Design Manual

Project Mozart

iOS Personal Voice/PlayHT/FastSpeech2 Synthetic Voice Detection and the Impacts of Noise

Contributing Authors: Branch Hill and Jacob Pomatto

Dec 7, 2023

Acronym	Meaning
AD	Audio Deepfake
ANN	Artificial Neural Network
AR-DAD	Arabic Diversified Audio Dataset
ASV	Automatic Speaker Verification
AWGN	Additive White Gaussian Noise
BPTT	Backpropagation Through Time
CNN	Convolution Neural Network
CQCC	Constant Q Cepstral Coefficients
DBiLSTM	Deep Bidirectional Long Short-Term Memory
DL	Deep Learning
DNN	Deep Neural Network
DT	Decision Tree
DTW	Dynamic Time Warping
EER	Equivalent Error Rate
FoR	Fake or Real
FQT	Final Qualification Test
GPG	GNU Privacy Guard
HLL	Human in Loop Learning
HMM	Hidden Markov Model
HTTPS	Hypertext Transfer Protocol Secure
KNN	K-Nearest Neighbors
LFCC	Linear Frequency Cepstral Coefficient
LGBM	Light Gradient-Boosting Machine
LLR	Log-Likelihood Ratios
LR	Linear/Logistic Regression
LSTM	Long Short-Term Memory
MFCC	Mel Frequency Cepstral Coefficient
RF	Random Forest
RNN	Recurrent Neural Network
SNR	Signal-Noise Ratio
SSAD	Semi-Supervised Anomaly Detection
SSH	Secure Shell Protocol
STFT	Short-Time Fourier Transform
STN	Spatial Transformer Network
SVM	Support Vector Machine
TCN	Temporal Convolution Network
WSL	Windows Subsystem for Linux

Table of Contents

Table of Figures

Table of Tables

Table of Scripts

Introduction

In recent years, generative AI has exploded in capabilities and fidelity that make the media they produce virtually indistinguishable from human-produced counterparts. While this may allow for the increased productivity across several domains, the technology may be leveraged in harmful ways. In the realm of speech synthesis, an emerging scam is to target some individual, collect audio recordings of a significant person in their life, clone that person's voice, and then beg for money, where the scammer pretends to be that loved one in a horrible situation. To counter unethical use of these technologies, detection mechanisms need to be implemented and made accessible to protect individuals from false information.

Objectives and Purpose

The primary goal of this project is to achieve a reasonable detection accuracy (>80%) with an acceptable equivalent error rate (EER) when detecting deep fake audio signals. As the project was developing, two secondary goals emerged. When there is significant signal noise, many detectors lose substantial detection accuracy, and consequently, the project adopted an objective to implement a model resilient to the effects of noise. Moreover, AI-based audio synthesis tools are becoming increasingly available to the public. With concerns of how this technology may be used, another goal of the project was to verify reasonable detection accuracy against accessible, contemporary generative speech algorithms.

Previous Work

When initially presented with the task of detecting deep faked speech, the team's initial thoughts were to explore a non-machine learning algorithm, since no one on the team had any experience with the subject. Preliminary research of traditional algorithms revealed two main categories: template matching and state-based statistical models.

Template matching is arguably the simpler algorithm in terms of theory and implementation. The fundamental behavior of the algorithm involves loading an input speech signal and comparing it with a list of pre-defined signal templates, where the best matching template is returned. In its most primitive form, the comparison of signal features uses Euclidean distance. That is, the time-domain is considered constant, and a time slice in the template is compared directly to the corresponding time slice in the signal, Figure 1. However, this implementation is vulnerable to accuracy loss when the time domain is scaled or shifted. If a person were to speak faster, the algorithm is susceptible to misclassifying the speech.

This drawback of Euclidean matching may be corrected by leveraging a dynamic time warping (DTW) algorithm [1]. This algorithm produces a discrete mapping between elements of one time series to another, such that the similarity of the signal is measured instead of the congruence, as illustrated in Figure 1. The calculation is performed by first initializing a matrix of size NxM, where N is the length of the target signal and M is the length of the template signal. Starting at the bottom-left corner, or the start of each signal, the distance between each sample and every other sample in the opposite signal is computed. Once the matrix has been completely populated, the minimum path spanning from the top-right corner of the matrix to the bottom-left corner may be backtraced, Figure 2. The resulting path represents the best mapping of elements in the template signal to the target signal.

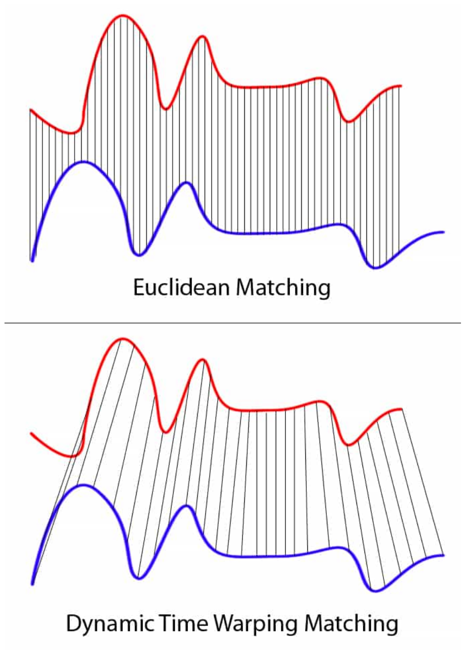
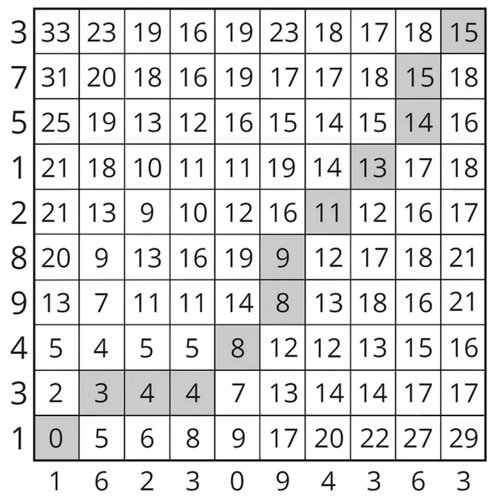


Fig. 1. Euclidean vs. Dynamic Time Warping Matching [3]



3	33	23	19	16	19	23	18	17	18	15
7	31	20	18	16	19	17	17	18	15	18
5	25	19	13	12	16	15	14	15	14	16
1	21	18	10	11	11	19	14	13	17	18
2	21	13	9	10	12	16	11	12	16	17
8	20	9	13	16	19	9	12	17	18	21
9	13	7	11	11	14	8	13	18	16	21
4	5	4	5	5	8	12	12	13	15	16
3	2	3	4	4	7	13	14	14	17	17
1	0	5	6	8	9	17	20	22	27	29
	1	6	2	3	0	9	4	3	6	3

Fig. 2. Dynamic Time Warping Matrix [3]

While template matching with DTW may yield accurate results, it has a major limitation when considering the use case of the project. The matching template set must be pre-defined. For a general deep fake classification algorithm, the resource requirements of storing and matching a complete set of real and fake speech signals is unfeasible. Instead, templates of common fake speech artifacts not shared by real speech signals would need to be identified and aggregated into a set. If a certain number of fake artifacts are detected, the classification algorithm would claim the signal is fake. Ultimately, this approach appeared to be a brute force solution that required significant manual configuration, and the team decided to pursue other implementations.

State-based statistical models give a more general classification algorithm, but they do not necessarily overcome the issues previously described for template matching [2]. The foundation of state-based statistical models are predominantly hidden Markov models (HMM), Figure 3. These models are probabilistic frameworks that contain all possible states of a system and the probability of transitioning to another state. Without machine learning, these models must be produced manually, leaving a lot of room for potential error. Additionally, further research revealed that these models are most commonly used for speech decoders, rather than a high-level classification algorithm.

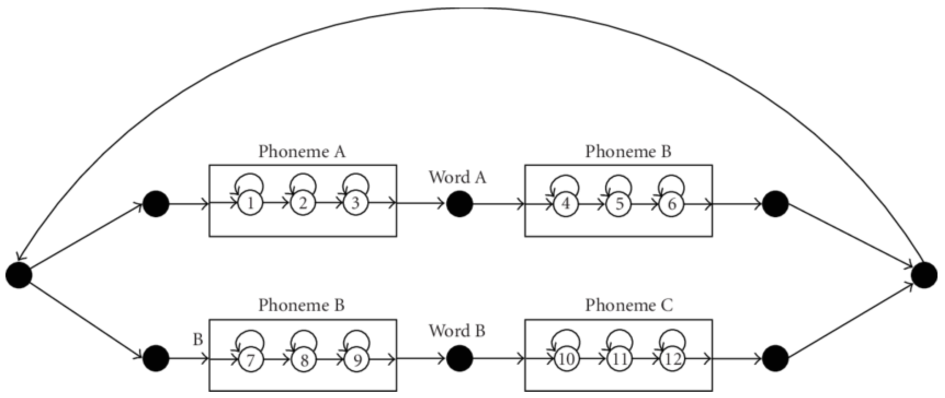


Fig. 3. Hidden Markov Model for Speech Recognition [7]

It did not appear that traditional speech recognition algorithms would satisfy the use case of the project, so the team decided to research machine learning and artificial intelligence oriented techniques and found an abundance of modern work. A recent summarization on the field of audio deep fakes (ADs) by Zaynab Almutairi and Hebah Elgibreen documented the currently known types of AD attacks, numerous leading detection methods, and the current challenges they are facing [4].

The paper outlines three main categories of attacks: imitation-based, synthetic-based, and replay-based AD attacks. Conceptually, imitation-based deep fakes are similar to voice masking modulators in the sense that an input speech is transformed to hide the identity of the originating speaker. To further mask the speaker, imitation attacks extract acoustic parameters from a similar target signal and convert the incoming speech into audio that sounds similar to the intended target, Figure 4 [4].

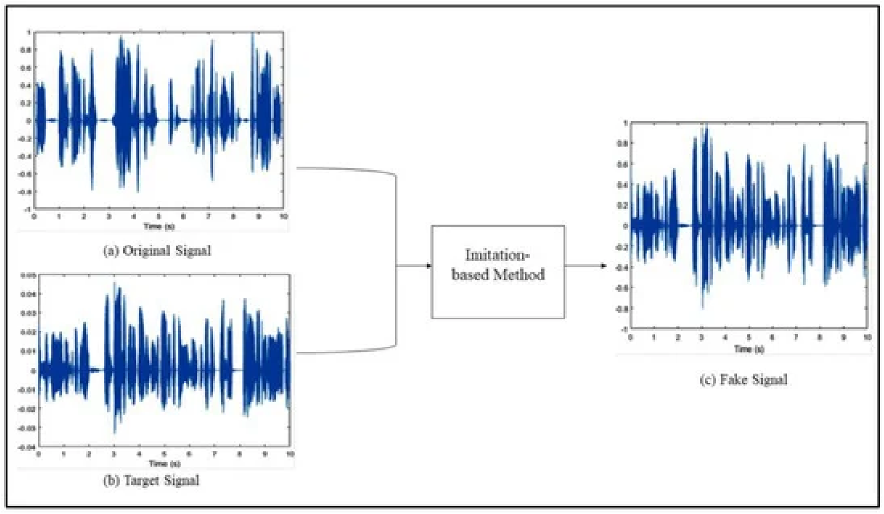


Fig. 4. Imitation-based Speech Generation [4]

Synthetic-based deep fakes transfer text directly into natural speech in real-time; however, these deep fakes have large data requirements. To produce the speech, a generative synthetic audio model, such as Tactoran 2 or FastSpeech2, must be trained with audio from the target speech [4]. Once completed, text may be supplied to the model where spectral parameters obtained during the training process produce a waveform with similar acoustic properties as the target speech, Figure 5 [4].

