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

Contributing Authors: Branch Hill and Jacob Pomatto

Final Version 1.0.0, Dec 7, 2023

Abstract–Generative artificial intelligence (AI) is becoming more prevalent and is easily accessible to anyone that has a basic understanding of modern technology. Even though it can be used for things like increasing productivity, it also has the potential to cause great harm. As this technology improves and becomes more popular, it's important that we have tools that can accurately detect synthetic speech and video, such as deep fakes. The term deep fakes refers to audio or video that has been generated or manipulated to sound realistic using machine learning technology. Some AI models have the ability to learn a person's mannerisms and voice, which makes it possible for a user to create fake recordings. This technology is already being misused for impersonation attempts and other devious acts. In this research, we focus on creating a neural network for detecting synthetic speech with acceptable accuracy. We also present some possible solutions for mitigating the issues posed by the presence of noise in the data. Our results indicate that training the model with various types of data is more effective than trying to remove the variation during pre-processing.

Keywords–Audio, Speech, TensorFlow, Keras, Artificial Intelligence (AI), Machine Learning, Deep Fakes

I. INTRODUCTION

As artificial intelligence and machine learning are becoming more popular, the amount of research related to these topics is increasing in certain domains. Topics like deep fake generation [9] and deep fake image detection [7,8] have received a lot of attention in this area. Other areas of research such as synthetic speech generation and detecting fake speech have not been widely researched.

Speech generation software can be used for great things like language learning applications, navigation systems, and text-to-speech (TTS) technology; however, it also presents major security threats. In the last year, there have been an increasing number of fraudulent attempts involving deep fakes with reports of a 3000% increase from 2022 to 2023 [10]. The significant increase in deep fakes attempts is shown in Figure I. As this software continues to improve, it is critical to develop systems that can accurately detect the presence of synthetic speech.

When our team started this research project, our goal was to develop a system that could classify fake and real speech with an acceptable accuracy of 80% or higher. This was attainable by leveraging a few popular machine learning frameworks like TensorFlow and training the model with our default dataset. When training a model, the full dataset is typically split into three partitions - training data, validation data, and test data, as shown in Figure II [14]. Training data is used to fit the model and it should be classified by the model with a high degree of accuracy after training is complete. Validation data is used to test the model during training and is used to determine when parameters should be adjusted. Validation data accuracy is a good indication of how the model will perform with test data. Test data is held for verification of the model after the completion of training. In the beginning of our research, all three partitions came from the default dataset and were not modified in any way; therefore, there was not much variation between each partition. We soon realized that we could easily deceive the model by making minor changes to the test dataset. One of the ways we did this was by adding several types of noise to the audio files. This posed a major issue as the detection accuracy of our model was decreased significantly by a very simple change in the data. Testing models in the presence of noise also seemed to be an area of research that was significantly lacking. While experimenting with how to deal with added noise, we explored two different paths - filtering noise and training with noise. Our main contribution is offering several possible solutions to maintaining a high detection accuracy when noise is added to test data.

II. 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. 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. While template matching with DTW may yield accurate results, the matching template set must be pre-defined. State-based statistical models give a more general classification algorithm, but without training, they also require significant manual configuration [2]. Ultimately, these approaches appeared to be brute force solutions, and the team decided to pursue other implementations.

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 [3] documented the current state of audio deep fake research. The paper outlines three main categories of attacks: imitation-based, synthetic-based, and replay-based AD attacks. To combat these attacks, the paper goes on to describe several detection methods and reveals that deep neural networks are a common pattern amongst the implementations to execute the detection mechanism. While the methods described performed the audio deep fake detection with a high degree of accuracy under ideal conditions, real world noise negatively impacted detection rates of the models. 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 [3].

DeepSonar, one of the detection methods reviewed by Almutair et al., uses a deep neural network composed of a convolution neural network (CNN) that feeds into a recurrent neural network (RNN) to perform the detection [4]. With this combination, DeepSonar can achieve an average accuracy of 98.1% with a false alarm rate of about 2% under ideal conditions [4]. 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% [4].

III. TECHNOLOGY

Training and deployment of neural networks have become central to the progress of Machine Learning, and TensorFlow has emerged as a widely used technology in this area. This framework has been used to improve popular applications such as Airbnb, Google, PayPal, and Spotify [11]. Because of the results and popularity of this framework, our team decided to use TensorFlow to develop the detection model.

The TensorFlow framework was developed by Google Brain and is one of the most popular frameworks for numerical computation, large-scale machine learning, deep learning, and other predictive analytics workloads [12]. It allows models to be designed with a series of layers that send and receive data. TensorFlow also supports parallel processing and GPU usage for model training and predictions. TensorFlow contains several interfaces that assist with creating and processing datasets. This framework is also tightly integrated with Keras which is a high-level neural network API that was adopted by TensorFlow and supports the ability to build convolutional and recurrent neural networks.

IV. DATA

A. *WaveFake*

The dataset used for this project was created in the WaveFake project, which was also related to synthetic speech detection and research. The main contribution of this research was the creation of a novel dataset across several different synthesis architectures as shown in figure III [5].

This dataset consists of over 115,000 generated speech files that were created using 6 different synthesis architectures. This data was generated from the LJSpeech dataset, which is a commonly used set of over 13,000 speech files [6]. This set was also used as reference data when training the model.

The WaveFake generated data that we used throughout this project was created using the Hifi-GAN architecture. Hifi-GAN audio is synthesized by inverting the spectrogram of real speech as shown in Figure IV [13].

B. *Data Architecture*

To properly train the model for classifying audio files, the data must be placed in the correct folder structure. We created a root dataset directory with two subdirectories for real and fake data. All the reference data from the LJSpeech dataset was placed in the real directory. Hifi-GAN generated files from the WaveFake dataset were placed in the fake directory along with some of the modified data that we added later in the project. The structure of the dataset is shown in Figure IV. When a dataset is loaded by TensorFlow, it utilizes the Keras API to give each batch of data a class name. By using this structure, TensorFlow will automatically create classes named “real” and “fake” based on the names of the subdirectories.

V. DATA PRE-PROCESSING

While most data processing occurs in the machine learning model, there are two vital pre-processing procedures that must be executed on the dataset before passing the information to the detection model.

The first procedure is separating the loaded dataset into three partitions: training, validation, and testing. When loading the dataset into the software, 80% of the data is allocated to the training partition, and the remaining 20% is loaded into a temporary partition. This temporary partition is then sharded equally into the validation and testing partitions. This ultimately yields an 80-10-10 split for the training, validation, and testing partitions, respectively.

