DeepFake-Voice Detection

Signal and Natural Language Processing with Deep Learning

Melanie Ullrich

12.12.2024

Contents

Abstract

This report presents an approach to detecting Deepfake voices in audio recordings using a dataset that was not specifically prepared for this purpose. The methodology focuses on data preparation and feature extraction, with mel spectrograms selected as the primary features, while briefly comparing them to Mel Frequency Cepstral Coefficients (MFCCs). A Convolutional Neural Network (CNN) was trained on the processed data, tackling challenges such as dataset imbalance and overfitting through various strategies. The results of the training are evaluated and visualized, highlighting the model's performance and limitations. By applying this methodology, the project explores the potential of CNNs in analyzing unprepared speech signals and opens the way for further research in this area.

1 Introduction

In a time of rapid technological progress, access to advanced algorithms and artificial intelligence is within everyone's reach. In today's connected world, the ability to tell the difference between deepfakes and real recordings is essential. Especially in times when elections are being manipulated, when wars are going on and when governments are failing. For this reason, this project attempts to detect Deepfake voice in recordings using the methods presented in the university course and the findings of recent papers introduced in the following chapters. This report describes the methodology and implementation of the project and the concrete results that were achieved.

The report begins with the background and motivation for this project. The main part includes the description of the dataset and methodology, which contains the data preprocessing and feature extraction. This is followed by the implementation, which presents code steps and challenges within the model training. Afterwards the results will be discussed. The final section concludes with a brief summary, recommendations and further work.

1.1 Background and Motivation

The idea for this project was driven by the need to detect Deepfakes, as current text-to-speech algorithms are capable of producing realistic fakes of human voices. Because of the easy access, there is a risk of misuse, such as cloning politicians' voices, which can lead to manipulation of society, like spreading misinformation.[1] An example of this was seen in the recent war between Russia and Ukraine, where there was a deeply fake video of President Zelenskyy telling soldiers to lay down their arms and surrender. [2]
The aim of the project is therefore to classify recordings into spoof and real voices. The idea for this project is based on the research paper from Müller et al. (2024) [1]. This research approach allows a deeper understanding of the performance of different models due to deepfake voice classification and attempts to understand and compare the main factors and drivers for these classification models. The main reason why this paper has been chosen is, that it points to a problem that is important to address. The majority of the latest models are based on the VCTK data set, which is recorded in a studio environment using professional speakers. The researchers have concluded that the dataset is not an optimal foundation for the detection of authentic audio recordings from social media sources. Therefore, they created a dataset for evaluation purposes, comprising examples drawn from the wild, for models trained on the VCTK dataset.[1]
Since real-world audio recordings are not produced in a studio environment, full of background noise, and even the best-trained models perform poorly on the 'in-the-wild' dataset, the objective of this project is to train a model on the dataset created by the researchers to gain insight into its potential effectiveness. The interest is to see how the model behaves when trained on this input, and to infer some necessary preprocessing pipeline steps that need to be considered when performing a classification task on real-world data.

The following chapter provides an overview of the current approaches to detecting Deepfake voices, offering insights into the latest developments in this field.

1.2 Current state of the art

The latest developments in the field of audio Deepfakes have resulted in the introduction of a range of state-of-the-art models. For example there are many long short-term memory(LSTM)-based models, which employ recurrent layers to identify time-based relationships in the audio. It is common for established models to be hybrid models, which combine the strengths of the different models. For example CRNNSpoof (1D convolutions combined with RNNs), are effective for capturing both local patterns and long-term dependencies. Furthermore, there are combinations of a CNN with attention pooling and bidirectional LSTM or transformer units. These rely on self-attention and positional encoding to effectively model global dependencies and temporal relationships, which are crucial for detecting inconsistencies in deepfake voices.[1]
In short, the most recent techniques combine innovative deep learning with domain-specific expertise in audio signals, thereby addressing the distinctive challenges of detecting Deepfake voices.

2 Material and Methods

This chapter describes the methodology and main implementation aspects of each project step. It also provides an overview of the materials and applications used in the project. The implementation steps that follow below are carried out using the python programming language including standard python libraries. Further libraries will be mentioned in the specific chapter.