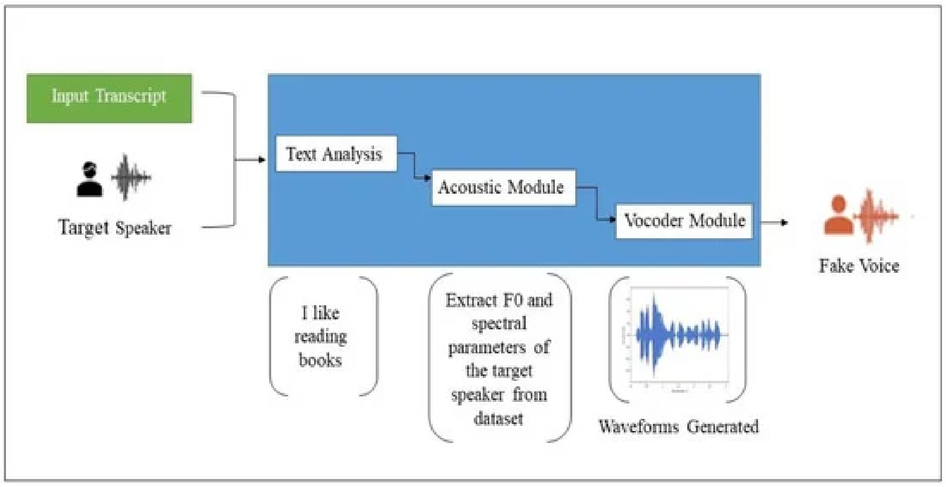


Fig. 5. Synthetic-based Speech Generation [4]

When compared to the previous two techniques, Replay-based deep fakes are the most primitive in implementation. These deep fakes are produced by manually cutting and pasting segments of target audio to form the desired speech. As these deep fakes use real target audio, rather than synthetically generated audio, it is often considered out of scope for most deep fake detection methods [4].

In viewing the summarized detection methods, it appeared to be a common pattern amongst the implementations to leverage deep neural networks (DNNs) to execute the detection mechanism, Table 1, and many methods demonstrated accuracies of greater than 90%, Table 2.

Compared to artificial neural networks (ANNs), DNNs contain multiple hidden layers that allow for exceptionally complex data processing and pattern recognition, Figure 6. This quality of DNNs enables a diverse realization of the technology that allows many modern detection methods to be unique, while remaining rooted by the same fundamental principles.

Table 1. Summary of AD detection methods studies [4]

Year	Ref.	Speech Language	Fakeness Type	Technique	Audio Feature Used	Drawbacks
2018	Yu et al. [12]	English	Synthetic	DNN-HLL	MFCC, LFCC, CQCC	The error rate is zero, indicating that the proposed DNN is overfitting.
				GMM-LLR	IMFCC, GFCC, IGFCC	Does not carry much artifact information in the feature representations perspective.
2019	Alzantot et al. [13]	English	Synthetic	Residual CNN	MFCC, CQCC, STFT	The model is highly overfitting with synthetic data and cannot be generalized over unknown attacks.
2019	C. Lai et al. [14]	English	Synthetic	ASSERT (SENet + ResNet)	Logspec, CQCC	The model is highly overfitting with synthetic data.
2020	P. RahulT et al. [15]	English	Synthetic	ResNet-34	Spectrogram	Requires transforming the input into a 2-D feature map before the detection process, which increases the training time and affects its speed.
2020	Lataifeh et al. [16]	Classical Arabic	Imitation	Classical Classifiers (SVM-Linear, SVMRBF, LR, DT, RF, XGBoost)	-	Failed to capture spurious correlations, and features are extracted manually so they are not scalable and needs extensive manual labor to prepare the data.
				DL Classifiers (CNN, BiLSTM)	MFCC spectrogram	DL accuracy was not as good as the classical methods, and they are an image-based approach that requires special transformation of the data.
2020	Rodríguez-Ortega et al. [17]	Spanish, English, Portuguese, French, and Tagalog	Imitation	LR	Time domain waveform	Failed to capture spurious correlations, and features are extracted manually so it is not scalable and needs extensive manual labor to prepare the data.
2020	Wang et al. [18]	English, Chinese	Synthetic	Deep-Sonar	High-dimensional data visualization of MFCC, raw neuron, activated neuron	Highly affected by real-world noises.
2020	Subramani and Rao [19]	English	Synthetic	EfficientCNN and RES-EfficientCNN	Spectrogram	They use an image-based approach that requires special transformation of the data to transfer audio files into images.
2020	Shan and	English	Synthetic	Bidirectional LSTM	MFCC	The method did not perform well over

	Tsai [20]					long 5 s edits.
2020	Wijethunga et al. [21]	English	Synthetic	DNN	MFCC, Mel-spectrogram, STFT	The proposed model does not carry much artifact information from the feature representations perspective.
2020	Jiang et al. [22]	English	Synthetic	SSAD	LPS, LFCC, CQCC	It needs extensive computing processing since it uses a temporal convolutional network (TCN) to capture the context features and another three regression workers and one binary worker to predict the target features.
2020	Chintha et al. [23]	English	Synthetic	CRNN-Spoof	CQCC	The model proposed is complex and contains many layers and convolutional networks, so it needs an extensive computing process. Did not perform well compared to WIRE-Net-Spoof.
				WIRE- Net-Spoof	MFCC	Did not perform well compared to CRNN-Spoof.
2020	Kumar-Singh and Singh [24]	English	Synthetic	Q-SVM	MFCC, Mel-spectrogram	Features are extracted manually so it is not scalable and needs extensive manual labor to prepare the data.
2020	Zhenchun Lei et al. [25]	English	Synthetic	CNN and Siamese CNN	CQCC, LFCC	The models are not robust to different features and work best with LFCC only.
2021	M. Ballesteros et al. [26]	Spanish, English, Portuguese, French, and Tagalog	Synthetic Imitation	Deep4SNet	Histogram, Spectrogram, Time domain waveform	The model was not scalable and was affected by the data transformation process.
2021	E.R. Bartusiak and E.J. Delp [27]	English	Synthetic	CNN	Spectrogram	They used an image-based approach, which required a special transformation of the data, and the authors found that the model proposed failed to correctly classify new audio signals indicating that the model is not general enough.
2021	Borrelli et al. [28]	English	Synthetic	RF, SVM	STLT	Features extracted manually so they are not scalable and needs extensive manual labor to prepare the data.
2021	Khalid et al. [29]	English	Synthetic	MesoInception-4, Meso-4, Xception, EfficientNet-B0, VGG16	Three-channel image of MFCC	It was observed from the experiment that Meso-4 overfits the real class and MesoInception-4 overfits the fake class, and none of the methods provided a satisfactory performance indicating that they are not suitable for fake audio detection.
2021	Khochare et al. [30]	English	Synthetic	Feature-based (SVM, RF, KNN, XGBoost, and LGBM)	Vector of 37 features of audio	Features extracted manually so they are not scalable and needs extensive manual labor to prepare the data.

				Image-based (CNN, TCN, STN)	Melspectrogram	It uses an image-based approach and could not work with inputs converted to STFT and MFCC features.
2021	Liu et al. [31]	Chinese	Synthetic	SVM	MFCC	Features extracted manually so it is not scalable and needs extensive manual labor to prepare the data.
				CNN	--	The error rate is zero indicating that the proposed CNN is overfitting.
2021	S. Camacho et al. [32]	English	Synthetic	CNN	Scatter plots	It did not perform as well as the traditional DL methods, and the model needed more training.
2021	T. Arif et al. [33]	English	Synthetic imitated	DBiLSTM	ELTP-LFCC	Does not perform well over an imitated-based dataset.

Table 2. Quantitative comparison between AD detection methods [4]

Measures	Dataset	Detection Method	Results (The Result Is Approximate from the Evaluation Test Published in the Study)
EER	ASV spoof 2015 challenge	DNN-HLLs [12]	12.24%
		GMM-LLR [12]	42.50%
	ASV spoof 2019 challenge	Residual CNN [13]	6.02%
		SENet-34 [14]	6.70%
		CRNN-Spoof [23]	4.27%
		ResNet-34 [15]	5.32%
		Siamese CNN [25]	8.75%
		CNN [25]	9.61%
		DBiLSTM [33] (Synthetic Audio)	0.74%
		DBiLSTM [33] (Imitation-based)	33.30%
		SSAD [22]	5.31%
	-	Bidirectional LSTM [20]	0.43%
	FoR	CNN [32]	11.00%
		Deep-Sonar [18]	2.10%
t-DCF	ASV spoof 2019 challenge	Residual CNN [13]	0.1569
		SENet-34 [14]	0.155
		CRNN-Spoof [23]	0.132
		ResNet-34 [15]	0.1514
		Siamese CNN [25]	0.211
		CNN [25]	0.217
		DBiLSTM [33]	0.008

		(Synthetic Audio)	
		DBiLSTM [33] (Imitation-based)	0.39
Accuracy	ASV spoof 2019 challenge	CNN [27]	85.99%
		SVM [28]	71.00%
	AR-DAD	CNN [16]	94.33%
		BiLSTM [16]	91.00%
		SVM [16]	99.00%
		DT [16]	73.33%
		RF [16]	93.67%
		LR [16]	98.00%
		XGBoost [16]	97.67%
		SVMRBF [16]	99.00%
		SVM-LINEAR [16]	99.00%
	FoR	DNN [21]	94.00%
		Deep-Sonar [18]	98.10%
		STN [30]	80.00%
		TCN [30]	92.00%
		SVM [30]	67%
		RF [30]	62%
		KNN [30]	62%
		XGBoost [30]	59%
		LGBM [30]	60%
		CNN [32]	88.00%
	FakeAVCeleb	EfficientNet-B0 [29]	50.00%
		Xception [29]	76.00%
		MesoInception-4 [29]	53.96%
		Meso-4 [29]	50.36%
		VGG16 [29]	67.14%
	H-Voice	LR [17]	98%
		Deep4SNet [26]	98.50%
	-	Q-SVM [24]	97.56%
	-	CNN [31]	99%
	-	SVM [31]	99%

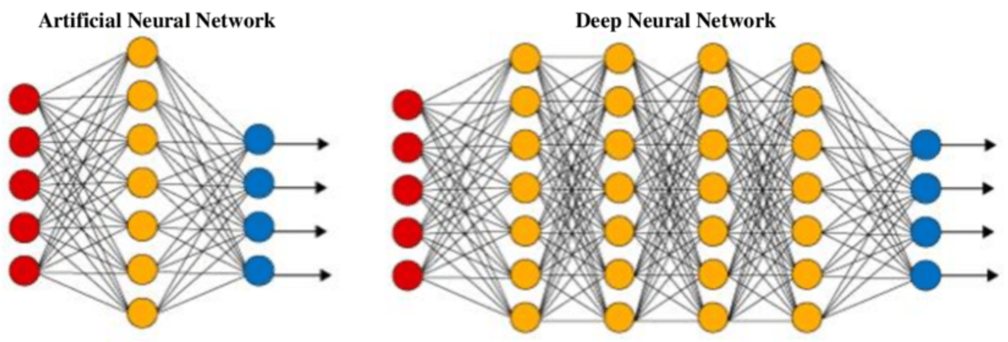


Fig. 6. Deep Neural Network Architecture[11]

Regardless of the detection method, they all appeared to share a common flaw. Real world noise negatively impacted detection rates of the models, and DeepSonar was notably impacted by real-world noise with accuracy losses up to 20% [6]. With such drastic losses in accuracy, it is crucial that audio deep fake detection methods are resilient to noise, especially as they are incorporated into real world applications. Despite its importance, there is a lack of documented research tackling this issue highlighting a key area for future research [4].

Of the deep fake detection methods discussed in Almutairi's et al. summary, DeepSonar is the most influential to the project. Because noise impacted DeepSonar's detection method the most, the team believed that advancements in resiliency would be most evident with this implementation.

