Listening Beyond Words: Transfer Learning for Audio Deepfake Detection

Master's Thesis

Gustav A.P. Bonvang

Aalborg University
M.Sc.Eng. Cyber Security
Spring Semester 2023

Department of Electronic Systems
Aalborg University
Frederikskaj 12, 2450 København SV
www.es.aau.dk

AALBORG
UNIVERSITY

STUDENT REPORT

Title:
Listening Beyond Words: Transfer Learning for Audio Deepfake Detection

Participant(s):
Gustav A.P. Bonvang

Project Type:
Master's Thesis

Supervisors:
Jens Myrup Pedersen
Ashutosh Dhar Dwivedi

Project Period:
Spring Semester 2023

Number of pages: 51

Date Completeted:
June 1, 2023

Abstract:
This project investigates the topic of audio deepfake detection. First, existing research in the field is examined through which it is found that residual neural networks and the use of transfer learning have the potential to perform well in the context of audio deepfake detection. Thus, a novel approach to the task is proposed which is based on the ResNet50 network architecture and the use of transfer learning. Models are trained and evaluated using the *In-The-Wild* dataset, which is converted to a set of mel spectrograms and rescaled prio to use in the neural network. Several models are trained using different hyperparameters, including a range of baseline models which do not use transfer learning. The best model used transfer learning and achieved an accuracy of 96.7% and an F1-score of 95.5%, while a comparison to the non-transfer learning baseline models showed an average 21.90% increase in accuracy and 44.01% increase in F1-score when using transfer learning.

This work is licensed under CC BY-SA 4.0
All figures and illustrations were created by the author unless otherwise stated in caption.

Table of Contents

Preface

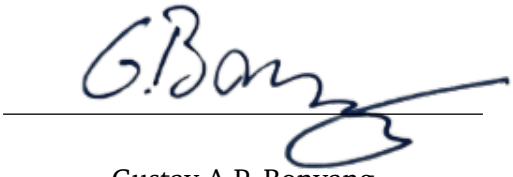
When considering the role that internet memes have played in my last five years as a university student, it seems only appropriate that my final project should be inspired by one. And while an Instagram reel of presidents swearing at each other while playing Minecraft together [1] might seem like harmless fun, it also highlights the dangers of how sophisticated generative AI models have become.

This thesis seeks to add to the research already conducted in the field by proposing a solution which utilises residual neural networks and transfer learning – tools in the deep learning toolbox which have shown great potential in related fields of research.

As my years as a student are drawing to an end, I am happy to have been able to immerse myself in a research field this interesting and relevant for my final project. Examining the research of previous work and the various creative approaches presented within it has been inspiring, and I am looking forward to watching the further developments which the future will undeniably bring.

Finally, I would like to thank my friends J and K for their professional sparring and motivational support during the course of the project. Your help has been invaluable, and I am lucky to have friends like you.

Copenhagen, June 1, 2023



Gustav A.P. Bonvang
gbonva18@student.aau.dk

NOMENCLATURE

ACC	Accuracy (as evaluation metric)
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASV	Automatic Speaker Verification
AUC	Area Under Curve
BiLSTM	Bidirectional Long Short-Term Memory
CNN	Convolutional Neural Network
DL	Deep Learning
EER	Equal Error Rate
FFT	Fast Fourier Transform
GAN	General Adversarial Network
KNN	K-Nearest Neighbors
LR	Logistic Regression
LSTM	Long Short-Term Memory
ML	Machine Learning
PRE	Precision (as evaluation metric)
QSVM	Quadratic Support Vector Machine
REC	Recall (as evaluation metric)
RNN	Recurrent Neural Network
ResNet	Residual Neural Network
SGD	Stochastic Gradient Descent
STFT	Short-Time Fourier Transform
SVM	Support Vector Machine
TTS	Text-to-speech
TTT	Time To Train
t-DCF	Tandem Detection Cost Function

Chapter 1
Introduction

Audio deepfakes refer to artificially generated audio which is created through the use of machine learning algorithms to replicate the voice of a real person. These techniques use deep neural networks to analyse and learn the unique patterns and characteristics of a person's voice, such as intonation, accent, and speech patterns, and then create new audio recordings that mimic the original voice with a high degree of accuracy [2].

Audio deepfakes have become a growing concern in recent years due to the potential for misuse, such as impersonation with criminal or political intent. An example of the former occurred in 2019 when cyber criminals called the CEO of a UK-based energy company, impersonating the CEO of the company's German parent company. This resulted in the CEO of the British company transferring €220 000 to an account controlled by the attackers in the belief that they were transferring money to a Hungarian supplier to the company [3]. A 2022 survey conducted by VMware [4] found that deepfake attacks were on the rise, with 66% of respondents reporting having experienced them. Of the respondents in question, 58% reported video as the most common type of deepfake experienced, while 42% reported audio as the most common type. Delivery methods included both email, mobile messaging, voice and social media [4].

In February 2023, a series of videos surfaced on the social media platform TikTok, in which current and previous presidents of the United States appear to be playing the video game Minecraft while voice chatting with each other and other prominent public figures such as political commentator Ben Shapiro and UFC commentator and podcaster Joe Rogan [1]. The videos utilise generative AI to mimic the voices of the public figures in situations which often involve bickering, profanity and controversial opinions. While this might seem harmless in an entertainment context, it illustrates a risk introduced by the increasingly sophisticated generative AI models which have started to appear in recent years – the potential for misuse to influence political agendas. An example of the gravity of this issue was seen in October 2019, when the Governor of California signed Assembly Bill 730 [5] – a bill which, ahead of the 2020 Presidential Elections, would:

> [...] prohibit a person, committee, or other entity, within 60 days of an election at which a candidate for elective office will appear on the

> ballot, from distributing with actual malice materially deceptive audio or visual media of the candidate with the intent to injure the candidate's reputation or to deceive a voter into voting for or against the candidate, unless the media includes a disclosure stating that the media has been manipulated [5].

Deepfake attacks do not only target companies and politicians. In April 2023, Danish newspaper *Berlingske* printed a story about a Canadian woman who was deceived by a criminal who called her claiming to be her grandson, using an AI generated model trained on audio clips of his voice [6]. The attacker posing as the grandson claimed to have been involved in a traffic accident, upon which the police had arrested him after finding illegal substances in the car. Another voice was then used to impersonate a police officer who instructed the woman on how to post bail, resulting with her transferring $58 350 to the attacker's account. Incidents like these have led the Federal Trade Commission to warn consumers about these kinds of calls from family members, recommending that they verify family members through other forms of communication [7].

1.1 Initial Problem Statement

With audio deepfakes posing a threat to both companies, political persons and private individuals, the need for a method for detecting deep fake attacks is becoming increasingly evident. As such, the main focus of this report will be surveying the existing research in the field and suggesting a method for audio deepfake detection.

This leads to the following initial problem statement:

> How are audio deepfakes generated, and how can they be detected? What are existing methods in the field, how effective are they, and how can they be improved?

This problem statement serves as a basis for chapter 2, which seeks to answer the questions presented in it. Said chapter then concludes in a final problem statement which serves as the basis for the remaining part of the project, in which a new method for audio deepfake detection is presented.

Chapter 2
Problem Analysis

This chapter examines the questions presented in the initial problem statement. First, a high-level introduction to deepfake generation is presented, upon which the concept of audio deepfake detection is investigated in further detail by looking at related literature. Relevant background information and theory is introduced and explained along the way, before the chapter concludes with a final problem statement, which serves as the basis for the remainder of the project.

2.1 Audio Deepfake Classification & Generation

Generally, spoofed audio can be divided into two main categories: AI generated and non AI generated. The non AI generated types include replay and mimicking, which refers to replaying recorded clips of a person's voice which may or may not have been altered, for nefarious purposes. While this type of attack has also been been proven detectable using deep learning techniques [8], non AI generated approaches are left out of the scope of this project, which instead focuses on deepfakes – spoofed audio created using deep neural networks. The focus area of the project is illustrated in red in figure 2.1.

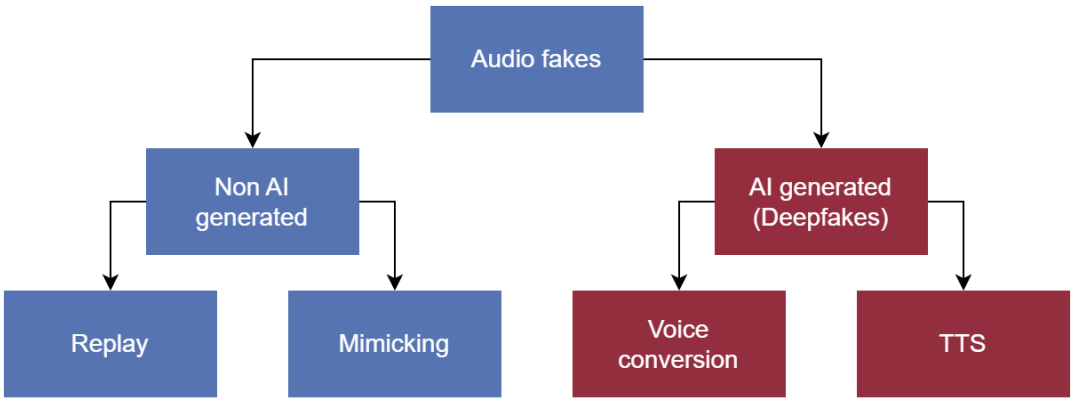
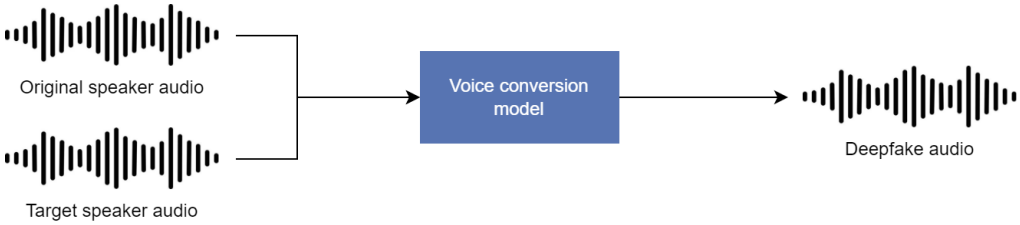
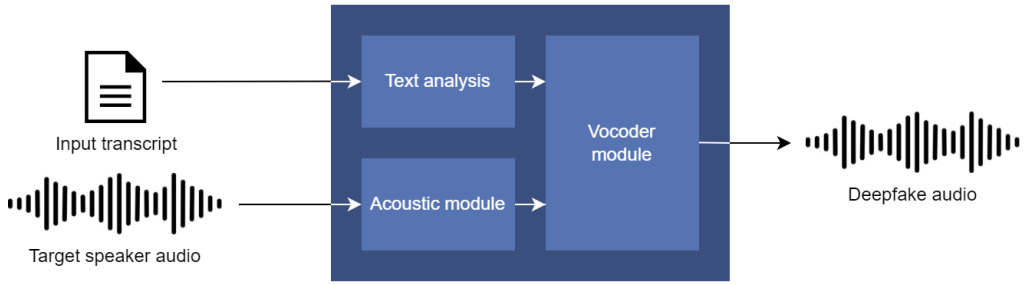


Figure 2.1: Types of audio spoofing attacks. Figure inspired by Khanjani et al. [2]

AI generated spoofed audio, hereafter referred to as audio deepfakes, is generally divided into two categories, as shown in figure 2.1: Voice conversion models and



(a) General approach for conversion based deepfake generation



(b) General approach for synthesis based deepfake generation

Figure 2.2: The two main approaches to audio deepfake generation. Figure inspired by Almutairi and Elgibreen [9].

voice synthesis models, also referred to as TTS (Text-To-Speech) models. The main difference between the two is that voice conversion models takes audio as a direct input and converts it to audio mimicking another persons voice, while TTS models are trained to accept text as an input, generating spoofed audio mimicking the voice of the target. Figure 2.2 illustrates the differences between the two.

Common for the two types is that they are generated using deep learning networks. In a 2023 survey on audio deepfake generation and detection, Khanjani et al. found that audio deepfakes are commonly generated using combinations of four types of neural networks: Convolutional neural networks (CNN), recurrent neural networks (RNN), general adversarial networks (GAN) and encoder-decoder (ED) networks [2].

Convolutional Neural Networks

A convolutional neural network (CNN) is a type of artificial neural network specifically designed for processing and analysing structured grid-like data, such as images and videos. The key component of a CNN is the convolutional layer, which applies filters or kernels to input data to extract relevant features. These filters excel at detecting local patterns and perform convolution operations to produce feature maps. CNNs are commonly used in the image classification realm but have also proven applicable in audio processing tasks. In this case, images known as spectrograms are typically generated from audio and fed to a CNN. This concept is further explored in section 3.3.

