performed best on the two DeepFake tasks. While they were prepared by different teams and were modified in slightly different ways, they still fundamentally belong to the same category. As such, it was the only category with a larger sample size, so the performance increase is expected. Nevertheless, the main goal was to detect DeepFakes, so this is satisfactory. Reseption V1 outperformed both SotA models on DeepFakeDetection and it outperformed Inception V3 on Deepfakes.

It is worrying that some manipulation methods are so far ahead of the rest. NeuralTextures consistently proved to be the most difficult method to detect. This is consistent with the results in the publication on the dataset [18]. New forgeries are being developed with every day and existing ones are being improved, so it is important to have the appropriate infrastructure in place to detect them as soon as possible. Moving forward, I would like to explore the detection of other video alteration techniques.

For future work, given the appropriate hardware, I would like to evaluate the scalability of the Reseption models. Given their nature, a larger depth should only serve to improve their performance. It would be interesting to see whether the Reseption V2 model would end up outperforming Reseption V1. I would also like to check how they perform given different levels of data augmentation.

Given how well ResNeXt101 performed with its input size of 224x224, it is worth looking into decreasing the input size of the models. It is well known that low resolution samples are more difficult to classify. The reason for this is that there is less information in the images and, if they need to be upscaled, the information that is there is significantly modified leaving the original characteristics less recognizable. Most of the extracted faces are smaller than 299x299, which is the current input size, so they need to be scaled up. As such, they are stretched, which distorts the features present in the image. If this were to be beneficial to performance, it would have the additional bonus of slightly decreasing model size and complexity.

In conclusion, I have contributed two novel architectures for image classification and used them for the task of DeepFake detection. These models are based on the concept of inception modules and residual connections. Both are less expensive than the models which inspired their creation and perform very similarly. Reseption V1 was the better of the two and performed very well with a classification accuracy score of 93.04% and a balanced accuracy score of 91.81%. Future investigations will entail exploring the impact of changes to model structure as well as the training process. I would also like to test the models on manipulation methods other than those explored here.

Bibliography

[1] Introduction to Artificial Neural Networks. Enzo Grossi, Massimo Buscema. European Journal of Gastroenterology & Hepatology, 2008.

[2] A Comprehensive Study of Artificial Neural Networks. Vidushi Sharma, Sachin Rai, Anurag Dev. International Journal of Advanced Research in Computer Science and Software Engineering, 2012.

[3] A logical calculus of the ideas immanent in nervous activity. Warren S. McCulloch, Walter Pitts. Bulletin of Mathematical Biophysics, 1943.

[4] Deep Learning. Yann LeCun, Yoshua Bengio, Geoffrey Hinton. Nature, 2015.

[5] Convolutional Networks for Images, Speech, and Time-Series. Yann Lecun, Yoshua Bengio. The handbook of brain theory and neural networks, 1998.

[6] Gradient-Based Learning Applied to Document Recognition. Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner. IEEE, 1998.

[7] ImageNet Classification with Deep Convolutional Neural Networks. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. NIPS, 2012.

[8] Large-scale Video Classification with Convolutional Neural Networks. Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, Li Fei-Fei. CVPR, 2014.

[9] Receptive Fields and Functional Architecture of Monkey Striate Cortex. D. H. Hubel, T. N. Wiesel. J Physiol, 1968.

[10] Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. Kunihiko Fukushima. Biological Cybernetics, 1980.

[11] DeepFakes. https://github.com/deepfakes/faceswap

[12] DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection. Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales, Javier Ortega-Garcia. arXiv, 2020.

[13] Deep Learning for Deepfakes Creation and Detection: A Survey. Thanh Thi Nguyen, Cuong M. Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, Saeid Nahavandi. arXiv, 2020.

[14] Generative Adversarial Nets. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. arXiv, 2014.