2.1 Dataset

The dataset, entitled 'in-the-wild' [1], which was created by the Fraunhofer AISEC Institut, has been utilized for the current Deepfake Voice Detection, which will be described in more detail in the following. The collection comprises audio deepfakes and corresponding benign audio for a set of 58 politicians and other public figures, sourced from publicly available digital platforms such as social networks and video streaming services. The .wav-files includes in total 20.8 hours of bona-fide and 17.2 hours of spoofed audio.[1] Additionally the dataset contains a comma-separated values(csv)-file with the metadata of the audio files. Those metadata shows the corresponding speaker and the label to each audio filename.
Following this brief insight into the dataset, the subsequent section presents the methodology and implementation of the project.

2.2 Methodology and Implementation

2.2.1 Data preparation

Before delving into the project, it is crucial to thoroughly understand the dataset being used. This involves analyzing the data structure and distribution to identify any challenges that could affect the models performance. In order to gain insights into the dataset, it was first necessary to visualize the distribution of the labels 'spoof' and 'bona fide'. This step highlights the representation of each class and helps identify potential imbalances in the dataset. As seen in the following Figure 1 it became evident that the dataset suffers from an imbalance. Specifically, the bona-fide class is significantly overrepresented compared to the spoof class. This imbalance presents a risk of falling into the accuracy paradox, where a model may achieve seemingly excellent accuracy like 90% by predominantly predicting the majority class without learning to distinguish between the two classes. Such a model would fail to generalize and perform effectively on unseen data.[3]

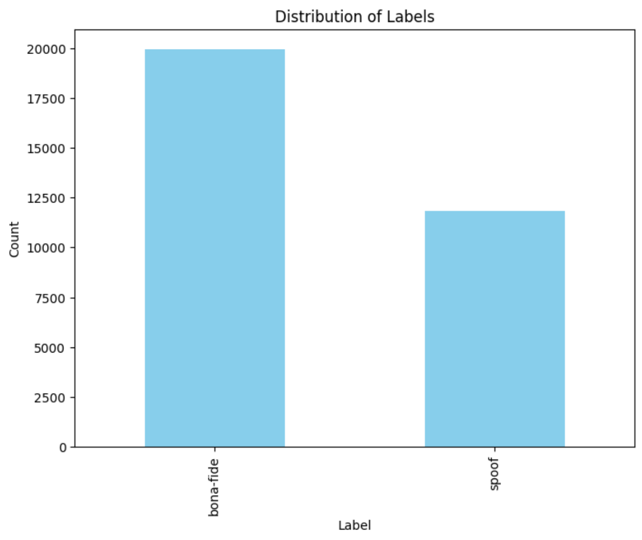


Figure 1: Distribution of the full dataset

There are several techniques available to address the imbalance issue during data preparation. The initial method chosen for this project involved the resampling process in order to create a balanced dataset. In particular, the method of under-sampling was employed, which involves the removal of instances from the overrepresented class i.e. the bona fide class. This approach is particularly suitable for this project due to the amount of data, which contains more than 30.000 data. Furthermore over-sampling, which involves duplicating instances from the underrepresented class (spoof), could lead to RAM limitations during training. Under-sampling avoids this issue by reducing the dataset size rather than increasing it.[3]
Following preparation, the balanced dataset comprises a total of 23,632 data points. The distribution can be seen in the following Figure 2.

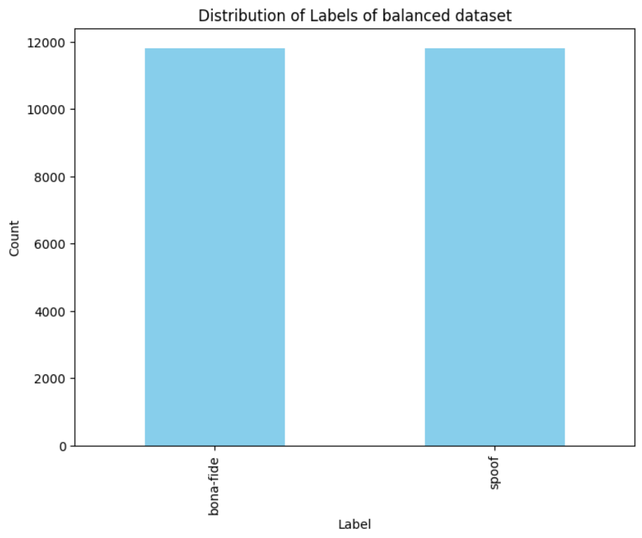


Figure 2: Distribution of the balanced dataset

Once the dataset has been prepared, the subsequent chapter will provide an explanation of the feature extraction process for model training.