Recurrent Neural Networks
Recurrent Neural Networks (RNNs) are a type of neural network that are well-suited for sequential data processing, making them suitable for various tasks in audio processing. RNNs have a unique ability to capture temporal dependencies in sequential data by maintaining an internal memory or "hidden state" that can persist information across time steps. This characteristic makes RNNs effective for processing audio signals, as audio data is inherently sequential in nature.

Generative Adversarial Networks
Generative Adversarial Networks (GANs) are popular in the world of deepfake generation due to their ability to produce realistic image, video and audio material. The concept of a GAN consists of two key components: a generator network and a discriminator network. The generator network generates synthetic data, while the discriminator network evaluates the authenticity of both real and generated data. GANs work through an adversarial training process where the generator and discriminator networks compete against each other, leading to the refinement of the generator's ability to produce more realistic data, as well as the discriminators ability to distinguish it.

Encoder-Decoder Networks
Encoder-Decoder (ED) networks, are another type of neural network commonly used in audio deepfake generation. This kind of network consists of an encoder and a decoder, which work together to transform input audio data into a desired output representation. As such, they are most commonly used in conversion based deepfake generation. In this context, ED networks are employed to convert the voice of one speaker to mimic the voice of another speaker. The encoder part of the network extracts relevant features from the input audio, while the decoder part synthesises a new audio signal that resembles the desired target speaker.

While deepfake generation methods is an entire field of research in itself, only a brief introduction is given here, as this project will focus mainly on the detection aspect. This prompts a closer look at existing methods for audio deepfake detection presented in related literature.

2.2 Detection Methods

While most focus within the realm of deepfake detection focuses on image and video material [2], efforts have been made to develop solutions for audio deepfake detection as well. These efforts can generally be divided into two main categories: Those employing traditional machine learning (ML) methods and those using a deep learning (DL) based approach.

2.2.1 ML-based approaches

Rodríguez-Ortega et al. [10] proposed a logistic regression (LR) approach by extracting entropy features of bona-fide and spoofed audio and training a model to distinguish between them, achieving an accuracy of 0.98 and correctly predicting all instances of spoofed audio. This approach however suffered from a high need for manual pre-processing. The researchers also published the dataset that they created under the name *H-Voice*[11].

Singh and Singh [12] later examined a series of different ML classifiers and compared their performance for both binary and multi-class classification scenarios. They found that the best performing classifier was the quadratic support vector machine (QSVM) for binary classification, which achieved an accuracy of 0.9756 and hence outperformed both the linear discriminant, quadratic discriminant, linear support vector machine (SVM), weighted K-nearest neighbors (KNN), boosted trees ensemble and LR models which were also trained.

Liu et al. [13] proposed a way to detect fake stereo audio, in which they implemented and compared an SVM approach and a deep learning approach utilising a custom designed convolutional neural network (CNN). They found that while both models performed well, the CNN was more robust than the SVM. Furthermore, the SVM suffered from the same need for manual pre-processing of the data as the LR approach suggested by Rodríguez-Ortega et al. [10].

Based on the existing research focusing on ML-based approaches, it seems that DL-based models might have some advantages over traditional ML-based models in terms of both robustness and less need for pre-processing. This prompts a deeper look into more of the existing DL-based solutions.

2.2.2 DL-based approaches

DL-based approaches differ from traditional ML approaches in several ways. DL models have more complex architectures with interconnected layers, allowing them to learn intricate patterns in the training data. DL models can combine feature extraction and classification, significantly reducing the need for manual pre-processing that exists with traditional ML-based approaches. While DL-based models typically work well with large amounts of data due to efficient hardware utilisation and optimization algorithms avaible, they also typically require more training data to achieve high performance.

Generally, most research in DL methods for audio deepfake detection uses variants

of either recurrent neural networks (RNN) or convolutional neural networks (CNN). RNNs are well-suited for sequential data processing, making them suitable for tasks involving audio, such as speech recognition or music generation. They can capture temporal dependencies and handle variable-length inputs. RNNs process audio sequentially, considering the context of previous inputs to make predictions. CNNs, meanwhile, are primarily designed for spatial data processing such as images. However, they can also be applied to audio recognition tasks by treating audio as a spectrogram or histogram. An example of the latter was proposed by Ballesteros et al. [14] who developed a model called *Deep4SNet* – a CNN-based classifier for analysing audio histograms, which achieved a global accuracy of 0.985 as well as a precision of 0.985 and recall of 0.944 for deep-voice based recordings.

In general, CNNs excel at capturing local patterns and features, making them useful in audio classification, and they can efficiently process large input volumes and extract hierarchical representations. However, they may not capture long-term temporal dependencies as effectively as RNNs.

Lataifeh et al. [15] compared the performance of the bidirectional long short-term memory (BiLSTM) RNN variant, a CNN and traditional ML-based approaches for deepfake detection. While both the BiLSTM and CNN models performed well (avg. F1-scores of 0.9776 and 0.9744 during validation, respectively), it was found that the CNN was the best model for the task when evaluated on a test data set. There, the CNN model achieved an accuracy of 0.9433 while the BiLSTM model achieved an accuracy of 0.9100.

While both RNNs and CNNs are viable approaches for audio deepfake detection, they can also be combined to utilise the qualities of them both. This was demonstrated by Chintha et al. [16] who developed a model called CRNN-Spoof which combines a CNN model and a BiLSTM model to classify audio deepfakes. This approach was considered a success with a 4.27% equal error rate (EER) using the ASVspoof 2019 dataset [17]. Wijethunga et al. [18] demonstrated a similar method of combining CNNs and RNNs and achieved a success rate of 94% in audio deepfake detection.

Another approach proposed by Subramani and Rao [19] used only CNN models, specifically the EfficientCNN and RES-EfficientCNN, the latter of which is a version of the former which utilises residual connections (explained in further detail in section 2.3). They achieved F1-scores of 97.61% for the RES-EfficientCNN model and 94.14% for the EfficientCNN model, illustrating that residual connections can improve performance when using CNNs for deepfake detection. This prompts a closer look into what makes residual neural networks unique and their use in the audio deep learning domain.

2.3 Residual Neural Networks

A residual neural network, or ResNet, is a type of convolutional neural network that learns residual functions with reference to the layer inputs. The concept was developed by He et al. [20] and won the ImageNet 2015 competition [21].

The core idea of ResNets is to use residual connections that allow information to bypass certain layers and be directly propagated to subsequent layers. This is achieved by adding *residual connections*, also often referred to as *skip connections*, that connect earlier layers to deeper layers. By doing so, ResNets enable a direct flow of information from shallower layers to deeper layers in the model, facilitating the training process. The concept of a skip connection is illustrated in figure 2.3.

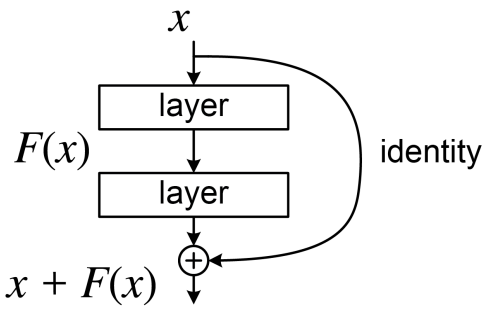


Figure 2.3: An example of a skip connection skipping two layers. Figure by "LunarLullaby" [22], licensed under CC BY-SA 4.0.

Residual connections help reduce the degradation problem, where deeper networks start to perform worse than shallower networks due to the difficulty of optimising deep architectures. With residual connections, it becomes easier for the network to learn residual mappings, which capture the difference between the desired output and the current output of a layer. By learning these residual mappings, the network can better adapt and improve its predictions as the depth increases.

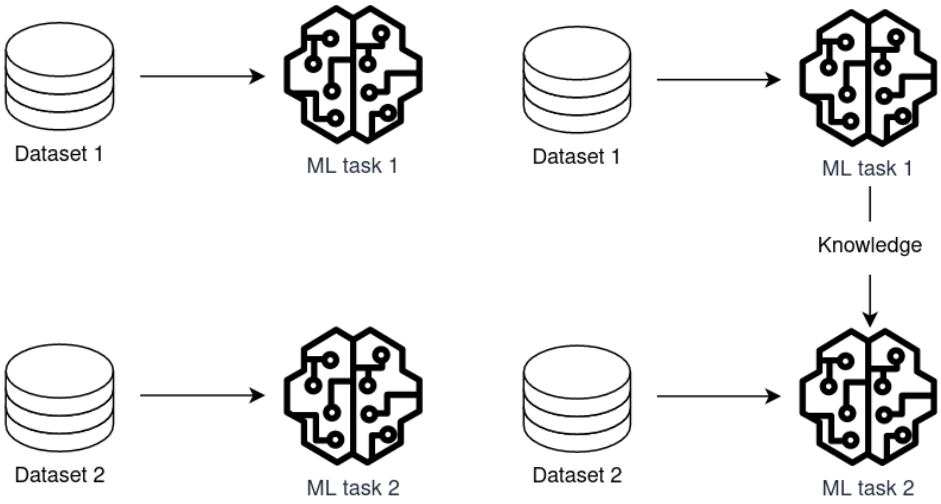
ResNets have been adopted for use in the audio recognition domain. Hershey et al. [23] compared four different CNN architectures on a dataset consisting of vast amounts of audio data obtained from YouTube: AlexNet, VGG, Inception V3 and ResNet50, the latter of which is a 50-layer ResNet variant. They found not only that these four CNNs, which are commonly used in image classification, performed well in the audio classification domain, but also that ResNet50 was the one of the models which performed best. While this specific study focused on identifying topics in the content of the audio, it seems reasonable to assume that the ResNet50 architecture would also perform well in identifying audio features in the deepfake detection domain.

Rahul T.P. et al. [24] also proposed an approach using residual neural networks, employing the 34-layer ResNet34 model. They succeeded in solving the vanishing gradient problem and achieved an EER of 5.32% and a t-DCF (Tandem Detection Cost Function) score of 0.1514%. The approach in this study differed from many others in that *transfer learning* was used, meaning that a pre-trained model was employed and fitted to the training data set. This concept is covered in further detail in the following section.

2.4 Transfer Learning

Transfer learning is a ML technique that allows a model to leverage knowledge gained from solving one task to improve its performance on a different but related task. Instead of starting the learning process from scratch, transfer learning enables a model to benefit from the knowledge gained by pre-training on a large dataset or a similar task.

In transfer learning, a model is typically first trained on a source task with a large dataset, such as a general image recognition task. It learns general patterns and features that are relevant to the source task. Then, instead of discarding the learned knowledge, the model reuses or adapts it to a target task with a smaller dataset or a different domain. In practice, this means reusing and adjusting the parameters/weights obtained during the original training process. The concept of transfer learning is illustrated in figure 2.4.



(a) Training a model for a different task without using transfer learning.

(b) Training a model for a different task using transfer learning.

Figure 2.4: Illustration of the concept of transfer learning.

By using transfer learning, models can often achieve better performance in the target task with less training data and time. The pre-trained model acts as a starting point, capturing lower-level features that are useful for various tasks, while the fine-tuning process adjusts the model's parameters to better suit the specifics of the target task.

Rahul T.P. et al. [24] were not the only ones to demonstrate the usefulness of transfer learning within the audio classification domain. Choi et al. [25] proposed a method for identifying genre, mood, instrumentation and other descriptors for music and found that the pre-trained model outperformed the model using randomly initialised weights in all test cases. Furthermore, Suratkar and Kazi [26] proposed a model called *EfficientNet* which combines a pre-trained CNN with an LSTM network for deepfake video detection. They achieved an accuracy of 98.69% and an AUC (area under curve) value of 97.26%, thus indicating that transfer learning may indeed be applicable in the deepfake detection domain.

2.5 Summary & Problem Statement

Throughout this chapter it was found that audio deepfake detection can generally be divided into ML-based and DL-based approaches. While ML-based methods have been proven to work well, they often require great amounts of manual preprocessing and hence most research focuses on DL-based approaches.

In DL-based approaches, variants of either RNNs or CNNs or combinations of the two are typically used. CNNs developed for use in image recognition, such as residual neural networks, have been proven to work well in audio classification scenarios when used in combition with visual audio representations such as spectrograms and histograms. Furthermore, utilising transfer learning has been shown to improve performance in audio classification and related deepfake detection tasks.

Based on these findings, this project will focus on applying transfer learning for audio deepfake detection using a ResNet architecture, based on the following problem statement:

> How can a residual neural network using transfer learning be applied for audio deepfake detection?

Along with the following sub-questions:

i) How well does the model perform?

ii) Does the use of transfer learning improve performance when compared to a similar model that uses randomly initialised weights instead of transfer learning?

The following section further defines the overall scope of the research, outlining the specific areas and boundaries that will be addressed to establish a clear foundation for the subsequent chapters.

2.5.1 Scoping & Focus

As described in section 2.5, the scope of the research presented in this project is to examine the impact of applying transfer learning when using a ResNet for audio deepfake detection.

As such, the project will not compare different model types. The ResNet architecture is chosen based on its merits both in the deepfake detection domain in general and in the audio classification domain, and as such is the only architecture implemented.