The second procedure is computing the linear frequency spectrograms of each audio signal for each partition. Leveraging a Short-Time Fourier Transform (STFT) with frame length of 255 and frame step of 128, a raw spectrogram representing the key frequencies of the signal is produced for each element of the dataset. The absolute value is then taken to highlight the magnitude of the signal rather than the polarity, and a ‘channels’ dimension is appended to the data element to mimic an image-like format. Once computed, the spectrograms are cached to accelerate subsequent training epochs.

VI. MODEL DESIGN

A. *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. The design overview of the project is shown in figure V.

B. *Data Organization*

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 VI. 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.

Following the rules enforced by Keras, the dataset for Project Mozart was constructed as seen in Figure IV. 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.

C. *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 VII.

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 VIII 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.

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 IX 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.

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 X. 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.

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 XII.

D. 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 XII.

Commonly used for classification problems, dense layers are simply a vector of neurons which receive input from all neurons of the previous layer, Figure XIII. 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.

E. Interpreting Model Output and Confidence Levels

The output of the model can be interpreted as a confidence interval of the predictions. As stated previously, the prediction output will be a value between 0 and 1. A value very close to 0 or 1 would indicate that the model is very confident in its prediction, whereas outputs closer to 0.5 represent a lower confidence level. Interpretations of the model output are shown in Table I.

VII. Fitting the Model

A. Training and Optimization

Fitting refers to the process of creating an instance of the designed model and training with the specified dataset. Periodically, the model is evaluated using validation data and the parameters are adjusted based on an optimization algorithm.

For this project, we utilized the Adam optimization algorithm for training. Adam stands for Adaptive Moment Estimation and is commonly used for training neural networks because of its ability to increase training speed [16], correct biases in the model [15], and improve the generalization of the model [15].

B. Early Stopping

The model training consists of a series of epochs in which the model makes a pass at the complete training dataset. After each epoch is complete, the parameters of the model are adjusted to correct biases and errors in the predictions. It is important to make several iterations to achieve optimal accuracy; however, overtraining the model can have negative impacts on performance.

In an attempt to solve this issue, we implemented an early stopping callback that would evaluate the validation accuracy of the model over the last three epochs. In the case that the accuracy does not improve or begins to decrease, the model will stop training. The number of epochs can vary between training sessions based on several factors such as the size of the dataset and the variation of the data. This algorithm prevents overtraining and improves the generalization of the model.

VIII. Making Predictions

In order to evaluate the performance of the model, we used the test partition of the dataset to make predictions. As stated previously, the raw output of the model is a value between 0 and 1, where values less than 0.5 correspond to synthetic classifications and values more than 0.5 correspond to organic classification. The algorithm for making predictions and determining confidence level is shown in table I.

IX. 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. The nominal waveform and spectrogram plots are shown in Figure XIV.

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 XIV and XV, the AWGN appears to make the spectrogram of the signal blurrier than the noise-free spectrogram potentially hiding key features. The white noise waveform and spectrogram plots are shown in Figure XV.

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 XVI, the burst noise appears to add vertical artifacts to the spectrogram that potentially hides key features.

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 XVI to Figure XVIII.

X. SYNTHETIC SPEECH GENERATORS

After completing the experimentation with noise processing and filtering, we created new datasets using synthetic speech generation software. The following applications was used to create these datasets.

A. iOS 17 Personal Voice

In the latest release of iOS, Apple has added a new feature called Personal Voice. This feature allows a user to record themselves saying several phrases and trains a local model to synthesize the speaker's voice. Once the model has learned the voice, it can be used for text-to-speech [17].

B. PlayHT

PlayHT is a voice AI company that is focused on building voice models to synthesize speech and replicate accents [18].

The results of testing and training with these datasets is shown in Table II.

XI. CONCLUSION AND RESULTS

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:

- A baseline trained on nominal data and predicted against nominal data
- A model trained on nominal data and predicted against white noise data
- A model trained on white noise data and predicted against white noise data
- A model trained on nominal data and predicted against filtered white noise data
- A model trained on nominal data and predicted against burst noise data
- A model trained on burst noise data and predicted against burst noise data
- 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 II.

When viewing the results from Table II, 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. 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.

XII. FUTURE WORK

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 for 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.

XIII. 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

A. Repository Contents

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 results

B. How to Contribute

Use the following command to clone the code repository.

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

ACKNOWLEDGEMENT

This research project was completed for CPE 657, taught by Dr. Jeffrey Kulick, at the University of Alabama in Huntsville. This is the capstone course requirement for the Master of Science in Software Engineering program.

REFERENCES

[1] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb. 1978, doi: 10.1109/tassp.1978.1163055.
[2] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," Proceedings of the IEEE, vol. 77, no. 2, pp. 257–286, Jan. 1989, doi: 10.1109/5.18626.
[3] Z. Almutairi and H. ElGibreen, "A review of Modern Audio Deepfake Detection Methods: Challenges and Future Directions," Algorithms, vol. 15, no. 5, p. 155, May 2022, doi: 10.3390/a15050155.
[4] R. Wang et al., "DeepSonar: Towards Effective and Robust Detection of AI-Synthesized Fake Voices," arXiv (Cornell University), May 2020, doi: 10.48550/arxiv.2005.13770.
[5] J. Frank and L. Schönherr, "WaveFake: a data set to facilitate audio deepfake detection," arXiv (Cornell University), Jun. 2021, doi: 10.5281/zenodo.4904579.
[6] "The LJ Speech Dataset." https://keithito.com/LJ-Speech-Dataset/
[7] C. Hsu, Y. Zhuang, and C.-Y. Lee, "Deep fake image detection based on pairwise learning," Applied Sciences, vol. 10, no. 1, p. 370, Jan. 2020, doi: 10.3390/app10010370.
[8] H. S. Shad et al., "Comparative analysis of deepfake image detection method using convolutional neural Network," Computational Intelligence and Neuroscience, vol. 2021, pp. 1–18, Dec. 2021, doi: 10.1155/2021/3111676.
[9] S. Salman, J. A. Shamsi, and R. Qureshi, "Deep Fake Generation and Detection: Issues, challenges, and solutions," IT Professional, vol. 25, no. 1, pp. 52–59, Jan. 2023, doi: 10.1109/mitp.2022.3230353.
[10] T. Macaulay, "Deepfake fraud attempts are up 3000% in 2023 — here's why," TNW | Data-Security, Nov. 15, 2023. [Online]. Available: https://thenextweb.com/news/deepfake-fraud-rise-amid-cheap-generative-ai-boom
[11] "Case studies and mentions," TensorFlow. https://www.tensorflow.org/about/case-studies