2.2.2 Feature Extraction

This chapter introduces the feature extraction process, which is a crucial step in preprocessing the data for model training. For this project, the use of mel spectrograms as features for the training of a convolutional neural network (CNN) was chosen. Although the state-of-the-art models are more complex with combinations between model architectures, the start in this project is to use a simple CNN. These types of models represent the optimal architecture for image processing.[2]
Accordingly, the mel spectrograms, which were introduced in the university class, were selected for use in this project. Mel spectrograms provide a powerful way to represent audio data. They capture the frequency content of an audio signal over time, but on a different frequency axis, while the color intensity represents the amplitude of a frequency at a certain point in time.[4] With this process it is possible to convert the information from audio files into images. To extract the mel spectrograms the python library *'librosa'*, which offers tools for processing audio data, was used. Initially, it was used for loading and reading of the wav-files. This stage involves the conversion of audio data into a format that can be processed automatically. Furthermore, the library's functions were employed to generate mel spectrograms from these audios, effectively transforming raw audio waveforms into a visual representation of the signal's frequency content over time. The *'librosa.power-to-db'*-function was applied to convert the spectrograms' power scale into a logarithmic decibel scale.

Figure 3 illustrates an example of two mel spectrograms from audio files of the speaker Donald Trump. One is labeled 'bona-fide' and the other is labeled 'spoof'.

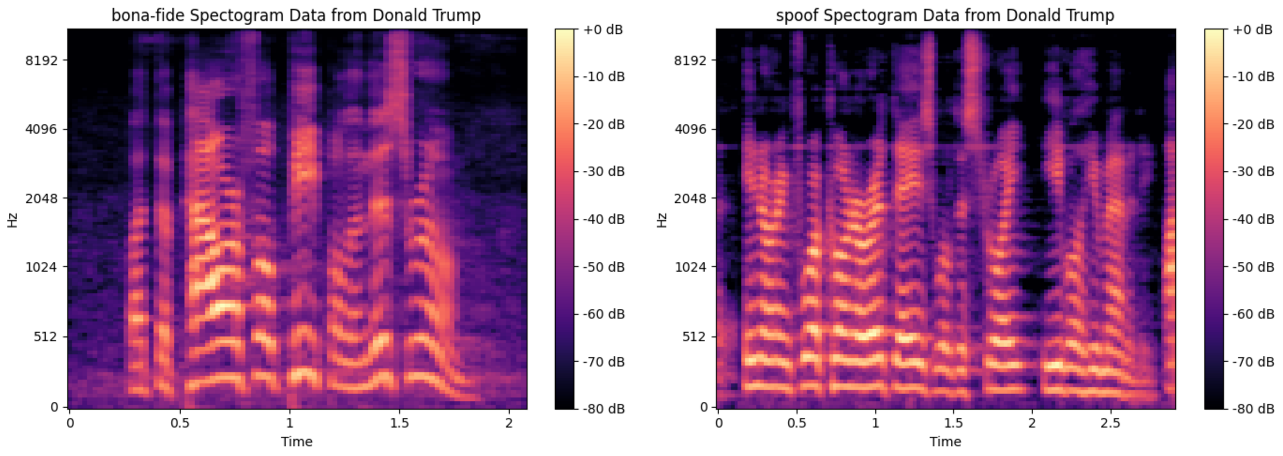


Figure 3: Mel spectograms from audio files of Donald Trump with labels bona-fide and spoof

A significant challenge encountered during feature extraction was, that the audio files varied in length. To address this, at first the *'duration'*-parameter in the loading-function of *'librosa'* was set to a fixed duration of 4 seconds for each audio clip. If an audio file was longer than these 4 seconds, it was truncated. This variability still resulted in spectrograms of different dimensions, which are incompatible with CNN training. To ensure that all spectrograms had consistent dimensions, padding was applied. Padding means that values of zeros were added to the shorter spectrograms, bringing them to a uniform size.[5] This approach allowed all features to be compatible with the CNN's input requirements without distorting the original data. Another issue that required attention was the need for the CNN to accept an input with four dimensions (batch size, number of mel-bands, time frames, and number of channels). To meet the input requirements, it was necessary to add one additional axis.