DeepSonar's deep neural network is composed of a convolution neural network (CNN) that feeds into a recurrent neural network (RNN) [6]. A convolution neural network is a two-dimensional neural network that is commonly leveraged for computer vision. While this may initially appear as poor choice for one-dimensional audio signals, preprocessing that signal into a spectrogram produces a two-dimensional representation of the spectral components of that signal. During the training process, the CNN adjusts the weights of the kernel matrix so that the optimal features of the spectrogram are identified for deep fake detection. During the prediction phase, the CNN is able to efficiently determine features in the audio signal, Figure 7.

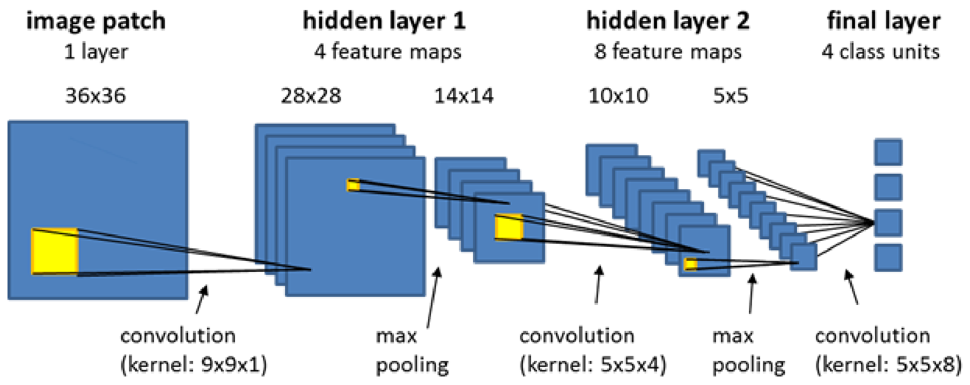


Fig. 7. Convolution Neural Network Architecture [8]

A recurrent neural network is a one-dimensional neural network where the output of some layers feeds into the input of preceding layers. This cyclic feature enables this network to detect temporal qualities, but introduces some difficulty training an RNN with a traditional backpropagation algorithm. Instead, a variation of the algorithm called backpropagation in time (BPTT) unfolds time to eliminate the cyclic tree during training; however, it is a computationally taxing process leading to slow training times [10]. Once trained, the RNN detects long-term dependencies in time variances of the CNN output, Figure 8.

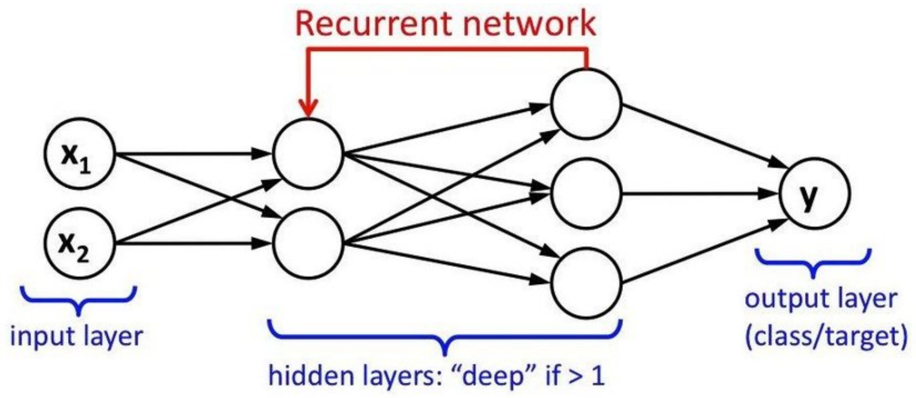


Fig. 8. Recurrent Neural Network Architecture [9]

With this combination, DeepSonar is able to achieve an average accuracy of 98.1% with a false alarm rate of about 2% under ideal conditions [6]. When the robustness of the detection method was evaluated by introducing indoor and outdoor noise to the signals, a significant decrease in detection accuracy was observed. Footsteps with a signal to noise ratio (SNR) of five reduced the accuracy by nearly 20%, while wind and rain with an SNR of five reduced the accuracy by 25% [6].

Beyond the architecture of the machine learning model, it is vital that there is enough data present to adequately train it. WaveFake is a project that claims that deep fake audio detection has been neglected in favor of its image counterpart [5]. To combat this, WaveFake provides several datasets of synthetically generated audio files of human speech using various algorithms so that researchers may have easily accessible datasets to train their models.

Contemporary Algorithms

With the release of iOS 17, Apple has made a Personal Voice feature available to Apple product users to synthesize an individual’s voice for phone calls or speaker playback. To produce a voice model, the user must read 150 generic phrases, and once the voice capture has been completed, the user’s local device will train the voice model while the device is charging and locked. The specific details of the algorithm’s implementation are not publicly available.

Available as a web tool, PlayHT is a modern and publicly available AI voice generator that targets media creators. While most packages offered by the platform require a subscription fee, there is a free plan that increases the accessibility to the tool, though with reduced capabilities. Similar to Apple’s Personal Voice the user must provide several recorded phrases spoken by the target voice. The PlayHT platform then uses a learning language model to generate the synthetic voice. The specific details of the algorithm’s implementation are not publicly available.

Project File Structure

```
CS657-Audio-Deep-Fake-Detector
├── .devcontainer          (Development Container Build Information)
│
├── scripts                (Utility Bash Scripts)
│   ├── *.sh
│
├── src
│   ├── data_processor     (Orchestrates Data Pipeline)
│   ├── deepfake_detector  (Orchestrates Data Predictions)
│   ├── model_utilities    (Organizes Model Layers)
│   ├── noise_filter       (Provides Noise Filtering Methods)
│   ├── noise_generator    (Provides Noise Generation Methods)
│   ├── plotter            (Provides Graphical Plotting Utilities)
│   ├── *.py
│
├── workflows              (High-Level Demonstrable Workflow Scripts)
│   ├── *.sh
│
├── .gitignore             (Specifies Files That Git Will Not Track)
├── README.md              (Instructions To Setup Environment)
├── requirements.txt       (Python Dependencies Necessary For Project)
```

Project Repository

The source code for this project is public on GitHub and can be found at the following location:
https://github.com/GuassianFlux/CS657-Audio-Deep-Fake-Detector
The project repository contains the following contents:

- Instructions for setting up the development environment
- DevContainer configuration files
- Scripts for installing all dependencies
- Source code for loading datasets, fitting the model, visualizing data, and making predictions
- High level workflow scripts
- Documentation:
  - Design Manual: Outlines the software design and project architecture
  - User Manual: Explains and demonstrates how the software is used
  - Final Qualification Test: Explains how to setup the development environment and reproduce results
  - Research Paper: Summaries the project from beginning to end and presents the final

Interested parties may clone and contribute to the project using the following link:

```
$ git clone https://github.com/GuassianFlux/CS657-Audio-Deep-Fake-Detector
```

Design Overview

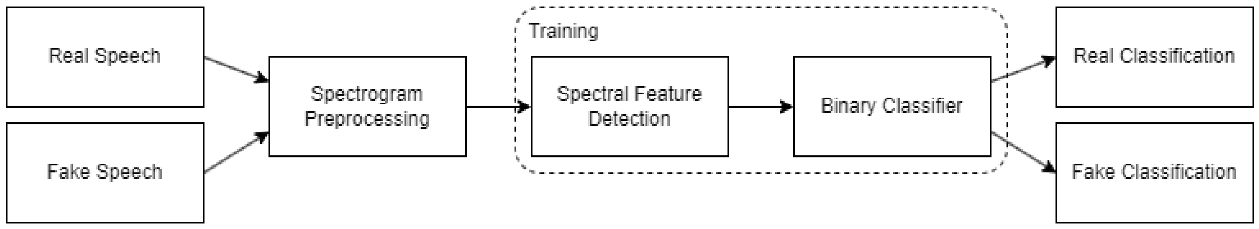


Fig. 9. Audio Detection Design Overview

At a high-level, the design of the detection algorithm consists of three main components: spectrogram preprocessing, spectral feature detection, and binary classification. The spectrogram preprocessing stage is responsible for accepting incoming speech data, both real and fake, and converting the one-dimensional data structure into a two-dimensional representation of the signal that illustrates the intensity of the signal's key frequencies with respect to time. This two-dimensional data is then fed into the spectral feature detection component, where a series of image convolutions are performed to highlight important features of the signal. With the signal reduced to key features, the binary classifier determines if the pattern is more similar to fake or real speech. During the training process, the weights of the convolution kernels and classification neurons are dynamic and are constantly changing to maximize a loss function, but once the training is complete, these weights are assumed to be optimal and may be applied directly without training for real-world applications.

Beyond the organization of the algorithm, the content of the training datasets consists of real and synthetic human speech produced using the HiFi-GAN-2 generative algorithm. To increase robustness against noise, additive white gaussian and burst noise have been mixed into the datasets.

Data Organization

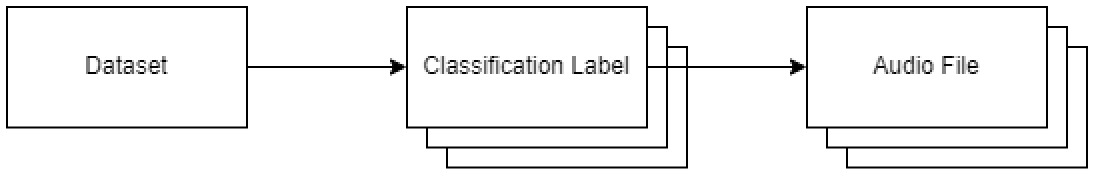


Fig. 10. Keras API Data Set Format

As previously mentioned, the organization of the project's data plays a vital role in performance of the model, and by leveraging Keras, a modern library for machine learning, a general format for the data was enforced, as shown in Figure 10. This format has three main components: the dataset, the classification labels, and the audio files. The dataset is a single directory that serves as an entry point to the data. This dataset folder may not contain any audio files directly, rather it contains sub-directories that represent categories of data. The name of these sub-directories act as the classification labels that will ultimately be referenced by the model. Each sub-directory then may contain the audio data. The audio files must be in a WAVE format, but they may be variable in length and bitrate.

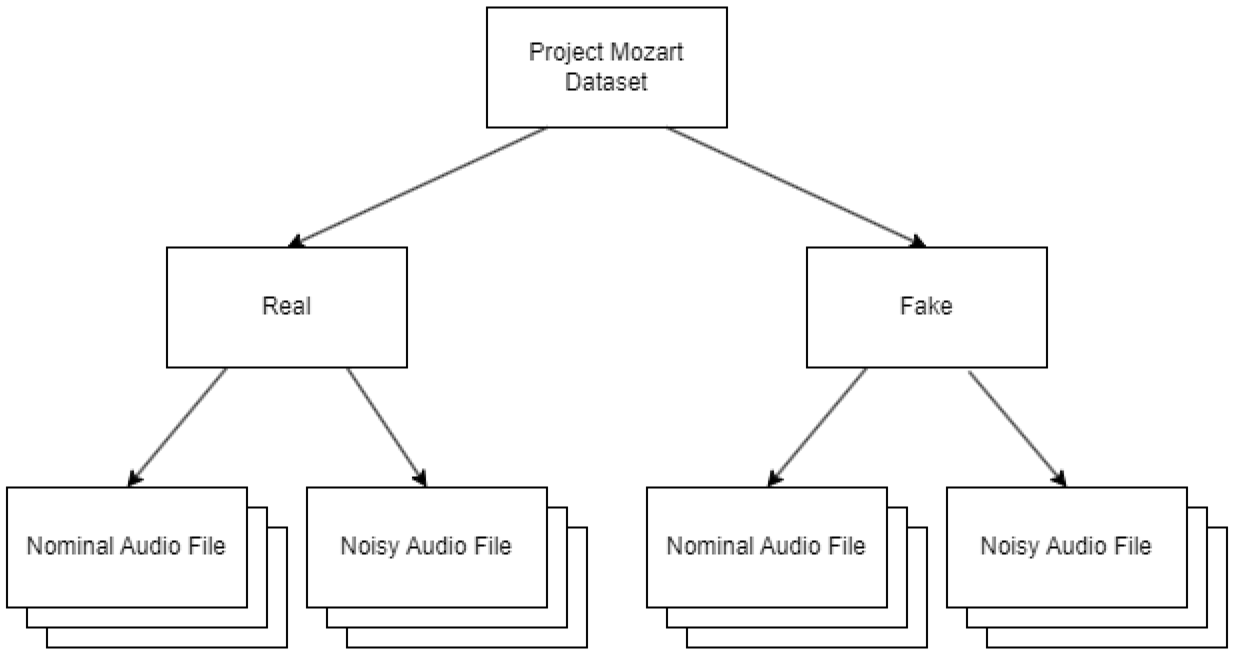


Fig. 11. Project Mozart Data Set Architecture

Following the rules enforced by Keras, the dataset for Project Mozart was constructed as seen in Figure 11. The objectives of the project were to determine if a given audio file was produced synthetically or organically; this ultimately boils down to a binary classification problem. Either it is a deep fake or it is not. Because of the nature of the problem, the dataset only needs to contain two classification labels: real and fake.