Instead of comparing different model architectures, the evaluation of the developed model will focus on overall performance and a comparison to a similar model which does not utilise transfer learning. This means that the aim of the project is not to develop the most accurate way of detecting audio deepfakes, but rather to build upon existing research by investigating the impact that the use of transfer learning can have when using ResNets for audio deepfake detection.

Furthermore, the scope of the project is limited to audio containing speech in the English language. While the need for robust audio deepfake detection is not limited in terms of geography, the focus of the project is demonstrating the effectiveness of transfer learning and not generalising across different languages.

Finally, the aim of the project is not to develop an end product for audio deepfake detection in practice. Rather, its aim is to demonstrate the use of ResNets and transfer learning as a viable approach to identifying audio deepfakes, as well as to evaluate the performance of the solution.

Chapter 3
Design & Implementation

This chapter introduces the overall solution design and covers relevant design choices that were made prior to and during the implementation phase. All steps in the implementation are then covered in detail with code snippets and appertaining explanations.

3.1 Solution Overview

Figure 3.1 illustrates the overall design of the proposed solution. The idea is to generate spectrograms from a dataset of audio data, preprocess them and feed them to a neural network consisting of a ResNet architecture and an output layer with a single node that uses a sigmoid activation function for binary classification. In this case, an output of 1 indicates that the input audio is a deepfake (spoof), while an output of 0 indicates that the audio clip is bona-fide (real).

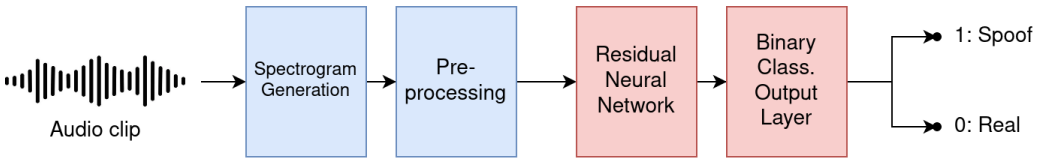


Figure 3.1: Overview of the proposed solution.

Different models can then be trained based on a number of varying factors such as:

- Model hyperparameters
- Parameters set during spectrogram generation
- Whether to use transfer learning, i.e. pre-trained weights for the ResNet model

Before implementing the solution, a number of design choices must be made. These are described in the following section.

3.2 Design Choices & Setup

Before designing and implementing the model, some choices must be made regarding how to obtain an appropriate dataset as well as which machine learning framework, specific ResNet model, programming language and hardware will be used in the implementation. These considerations are explained in this section.

3.2.1 Training Data

Several different data sets containing audio files for deepfake detection exist. They differ greatly in parameters such as size, language, number of speakers, gender of speakers, generative model used and amount of preprocessing done beforehand. This section introduces some of the most popular, publicly available ones and discusses pros and cons of each.

ASVspoof
The ASVspoof dataset [17] is a collection of audio recordings and corresponding metadata, designed to support research into automatic speaker verification (ASV) systems. The dataset was initially created to address the problem of spoofing attacks on ASV systems but also includes a version specifically intended for use in training and evaluating audio deepfake detection systems. As the dataset was created as part of a challenge, it has been widely used in research on the topic, and several papers have been published with results achieved using this dataset. A drawback to the dataset within the context of this project is that it contains audio in multiple languages, which makes it less suitable in the scope of this project as described in section 2.5.1.

Fake-or-Real (FoR)
The Fake-or-Real dataset [27] was developed by researchers at the Audio Processing Techniques Lab at York (APTLY). It consists of 195 000 audio files containing both bona-fide and deepfake speech, the latter of which was created using modern TTS models like Deep Voice 3 and Google's Wavenet. The dataset exists in several versions with varying degress of preprocessing, including a normalised version which is balanced in terms of the speakers' genders and has a standardised sample rate and volume.

H-Voice
The H-Voice dataset created by Ballesteros et al. [11] and differs from the datasets mentioned above in that it does not consist of audio files. Instead, the audio is represented as histograms. It consists of a total of 6672 histograms, of which the deepfake ones are generated using both conversion- and synthesis based methods.

WaveFake
WaveFake is another dataset for audio deepfake detection presented by Frank and Schönherr [28]. It consists of 117 985 audio clips from various sources – some are recordings of actual human speech while others are generated using various TTS models. The dataset however contains speech in both English and Japanese, meaning that it is less suitable for use in this project.

In-the-Wild
In-the-Wild [29] is a dataset created in 2022 by Müller et al. [30] for the primary purpose of evaluating the performance of audio deepfake detection models on "in-the-wild" data – audio which has been gathered from around the internet and which one might encounter in real life. It differs from the other datasets mentioned here primarily in that it focuses on public persons. The dataset contains collected audio – both bona-fide and deepfake – for 58 celebrities and politicians, ranging from Tupac Shakur to Queen Elizabeth II and contains 20.8 hours of bona-fide audio and 17.2 hours of deepfake audio, resulting in an average of 23 minutes of bona-fide and 18 minutes of deepfake audio per speaker. It is standardised in terms of sample rate and file type, however besides this little preprocessing has been done.

Ultimately, In-The-Wild is chosen as the dataset for use in this project, as it provides a decent amount of high-quality, labeled audio data in English. Furthermore, while the audio is preprocessed in terms of sample rate and file type, it has not been preprocessed further, allowing for experimentation with different types and degrees of preprocessing. In addition, the audio samples being recordings of famous people plays well into the need for robust audio deepfake detection to avoid misuse for political purposes which is described in chapter 1.

3.2.2 Machine Learning Framework & Architecture

In order to build, train and evaluate models, a framework is needed. For this purpose, TensorFlow is used, based on number of reasons. TensorFlow is a popular and well-documented framework developed by Google Brain for building and deploying machine learning models. It allows for efficient execution on both CPUs and GPUs, and offers a collection of pre-built neural network layers, optimization algorithms, and tools for model visualisation and deployment.

Another advantage to using TensorFlow is the Keras API, which is a high-level library that runs on top of TensorFlow and provides a user-friendly and intuitive API for building and training deep learning models. It simplifies the implementation process by offering a modular approach to model design and provides a simple way to define and stack layers, specify activation functions, and configure various parameters of the model.

Among the pre-built models available in the Keras API is a range of ResNet models than can be loaded and modified to suit the specific needs for the task at hand. Table 3.1 shows an overview of the ResNet variants included with Keras.

Model	Size (MB)	Accuracy		Parameters	Depth	Time per inference step (ms)	
		Top-1	Top-5			CPU	GPU
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6

Table 3.1: ResNet variants available in the Tensorflow Keras API [31].

Variants are provided with different depths, resulting in different complexity and training times. For this project, ResNet50 is chosen as it provides a balance between performance and complexity. Furthermore, since the shallower ResNet34 model has been proven viable in the context of audio classification [24], a deeper model is not deemed necessary in the scope of this project.

ResNet50

ResNet50, as the name implies, is a ResNet variant which consists of 50 layers – one initial convolution layer, 16 primary building blocks of three convolution layers each, and an output layer. The architecture of the model is presented in figure 3.2.

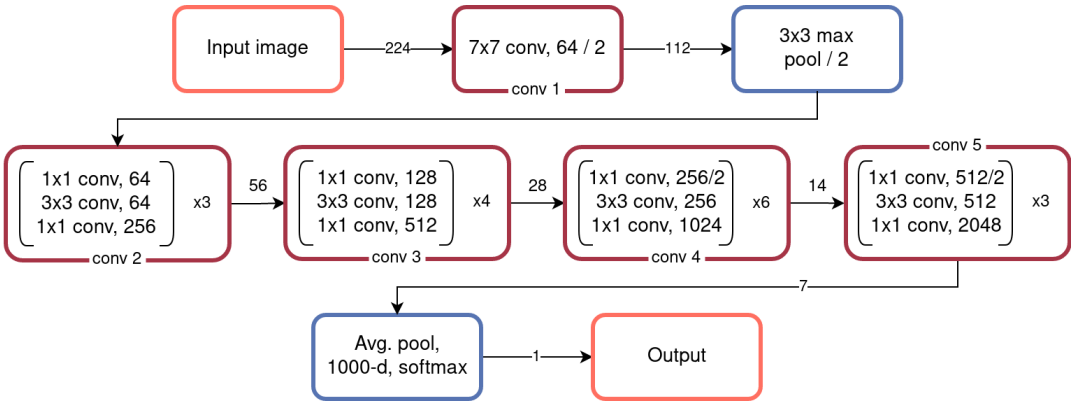


Figure 3.2: ResNet 50 architecture based on table by He et al. [20].

The model takes an input image (or other two-dimensional array-like type of data) with a dimension of 224 by 224. The initial convolution layer has a kernel size of

7x7, 64 different kernels and a stride (kernel step size) of 2, yielding an output of 112 by 112. The initial convolution layer is followed by a max pooling layer with a kernel size of 3x3 and stride 2, which in turn is followed by the functional building blocks.

The building blocks consist of three layers each and are repeated 3, 4, 6 and 3 times respectively. This is where the residual connections explained in section 2.3 are introduced.

After the functional building blocks comes the global average pooling layer which is applied to reduce the spatial dimensions to a 1x1 feature map. The output of this layer is flattened and connected to a fully connected layer. This layer combines the features learned by the previous layers to make predictions.

As the stock ResNet50 architecture was developed for multi-class image classification, the fully connected layer ends with a softmax activation function to produce class probabilities. This can however be changed to use a sigmoid activation function instead for binary classification purposes.

3.2.3 Development Setup

Table 3.2 shows an overview of the system used for training and evaluating models. This information is included for reference, as the time to train (TTT) relies heavily on the hardware used in the training process, and to facilitate reproduction of the results of the project by following the same implementation steps.

System	**OS**	Ubuntu 22.04.5 LTS
	Kernel	5.19.0-42-generic
CPU	**Model**	Intel Core i7-13700F
	Cores	16 (8 @ 4.1GHz + 8 @ 5.1GHz)
GPU	**Model**	NVIDIA RTX3070
	Cores	5888
	Memory	8GB GDDR6
RAM		32 GB DDR4 @ 3600MHz

Table 3.2: Overview of the system used to train and evaluate the models.

The development environment consists of a docker container running Ubuntu 22.04.5 with the following relevant software installed:

- Python 3.8.10
- Tensorflow 2.12.0
- NVIDIA System Management Interface (SMI) 530.30.02
- NVIDIA CUDA 12.1

The NVIDIA SMI and CUDA drivers allow for GPU acceleration which, as shown in table 3.1, greatly the reduces time per inference step – i.e. the time needed to train models. Besides the above mentioned software packages, various Python libraries such as NumPy, Matplotliib and PIL (Python Imaging Library).

3.3 Spectrogram Generation

In order to be able to process audio data using the ResNet50 model, it must be transformed into an image. This is done by generating a *spectrogram* – a visual representation of the spectrum of frequencies in a signal as it varies with time. It is a 3-dimensional plot of time, frequency, and amplitude, where the amplitude of a signal is represented by the color or intensity of the plot at each point in time and frequency.

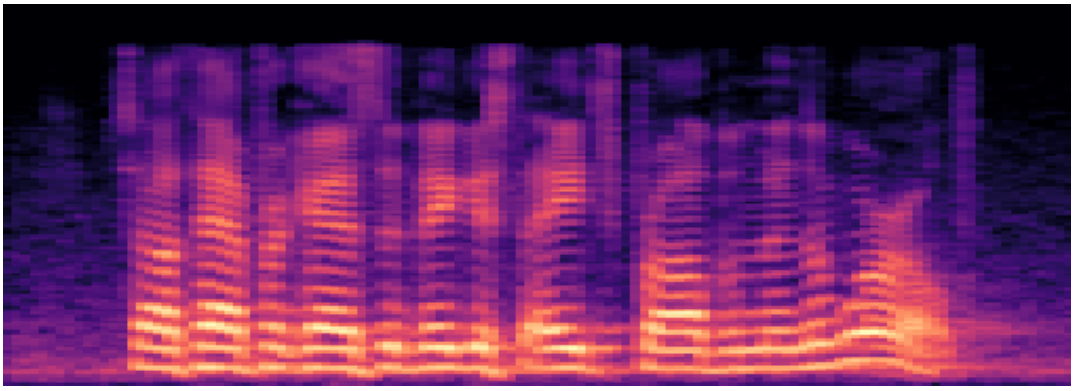


Figure 3.3: An example of a spectrogram. The x-axis represents time and the y-axis represents frequency, while the color indicates amplitude.

Spectrograms can be generated in a number of different ways with the most common one being the short-time Fourier transform (STFT). This method involves dividing

the signal into overlapping segments of fixed length, applying a Fourier transform to each segment to obtain its frequency content, and then plotting the resulting frequency content over time. In practice, this is done by applying a window function to each segment of the signal. The window function is typically a smooth, tapered function that gradually reduces the amplitude of the signal towards the edges of the segment, with popular choices being the Hamming window, Hanning window, and Blackman window. The entire process is illustrated in figure 3.4.