A further critical factor was the restriction of resources. In Google Colab, the maximum available

RAM was 12GB. This is insufficient for processing such a large dataset. To further mitigate this issue, the *'numpy'*-library-function *'np.savez-compressed'* was utilized to compress the features with their respective labels and save them to a file, reducing memory usage. This function generates a lightweight *'*.npz'*-file, which can be easily accessed later during the training and testing of the model.[4]

Once the necessary features are created, the implementation of the model training is described in the subsequent chapter.

2.2.3 Implementation of Model Training

The following chapter will provide an overview of the implementation strategy for the model training. It is essential to begin by establishing a clear separation between the train, validation and test datasets. As no separate test data is available, the data was initially split into 70% training data and 30% rest data. The remaining data was then divided into two sets: 90% for validation and 10% for testing. In order to normalize the data, a z-score normalization was applied, utilizing the mean and standard deviation. It is important to obtain the statistics for the training data and apply the same normalization to the validation and test data to ensure consistency. The decision to normalize these data using the z-score was made in view of the potential for significant variation in values, which can be more effectively managed using a globally standardized scale.
In addition, the labels had to be adapted for encoding. At first the *'LabelEncoder'*-function from *'sklearn.preprocessing'* was used to transform the categorical labels (bona-fide and spoof) into numerical values. Each unique class is now represented by an integer. After label encoding, the numerical labels are converted into one-hot vectors using the *'keras'*-function *'to-categorical'*. A one-hot vector is a binary vector in which only the position of the corresponding class has the value 1, all other positions are 0. The one-hot vector enables the model to represent each class uniquely without implying an order between the classes.

The next stage was to define the architecture of the convolutional neural network model. These project research indicated that a 'big' model with many convolutional layers is not necessarily an optimal choice in terms of performance. Following numerous modifications to the architectural design due to overfitting issues, which will be outlined in the subsequent chapter, the final structure is described in the following. As seen in Figure 4 the architecture comprises one convolutional layer with kernel size 3x3 and activation function *'relu'*. A MaxPooling operation with a pool size of 2x2 and 2 strides is then applied. This is followed by a flatten layer and finally two Dense layers. The final layer is the output layer, which contains the number of classes and the activation function *'softmax'*. To avoid overfitting problems a dropout-layer with a rate of 0.5 was applied before the classification output layer. The model was compiled with the optimizer *'adam'*, the loss function *'categorical-crossentropy'* and the metric *'accuracy'*. For monitoring reasons a early stopping was applied during training. The training was performed with a batch size of 32 and the number of epochs was 10.

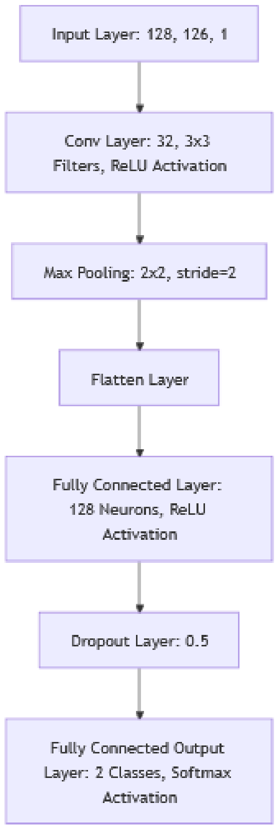

Input Layer: 128, 126, 1
Conv Layer: 32, 3x3 Filters, ReLU Activation
Max Pooling: 2x2, stride=2
Flatten Layer
Fully Connected Layer: 128 Neurons, ReLU Activation
Dropout Layer: 0.5
Fully Connected Output Layer: 2 Classes, Softmax Activation

Figure 4: Final CNN architecture for training

After diving into the model architecture, the results of the models training performance will be discussed in the next chapter.