Moreover, as seen in the prior work section, Deep Sonar was notably impacted by real-world noise, and it became an objective of the project to incorporate some level of resiliency against it. Initial thoughts by the team were that filtering the noise to produce a pure dataset of nominal audio files would produce the best results, but testing revealed that the filtering mechanisms used altered the spectral composition of the audio files far greater than the original noise. While the audio signal sounded much clearer to humans, it was unrecognizable to the model. The alternative solution was to incorporate noisy audio files into each category so that the model would be familiar with this type of data.

Spectrogram Preprocessing

To prepare the audio signal for processing by the model, the one-dimensional signal is converted into a two-dimensional spectrogram, where the spectral components of the signal are illustrated with respect to time. This transformation is performed with a short-time fourier transform (STFT) with a frame length of 255 and frame step of 128. The STFT algorithm is identical to a simple Fourier transform with the exception that small windows of the signal are considered, rather than the complete signal Figure 12. This allows for a continuous application of the algorithm to be performed, enabling real-time or dynamic length processing inherently. The absolute value of the aggregated output of this computation is then obtained before passing the data to the feature detection model. This operation shifts focus to the magnitude of spectral features rather than polarity.

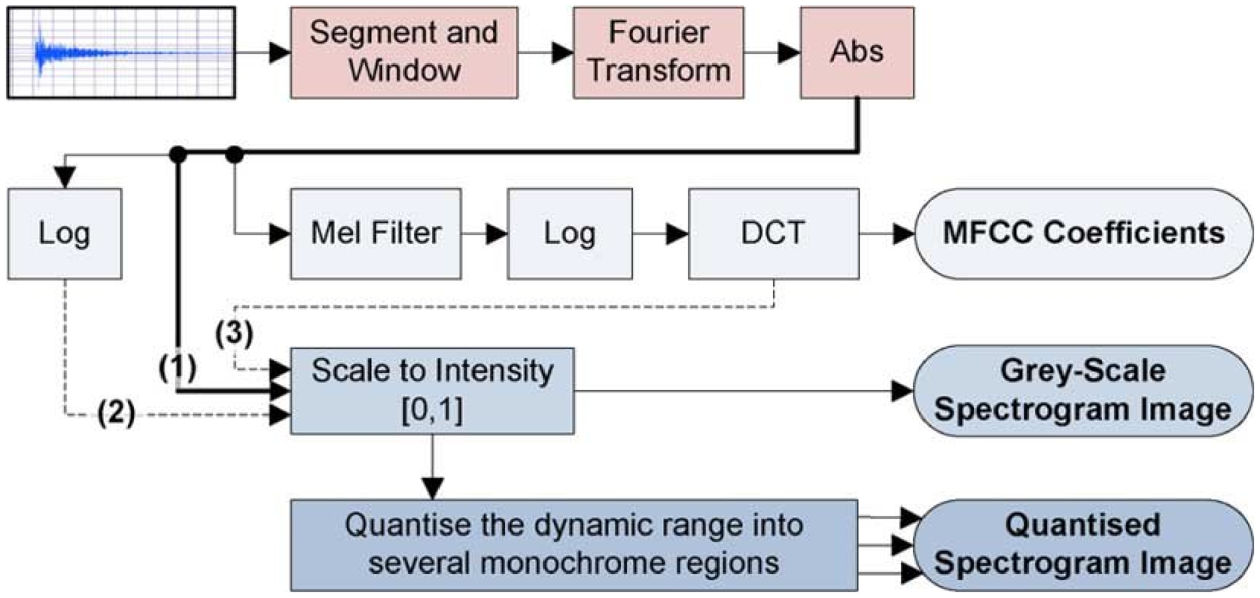

Segment and Window
Fourier Transform
Abs
Log
Mel Filter
Log
DCT
MFCC Coefficients
(3)
(1)
(2)
Scale to Intensity [0,1]
Grey-Scale Spectrogram Image
Quantise the dynamic range into several monochrome regions
Quantised Spectrogram Image

Fig. 12. Spectrogram Preprocessing Overview [34]

Spectral Feature Detection

The spectral feature detection component is the first stage of the machine learning model designed for the project and receives the previously produced spectrograms as input. This detection algorithm is responsible for learning patterns in the spectral components of the real and fake audio signals, while also transforming the two-dimensional input into a one-dimensional output. These functions are performed across four unique layers, as illustrated in Figure 13.

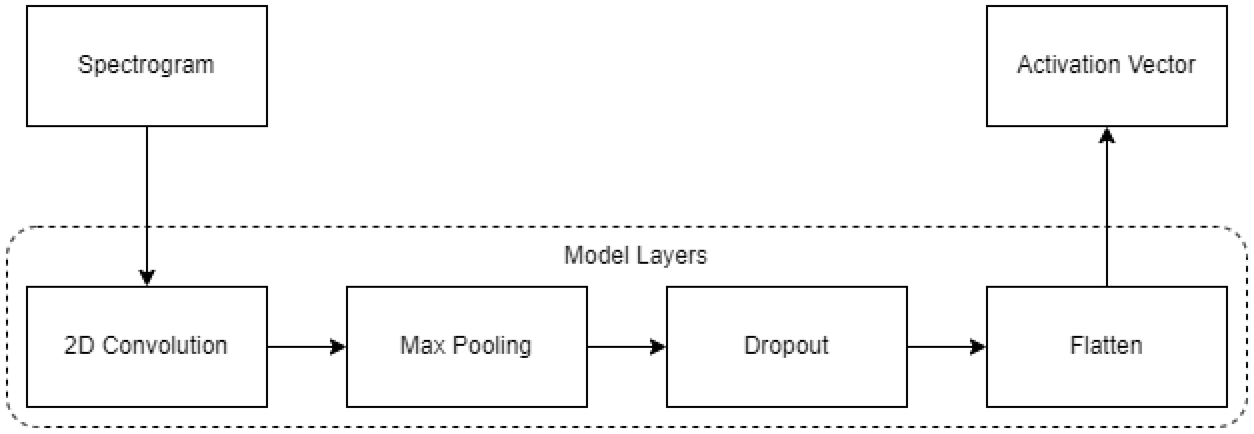


Fig. 13. Spectral Feature Detection Model Layers Overview

Commonly used for image processing, the first layer is a convolution layer, and during its initialization, the dimensions of the previously produced spectrograms are sampled to determine the input buffer shape. Additionally, the layer is instantiated to have 32 filters with kernel size of

3x3. Figure 14 demonstrates this operation on a reduced scale to highlight the process. Denoted by the red outline, the kernel is slid across the spectrogram, where each matrix value, or pixel, is multiplied by the corresponding value in the kernel and accumulated into a single value for the window. The values, or weights, in the kernel are the variable tuning parameters that the model adjusts during the training process to produce different filtered outputs. This process is repeated for each filter declared during instantiation allowing a single model to detect a handful of unique features in the provided datasets.

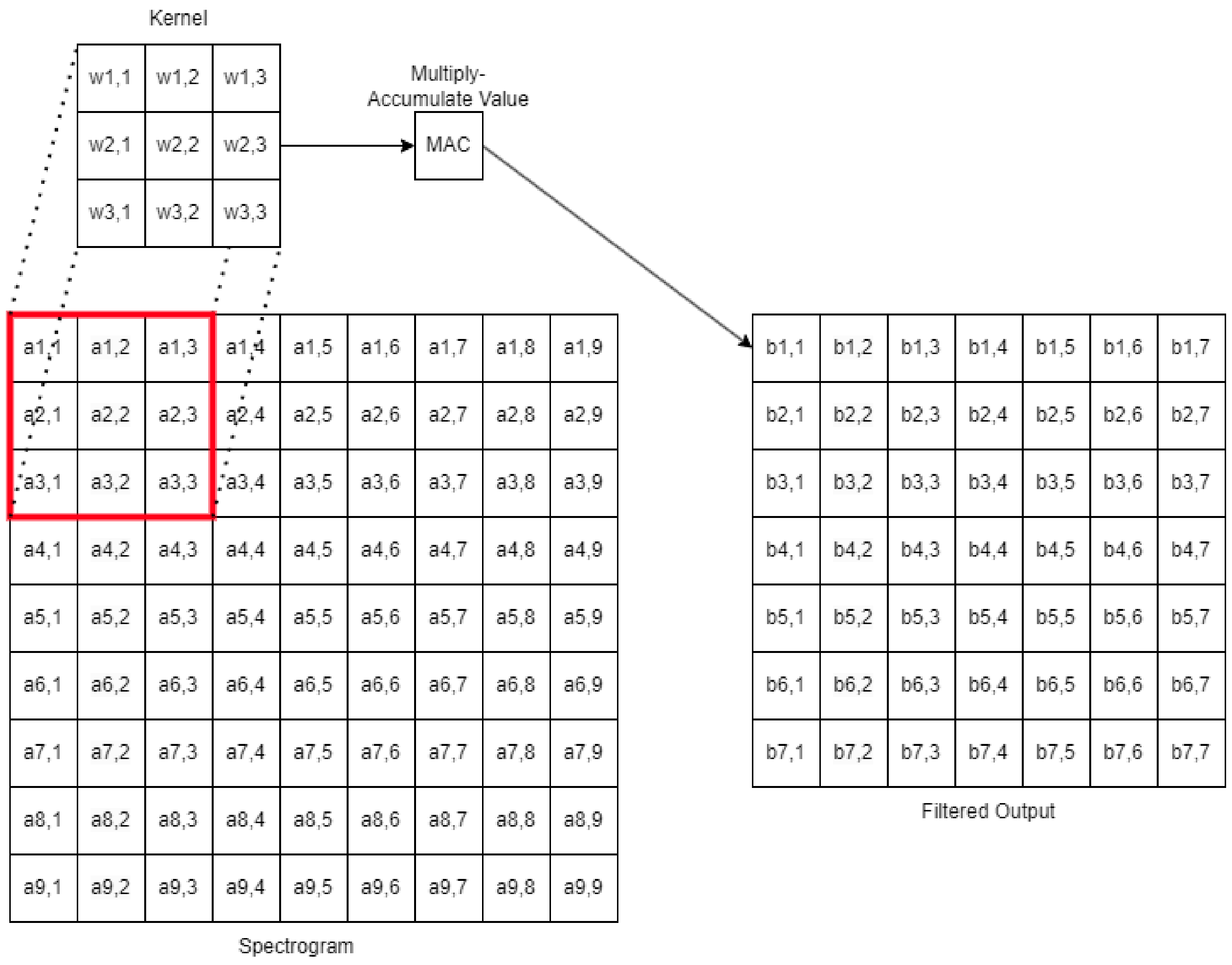


Fig. 14. 2D Convolution Operation Overview

To improve processing efficiency by eliminating unnecessary information, the output from the convolution layer is passed into a max pooling layer instantiated with a 2x2 window. This layer downsamples the input data by taking the maximum value within the window; thus, preserving the most prominent features while minimizing noise. Moreover, this operation yields decreased processing times by reducing the magnitude of the data to be processed. Using Figure 15 as an example, the number of data elements to be processed in the matrix was reduced by 67%. Though simple in purpose, the max pooling layer is vital for the model to operate effectively.