[12] “Everything you wanted to know about TensorFlow,” Databricks. https://www.databricks.com/glossary/tensorflow-guide#:~:text=Because%20TensorFlow%20was%20developed%20by,to%20speed%20up%20TensorFlow%20jobs
[13]Tan, Xu, et al. “FastSpeech 2: Fast and High-Quality End-to-End Text to Speech.” *Microsoft Research*, 29 Apr. 2021, www.microsoft.com/en-us/research/lab/microsoft-research-asia/articles/fastspeech-2-fast-and-high-quality-end-to-end-text-to-speech/.
[14] L. Y. Data, “Machine learning and training data: what you need to know,” Label Your Data, Mar. 17, 2021. https://labelyourdata.com/articles/machine-learning-and-training-data
[15] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv (Cornell University), Dec. 2014, doi: 10.48550/arxiv.1412.6980.
[16] R. Agarwal, “Complete guide to the ADAM Optimization Algorithm,” Built In, Sep. 13, 2023. https://builtin.com/machine-learning/adam-optimization
[17] “Record your Personal Voice on iPhone,” Apple Support. https://support.apple.com/guide/iphone/record-a-personal-voice-iph51936468d/ios#:~:text=With%20Personal%20Voice%2C%20you%20can,follow%20the%20series%20of%20prompts.
[18] “PlayHT: We build human-like Voice AI that customers love speaking with. | Y Combinator,” Y Combinator. https://www.ycombinator.com/companies/playht

FIGURE I
Deep fake attempts 2022 to 2023 [10]

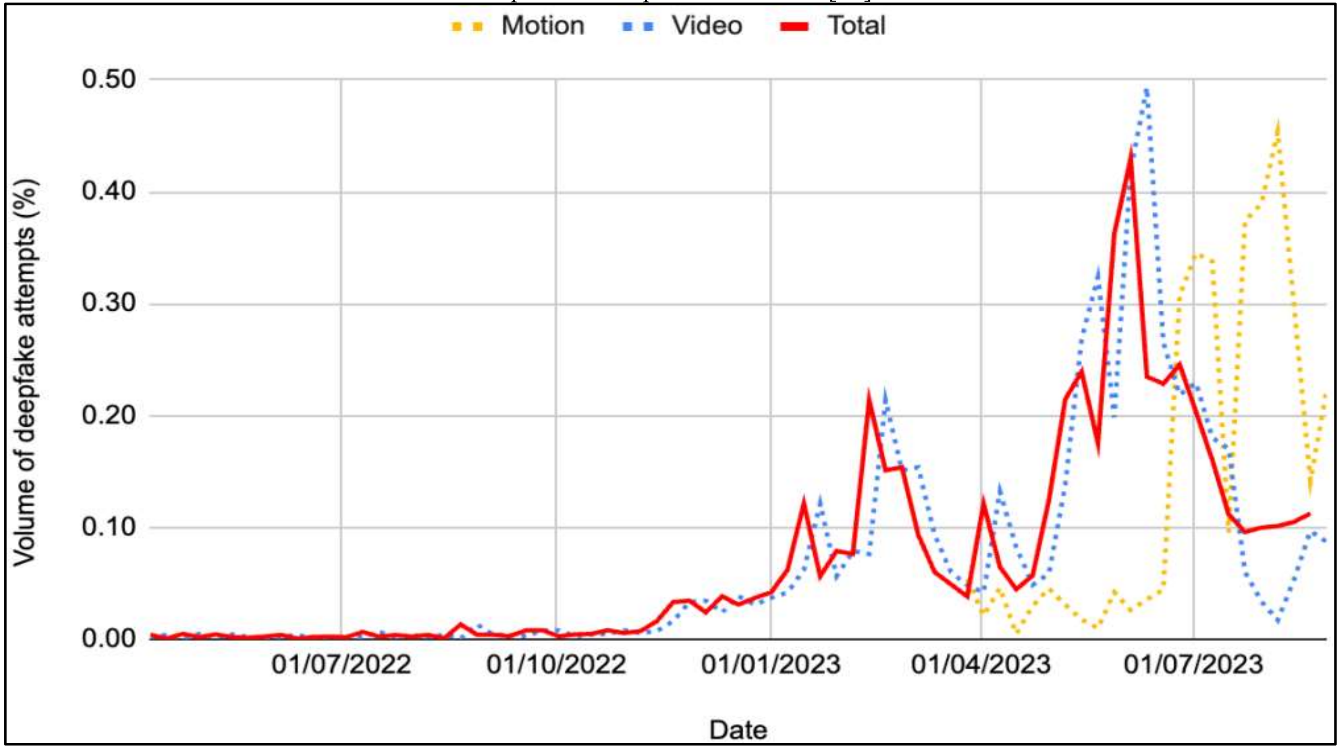

FIGURE II
Dataset Split [14]

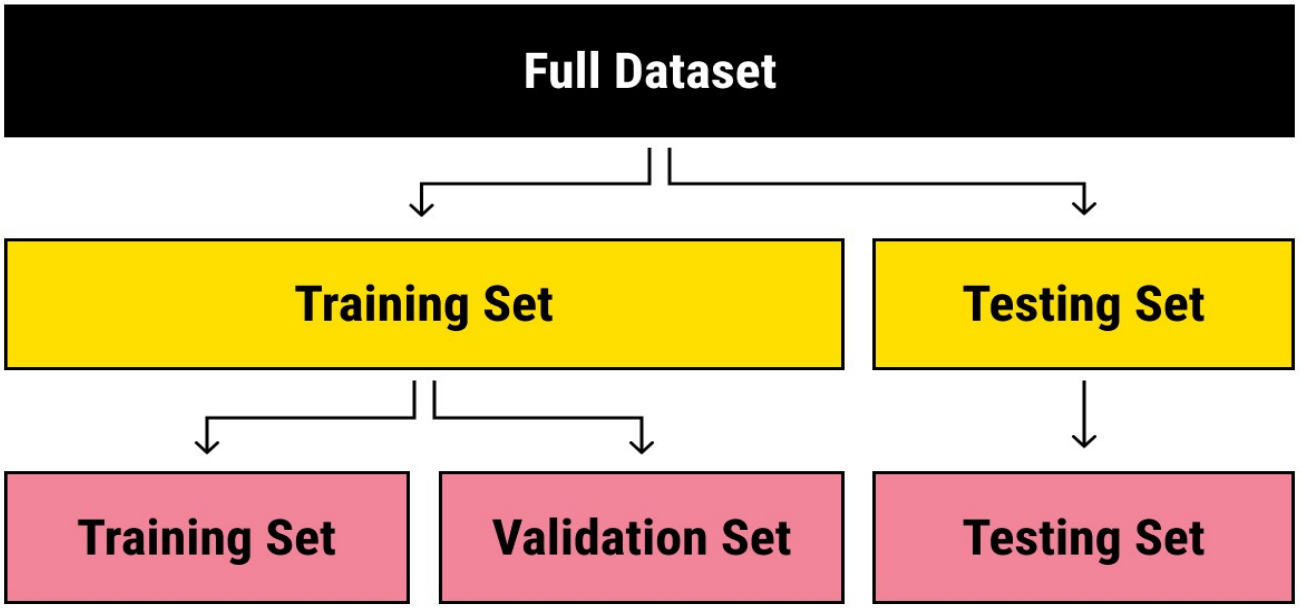

FIGURE III
FastSpeech2 Architecture[13]