3 Performance Results

The model performance results are presented in detail in this chapter. Before delving directly into performance accuracy, it is necessary to outline some of the challenges in the training process. Initially, training process faced a significant challenge with overfitting. At first various hyperparameter configurations were tested using GridSearchCV. For this test it was only possible to use 5% of the balanced data set (1180 samples) instead of the recommended 10% due to the limitations of the RAM. Through experimentation, it became clear that using too many convolutional layers led to immediate overfitting, with training accuracy exceeding 90% within the first three epochs. Because of that inspired by insights from a GitHub repository[5], a reduced number of convolutional layers proved more effective, delaying overfitting until around epoch 5.
In addition to that Dropout Layers were used to reduce the risk of overfitting within training. Furthermore AvgPooling layers instead of the MaxPooling layers were tested to simplify feature maps, which was not successful. An additional consideration was the potential use of different features. As the researchers found that Mel Frequency Cepstral Coefficients (MFCCs) were more powerful than mel spectrograms, these were also tested in further steps. This type of feature didn't perform well at all, so it was removed. [2] At the end the hyperparameter-tuning and the Dropout Layers could help against overfitting.

The performance results of the final model architecture with the raw mel spectrogram as input is illustrated in Figures 5. It can be seen that the model training accuracy starts with a high accuracy at around 85% in the first epoch, increases and stagnates at 90% by epoch 4. The validation accuracy follows the trend with a slightly bad performance within the first 2 epochs, but then tending to follow the train accuracy. A early stopping for monitoring purposes and preventing overfitting was implemented, which results to a early stopping at epoch 9 instead of 10.