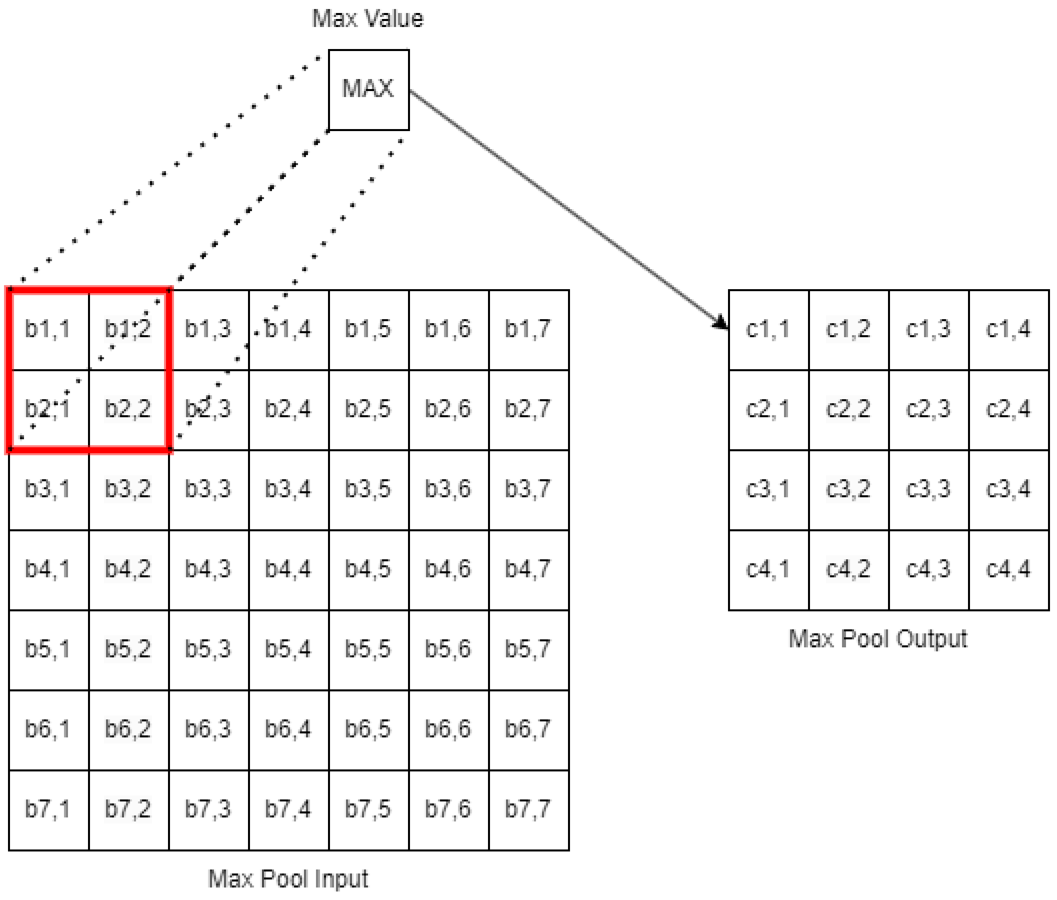

Max Value
MAX
b1,1 b1,2 b1,3 b1,4 b1,5 b1,6 b1,7
b2,1 b2,2 b2,3 b2,4 b2,5 b2,6 b2,7
b3,1 b3,2 b3,3 b3,4 b3,5 b3,6 b3,7
b4,1 b4,2 b4,3 b4,4 b4,5 b4,6 b4,7
b5,1 b5,2 b5,3 b5,4 b5,5 b5,6 b5,7
b6,1 b6,2 b6,3 b6,4 b6,5 b6,6 b6,7
b7,1 b7,2 b7,3 b7,4 b7,5 b7,6 b7,7
Max Pool Input
c1,1 c1,2 c1,3 c1,4
c2,1 c2,2 c2,3 c2,4
c3,1 c3,2 c3,3 c3,4
c4,1 c4,2 c4,3 c4,4
Max Pool Output

Fig. 15. Max Pooling Operation Overview

Moving into the dropout layer, its functionality may seem counterintuitive at first. During the training process, this layer disables information from passing to subsequent layers, Figure 16. For the project, this layer is instantiated with a dropout rate of 50%. For most systems losing information is considered problematic, however, in the learning phase of the model, the algorithm may easily fixate on specific, and potentially fragile, patterns. The problem is commonly referred to as overfitting. By randomly eliminating information with each training cycle, the model must look for stronger, more general patterns that exist in the information, so when presented with novel information, it has a better chance of recognizing a strong pattern.

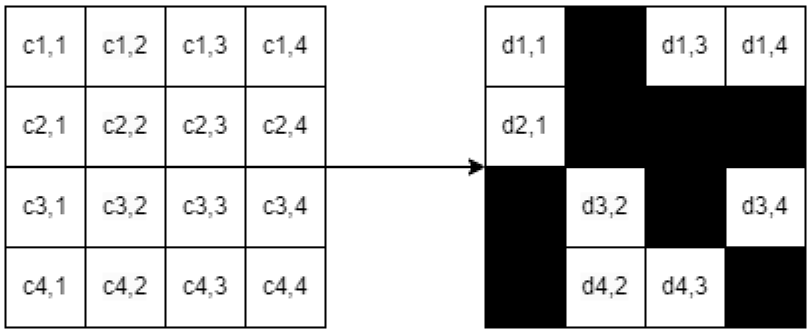


Fig. 16. Dropout Operation Overview

To prepare the detected features for the classification stage, the two-dimensional data must be converted into a one-dimensional container. Using a flatten layer, the matrix is unwrapped into a vector of all the values remaining from the previous dropout layer, Figure 17.

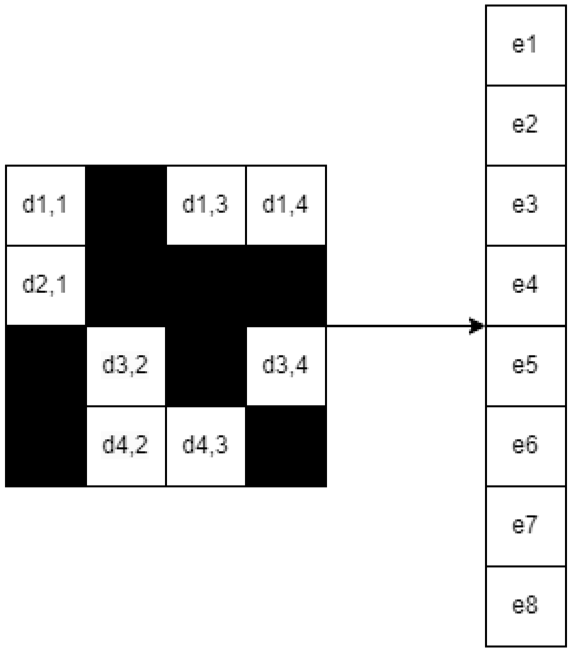


Fig. 17. Flatten Operation Overview

Binary Classifier

The second stage of the model is responsible for the binary classification of the audio files specifying whether the provided speech is synthetic or organic in origin. To perform this classification, the vector from the previous spectral detection stage is received as input and passed directly into a series of dense layers, Figure 18.

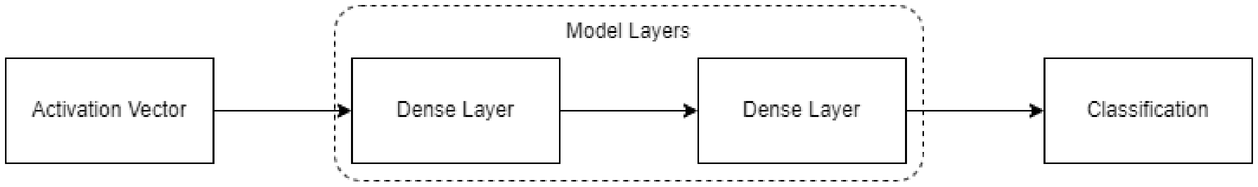


Fig. 18. Binary Classification Model Overview

Commonly used for classification problems, dense layers are simply a vector of neurons which receive input from all neurons of the previous layer, Figure 19. The project leverages two dense layers. The first dense layer is initialized with 128 neurons activated with a rectified linear unit function to determine patterns in the received input, and the second dense layer is initialized with only a single neuron activated by a sigmoid function to provide a binary classification. At each layer, the received value is multiplied by a weight contained by the neuron, passed into the activation function, and forwarded to the next layer of the model until the end of the model is reached. The final output is a value bounded between 0 and 1, where values less than 0.5

correspond to synthetic classifications and values more than 0.5 correspond to organic classification.

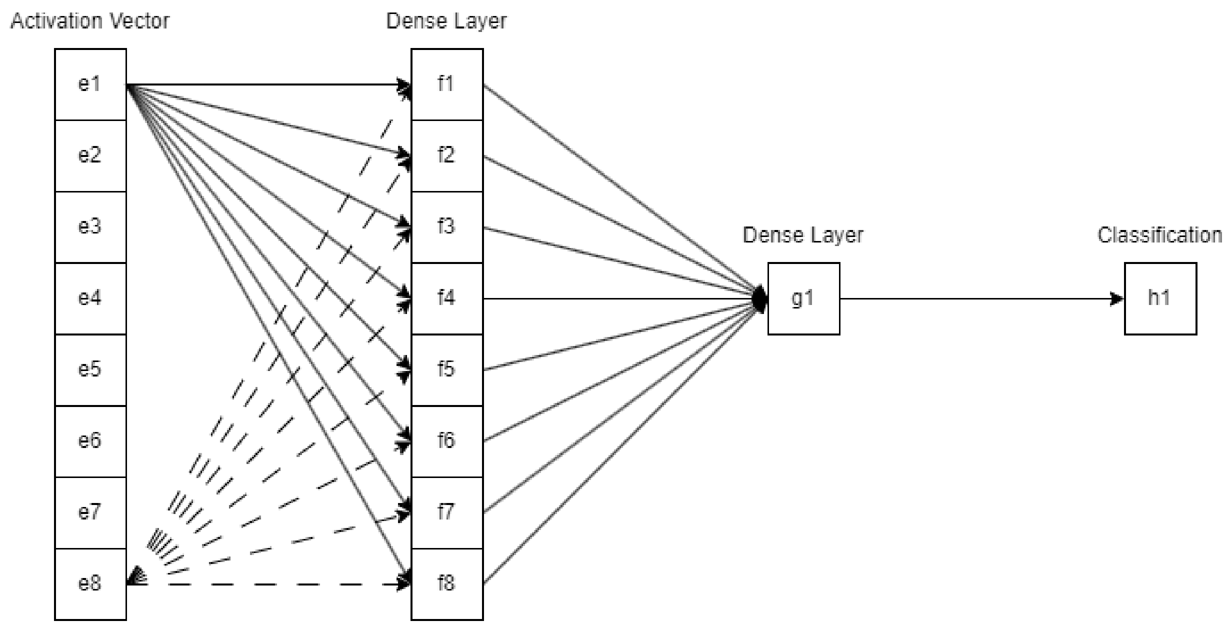


Fig. 19. Dense Layer Overview

Noise Processing

For most of the experimentation performed during the project, noise-related processing played a significant role in altering the supplied datasets. This processing may be split into two main categories: noise generation and noise filtering.

As the name suggests, the noise generation processing involved production of some audio signal that could be combined with the original speech signal to reduce clarity. This noise manifested in two forms: additive white gaussian noise and burst noise.

Additive white gaussian noise (AWGN) is a common noise model used to imitate the noise produced by several naturally occurring processes. The descriptors of the noise signify different characteristics. The noise is additive because it is contributing new information into the signal. Similar to light, “white” refers to the uniformity of the noise’s power spectral density, and gaussian denotes the normal distribution of the noise signal in the time domain.