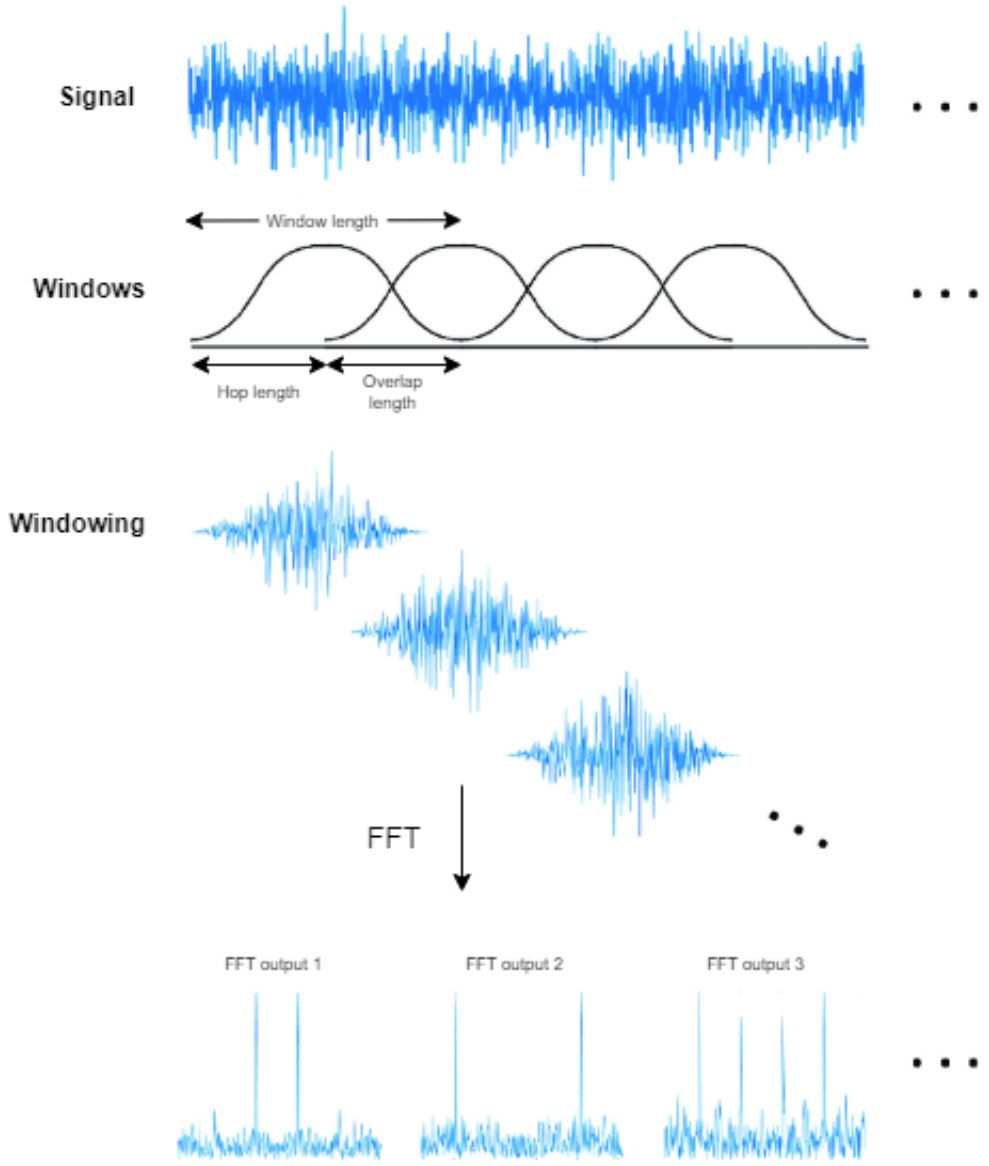


Figure 3.4: Illustration of the short-time Fourier transform. Figure adapted from Jeon et al. [32], licensed under CC BY 4.0.

The window function is applied to the signal, upon which STFT is performed on the resulting signal segments using the FFT (Fast Fourier Transform) algorithm. This results in a number of graphs representing amplitude of individual frequencies, which are then mapped along the x-asis with amplitude represented in color to form a spectrogram.

3.3.1 The mel scale

When working with audio recognition, it can be useful to convert the frequency of the input signal from hertz to *mel* during preprocessing. The mel scale was proposed by Stevens et al. in 1937 [33] and is a non-linear frequency scale based on the idea that humans are more sensitive to changes in lower frequencies than higher frequencies. It takes into account the way the human ear perceive pitch and groups frequencies together based on their perceptual similarity. This makes it a useful scale for tasks like speech and audio processing, where human perception is an important consideration. The mel scale is defined in equation 3.1 and illustrated in figure 3.5.

$$m = 2595 \log(1 + \frac{f}{700}) \tag{3.1}$$

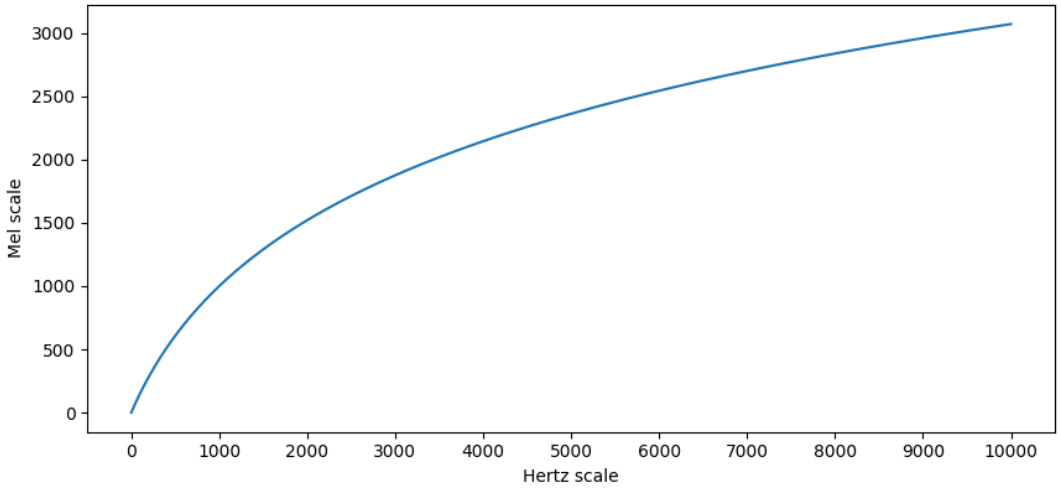
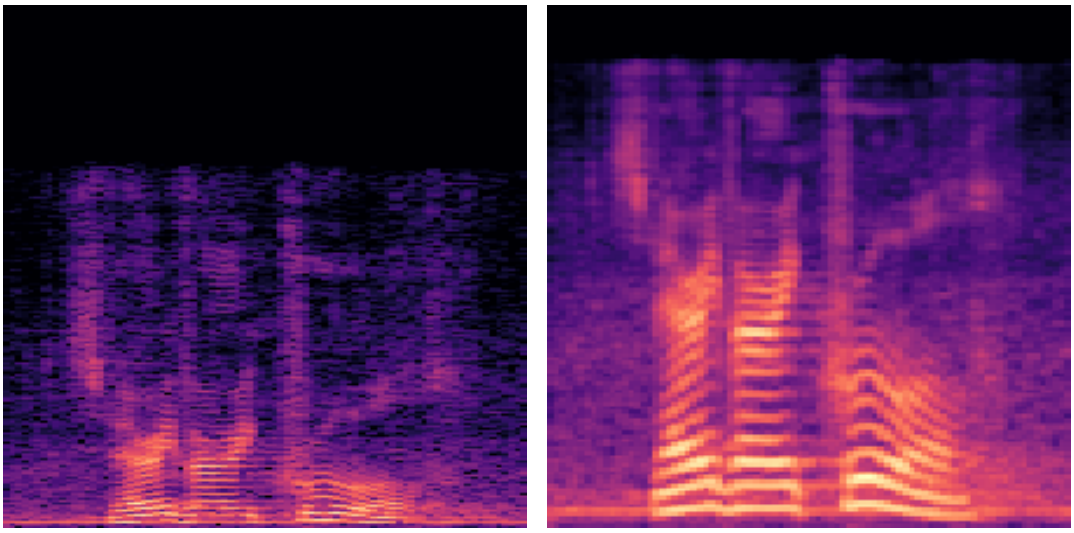


Figure 3.5: Graph showing the correlation between frequency represented in Hz and in mel.

When plotting a spectrogram generated from a sound clip containing human speech, individual frequencies can be more clearly discerned from each other when the sound is converted from hertz to mel. This is illustrated in figure 3.6 which shows two spectrograms of the same audio clip – one with frequency represented in hertz and the other in mel. Seeing as the ResNet50 model is limited to an input image with a size

of 224x224 pixels, this can be useful in making sure that the frequency ranges in which human speech is present are dominant in the generated spectrograms, hopefully leading to more accurate learning of features in the sound waves and better performance for the model.



(a) A spectrogram of human speech shown in the hertz scale.

(b) A spectrogram of the same file with frequencies converted to the mel scale.

Figure 3.6: Two spectrograms of the same audio clip showing frequency in hertz and mel, respectively.

3.3.2 Implementation

Figure 3.7 shows an overview of the implementation with the spectrogram generation process marked in green. Audio clips are loaded from the original dataset and used to generate spectrograms which end up comprising the image dataset used to train and evaluate the model. The entire code for this part of the implementation is included in appendix A.3.

In practice, the spectrograms are generated using the `librosa` library for Python. It is a high-level library built on top of other scientific computing libraries in Python, such as NumPy and SciPy, and provides a wide range of functions and tools for tasks such as loading and analysing audio files. In this case, the built-in function for generating spectrograms using the FFT algorithm is used.

Listing 3.1 shows how the parameters for spectrogram generation are specified. The window size – i.e. number of samples pr. FFT – is defined as `n_fft`, while `hop_length` represents hop length, i.e. distance between windows (for reference see figure 3.4). The `factor` variable is a value which specifies the ratio between the length of the

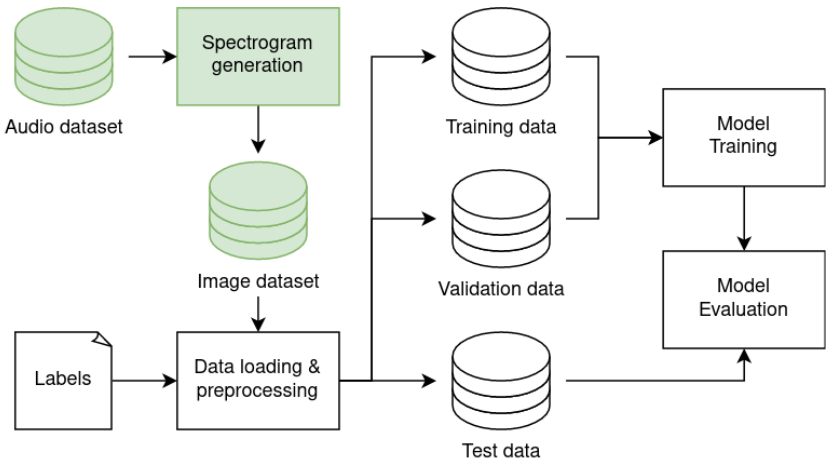


Figure 3.7: Overview of the implementation with spectrogram generation process marked in green.

input audio clip and the width of the resulting spectrogram. A factor of one means that 1 second of audio results in 224 pixels of width.

```
samplecount = 31779
n_fft = 2048
hop_length = 512
factor = 1
outpath = "/tf/spectrograms/1_NOISY_N2048_H512"

if os.path.isdir(outpath):
    sys.exit("Path already exists, exiting")
else:
    os.mkdir(outpath)
```

Listing 3.1: Code for defining parameters and paths for use in spectrogram generation.

As shown in listing 3.2, the program then enters a loop where it iterates through all the files in the dataset. For each file, it checks if a spectrogram already exists in the output folder so as not to perform any redundant operations. The audio is then loaded into memory and the length of each clip is saved in the `audiolength` variable. The audio is then converted from hertz to mel and a the spectrogram is created using librosa's `melspectrogram` function. Finally, in line 21 `np.abs` converts all values in the spectrogram instance to positive values, before the `power_to_db` function converts the amplitude to decibel.

```
for i in range(samplecount):

    input_file = "/tf/release_in_the_wild/" + str(i) + ".wav"
    output_file = outpath + "/" + str(i) + ".png"

```

```
    if not(os.path.isfile(output_file)):

        signal, sr = librosa.load(input_file)
        audiolength = librosa.get_duration(
            y=signal,
            sr=sr,
            hop_length=hop_length,
            n_fft=n_fft)

        mel_signal = librosa.feature.melspectrogram(
            y=signal,
            sr=sr,
            hop_length=hop_length,
            n_fft=n_fft)

        spectrogram = np.abs(mel_signal)
        power_to_db = librosa.power_to_db(spectrogram, ref=np.max)
```

Listing 3.2: Loop for spectrogram generation.