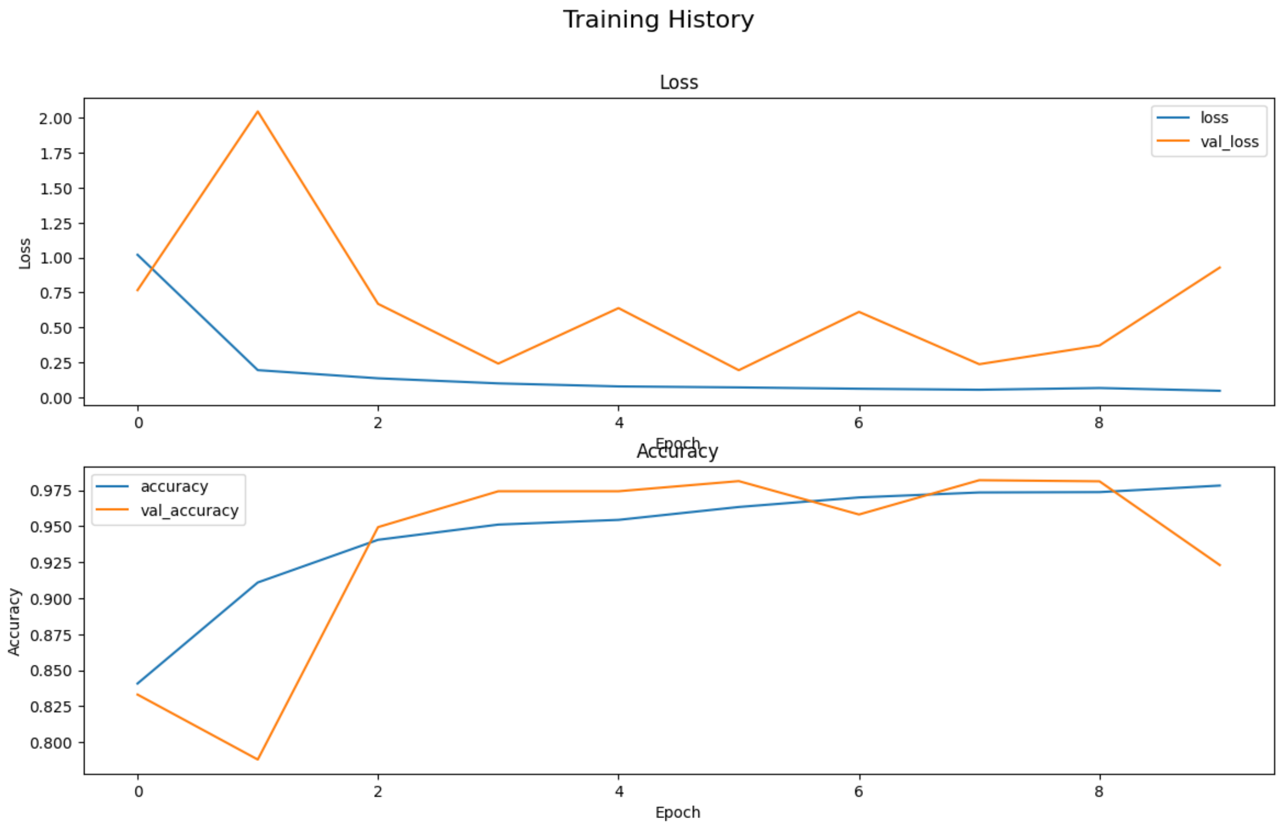


Figure 5: Model performance when using raw mel spectrograms as input feature

The model's performance was found to be quite good, which was an unexpected result. It was hypothesized that the performance would be suboptimal due to the presence of noise and the absence of an effective audio data cleaning process. The question arises why the noise has no impact on the pattern findings. The objective was to ascertain what the outcome would be in the absence of noise. To address this, a simple noise reduction function from a python library called *'noisereduce'*[6] was applied to the data. This function is based on spectral gating, which calculates a signal spectrogram and estimates a noise threshold for each frequency band of that noise. This threshold is used to calculate a mask that will gate out the noise below the threshold that varies with frequency.[6]
As shown in Figure 6, the raw audio files show some differences before and after noise reduction.

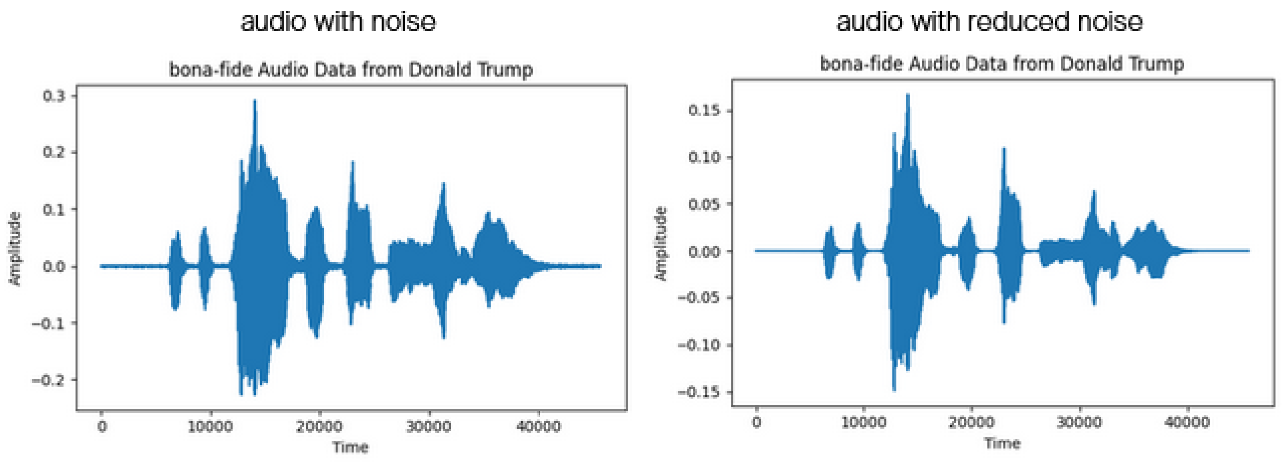


Figure 6: Audio data compared with noise (left) and noise reduced (right)

The performance results of the model, which was trained on the noise reduced mel spectrogram are illustrated in the following Figure 7.