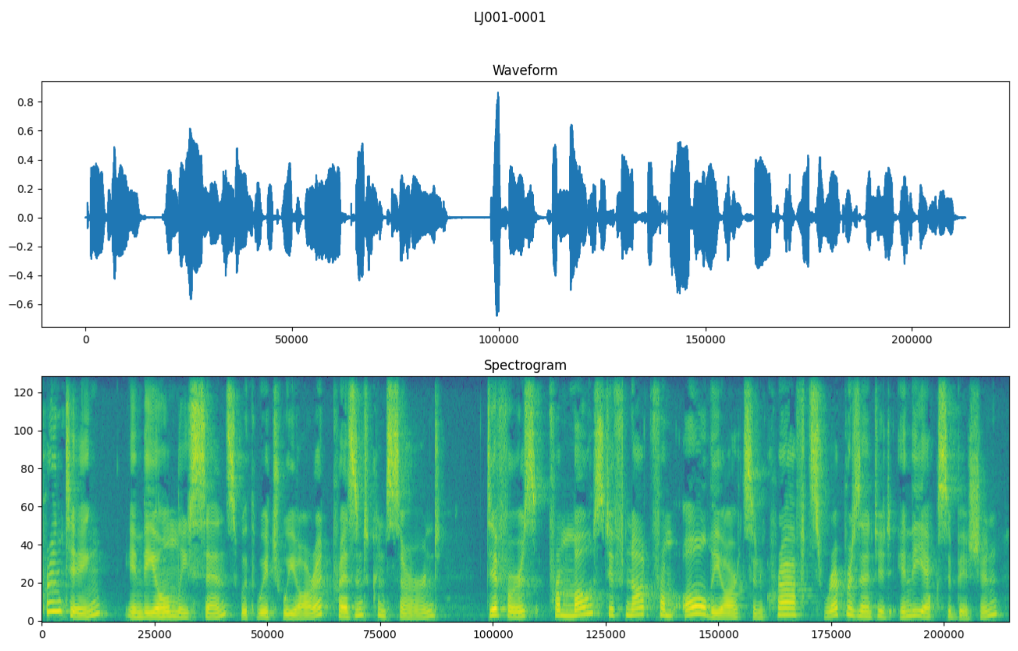


Fig. 20. Nominal Signal Waveform and Spectrogram

To produce this type of noise for the project, a script is used to iterate through a user-specified input directory, where each audio file is read, processed, and written to a user-specified output directory. For each file in the input directory, the absolute value of each sample of the signal is computed, and the result is used to calculate the mean amplitude. Using a user-specified signal-to-noise ratio, the mean amplitude is adjusted to produce a corresponding maximum noise amplitude. A normal distribution bounded by 0 and the maximum noise amplitude is sampled, which is then added to each signal sample. Viewing Figure 20 and 21, the AWGN appears to make the spectrogram of the signal blurrier than the noise-free spectrogram potentially hiding key features.

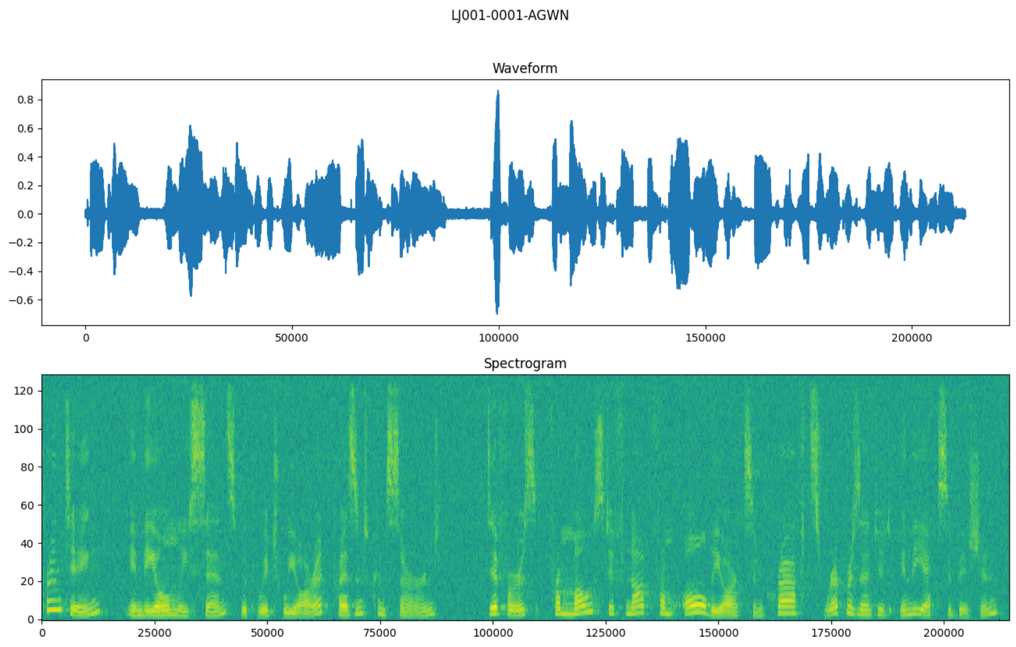


Fig. 21. Additive White Gaussian Noise Signal Waveform and Spectrogram

Burst noise, sometimes referred to as popcorn noise, is an electronic noise type that consists of sudden discrete transitions between signal amplitudes. Similar to AWGN, a similar script with the same input parameters is leveraged to quickly add this noise type to a user-specified dataset. For each file in the input directory, the absolute value of each sample of the signal is computed, and the result is used to calculate the mean amplitude. Using a user-specified signal-to-noise ratio, the mean amplitude is adjusted to produce a corresponding maximum noise amplitude. For each sample, a boolean flag that is initialized to false is evaluated. If it is true, it adds the maximum noise amplitude to the sample. If it is false, it does not alter the sample. Then a random number between 0 and 1 is produced and compared against a constant toggle rate. If the random number is less than the specified threshold, the boolean flag is flipped. Viewing Figure 22, the burst noise appears to add vertical artifacts to the spectrogram that potentially hides key features.

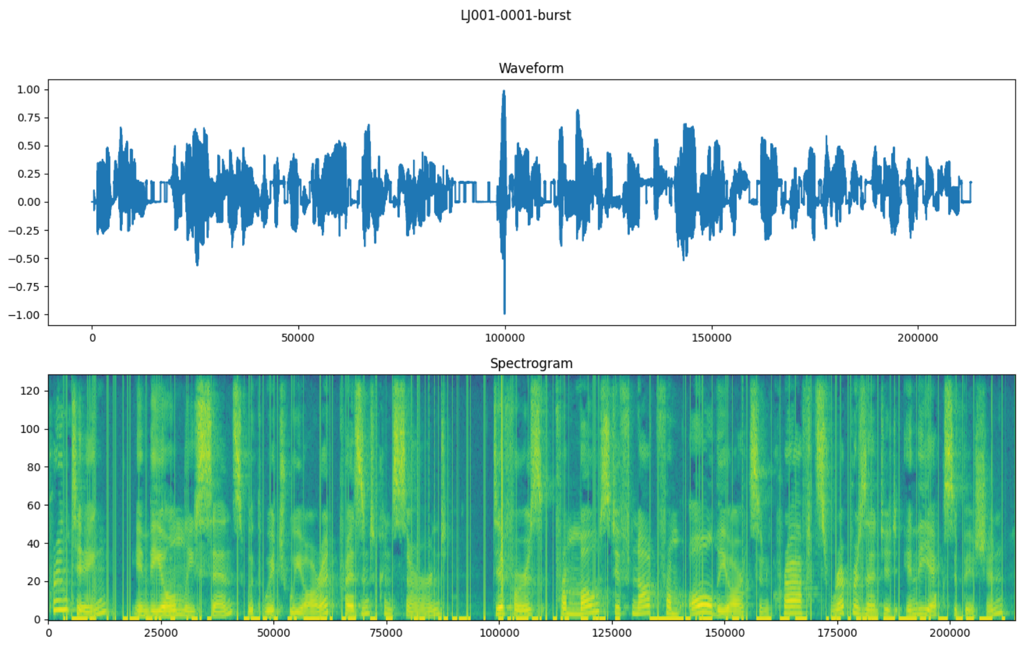


Fig. 22. Burst Noise Signal Waveform and Spectrogram

Once the ability to incorporate noise was established, the team needed a mechanism to filter that noise from the signal to evaluate what performance impacts it may have for the project. Spectral gating is a form of noise gating, where the noise for each gate is limited to some specified threshold. Spectral gating is a variation where these gates depict various frequency bands. To determine these thresholds, a spectrogram of the signal is first computed, and then a time-smoothed version is computed using an infinite impulse response filter applied forward and backward to each frequency channel. Using this time-smoothed spectrogram, a mask is generated and smoothed over frequency and time. The mask is then applied to the signal and inverted. While the produced audio signal sounds clearer to human ears, the produced output signal's spectrogram varies dramatically from the original signal, as seen by comparing Figure 22 to Figure 24.