In listing 3.3, the spectrogram is finally rendered using matplotlib. The default axes are removed so as to only include the actual spectrogram content in the output data which is subsequently fed to the model.

```
        # Plotting the mel spectrogram
        fig = plt.figure()
        fig.set_size_inches(factor*audiolength, 1)
        ax = plt.Axes(fig, [0., 0., 1., 1.])
        ax.set_axis_off()
        fig.add_axes(ax)

        # Display and save the final spectrogram
        librosa.display.specshow(
            power_to_db,
            sr=sr,
            hop_length=hop_length,
            cmap="magma")

        plt.savefig(
            output_file,
            dpi=224,
            bbox_inches="tight",
            pad_inches=0)

        plt.close()
```

Listing 3.3: Rendering and exporting spectrogram using matplotlib.

3.4 Data Loading & Preprocessing

After the audio dataset has been converted into spectrograms, the resulting image dataset is almost ready to be used for training and evaluating the model. It is however still necessary to perform some preprocessing on the data, which is done when it is loaded into the main program used to train and evaluate models. Figure 3.8 shows an overview of the entire implementation with the data loading process marked in blue. The code for this part of the implementation can be seen in its entirety in appendix A.1.

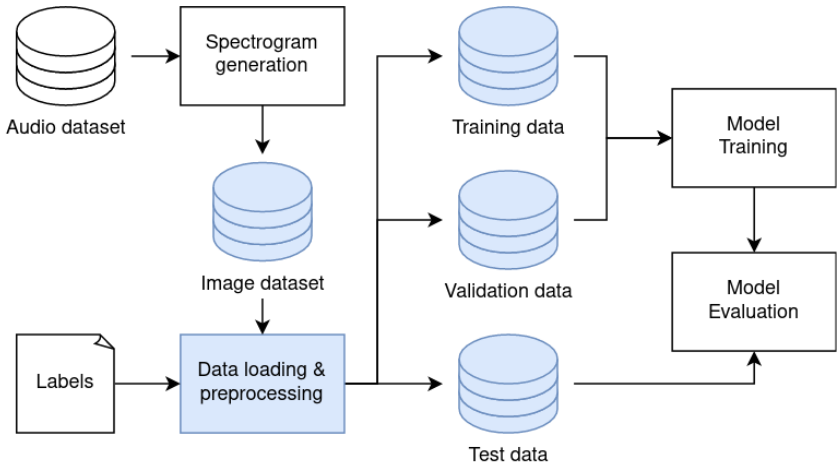


Figure 3.8: Overview of the solution with the part of the program used to load data marked in blue.

In the code shown in listing 3.4, the path for the spectrograms is first defined along with the path for the .csv file containing labels and the fraction of the dataset to reserve for testing when loading the data.

A function, `dataSplit()`, is then called. This function imports the data to a working directory called "workdir" and sorts the files into subfolders according to their label and whether they are to be used for training/validation or testing. Furthermore, it clears the working directory on each run, which enables loading the data with different ratios between training, validation and test data. The resulting folder structure of the "workdir" directory is shown in figure 3.9. The `dataSplit()` function is not explained in detail here but can be seen in its entirety in appendix A.2.

```
1 inputData = "/tf/spectrograms/1_NOISY_N2048_H512"
2 labels = "/tf/labels.csv"
3 testSplit = 0.1
4
5 dataSplit(inputData, labels, testSplit)
6
```

```
training_dir = pathlib.Path("/tf/spectrograms/workdir/training").
    with_suffix('')
test_dir = pathlib.Path("/tf/spectrograms/workdir/test").with_suffix('
    ')

test_count = len(list(test_dir.glob('*/*.png')))
train_count = len(list(training_dir.glob('*/*.png')))
validation_split = test_count / train_count
```

Listing 3.4: Code for splitting dataset in training/evaluation and test sets.

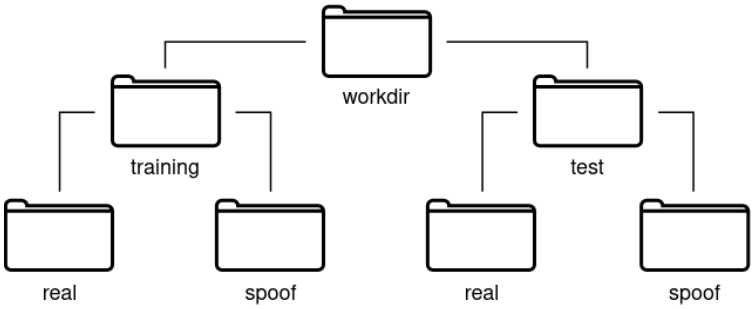


Figure 3.9: Folder structure of the "workdir" directory.

Executing the code shown in listing 3.4 results in the following output:

```
Cleared working directory, loading new files...
Found 31779 files in /tf/spectrograms/1_NOISY_N2048_H512
Reserving 3177 files for testing and using 28602 files for training
and validation.
Succesfully loaded 31779 files.
```

Listing 3.5: Output when executing the code shown in listing 3.4

In this case, 10% of the dataset is reserved for testing and moved into the `test` subfolder in the `workdir` directory.

Next – as shown in listing 3.6 – image size and and batch size are specified, before three dataset objects are created for training, validation and testing, respectively. As of now, the code reserves as large a portion of the dataset for validation as for testing, however this can easily be changed in order to experiment with different division ratios.

During the dataset creation process, the spectrograms are resized by cropping them at a random point along the x-asis. As explained in section 3.3, the spectrograms generated have a height of 224 pixels and a width corresponding to 224 pixels multiplied by the length of the audio clip in seconds, meaning that a random crop will result in a square image of 224x224 pixels.

```
batch_size = 50
img_height = 224
img_width = 224

print("\033[1mCreating training and validation datasets:\033[0m")
training_ds, validation_ds = tf.keras.utils.
    image_dataset_from_directory(
    training_dir,
    validation_split=validation_split,
    subset="both",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    crop_to_aspect_ratio=True
)

print("\n\033[1mCreating test dataset:\033[0m")
test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    crop_to_aspect_ratio=True
)

class_names = training_ds.class_names
print("\nNames of",str(len(class_names)), "classes:", class_names)
```

Listing 3.6: Code for loading spectrograms as Tensorflow dataset objects.

The code shown in listing 3.6 produces the output shown in listing 3.7 below:

```
Creating training and validation datasets:
Found 28601 files belonging to 2 classes.
Using 25423 files for training.
Using 3178 files for validation.

Creating test dataset:
Found 3178 files belonging to 2 classes.

Names of 2 classes: ['real', 'spoof']
```

Listing 3.7: Output of code shown in listing 3.6.

In order to manually verify that the spectrograms have been loaded and preprocessed correctly, nine random spectrograms from the training dataset are selected and plotted along with their corresponding labels, resulting in the image shown in figure 3.10.

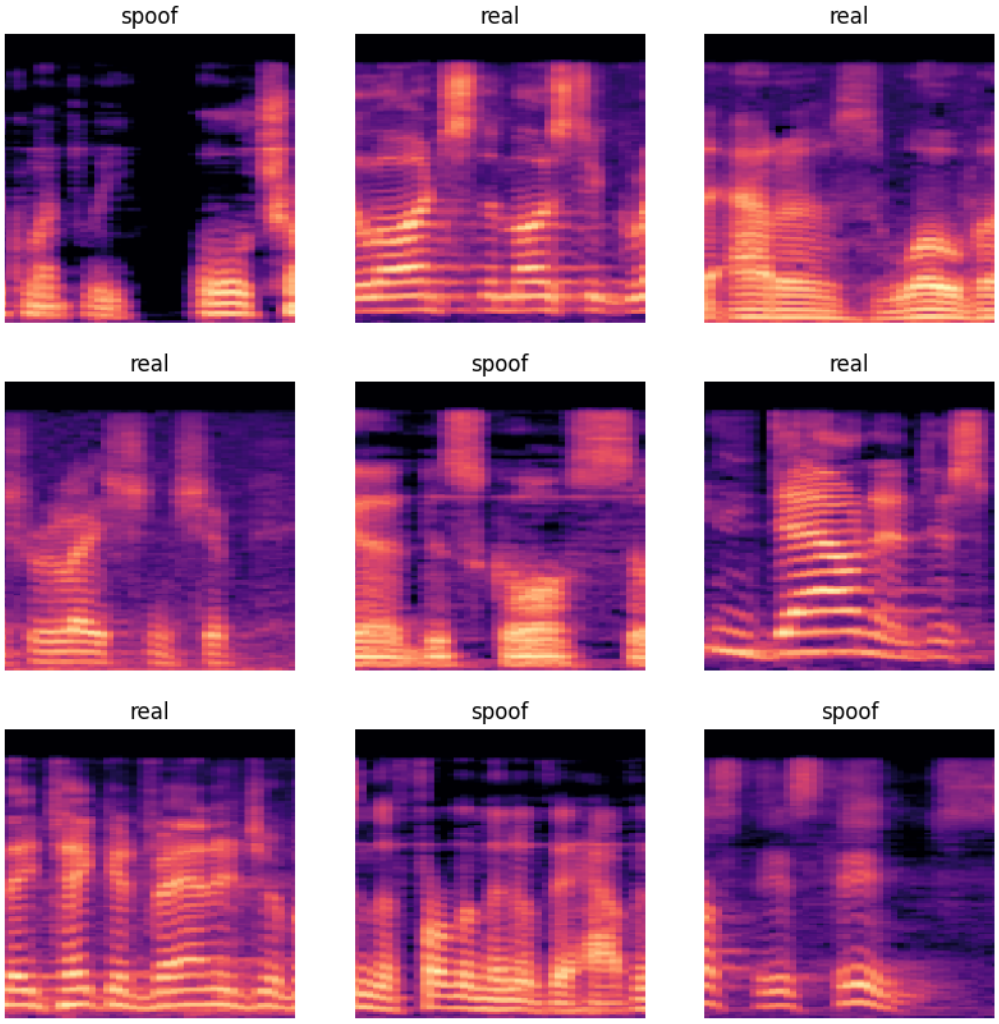


Figure 3.10: A selection of nine randomly chosen spectrograms from the final training dataset.

3.5 Model Design & Training

Now that the data has been loaded and preprocessed, it is ready to be used for training and evaluating models. This part of the implementation phase is shown in red in figure 3.11, and the code is included in appendix A.1.

Listing 3.8 shows the code in which the model is defined. First, a model instance is created using the `keras.Sequential()` function. The Keras Sequential model is a fundamental building block in the Keras library, as it provides a simple way to create neural networks by stacking multiple layers sequentially.

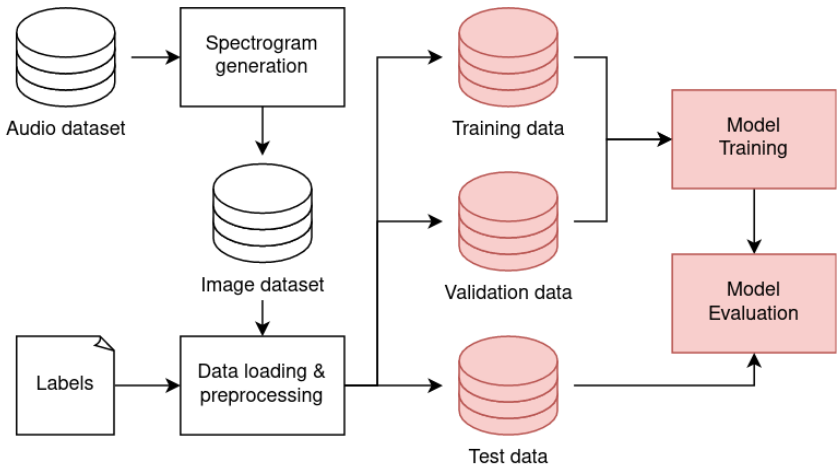


Figure 3.11: Overview of the solution with the part of the program used to train and evaluate models marked in red.

```
model = keras.Sequential()

# Add pretrained ResNet50 model:
model.add(keras.applications.resnet50.ResNet50(
    include_top = False,
    weights = "imagenet",
    pooling = "avg")
)

# Add output layer with sigmoid activation function for binary
    classification:
model.add(keras.layers.Dense(1, activation = "sigmoid"))
model.layers[0].trainable = False
```

Listing 3.8: Defining the model.

Next, the pre-built ResNet50 model is added using the `add()` method of the sequential model. The `weights` parameter specifies whether to use pre-trained weights for transfer learning (by using weights obtained by training on the *imagenet* dataset which consists of millions of natural images) or to train the model from scratch by setting the parameter to `None`.