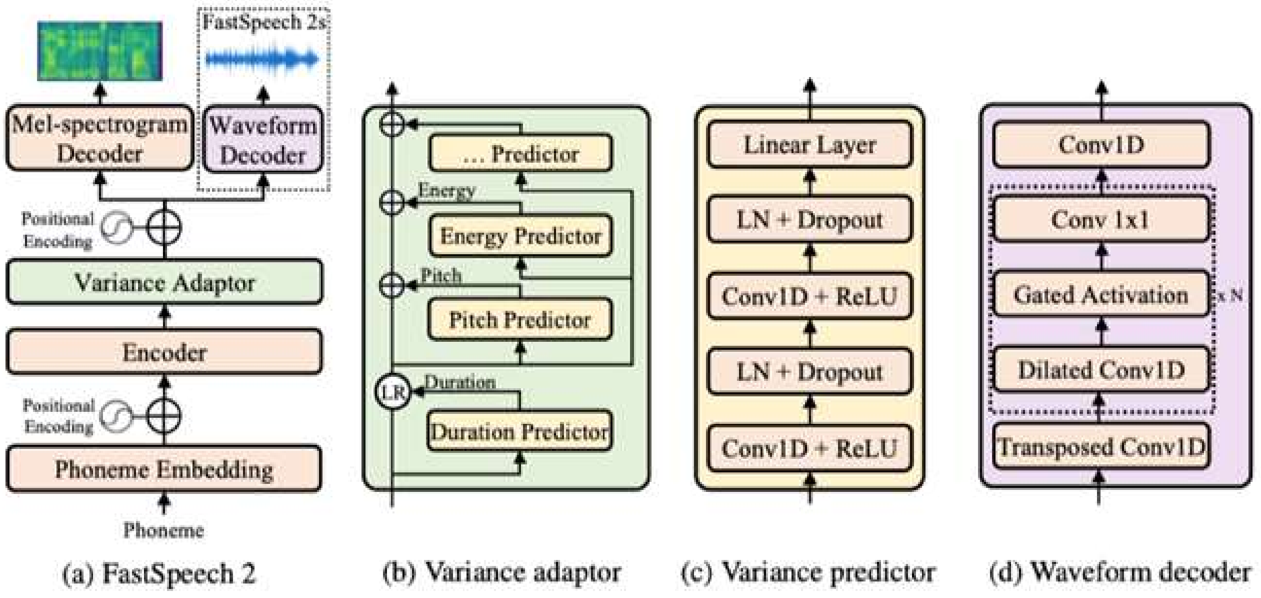

FIGURE IV
Dataset Folder Structure

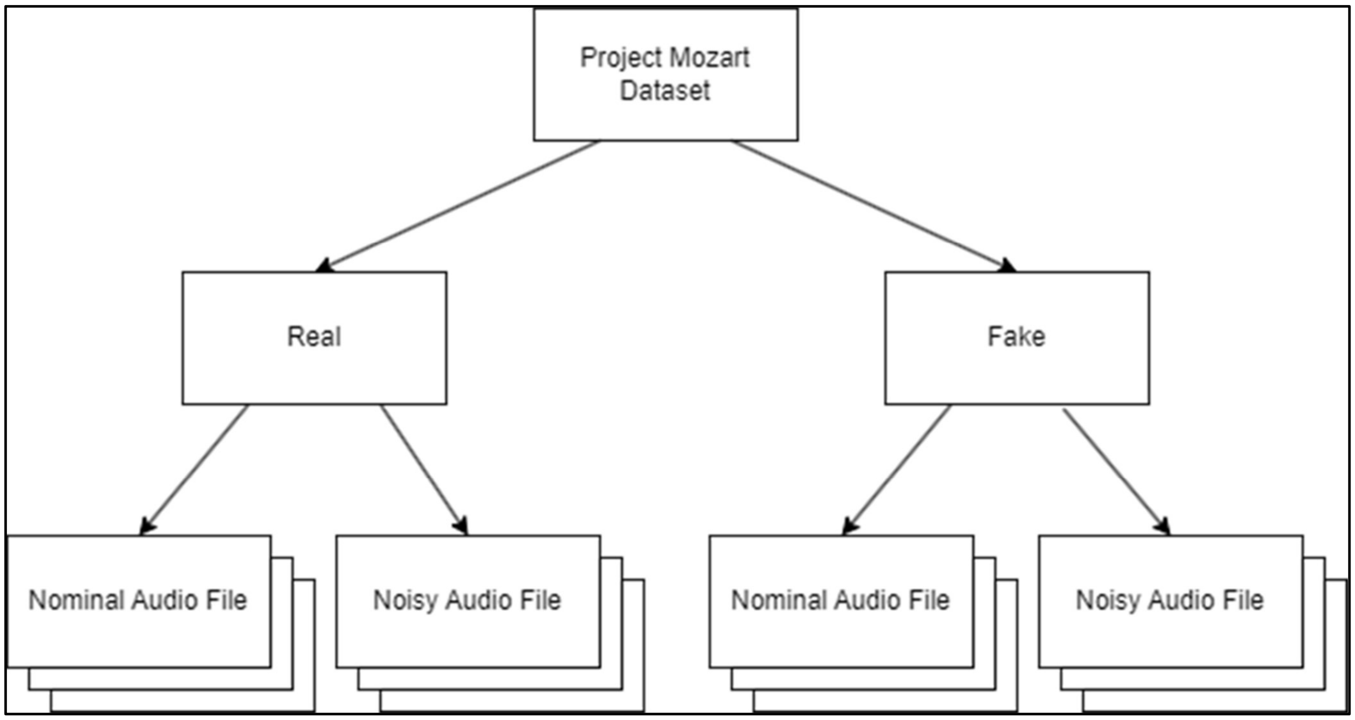

FIGURE V
Design Overview

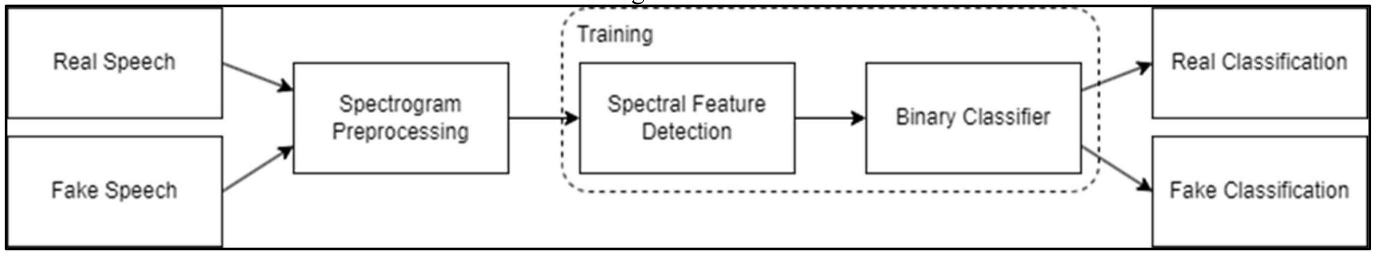

FIGURE VI
Keras Dataset Format

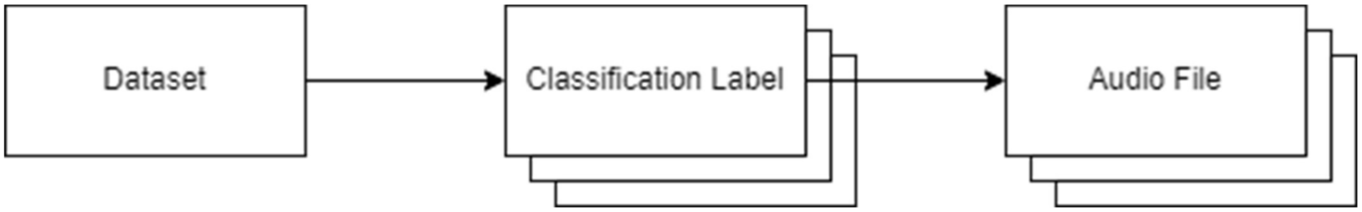

FIGURE VII
Spectral Detection Model Layers