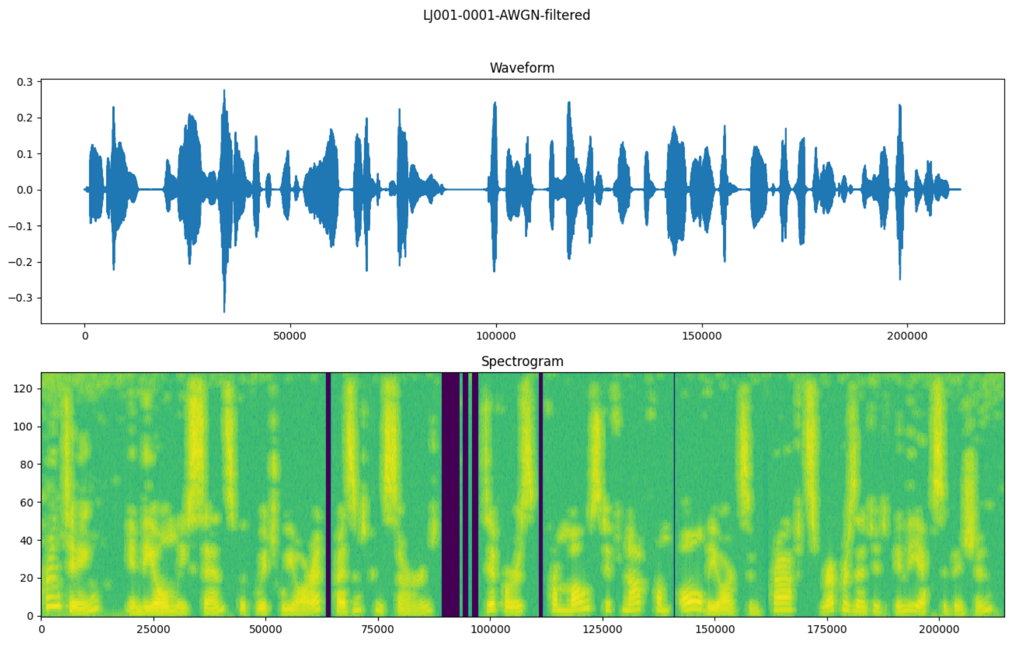


Fig. 23. Additive White Gaussian Noise Filtered Signal Waveform and Spectrogram

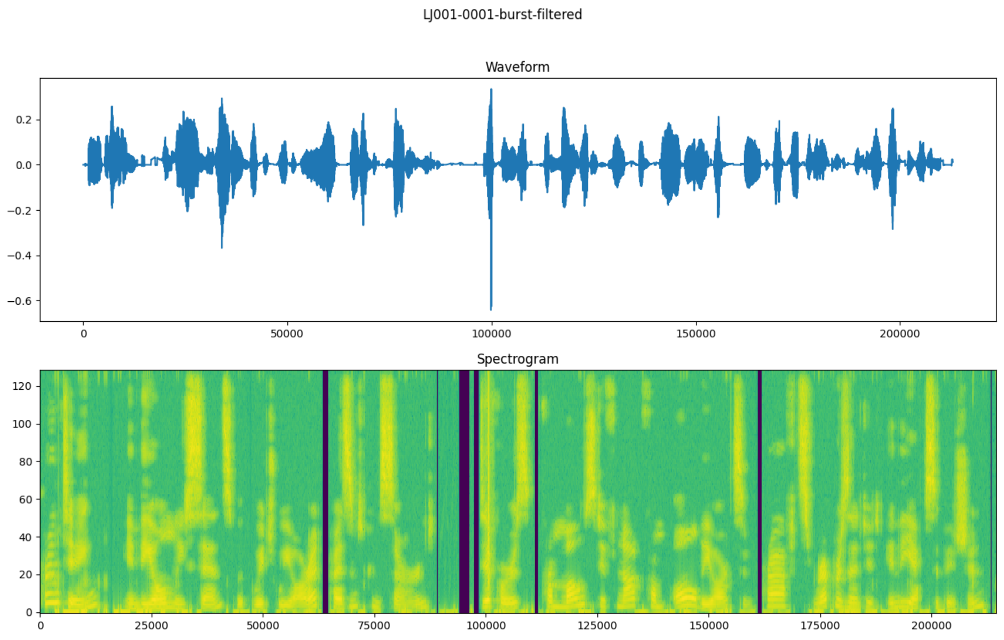


Fig. 24. Burst Noise Filtered Signal Waveform and Spectrogram

Results and Discussion

Table 3. Project Mozart Model Evaluation Results

Scenario	Avg Prediction Accuracy	Equivalent Error Rate	Accuracy Delta
FastSpeech2			
Baseline	75.0000	11.3281	0.0000
White Noise - No Training	73.1818	12.2656	-1.8182
White Noise - Training	75.6818	10.9766	0.6818
White Noise - Filtered	68.7879	17.1094	-6.2121
Burst Noise - No Training	63.7879	17.1094	-11.2121
Burst Noise - Training	70.2273	13.7891	-4.7727
Burst Noise - Filtered	66.2879	15.8203	-8.7121
iOS Personal Voice			
Baseline	96.4286	0.0000	0.0000
White Noise - No Training	92.9464	1.8056	-3.4822
White Noise - Training	96.4286	0.0000	0.0000
White Noise - Filtered	96.0715	0.1852	-0.3571
Burst Noise - No Training	80.9822	8.0093	-15.4464
Burst Noise - Training	96.4286	0.0000	0.0000
Burst Noise - Filtered	94.5536	0.9722	-1.8750
PlayHT			
Baseline	81.8750	9.0605	0.0000
White Noise - No Training	76.2480	11.8745	-5.6270
White Noise - Training	78.1250	10.9375	-3.7500
White Noise - Filtered	69.3750	15.3115	-12.5000
Burst Noise - No Training	71.8750	14.0625	-10.0000
Burst Noise - Training	75.6250	12.1875	-6.2500
Burst Noise - Filtered	71.8500	14.0625	-10.0250
Combined (FastSpeech2, Personal Voice, PlayHT)			
Baseline	84.9615	6.8555	0.0000
White Noise - No Training	83.9231	7.3828	-1.0384
White Noise - Training	85.5000	6.5820	0.5385
White Noise - Filtered	81.2308	8.7500	-3.7307
Burst Noise - No Training	82.3846	8.1641	-2.5769
Burst Noise - Training	80.5000	9.1211	-4.4615
Burst Noise - Filtered	78.8846	9.9414	-6.0769

When evaluating the performance of the detection model, four groupings of data were constructed: one for each generative algorithm and one that combined all of the generative algorithms. For each grouping, several scenarios were constructed:

1. A baseline trained on nominal data and predicted against nominal data
2. A model trained on nominal data and predicted against white noise data
3. A model trained on white noise data and predicted against white noise data
4. A model trained on nominal data and predicted against filtered white noise data
5. A model trained on nominal data and predicted against burst noise data
6. A model trained on burst noise data and predicted against burst noise data
7. A model trained on nominal data and predicted against filtered burst noise data

These scenarios were chosen to determine the impact of different noise models on the prediction accuracy of the project's detection model and how it may be mitigated. Each scenario was executed for 40 runs, where the average prediction accuracy and equivalent error rate are demonstrated in Table 3.

When viewing the results from Table 3, a few observations may be made. When comparing the baseline performance for each algorithm, the model performs best against Apple's Personal Voice, followed by PlayHT, and then by FastSpeech2. When comparing noise, burst noise appears to impact the prediction accuracy more severely than white noise for an equivalent SNR. This may be mitigated by introducing the noise into the training set, and in general, filtering the noise before predictions reduces accuracy. When comparing the combined group to its individual counterparts, the baseline prediction accuracy is nearly equivalent to the averaged individual baselines (84.4345%); however, the impact of noise is not as extreme, such that the accuracy deltas are on average lower in magnitude. Though this may not be true with additional test cycles, the combined group performed best when trained with white noise and predicted against white noise.

When comparing the results of this experiment to DeepSonar, an alternative detection method, the combined baseline results for this project's prediction model are 13.1385% lower; however, DeepSonar was very susceptible to noise, such that prediction accuracy would be as low as 73% when it was present. While this project's nominal prediction accuracy is lower than DeepSonar's, its noisy prediction accuracy is nearly 11% greater on average, and the worst accuracy loss when noise is present is nearly 2.5% rather than DeepSonar's 25%. Under ideal conditions DeepSonar certainly has the ability to perform better, but this project's detection model performs most consistently.

Conclusion

Reviewing the objectives set for the project, the team was able to successfully achieve the initial plan. The combined group detection accuracy of 84.9615% and EER of 9.0605 exceed the desired minimum threshold of 80% detection accuracy. Additionally, the model was fairly resilient to noise interference; when compared to a modern counterpart, the performance loss was reduced by a factor of 10. And last, these results were collected against modern, publicly available generative algorithms providing contemporary data points for audio deep fakes. However successful in the scope of the semester, the project is far from solving the issues presented by audio deep fakes. If given more time, objectives improving the detection accuracy, adding the ability to classify algorithms, and increasing the productization of the model would outline the next steps for the project moving forward.

Appendix

install.sh

```
pip install -r requirements.txt
```

1_Loading_Data.sh

```
#!/bin/bash

# Workflow 1 - Loading Data: Test for loading a dataset from a directory
structure
echo "=== Workflow 1 - Loading Data: Test for loading a dataset from a
directory structure ==="

dataset_path='/workspaces/data_sets'

# Fit a new model
python /workspaces/src/load_data.py $dataset_path
```

2_Fitting_Model.sh

```
#!/bin/bash

# Workflow 2 - Fitting Model: Normal training of the model with the default
dataset
echo "=== Workflow 2 - Fitting Model: Normal training of the model with the
default dataset ==="

dataset_path='/workspaces/data_sets'
trained_model_path='/workspaces/trained_model'

# Fit a new model
python /workspaces/src/fit_model.py $dataset_path -tmp $trained_model_path
```

3_Load_Saved_Model.sh

```
#!/bin/bash

# Workflow 3 - Load Saved Model: Loading a model that has been saved to file
echo "=== Workflow 3 - Load Saved Model: Loading a model that has been saved
to file ==="

trained_model_path='/workspaces/trained_model'

# Load saved model
python /workspaces/src/load_saved_model.py $trained_model_path
```

4_Making_Predictions.sh

```
#!/bin/bash

# Workflow 4: Prediction of the dataset with existing model
echo "=== Workflow 4: Prediction of the dataset with existing model ==="

dataset_path='/workspaces/data_sets'
trained_model_path='/workspaces/trained_model'
prediction_data_path='/workspaces/prediction_data'

# Make predictions
python /workspaces/src/make_predictions.py $dataset_path -tmp
$trained_model_path -pdp $prediction_data_path
```

5_Training_And_Predictions.sh

```
#!/bin/bash

# Workflow 5 - Making Predictions w/ Data Split: Loads data, creates and
trains model, makes predictions with the initial data split
echo "=== Workflow 5 - Making Predictions w/ Data Split: Loads data, creates
and trains model, makes predictions with the initial data split ==="

# Load saved model
python /workspaces/src/__main__.py
```

6_Adding_White_Noise.sh

```
#!/bin/bash

# Workflow 6 - Adding White Noise: Create white noise datasets
echo "=== Workflow 6 - Adding White Noise: Create white noise datasets ==="

root_noise_dir='/workspaces/white_noise_data_sets'
if [ -d $root_noise_dir ]; then
    rm -r $root_noise_dir
fi
mkdir $root_noise_dir

training_dataset_path='/workspaces/data_sets'
real_dataset_path='/workspaces/data_sets/real'
real_white_noise_dataset_path='/workspaces/white_noise_data_sets/real'
fake_dataset_path='/workspaces/data_sets/fake'
```

```
fake_white_noise_dataset_path='/workspaces/white_noise_data_sets/fake'

# Add white noise to real data
python /workspaces/src/make_noise_dataset.py $real_dataset_path
$real_white_noise_dataset_path --snr 15 --type white

# Add white noise to fake data
python /workspaces/src/make_noise_dataset.py $fake_dataset_path
$fake_white_noise_dataset_path --snr 15 --type white
```

7_Adding_Burst_Noise.sh

```
#!/bin/bash

# Workflow 7 - Adding Burst Noise: Create burst noise datasets
echo "=== Workflow 7 - Adding Burst Noise: Create burst noise datasets ==="

root_noise_dir='/workspaces/burst_noise_data_sets'
if [ -d $root_noise_dir ]; then
    rm -r $root_noise_dir
fi
mkdir $root_noise_dir

training_dataset_path='/workspaces/data_sets'
real_dataset_path='/workspaces/data_sets/real'
real_burst_noise_dataset_path='/workspaces/burst_noise_data_sets/real'
fake_dataset_path='/workspaces/data_sets/fake'
fake_burst_noise_dataset_path='/workspaces/burst_noise_data_sets/fake'

# Add burst noise to real data
python /workspaces/src/make_noise_dataset.py $real_dataset_path
$real_burst_noise_dataset_path --snr 15 --type burst

# Add burst noise to fake data
python /workspaces/src/make_noise_dataset.py $fake_dataset_path
$fake_burst_noise_dataset_path --snr 15 --type burst
```

8_Noise_Filtering.sh

```
#!/bin/bash

# Workflow 8 - Noise Filtering: Filtering the white noise dataset
echo "=== Workflow 8 - Noise Filtering: Filtering the white noise dataset ==="


real_white_noise_dataset_path='/workspaces/white_noise_data_sets/real'
fake_white_noise_dataset_path='/workspaces/white_noise_data_sets/fake'


root_filtered_noise_dir='/workspaces/filtered_noise_data_sets'
if [ -d $root_filtered_noise_dir ]; then
```

```
    rm -r $root_filtered_noise_dir
fi
mkdir $root_filtered_noise_dir

real_filtered_noise_dir='/workspaces/filtered_noise_data_sets/real'
if [ -d $real_filtered_noise_dir ]; then
    rm -r $real_filtered_noise_dir
fi
mkdir $real_filtered_noise_dir

fake_filtered_noise_dir='/workspaces/filtered_noise_data_sets/fake'
if [ -d $fake_filtered_noise_dir ]; then
    rm -r $fake_filtered_noise_dir
fi
mkdir $fake_filtered_noise_dir

# Filter noise for real data
python /workspaces/src/filter_noise_dataset.py $real_white_noise_dataset_path
$real_filtered_noise_dir

# Filter noise for fake data
python /workspaces/src/filter_noise_dataset.py $fake_white_noise_dataset_path
$fake_filtered_noise_dir
```