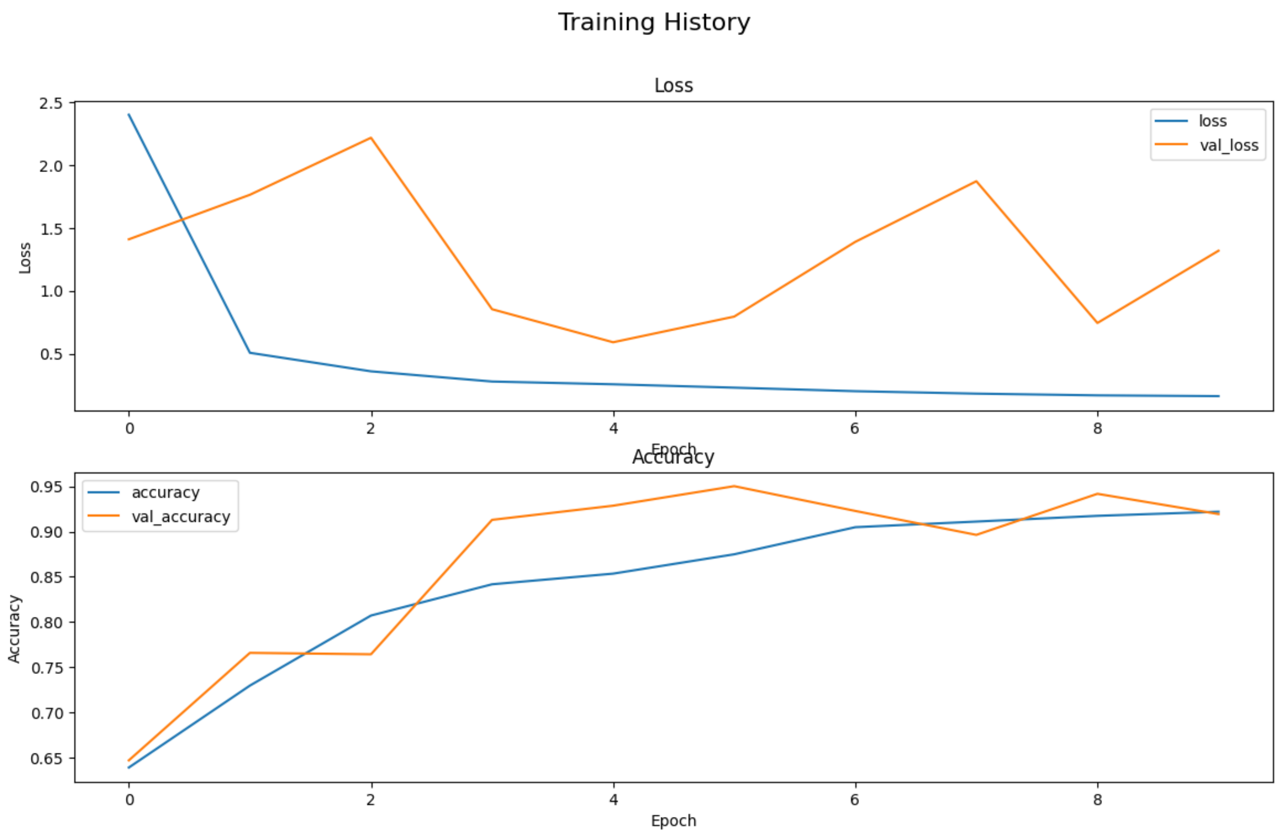
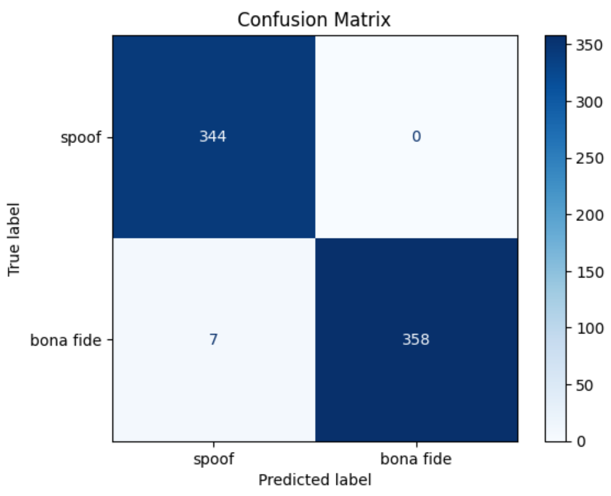


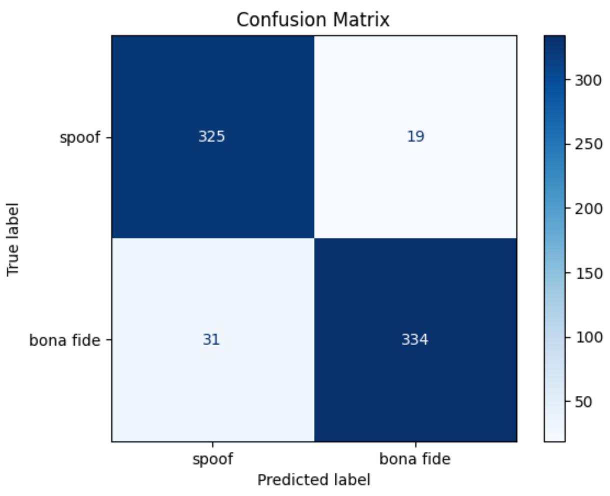
Figure 7: Model performance when using noise reduced mel spectrograms

In Figure 7 it can be seen that the validation accuracy is at some point higher in comparison to the train accuracy. It is assumed that this is due to the inclusion of the dropout, which is inactive during validation. The training accuracy increases more gradually, starting at 70% in the first epoch and reaching 90% by around epoch 6. The model performance is comparable to that of the other model, with the exception of a slower training period and higher peaks in the validation loss.