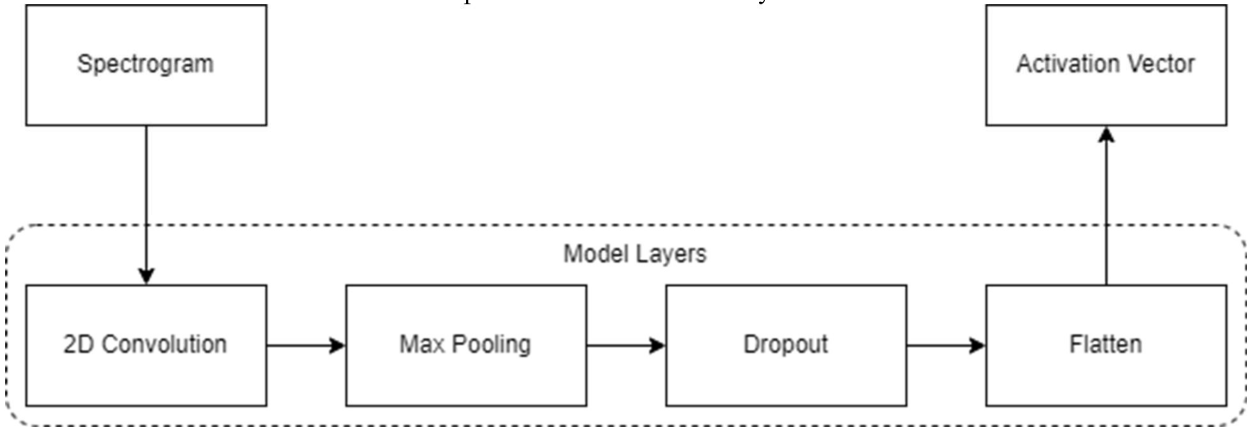

FIGURE VIII
2D Convolution Operation Overview

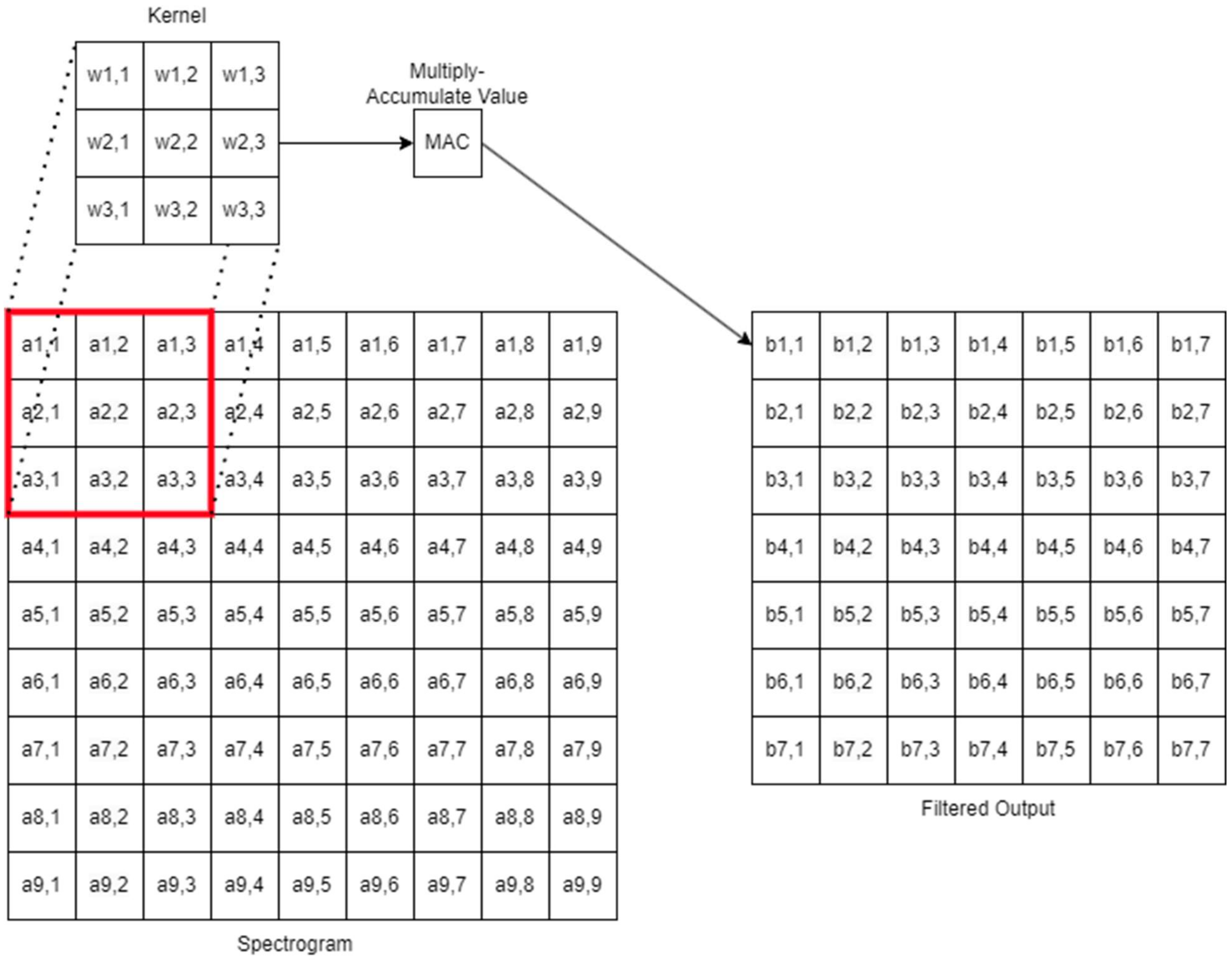

FIGURE IX
Max Pooling Operation Overview

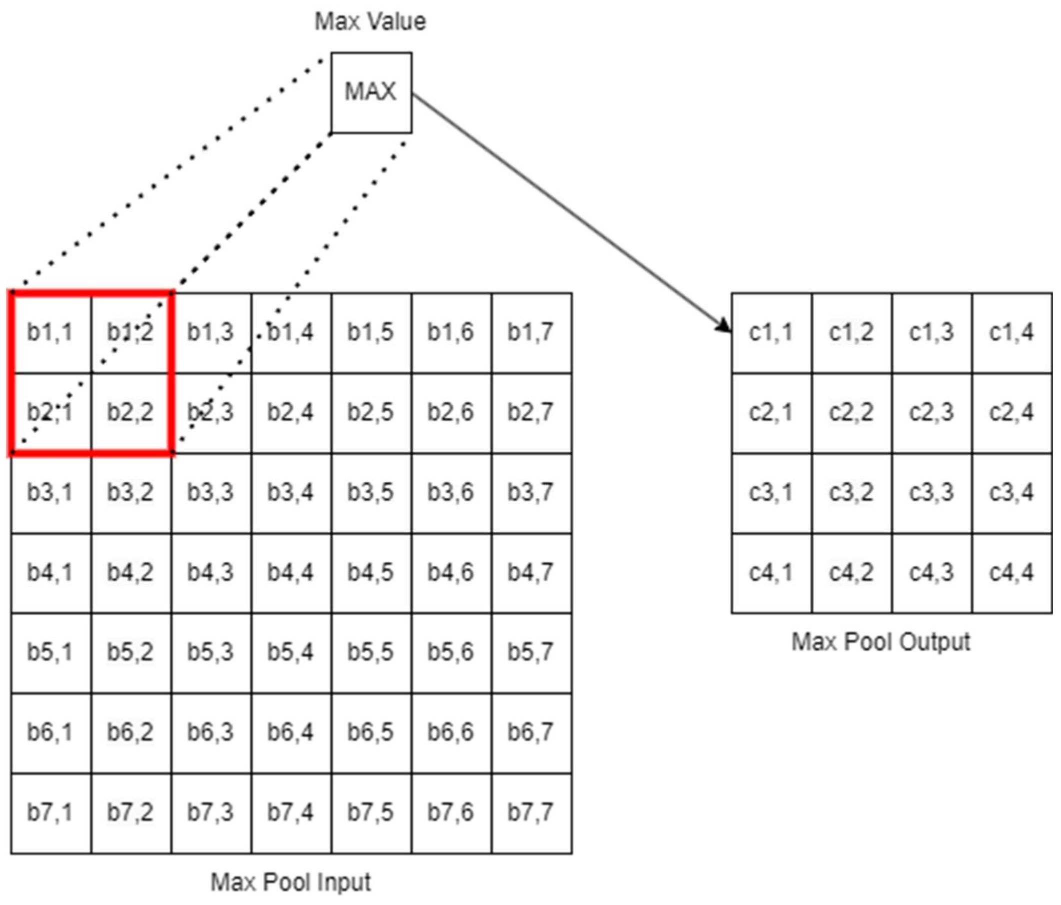
Max Value
MAX
b1,1 b1,2 b1,3 b1,4 b1,5 b1,6 b1,7
b2,1 b2,2 b2,3 b2,4 b2,5 b2,6 b2,7
b3,1 b3,2 b3,3 b3,4 b3,5 b3,6 b3,7
b4,1 b4,2 b4,3 b4,4 b4,5 b4,6 b4,7
b5,1 b5,2 b5,3 b5,4 b5,5 b5,6 b5,7
b6,1 b6,2 b6,3 b6,4 b6,5 b6,6 b6,7
b7,1 b7,2 b7,3 b7,4 b7,5 b7,6 b7,7
Max Pool Input
c1,1 c1,2 c1,3 c1,4
c2,1 c2,2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