The `include_top` parameter is set to false, because we do not want to include the fully connected top layer (output layer) of the pre-built model, as the pre-trained model is trained for multiclass classification and uses a softmax activation function in the output layer. Instead, an output layer for binary classification is manually added in the form a fully connected (dense) layer with just one node using a sigmoid activation function. This results in an output between zero and one with zero indicating bona-fide audio and one indicating spoofed audio.

The model is compiled as shown in listing 3.9. The Adam optimiser, which is an extension of the stochastic gradient descent (SGD) algorithm, is chosen due to its fast convergence and ability to work well on noisy and sparse datasets. Furthermore, Adam uses a combination of adaptive learning rates and momentum to make adjustments to the network's parameters during training, which helps the model learn faster and converge more quickly towards the optimal set of parameters that minimise the loss function [34].

```
# Compile model
model.compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = [
        keras.metrics.BinaryAccuracy(),
        keras.metrics.Precision(),
        keras.metrics.Recall(),
        keras.metrics.TruePositives(),
        keras.metrics.FalsePositives(),
        keras.metrics.TrueNegatives(),
        keras.metrics.FalseNegatives()
    ])
```

Listing 3.9: Specifying optimiser, loss and compiling the model.

The loss function chosen is *binary cross-entropy*. It is chosen because it is a widely used loss function in deep learning for binary classification problems. Cross-entropy calculates a score that summarises the average difference between the actual and predicted probability distributions for predicting class 1 (deepfake). The score is minimised, and a perfect cross-entropy value is 0 [35].

Moreover, evaluation metrics are also specified during compilation. The metrics used to evaluate the model, as well as the reasoning behind choosing them, are described in further detail in section 4.1.1. After compiling, a summary of the model is printed using the `model.summary()` method, yielding the output shown in listing 3.10 below:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 dense (Dense)               (None, 1)                 2049
=================================================================
Total params: 23,589,761
Trainable params: 2,049
Non-trainable params: 23,587,712
```

Listing 3.10: Output of the `model.summary()` method.

Finally, the model is trained using the `model.fit()` function, which is shown in listing 3.11. First, the number of epochs is defined, upon which steps per epoch is calculated from the total number of batches in the dataset.

```
epochs = 1
steps_per_epoch = len(training_ds)/epochs

model.fit(
    training_ds,
    steps_per_epoch = steps_per_epoch,
    validation_data = validation_ds,
    validation_steps = 1,
    epochs = epochs
)
```

Listing 3.11: Training the model using the `model.fit()` function.

Hyperparameters such as batch size, steps per epoch and number of epochs can have a great impact on the performance of the model. As such, a number of models are trained, evaluated and compared to each other. This process is described in the following section, and an overview of the results is shown in section 4.2.

3.6 Model Evaluation

After the models have been trained, their performance is evaluated using the test data. A detailed description of the methodology followed and metrics used in the evaluation can be seen in chapter 4, while this section describes only the implementation aspect.

The models are evaluated using the `evaluate()` method of the model instance:

```
results = model.evaluate(
    test_ds,
    return_dict=True
)
```

Listing 3.12: Evaluating performance using the `model.evaluate() function.`

This function makes predictions for the test data and compares them to the labels in the dataset. It returns a dictionary of the metrics specified during model compilation (see listing 3.9). Furthermore, as seen in listing 3.13, a function is defined which calculates the F1-score of the model:

```
def f1score(p,r):
    f1 = 2/((1/p)+(1/r))
    return f1
```

Listing 3.13: Function for calculating F1-score of a model.

Finally, a confusion matrix is plotted to visualise the performance of the model. The code used to generate the confusion matrix is seen in listing 3.14:

```
tp, fp = results['true_positives'], results['false_positives']
fn, tn = results['false_negatives'], results['true_negatives']
cmx = np.array([[tp, fp],[fn, tn]], np.int32)