[15] FaceForensics. https://github.com/ondyari/FaceForensics

[16] Official FaceForensics dataset splits. https://github.com/ondyari/FaceForensics/tree/master/dataset/

[17] Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. FaceForensics: A large-scale video dataset for forgery detection in human faces. arXiv, 2018.

[18] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, Matthias Nießner. FaceForensics++: Learning to Detect Manipulated Facial Images. arXiv, 2019.

[19] Brian Dolhansky, Russ Howes, Ben Pflaum, Nicole Baram, Cristian Canton Ferrer, AI Red Team, Facebook AI. The Deepfake Detection Challenge (DFDC) Preview Dataset. arXiv, 2019.

[20] Kaggle DeepFake Detection Challenge. https://www.kaggle.com/c/deepfake-detection-challenge

[21] Kaggle DeepFake Detection Challenge Leaderboard. https://www.kaggle.com/c/deepfake-detection-challenge/leaderboard

[22] Google contribution to FaceForensics++. https://ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html

[23] Face2Face: Real-Time Face Capture and Reenactment of RGB Videos. Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. IEEE, 2016.

[24] FaceSwap. https://github.com/MarekKowalski/FaceSwap/

[25] Justus Thies, Michael Zollhofer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. arXiv, 2019.

[26] ImageNet Large Scale Visual Recognition Challenge. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei. arXiv, 2015.

[27] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam. arXiv, 2017.

[28] SSD: Single Shot MultiBox Detector. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. arXiv, 2016.

[29] Mobilenet Face Detector. https://github.com/yeephycho/tensorflow-face-detection

[30] WIDER FACE: A Face Detection Benchmark. Shuo Yang, Ping Luo, Chen Change Loy, Xiaoou Tang. IEEE, 2016.

[31] Real-Time Multi-Scale Face Detector on Embedded Devices. Xu Zhao, Xiaoqing Liang, Chaoyang Zhao, Ming Tang, Jinqiao Wang. MDPI, 2019.

[32] Face Recognition Based on The Improved MobileNet. You Zhou, Yiyue Liu, Guijin Han, Yiping Fu. IEEE, 2019.

[33] An Evaluation of Convolutional Neural Network Models for Object Detection in Images on Low-End Devices. David Foley, Ruairi O'Reilly. CEUR-WS, 2018.

[34] JavaScript Object Notation (JSON). https://www.json.org/json-en.html

[35] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, Yi Yang. Random Erasing Data Augmentation. arXiv, 2017.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 2014.

[37] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. arXiv, 2017.

[38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Going deeper with convolutions. arXiv, 2014.

[39] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. arXiv, 2015.

[40] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. arXiv, 2016.

[41] Mingxing Tan, Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv, 2019.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. arXiv, 2015.

[43] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. arXiv, 2017.

[44] PyTorch torchvision pretrained models. https://pytorch.org/docs/stable/torchvision/models.html

[45] Pretrained models for PyTorch. https://github.com/Cadene/pretrained-models.pytorch

[46] EfficientNet model architecture implementation. https://github.com/qubvel/efficientnet

[47] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, Jiawei Han. On the Variance of the Adaptive Learning Rate and Beyond. arXiv, 2020.

[48] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, James Martens. Adding Gradient Noise Improves Learning for Very Deep Networks. arXiv, 2015.

[49] Xavier Glorot, Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. Thirteenth International Conference on Artificial Intelligence and Statistics, May 2010.

[50] Example image of a simple neural network. https://medium.com/ai-in-plain-english/my-notes-on-neural-networks-adf3e49657f8

[51] Stanford lecture cs231n on convolutional neural networks. https://cs231n.github.io/convolutional-networks/

[52] THOP module for counting operations in PyTorch models. https://github.com/Lyken17/pytorch-OpCounter

[53] LIME tool for model explainability analysis. https://github.com/marcotcr/lime

[54] "Why Should I Trust You?": Explaining the Predictions of Any Classifier. Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. arXiv, 2016.