FIGURE X
Dropout Operation Overview

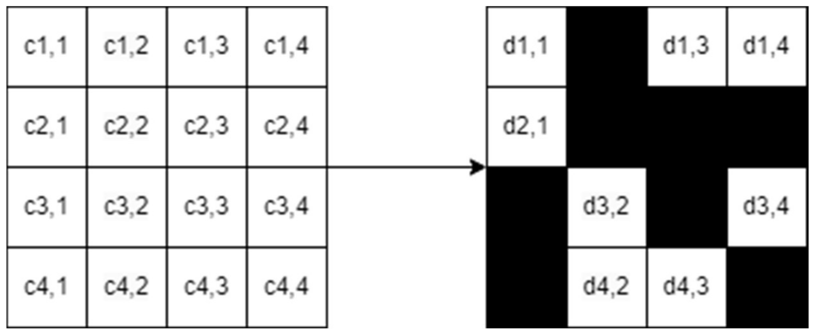

FIGURE XI
Flatten Operation Overview

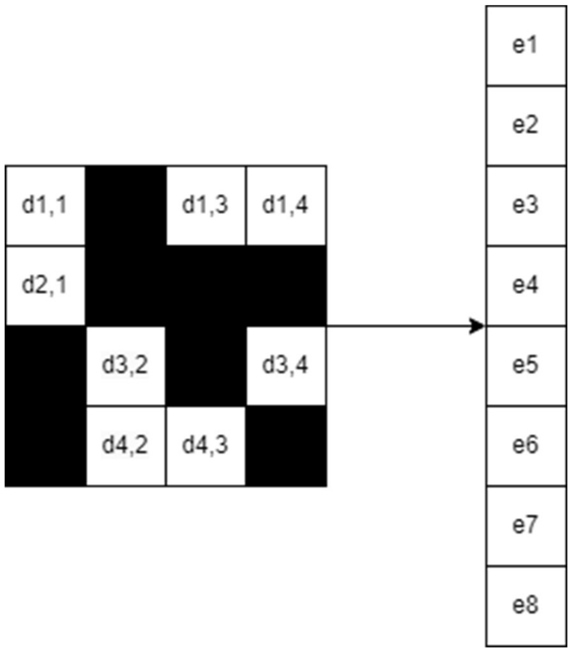

FIGURE XII
Binary Classification Model Overview

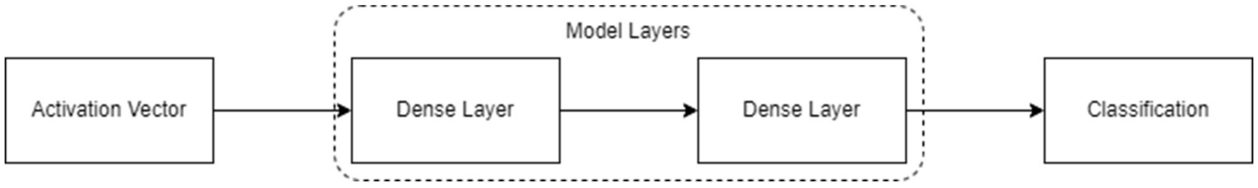

FIGURE XIII
Dense Layer Overview

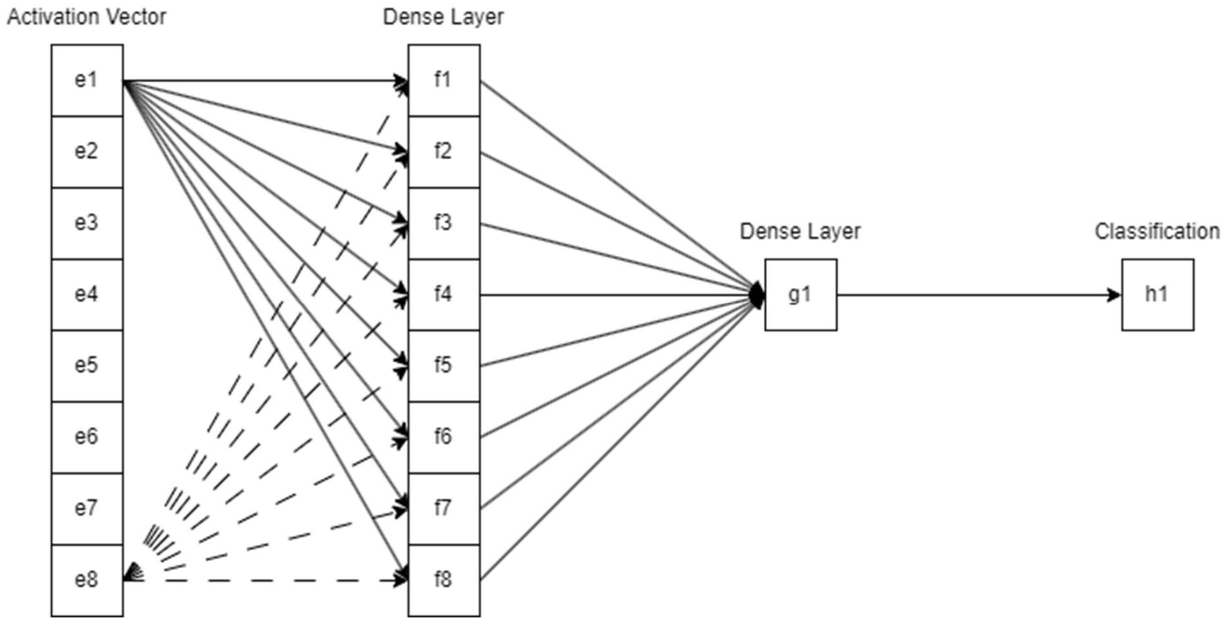

FIGURE XIV
Nominal Signal Waveform and Spectrogram