Upon testing the models with the test set, it became evident that the first model outperforms the other. Figure 8 presents the confusion matrix for both models after the prediction. The model that used the original data as input demonstrates an impressive ability to classify nearly every predicted audio file correctly. The alternative model yielded a higher number of incorrect predictions.



(a) ConfusionMatrix with raw mel spectrograms



(b) ConfusionMatrix with noise reduced mel spectrograms

Figure 8: Comparison of prediction results

4 Conclusion and Future work

This section outlines possible future work and the conclusion of this project.
There are a number of directions that can be explored to extend the findings based on the dataset in the current deepfake voice recognition system. With more resources, it would be more beneficial to utilize the full audio data rather than just a portion of 4 seconds. Furthermore, switching from under-sampling to over-sampling could help achieve a balanced dataset without losing valuable information. In addition to that Data Augmentation would be a interesting method to increase the dataset.
For feature extraction, trying alternative features as constant-Q transform spectrogram (cqtspec), which are even more powerful features might yield better results or complement the current mel spectrogram approach. [1] These features have the potential to capture different aspects of the audio signal. A combination of features could also prove beneficial.
In terms of model training, experimenting with new architectures could unlock unexploited potential. With more time and research, it may be possible to design a more optimal model structure. Additionally, it would be interesting to fine-tuning pre-trained models, such as those trained on datasets like VCTK. It is interesting if it would be possible to adapt to datasets with noise and achieve a generalization. A further really interesting approach would be how the model behave when predicting data from VCTK dataset, that don't show some noises in the data. Could the model predict these

without having the noise as pattern?

In conclusion, the results of the project show that noise in data can help the model to generalize on unseen data. But it is important to consider, that only a moderate amount of noise could be beneficial. Clean data is necessary for train a model and more that a moderate amount would lead to bad model performance. In research there are also approaches which shows that moderate noise helps. [7] Additionally, the quantity of data plays a significant role; having a larger and more diverse dataset enables models to learn robustly. Finally, sufficient computational resources, particularly RAM, are essential for handling large datasets and implementing techniques like hyperparameter optimization or oversampling.
It is important to address these aspects in order to improve model performance and advance capabilities in the detection of Deepfake voices.

References

[1] Nicolas M. Müller et al. *Does Audio Deepfake Detection Generalize?* 2024. arXiv: 2203.16263 [cs.SD]. URL: https://arxiv.org/abs/2203.16263.

[2] Mvelo Mcuba et al. "The Effect of Deep Learning Methods on Deepfake Audio Detection for Digital Investigation". In: *Procedia Computer Science* 219 (2023). CENTERIS – International Conference on ENTERprise Information Systems / ProjMAN – International Conference on Project MANagement / HCist – International Conference on Health and Social Care Information Systems and Technologies 2022, pp. 211–219. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2023.01.283. URL: https://www.sciencedirect.com/science/article/pii/S1877050923002910.

[3] Jason Brownlee. *8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset.* 2020. URL: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.

[4] Scott Duda. *Urban Environmental Audio Classification Using Mel Spectrograms.* 2020. URL: https://scottmduda.medium.com/urban-environmental-audio-classification-using-mel-spectrograms-706ee6f8dcc1.

[5] sksmta. *Audio DeepFake Detection.* 2023. URL: https://github.com/sksmta/audio-deepfake-detection/blob/main/main.ipynb.

[6] Tim Sainburg. *timsainb/noisereduce: v1.0.* Version db94fe2. June 2019. DOI: 10.5281/zenodo.3243139. URL: https://doi.org/10.5281/zenodo.3243139.

[7] Shi Yin et al. "Noisy training for deep neural networks in speech recognition". In: *EURASIP Journal on Audio, Speech, and Music Processing* 2015 (Dec. 2015). DOI: 10.1186/s13636-014-0047-0.