cmx_plot = sns.heatmap(
    cmx/np.sum(cmx),
    cmap='Blues',
    annot=True,
    fmt=".1%",
    linewidth=5,
    cbar=False,
    square=True,
    xticklabels = ['Spoof (1)', 'Real (0)'],
    yticklabels = ['Spoof (1)', 'Real (0)']
)
cmx_plot.set(xlabel="Actual", ylabel="Predicted")
```

Listing 3.14: Code for generating the confusion matrix.

CHAPTER 4
TEST & EVALUATION

This chapter first describes the methodology followed when testing and evaluating models both with and without utilising transfer learning, as well as the metrics used when evaluating them. The results are then presented and discussed, upon which follows a broader reflection on the solution within the context in which it exists.

4.1 Methodology

In order to determine whether transfer learning can improve performance in detection of deepfake audio, models are trained both with and without using pre-trained weights for the ResNet50 model. Furthermore, different batch sizes and numbers of epochs are tested to observe their influence on the performance of the models. In total, 18 models are trained using all combinations of the following variables:

- **Transfer Learning (TL):** *Yes (imagenet pre-trained weights), no (no pre-trained weights)*
- **Batch size:** *10, 50, 100*
- **Number of epochs:** *1, 5, 10*

The learning rate is not included as a variable factor as it is automatically updated by the Adam optimiser. Furthermore, as the task is a binary classification problem, the activation function is not a variable parameter either – instead the sigmoid activation function is used in the output layer of all models trained.

All models are trained with a training/validation/test ratio of 80:10:10. While the optimal ratio differs by task and circumstances, 80:10:10 is a commonly used ratio [36], and preliminary tests showed little variation in results when using other ratios in the data splitting process. The ratio used results in 3 178 files being used for testing. The models are then evaluated and compared based on a selection of relevant metrics, all of which are described in 4.1.1.

4.1.1 Evaluation Metrics

Loss
Loss is a metric used measure the error between the predicted outputs of a model and the actual values. It quantifies how well the model is performing in terms of its ability to make accurate predictions. The loss function (in this case binary cross-entropy (BCE)) calculates the difference between the predicted values and the ground truth values for the inputs of the dataset. As the model improves during training, the loss decreases, indicating better alignment between the predicted and actual values. While loss is primarily an internal metric used during the training process to optimise the model, it is still included in the results shown in section 4.2.

Accuracy
One of the most simple metrics, accuracy measures the proportion of correctly predicted instances to the total number of instances in a dataset. It provides an overall assessment of the model's correctness by calculating the ratio of correctly classified samples to the total number of samples.

While accuracy is a straightforward metric, it may not always be sufficient for evaluating the performance of a model. In cases where the classes are imbalanced, where the cost of false positives or false negatives is significantly different, or where different types of errors have varying importance, other metrics such as precision, recall or F1-score may provide a more comprehensive assessment of the model's performance.

Precision
Precision is a metric used to measure the accuracy of positive predictions made by a model. It calculates the ratio of true positive predictions (correctly predicted positive instances) to the total number of positive predictions made by the model, as shown in equation 4.1. TP indicates true positives while FP indicates false positives.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.1}$$

Precision focuses on the precision of positive predictions, indicating how well the model correctly identifies positive instances. A higher precision value suggests a lower rate of false positives, meaning that the model has a lower tendency to classify negative instances as positive.

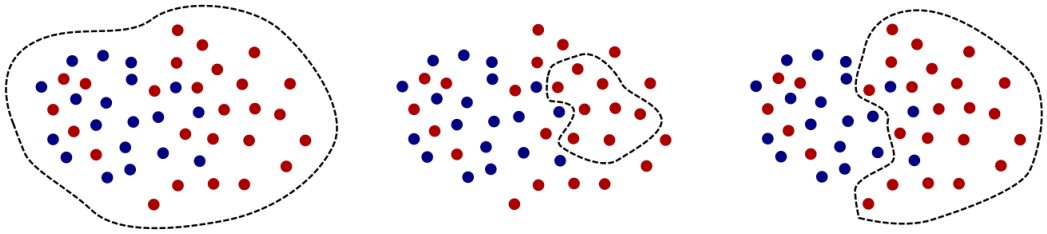
Precision is especially important in scenarios where false positives carry significant consequences or where the goal is to minimise incorrect positive predictions. However, it should be considered along with other metrics such as recall or F1-score to get a more comprehensive evaluation of the model's performance.

Recall

Recall is another metric used to measure the ability of a model to correctly identify positive instances. While precision calculates the ratio of true positives to the total number of positive predictions made by the model, recall calculates the ratio of true positives to the total number of *actual* positive instances. The formula for calculating recall is shown in equation 4.2:

$$\text{Recall} = \frac{TP}{TP+FN} \tag{4.2}$$

In simple terms, precision evaluates how well the model avoids false positives, while recall evaluates how well it avoids false negatives. A high precision indicates that the model has a low rate of false positives, while a high recall indicates that the model can effectively capture most of the positive instances, minimising false negatives. Figure 4.1 visualises precision and recall and the reason why a model's performance cannot be evaluated based on only one or the other. For this reason, the F1-score metric is introduced.



(a) Perfect recall, low precision. **(b)** Perfect precision, low recall. **(c)** Balanced precision and recall.

Figure 4.1: Illustrating precision and recall.

F1-score

The F1 score is a metric that combines precision and recall into a single value. It provides a balanced measure of a model's performance by considering both its ability to accurately identify positive instances (precision) and to capture all positive instances (recall).

The F1 score is calculated as the harmonic mean of precision and recall. It ranges from 0 to 1, with 1 being the best possible score. A higher F1 score indicates a better balance between precision and recall, meaning the model can effectively identify positive instances while also minimising false positives and false negatives.

$$\text{F1-score} = \frac{2}{\frac{1}{\mathit{Precision}} + \frac{1}{\mathit{Recall}}} = \frac{TP}{TP + \frac{1}{2}(FP+FN)} \tag{4.3}$$

The F1 score is particularly useful when there is an imbalance between the number of positive and negative instances in the dataset – as is the case here – as it provides a comprehensive evaluation of the model's performance by considering both types of errors.

4.2 Results

Table 4.1 shows loss, accuracy, precision, recall, F1-score and time to train (TTT) for all 18 models trained and evaluated.

#	**Parameters**			**Results**					
	TL	Batch size	Epochs	Loss	ACC	PRE	REC	F1	TTT(s)
1	Yes	10	1	0.103	0.966	0.986	0.921	0.953	45.18
2			5	0.105	0.965	0.986	0.92	0.952	45.72
3			10	0.099	0.966	0.986	0.922	0.953	45.23
4		50	1	0.112	0.964	0.968	0.933	0.951	36.66
5			5	0.106	0.967	0.960	0.951	0.955	37.38
6			10	0.109	0.966	0.958	0.951	0.955	39.12
7		100	1	0.134	0.957	0.975	0.906	0.940	36.92
8			5	0.150	0.951	0.962	0.905	0.933	37.72
9			10	0.120	0.963	0.964	0.937	0.950	38.82
10	No	10	1	0.426	0.808	0.935	0.521	0.669	45.98
11			5	0.381	0.829	0.901	0.607	0.726	45.34
12			10	0.430	0.817	0.685	0.940	0.793	45.64
13		50	1	0.471	0.772	0.797	0.520	0.630	38.91
14			5	0.448	0.779	0.838	0.505	0.630	39.14
15			10	0.434	0.789	0.850	0.525	0.649	39.68
16		100	1	0.514	0.720	0.883	0.287	0.433	34.89
17			5	0.471	0.803	0.775	0.664	0.715	37.63
18			10	0.490	0.789	0.772	0.615	0.685	38.91

Table 4.1: Results of all models trained, showing whether transfer learning/pre-trained weights were used (TL), batch size, epochs, loss, accuracy (ACC), precision (PRE), recall (REC), F1-score and time to train (TTT) in seconds.

The best performing model trained was model #5 which achieved both the highest accuracy (0.967) and F1-score (0.955) and a decently low loss (0.106). This means that it only misclassified 3.3% of the test data instances and that both precision and recall values are high – and that they are close to each other.

We observe that the test performance for all models trained using transfer learning is significantly better than for those trained using randomly initialised weights. To visualise this, figure 4.2 shows a bar graph of the F1-scores for all models.

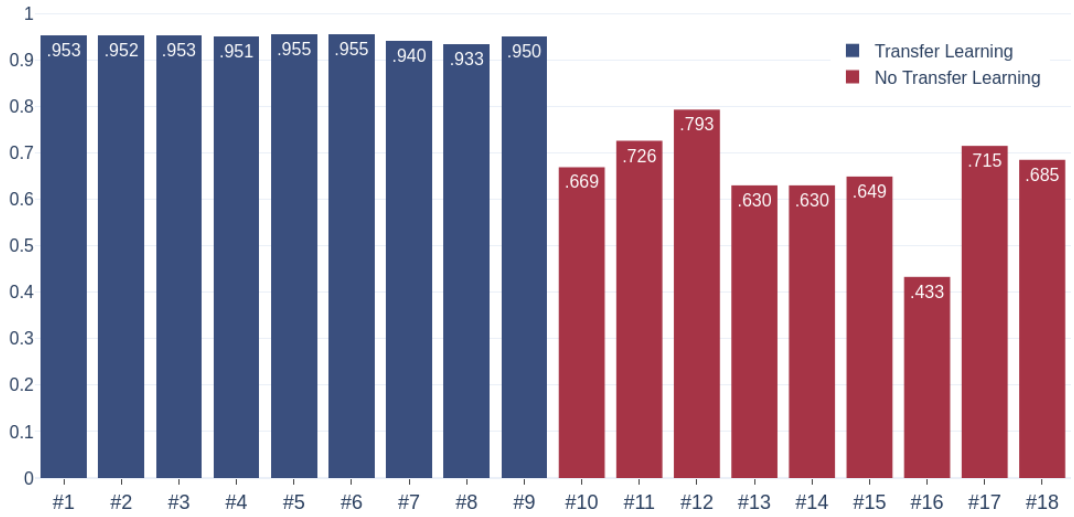


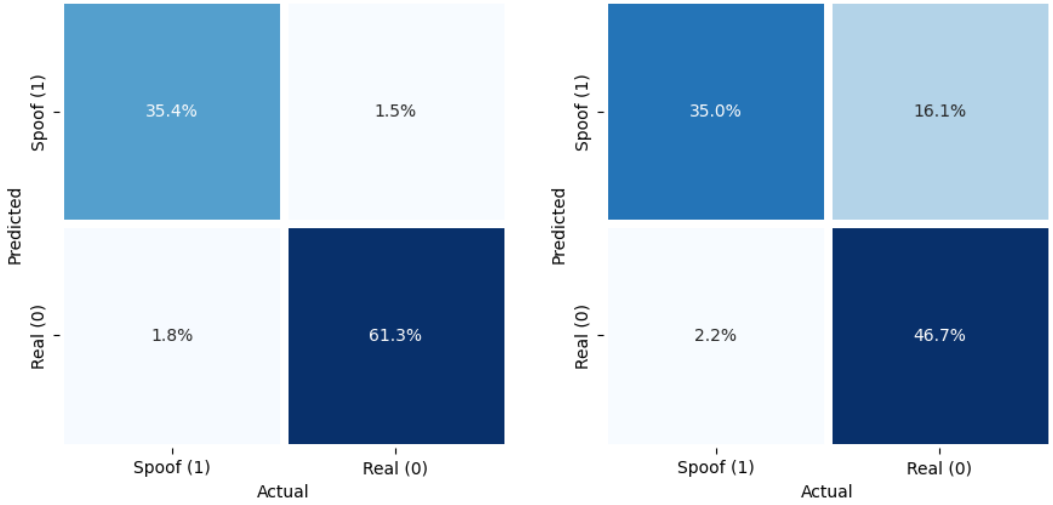
Figure 4.2: F1-score (y-axis) for all models trained (y-axis). The models trained using transfer learning are shown in blue while the models not using transfer learning are shown in red.

While the best performing of the non-transfer learning models (model #12) achieved an accuracy of 0.817 and an F1-score of 0.793, the best performing of the transfer learning models (model #5) achieved an accuracy of 0.976 and an F1-score of 0.955, corresponding to a 19.46% increase in accuracy and a 20.43% increase in F1-score when using transfer learning. Figure 4.3 shows the confusion matrices for models #5 and #12.

Interestingly, both models shown in figure 4.3 have low false negative rates of 1.8% and 2.2%, respectively, in identifying deepfake audio. Conversely, model #12 performs worse in terms of false positives with a false positive rate of 16.1% compared to 1.5% for model #5, significantly lowering its F1-score. Models using transfer learning also achieved a lower loss with an average of 0.115 across models compared to 0.452 for non-transfer learning.

When looking at averages for all models trained, the average accuracy and F1-score across transfer learning models (#1-9) were 0.963 and 0.949, respectively, while the average values across non-transfer learning models (#10-18) were 0.790 for

accuracy and 0.659 for F1-score. This corresponds to a 21.90% increase in accuracy and a 44.01% increase in F1-score when using transfer learning.



(a) Confusion matrix for the best performing transfer learning model (model #5).

(b) Confusion matrix for the best performing transfer learning model (model #12).

Figure 4.3: Confusion matrices for the best performing models trained with and without the use of transfer learning, respectively.

4.3 Discussion

It is clear from the results of the evaluation of the 18 models that transfer learning yields positive results when used in conjunction with residual neural networks for audio deepfake detection. There are however some things which should be taken into consideration and could be explored further in future work.

First, audio files from the dataset were randomly cropped during dataset generation to fit the input size required by the model. This means that only one second of each file was used, resulting in only 43% of the total amount of data in the dataset being utilised. This could be addressed by changing the preprocessing steps to use multiple crops of each file, which would provide more data for use in the training process.

Second, more different combinations of hyperparameters could be tested to observe their impact on the performance of the models. While this aspect was not a primary focus area of the project, the results show that the choice of hyperparameters had an impact on model performance in terms of both F1-score, accuracy and loss.

Furthermore, only one dataset was used, meaning that the models' abilities to generalise cannot be confidently determined based on the results achieved in this project. While the dataset does comprise data from various sources and was developed for use in generalisation evaluation, the model should be evaluated on data from other sources in order to determine how well it generalises. Future work could also include the use of data from various different datasets in the training phase in order to develop a more robust model and ensure generalisation abilities.

Another potential drawback to the dataset used is that it – while containing data from various sources – only contains speech from 58 different people. It could be investigated whether this impacts the models' abilities to generalise and whether a more diverse dataset could be used to improve generalisation.

The amount and type of preprocessing carried out on the dataset is another aspect which could be further investigated in future work. For spectrogram generation, window length and hop length could be varied to study their impact on model performance. In addition, further preprocessing such as volume normalisation or noise reduction could be implemented.

Despite these considerations, the results prove that an effective solution for audio deepfake detection can be implemented using residual neural networks and transfer learning.

4.4 Final Thoughts

At this point, it has been demonstrated that the combination of residual neural networks and transfer learning is a viable method of detecting deepfake audio – albeit not necessarily the only one. While the scope if this project has been deep learning based methods for securing audio authenticity, completely different approaches could potentially be just as effective.

One interesting approach which has not been investigated in this project is digital watermarking – a process which consists of embedding a unique, imperceptible digital watermark in audio files during the recording or production process which can serve as a form of authentication. While this would not be able to prevent all kinds of harmful use of deepfakes, it provides a different approach that does not rely on deep learning knowledge and the availability of large datasets.

During the writing of this thesis, Apple Inc. filed a patent describing a system which would let users of their products train a personal voice model, which can read aloud messages that they send to their contacts [37]. While the idea might seem innova-

tive – and to some, perhaps even beautiful – it would effectively put several deepfake generation models in the pocket of every single iPhone user on Earth. Synthesised audio is becoming an increasingly big part of our lives in the form of audio books, voice assistants and entertainment media, and so should solutions which can distinguish synthesised audio from real.

A perfect solution for deepfake detection – not just in the audio domain, but in the broader sense – could pave the way for a safer and more trusting world. We would be able to browse the internet without having to doubt whether a video of a celebrity or politician could be trusted to be real, and we would remove a weapon from the arsenal of tools which can be used to influence political agendas in undemocratic ways. That is, however, outside of the scope of this project – although I believe that the work presented here represents a small step in the right direction.

CHAPTER 5
CONCLUSION

This project has investigated the use of machine learning for audio deepfake detection. By examining existing research in the field, it was found that residual neural networks and transfer learning both are techniques which have proved effective within this domain.

A novel solution for audio deepfake detection was then proposed. The solution consists of converting audio to spectrograms and feeding it to an implementation of the ResNet50 network architecture which has been pre-trained on a large dataset of natural images and further trained for audio classification using a dataset comprising both synthesised and real audio.

The best performing model achieved an accuracy of 96.7% and an F1-score of 95.5%, proving that the combination of residual neural networks and transfer learning is an effective approach to audio deepfake detection.

Furthermore, a similar model was trained without the use of transfer learning in order to study the difference. It was found that, across all models trained, the use of transfer learning increased the prediction accuracy of models by an average of 21.90% and the F1-score by an average of 44.01%, leading to the conclusion that transfer learning can improve performance in audio deepfake detection tasks.

BIBLIOGRAPHY

[1] A. Diaz, "TikTok AI voice generator turns US presidents into Discord goblins - Polygon," [Online]. Available: https://www.polygon.com/23610381/presidents-play-minecraft-ai-voice-meme-joe-biden-trump, *Polygon*.

[2] Z. Khanjani, G. Watson, and V. P. Janeja, "Audio deepfakes: A survey," 2023. DOI: 10.3389/fdata.2022.1001063.

[3] C. Stupp, "Fraudsters Used AI to Mimic CEO's Voice in Unusual Cybercrime Case - WSJ," [Online]. Available: https://www.wsj.com/articles/fraudsters-use-ai-to-mimic-ceos-voice-in-unusual-cybercrime-case-11567157402, *Wall Street Journal*.

[4] VMware, "VMware Global Incident Response Threat Report 2022," 2022.

[5] M. Berman, "An act to amend, repeal, and add Section 35 of the Code of Civil Procedure, and to amend, add, and repeal Section 20010 of the Elections Code, relating to elections." [Online]. Available: https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201920200AB730.

[6] N. Schaumann, "Det lyder som din datter, der ringer efter penge. I virkeligheden er det en helt anden," [Online]. Available: https://www.berlingske.dk/internationalt/det-lyder-som-din-datter-der-ringer-efter-penge-i-virkeligheden-er, *Berlingske*.

[7] A. Puig, "Scammers use AI to enhance their family emergency schemes | Consumer Advice," [Online]. Available: https://consumer.ftc.gov/consumer-alerts/2023/03/scammers-use-ai-enhance-their-family-emergency-schemes?utm_source=govdelivery, *FTC Consumer Advice*.

[8] F. Tom, M. Jain, and P. Dey, "End-To-End Audio Replay Attack Detection Using Deep Convolutional Networks with Attention," *Interspeech 2018*, 2018. DOI: 10.21437/Interspeech.2018-2279.

[9] Z. Almutairi and H. Elgibreen, "A Review of Modern Audio Deepfake Detection Methods: Challenges and Future Directions," *Algorithms 2022, Vol. 15, Page 155*, vol. 15, no. 5, p. 155, May 2022, ISSN: 1999-4893. DOI: 10.3390/A15050155. [Online]. Available: https://www.mdpi.com/1999-4893/15/5/155/htmhttps://www.mdpi.com/1999-4893/15/5/155.

[10] Y. Rodríguez-Ortega, D. M. Ballesteros, and D. Renza, "A Machine Learning Model to Detect Fake Voice," *Communications in Computer and Information Science*, vol. 1277 CCIS, pp. 3–13, 2020, ISSN: 18650937. DOI: 10.1007/978-

3-030-61702-8{_}1/COVER. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-61702-8_1.

[11] D. M. Ballesteros L, Y. P. Rodriguez, and D. Renza, "H-Voice: Fake voice histograms (Imitation+DeepVoice)," *Mendeley Data*, 2020. DOI: 10.17632/K47YD3M28W.4.

[12] A. K. Singh and P. Singh, "Detection of AI-Synthesized Speech Using Cepstral & Bispectral Statistics," *Proceedings - 4th International Conference on Multimedia Information Processing and Retrieval, MIPR 2021*, pp. 412–417, 2021. DOI: 10.1109/MIPR51284.2021.00076.

[13] T. Liu, D. Yan, R. Wang, N. Yan, and G. Chen, "Identification of Fake Stereo Audio Using SVM and CNN," *Information 2021, Vol. 12, Page 263*, vol. 12, no. 7, p. 263, Jun. 2021, ISSN: 2078-2489. DOI: 10.3390/INFO12070263. [Online]. Available: https://www.mdpi.com/2078-2489/12/7/263/htmhttps://www.mdpi.com/2078-2489/12/7/263.

[14] D. M. Ballesteros, Y. Rodriguez-Ortega, D. Renza, and G. Arce, "Deep4SNet: deep learning for fake speech classification," *Expert Systems with Applications*, vol. 184, p. 115 465, Dec. 2021, ISSN: 0957-4174. DOI: 10.1016/J.ESWA.2021.115465.

[15] M. Lataifeh, A. Elnagar, I. Shahin, and A. B. Nassif, "Arabic audio clips: Identification and discrimination of authentic Cantillations from imitations," *Neurocomputing*, vol. 418, pp. 162–177, Dec. 2020, ISSN: 0925-2312. DOI: 10.1016/J.NEUCOM.2020.07.099.

[16] A. Chintha, B. Thai, S. J. Sohrawardi, *et al.*, "Recurrent Convolutional Structures for Audio Spoof and Video Deepfake Detection," *IEEE Journal on Selected Topics in Signal Processing*, vol. 14, no. 5, pp. 1024–1037, Aug. 2020, ISSN: 19410484. DOI: 10.1109/JSTSP.2020.2999185.

[17] J. Yamagishi, M. Todisco, M. Sahidullah, *et al.*, "ASVspoof 2019: The 3rd Automatic Speaker Verification Spoofing and Countermeasures Challenge database," [Online]. Available: https://datashare.ed.ac.uk/handle/10283/3336.

[18] R. L. Wijethunga, D. M. Matheesha, A. A. Noman, K. H. De Silva, M. Tissera, and L. Rupasinghe, "Deepfake audio detection: A deep learning based solution for group conversations," *ICAC 2020 - 2nd International Conference on Advancements in Computing, Proceedings*, pp. 192–197, Dec. 2020. DOI: 10.1109/ICAC51239.2020.9357161.

[19] N. Subramani and D. Rao, "Learning Efficient Representations for Fake Speech Detection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 5859–5866, Apr. 2020, ISSN: 2374-3468. DOI: 10.1609/AAAI.V34I04.6044. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6044.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December, pp. 770–778, Dec. 2016, ISSN: 10636919. DOI: 10.1109/CVPR.2016.90.

[21] ImageNet, "ImageNet," [Online]. Available: https://www.image-net.org/index.php, *Last accessed: 29/05/2023*.

[22] "LunarLullaby", "A building block in a deep residual network," [Online]. Available: https://commons.wikimedia.org/wiki/File:ResBlock.png.

[23] S. Hershey, S. Chaudhuri, D. P. W. Ellis, *et al.*, "CNN architectures for large-scale audio classification," 2016.

[24] R. T. P, P. R. Aravind, R. C, U. Nechiyil, and N. Paramparambath, "Audio Spoofing Verification using Deep Convolutional Neural Networks by Transfer Learning," Aug. 2020. [Online]. Available: https://arxiv.org/abs/2008.03464v1.

[25] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "Transfer learning for music classification and regression tasks," Tech. Rep., 2017. [Online]. Available: https://github.com/keunwoochoi/transfer_.

[26] S. Suratkar and F. Kazi, "Deep Fake Video Detection Using Transfer Learning Approach," *Arabian Journal for Science and Engineering*, pp. 1–11, Oct. 2022, ISSN: 21914281. DOI: 10.1007/S13369-022-07321-3/FIGURES/8. [Online]. Available: https://link.springer.com/article/10.1007/s13369-022-07321-3.

[27] APTLY and LaSSoftE, "Datasets – APTLY and LaSSoftE," [Online]. Available: https://bil.eecs.yorku.ca/datasets/, *Last accessed: 30/05/2023*.

[28] J. Frank and L. Schönherr, "WaveFake: A Data Set to Facilitate Audio Deepfake Detection," Nov. 2021. [Online]. Available: https://arxiv.org/abs/2111.02813v1.

[29] Fraunhofer AISEC, "In-the-Wild Audio Deepfake Dataset," [Online]. Available: https://deepfake-demo.aisec.fraunhofer.de/in_the_wild, *Last accessed: 27/04/2023*.

[30] N. M. Müller, P. Czempin, F. Dieckmann, A. Froghyar, and K. Böttinger, "Does Audio Deepfake Detection Generalize?" *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2022-September, pp. 2783–2787, Mar. 2022, ISSN: 19909772. DOI: 10.48550/arxiv.2203.16263. [Online]. Available: https://arxiv.org/abs/2203.16263v3.

[31] Keras Team, "Keras Applications," [Online]. Available: https://keras.io/api/applications/, *Last accessed: 29/05/2023*.

[32] H. Jeon, Y. Jung, S. Lee, and Y. Jung, "Area-efficient short-time fourier transform processor for time–frequency analysis of non-stationary signals," *Applied Sciences*, vol. 10, no. 20, pp. 1–10, Oct. 2020, ISSN: 20763417. DOI: 10.3390/APP10207208. [Online]. Available: https://www.researchgate.net/publication/346243843_Area-Efficient_Short-Time_Fourier_Transform_Processor_for_Time-Frequency_Analysis_of_Non-Stationary_Signals.

[33] S. S. Stevens, J. Volkmann, and E. B. Newman, "A Scale for the Measurement of the Psychological Magnitude Pitch," *Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937, ISSN: NA. DOI: 10.1121/1.1915893.

[34] S. Mahendra, "What is the Adam Optimizer and How is It Used in Machine Learning - Artificial Intelligence +," [Online]. Available: https://www.aiplusinfo.com/blog/what-is-the-adam-optimizer-and-how-is-it-used-in-machine-learning/, *Last accessed: 26/05/2023*.

[35] J. Brownlee, "A Gentle Introduction to Cross-Entropy for Machine Learning," [Online]. Available: https://machinelearningmastery.com/cross-entropy-for-machine-learning/, *Last accessed: 26/05/2023*.

[36] V. R. Joseph, "Optimal ratio for data splitting," *Statistical Analysis and Data Mining*, vol. 15, no. 4, pp. 531–538, Aug. 2022, ISSN: 19321872. DOI: 10.1002/SAM.11583.

[37] A. Orr, "Apple working on how to read back iMessages in the sender's voice | AppleInsider," [Online]. Available: https://appleinsider.com/articles/23/02/16/apple-working-on-how-to-read-back-imessages-in-the-senders-voice, *Last accessed: 31/05/2023*.

Appendix A

Code

A.1 Program used to train and evaluate models