9_Collect_Metrics.sh

```
#!/bin/bash

# Workflow 9 - Collect Metrics: Evaluate the model against datasets
echo "=== Workflow 9 - Collect Metrics: Evaluate the model against datasets
==="
default=1
count=${1:-$default}

dataset_path='/workspaces/data_sets'
white_noise_dataset_path='/workspaces/white_noise_data_sets'
burst_noise_dataset_path='/workspaces/burst_noise_data_sets'
filtered_noise_dataset_path='/workspaces/filtered_noise_data_sets'

trained_model_path='/workspaces/trained_model'
prediction_data_path='/workspaces/prediction_data'

metrics_root_path='/workspaces/metrics'

if [ -d $metrics_root_path ]; then
    rm -r $metrics_root_path
fi
mkdir $metrics_root_path
mkdir $metrics_root_path/baseline
mkdir $metrics_root_path/white_noise_train
mkdir $metrics_root_path/burst_noise_train
mkdir $metrics_root_path/white_noise_no_train
mkdir $metrics_root_path/burst_noise_no_train
mkdir $metrics_root_path/filtered_noise

for i in $(seq 1 $count)
do
    python /workspaces/src/fit_model.py $dataset_path -tmp $trained_model_path
    python /workspaces/src/make_predictions.py $dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/baseline/metric_$i.txt
    python /workspaces/src/make_predictions.py $white_noise_dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/white_noise_no_train/metric_$i.txt
    python /workspaces/src/make_predictions.py $burst_noise_dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/burst_noise_no_train/metric_$i.txt
    python /workspaces/src/make_predictions.py $filtered_noise_dataset_path
-tmp $trained_model_path -pdp $prediction_data_path >
$metrics_root_path/filtered_noise/metric_$i.txt
    python /workspaces/src/fit_model.py $white_noise_dataset_path -tmp
$trained_model_path
    python /workspaces/src/make_predictions.py $white_noise_dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/white_noise_train/metric_$i.txt
    python /workspaces/src/fit_model.py $burst_noise_dataset_path -tmp
$trained_model_path
    python /workspaces/src/make_predictions.py $burst_noise_dataset_path -tmp
$trained_model_path -pdp $prediction_data_path >
$metrics_root_path/burst_noise_train/metric_$i.txt
done

echo 'Baseline Training and Prediction:'
python /workspaces/src/parse_metrics.py $metrics_root_path/baseline
echo '\nBaseline Training and White Noise Prediction:'
python /workspaces/src/parse_metrics.py
$metrics_root_path/white_noise_no_train
echo '\nBaseline Training and Burst Noise Prediction:'
python /workspaces/src/parse_metrics.py
$metrics_root_path/burst_noise_no_train
echo '\nBaseline Training and Filtered Noise Prediction:'
python /workspaces/src/parse_metrics.py $metrics_root_path/filtered_noise
echo '\nWhite Noise Training and Prediction:'
python /workspaces/src/parse_metrics.py $metrics_root_path/white_noise_train
echo '\nBurst Noise Training and Prediction:'
python /workspaces/src/parse_metrics.py $metrics_root_path/burst_noise_train
```

Citations

1. Sakoe, H., and S. Chiba. “Dynamic programming algorithm optimization for spoken word recognition.” IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 26, no. 1, 1978, pp. 43–49, https://doi.org/10.1109/tassp.1978.1163055.
2. Rabiner, L.R. “A tutorial on Hidden Markov models and selected applications in speech recognition.” *Proceedings of the IEEE*, vol. 77, no. 2, 1989, pp. 257–286, https://doi.org/10.1109/5.18626.
3. Bhan, Sarthak. “What Is Dynamic Time Warping?” *Medium*, MLearning.ai, 18 July 2022, medium.com/mlearning-ai/what-is-dynamic-time-warping-253a6880ad12.
4. Almutairi, Zaynab, and Hebah Elgibreen. “A review of modern audio deepfake detection methods: Challenges and future directions.” Algorithms, vol. 15, no. 5, 2022, p. 155, https://doi.org/10.3390/a15050155.
5. Frank, Joel, and Lea Schönherr. “WaveFake: A Data Set to Facilitate Audio Deepfake Detection.” *CoRR*, abs/2111.02813, 2021, https://doi.org/https://doi.org/10.48550/arXiv.2111.02813.
6. Wang, Run, et al. “DeepSonar: Towards Effective and Robust Detection of AI-Synthesized Fake Voices.” *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, https://doi.org/https://doi.org/10.48550/arXiv.2005.13770.
7. Dimitrakakis, Christos & Bengio, Samy. (2011). Phoneme and Sentence-Level Ensembles for Speech Recognition. EURASIP J. Audio, Speech and Music Processing. 2011. 10.1155/2011/426792.
8. “Using Deep Learning Models / Convolutional Neural Networks.” *Using Deep Learning Models / Convolutional Neural Networks*, 2023, docs.ecognition.com/eCognition_documentation/User%20Guide%20Developer/8%20Classification%20-%20Deep%20Learning.htm.
9. Mishra, Vidushi & AGARWAL, SMT & PURI, NEHA. (2018). COMPREHENSIVE AND COMPARATIVE ANALYSIS OF NEURAL NETWORK. INTERNATIONAL JOURNAL OF COMPUTER APPLICATION. 2. 10.26808/rs.ca.i8v2.15.
10. Saeed, Mehreen. “An Introduction to Recurrent Neural Networks and the Math That Powers Them.” *MachineLearningMastery.Com*, Guiding Tech Media, 5 Jan. 2023, machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/.
11. K, Bharath. “Introduction to Deep Neural Networks.” *DataCamp*, DataCamp, 4 July 2023, www.datacamp.com/tutorial/introduction-to-deep-neural-networks.
12. Yu, H.; Tan, Z.-H.; Ma, Z.; Martin, R.; Guo, J. Guo spoofing detection in automatic speaker verification systems using DNN classifiers and dynamic acoustic features. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 4633–4644. [**Google Scholar**] [**CrossRef**][**Green Version**]
13. Alzantot, M.; Wang, Z.; Srivastava, M.B. Deep residual neural networks for audio spoofing detection. *arXiv CoRR* **2019**, arXiv:1907.00501. [**Google Scholar**]
14. Lai, C.-I.; Chen, N.; Villalba, J.; Dehak, N. ASSERT: Anti-spoofing with squeeze-excitation and residual networks. *arXiv* **2019**, arXiv:1904.01120. [**Google Scholar**]
15. Aravind, P.R.; Nechiyil, U.; Paramparambath, N. Audio spoofing verification using deep convolutional neural networks by transfer learning. *arXiv* **2020**, arXiv:2008.03464. [**Google Scholar**]
16. Lataifeh, M.; Elnagar, A.; Shahin, I.; Nassif, A.B. Arabic audio clips: Identification and discrimination of authentic cantillations from imitations. *Neurocomputing* **2020**, *418*, 162–177. [**Google Scholar**] [**CrossRef**]
17. Rodríguez-Ortega, Y.; Ballesteros, D.M.; Renza, D. A machine learning model to detect fake voice. In *Applied Informatics*; Florez, H., Misra, S., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 3–13. [**Google Scholar**]
18. Wang, R.; Juefei-Xu, F.; Huang, Y.; Guo, Q.; Xie, X.; Ma, L.; Liu, Y. Deepsonar: Towards effective and robust detection of ai-synthesized fake voices. In Proceedings of the the 28th ACM International

Conference on Multimedia, Seattle, WA, USA, 12–16 October 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1207–1216. [**Google Scholar**]

19. Subramani, N.; Rao, D. Learning efficient representations for fake speech detection. In Proceedings of the The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, 7–12 February 2020; pp. 5859–5866. [**Google Scholar**]
20. Shan, M.; Tsai, T. A cross-verification approach for protecting world leaders from fake and tampered audio. *arXiv* **2020**, arXiv:2010.12173. [**Google Scholar**]
21. Wijethunga, R.L.M.A.P.C.; Matheesha, D.M.K.; Al Noman, A.; De Silva, K.H.V.T.A.; Tissera, M.; Rupasinghe, L. Rupasinghe deepfake audio detection: A deep learning based solution for group conversations. In Proceedings of the 2020 2nd International Conference on Advancements in Computing (ICAC), Malabe, Sri Lanka, 10–11 December 2020; Volume 1, pp. 192–197. [**Google Scholar**]
22. Jiang, Z.; Zhu, H.; Peng, L.; Ding, W.; Ren, Y. Self-supervised spoofing audio detection scheme. In Proceedings of the INTERSPEECH 2020, Shanghai, China, 25–29 October 2020; pp. 4223–4227. [**Google Scholar**]
23. Chintha, A.; Thai, B.; Sohrawardi, S.J.; Bhatt, K.M.; Hickerson, A.; Wright, M.; Ptucha, R. Ptucha recurrent convolutional structures for audio spoof and video deepfake detection. *IEEE J. Sel. Top. Signal. Process.* **2020**, *14*, 1024–1037. [**Google Scholar**] [**CrossRef**]
24. Singh, A.K.; Singh, P. Detection of ai-synthesized speech using cepstral & bispectral statistics. In Proceedings of the 2021 IEEE 4th International Conference on Multimedia Information Processing and Retrieval (MIPR), Tokyo, Japan, 8–10 September 2021; pp. 412–417. [**Google Scholar**]
25. Lei, Z.; Yang, Y.; Liu, C.; Ye, J. Siamese convolutional neural network using gaussian probability feature for spoofing speech detection. In Proceedings of the INTERSPEECH, Shanghai, China, 25–29 October 2020; pp. 1116–1120. [**Google Scholar**]
26. Ballesteros, D.M.; Rodriguez-Ortega, Y.; Renza, D.; Arce, G. Deep4SNet: Deep learning for fake speech classification. *Expert Syst. Appl.* **2021**, *184*, 115465. [**Google Scholar**] [**CrossRef**]
27. Bartusiak, E.R.; Delp, E.J. Frequency domain-based detection of generated audio. In Proceedings of the Electronic Imaging; Society for Imaging Science and Technology, New York, NY, USA, 11–15 January 2021; Volume 2021, pp. 273–281. [**Google Scholar**]
28. Borrelli, C.; Bestagini, P.; Antonacci, F.; Sarti, A.; Tubaro, S. Synthetic speech detection through short-term and long-term prediction traces. *EURASIP J. Inf. Secur.* **2021**, *2021*, 2. [**Google Scholar**] [**CrossRef**]
29. Khalid, H.; Kim, M.; Tariq, S.; Woo, S.S. Evaluation of an audio-video multimodal deepfake dataset using unimodal and multimodal detectors. In Proceedings of the 1st Workshop on Synthetic Multimedia, ACM Association for Computing Machinery, New York, NY, USA, 20 October 2021; pp. 7–15. [**Google Scholar**]
30. Khochare, J.; Joshi, C.; Yenarkar, B.; Suratkar, S.; Kazi, F. A deep learning framework for audio deepfake detection. *Arab. J. Sci. Eng.* **2021**, *47*, 3447–3458. [**Google Scholar**] [**CrossRef**]
31. Liu, T.; Yan, D.; Wang, R.; Yan, N.; Chen, G. Identification of fake stereo audio using SVM and CNN. *Information* **2021**, *12*, 263. [**Google Scholar**] [**CrossRef**]
32. Camacho, S.; Ballesteros, D.M.; Renza, D. Fake speech recognition using deep learning. In *Applied Computer Sciences in Engineering*; Figueroa-García, J.C., Díaz-Gutierrez, Y., Gaona-García, E.E., Orjuela-Cañón, A.D., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 38–48. [**Google Scholar**]

33. Arif, T.; Javed, A.; Alhameed, M.; Jeribi, F.; Tahir, A. Voice spoofing countermeasure for logical access attacks detection. *IEEE Access* **2021**, *9*, 162857–162868. [**Google Scholar**] [**CrossRef**]

34. Dennis, Jonathan & Dat, Tran & Li, Haizhou. (2011). Spectrogram Image Feature for Sound Event Classification in Mismatched Conditions. Signal Processing Letters, IEEE. 18. 130 - 133. 10.1109/LSP.2010.2100380.