```
# Import dependencies
# ------------------------------------------------------------------

import numpy as np
import os
import PIL
import PIL.Image
import pathlib
import matplotlib.pyplot as plt
import seaborn as sns
import time

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from dataSplit import dataSplit

# Load and split data
# ------------------------------------------------------------------

inputData = "/tf/spectrograms/1_NOISY_N2048_H512"
labels = "/tf/labels.csv"
testSplit = 0.1

dataSplit(inputData, labels, testSplit)

training_dir = pathlib.Path("/tf/spectrograms/workdir/training").
    with_suffix('')
test_dir = pathlib.Path("/tf/spectrograms/workdir/test").with_suffix('
    ')

test_count = len(list(test_dir.glob('*/*.png')))
train_count = len(list(training_dir.glob('*/*.png')))
validation_split = test_count / train_count


```

```
# Define parameters and create datasets
# ---------------------------------------------------------------------

batch_size = 100
epochs = 10
img_height = 224
img_width = 224

print("\033[1mCreating training and validation datasets:\033[0m")
training_ds, validation_ds = tf.keras.utils.
    image_dataset_from_directory(
    training_dir,
    validation_split=validation_split,
    subset="both",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    crop_to_aspect_ratio=True
)

print("\n\033[1mCreating test dataset:\033[0m")
test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    crop_to_aspect_ratio=True
)

class_names = training_ds.class_names
print("\nNames of",str(len(class_names)), "classes:", class_names)

# Build and compile model
# ---------------------------------------------------------------------

# Instantiate sequential model type:
model = keras.Sequential()

# Add pretrained ResNet50 model:
model.add(keras.applications.resnet50.ResNet50(
    include_top = False,
    weights = None,
    pooling = "avg")
)

# Add output layer with sigmoid act. function for binary clasf.:
model.add(keras.layers.Dense(1, activation = "sigmoid"))
model.layers[0].trainable = False

# Compile model
model.compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
```

```
91      metrics = [
92          keras.metrics.BinaryAccuracy(),
93          keras.metrics.Precision(),
94          keras.metrics.Recall(),
95          keras.metrics.TruePositives(),
96          keras.metrics.FalsePositives(),
97          keras.metrics.TrueNegatives(),
98          keras.metrics.FalseNegatives(),
99      ])
100
101 model.summary()
102
103 # Model training
104 # ----------------------------------------------------------------------
105
106 steps_per_epoch = len(training_ds)/epochs
107
108 t0 = time.time()
109
110 model.fit(
111     training_ds,
112     steps_per_epoch = steps_per_epoch,
113     validation_data = validation_ds,
114     validation_steps = 1,
115     epochs = epochs
116 )
117
118 t1 = time.time()
119 dt = (t1 - t0)
120
121 # Model evaluation
122 # ----------------------------------------------------------------------
123
124 results = model.evaluate(
125     test_ds,
126     return_dict=True
127 )
128
129 def f1score(p,r):
130     f1 = 2/((1/p)+(1/r))
131     return f1
132
133 print("----------------------------------------")
134 print('\033[1m'+"Model metrics:"+'\033[0m')
135 for i in results:
136     print(i + ": " + str(results[i]))
137 print("----------------------------------------")
138 print("F1 Score: " + str(f1score(results['precision'],results['recall'
    ])))
139 print("Time to train:", dt)
140 print("----------------------------------------")
141
142 tp, fp = results['true_positives'], results['false_positives']
```

```
fn, tn = results['false_negatives'], results['true_negatives']
cmx = np.array([[tp, fp],[fn, tn]], np.int32)

cmx_plot = sns.heatmap(
    cmx/np.sum(cmx),
    cmap='Blues',
    annot=True,
    fmt=".1%",
    linewidth=5,
    cbar=False,
    square=True,
    xticklabels = ['Spoof (1)', 'Real (0)'],
    yticklabels = ['Spoof (1)', 'Real (0)']
)
cmx_plot.set(xlabel="Actual", ylabel="Predicted")
```

A.2 DataSplit function used for data sorting

```
import os.path
import os
import csv
import sys
import shutil
import glob

def dataSplit(datadir, labelfile, test_share):

    # Clear workdir folder of existing files
    oldfiles = glob.glob('/tf/spectrograms/workdir/*/*/*')
    for f in oldfiles:
        os.remove(f)
    print("Cleared working directory, loading new files...")

    # Define paths to sort to
    p_test_real = "/tf/spectrograms/workdir/test/real"
    p_test_spoof = "/tf/spectrograms/workdir/test/spoof"
    p_train_real = "/tf/spectrograms/workdir/training/real"
    p_train_spoof = "/tf/spectrograms/workdir/training/spoof"

    # Get number of files from labels csv file
    csv_file = open(labelfile)
    filecount = len(csv_file.readlines())
    csv_file.close()

    # Define numbers of files to be used for test and training
    test_num = int(test_share*filecount)
    train_num = int(filecount - test_num)

    print("Found", filecount, "files in", datadir)
    print(
```

```
        "Reserving",
        test_num,
        "files for testing and using",
        train_num,
        "files for training and validation."
    )

    # Read from labels file and copy files to appropriate folders

    csv_file = open(labelfile)
    csv_reader = csv.reader(csv_file, delimiter=',')

    for row in csv_reader:

        n = row[0].partition(".")[0]
        filename = "/"+ str(n) + ".png"

        if(int(n) <= test_num):
            if(row[2] == "spoof"):
                shutil.copy((datadir + filename), p_test_spoof +
    filename)
            elif(row[2] == "bona-fide"):
                shutil.copy((datadir + filename), p_test_real +
    filename)

        else:
            if(row[2] == "spoof"):
                shutil.copy((datadir + filename), p_train_spoof +
    filename)
            elif(row[2] == "bona-fide"):
                shutil.copy((datadir + filename), p_train_real +
    filename)

    csv_file.close()

    print("Succesfully loaded", filecount, "files.")
```

A.3 Program used for spectrogram generation

```
import librosa
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import os.path
import os
import sys

%matplotlib inline
matplotlib.use('Agg')

```

```
samplecount = 31779
n_fft = 2048
hop_length = 512
factor = 1

outpath = "/tf/spectrograms/1_NOISY_N2048_H512"

if os.path.isdir(outpath):
    sys.exit("Path already exists, exiting")
else:
    os.mkdir(outpath)

for i in range(samplecount):

    percentcomplete = (i/(samplecount-1))*100

    print("Generating spectrogram " + str(i) + "/" + str(samplecount
    -1) + " (" + str(int(percentcomplete)) + "% done)", end="\r")

    input_file = "/tf/release_in_the_wild/" + str(i) + ".wav"
    output_file = outpath + "/" + str(i) + ".png"

    if not(os.path.isfile(output_file)):

        signal, sr = librosa.load(input_file)
        audiolength = librosa.get_duration(
            y=signal,
            sr=sr,
            hop_length=hop_length,
            n_fft=n_fft
        )

        mel_signal = librosa.feature.melspectrogram(
            y=signal,
            sr=sr,
            hop_length=hop_length,
            n_fft=n_fft
        )

        spectrogram = np.abs(mel_signal)
        power_to_db = librosa.power_to_db(spectrogram, ref=np.max)

        # Plotting the mel spectrogram
        fig = plt.figure()
        fig.set_size_inches(factor*audiolength, 1)
        ax = plt.Axes(fig, [0., 0., 1., 1.])
        ax.set_axis_off()
        fig.add_axes(ax)

        # Display and save the final spectrogram
        librosa.display.specshow(
            power_to_db,
            sr=sr,
```

```
64              hop_length=hop_length,
65              cmap="magma"
66          )
67
68          plt.savefig(
69              output_file,
70              dpi=224,
71              bbox_inches="tight",
72              pad_inches=0
73          )
74          plt.close()
```