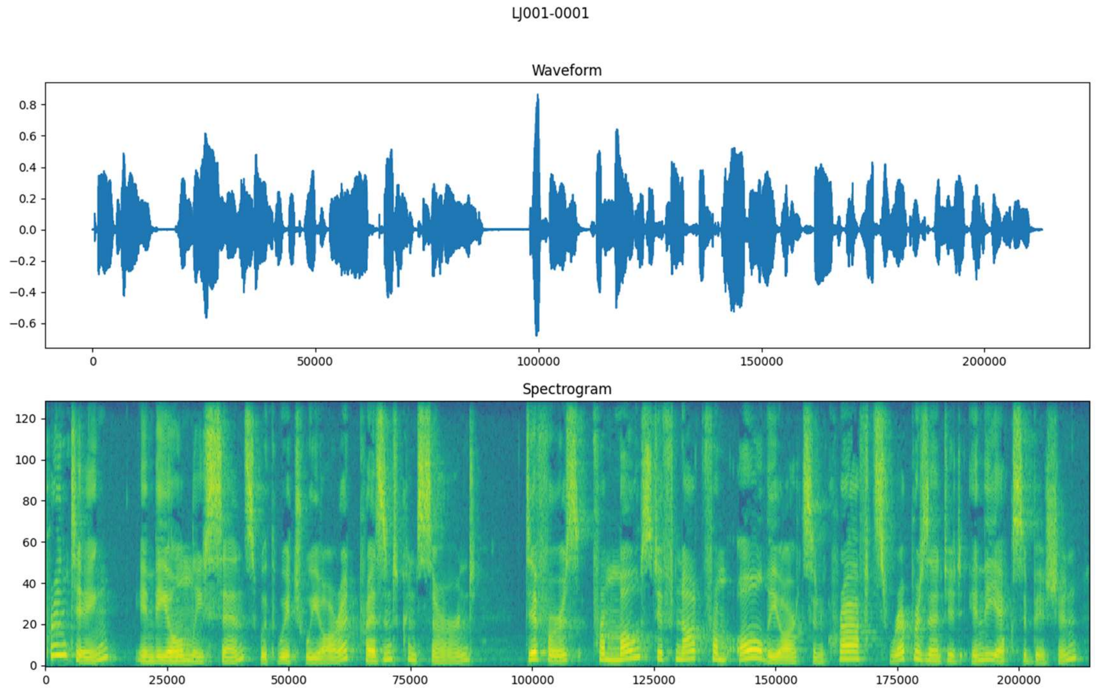

FIGURE XV
Additive White Gaussian Noise Signal Waveform and Spectrogram

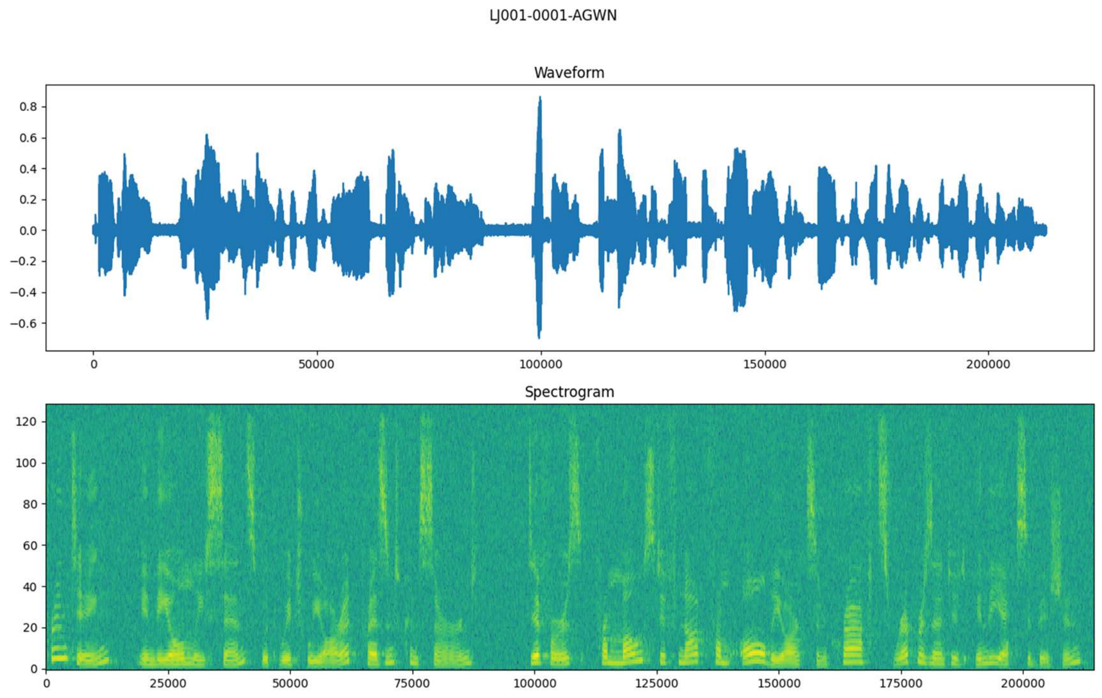

FIGURE XVI
Burst Noise Signal Waveform and Spectrogram

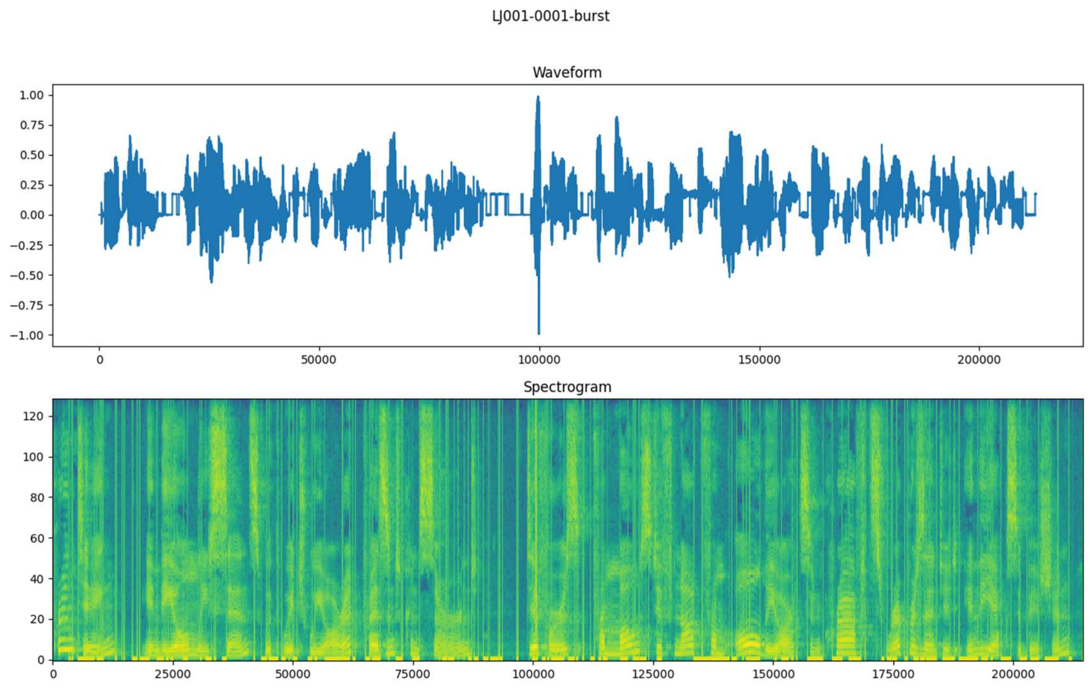

FIGURE XVII
Additive White Gaussian Noise Filtered Signal Waveform and Spectrogram

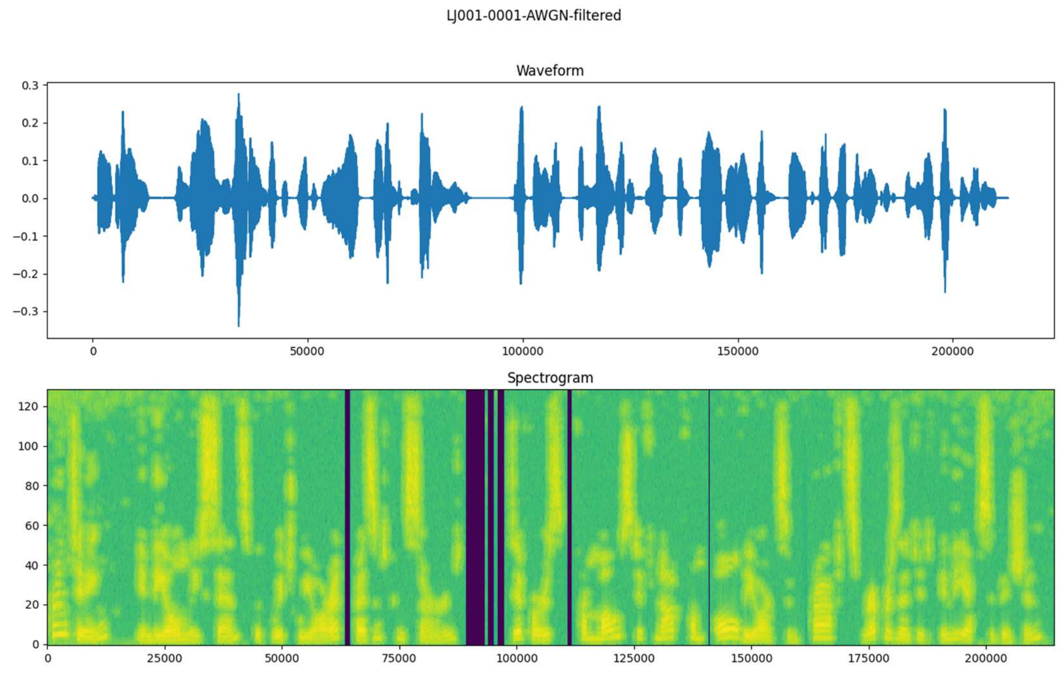

FIGURE XVIII
Burst Noise Filtered Signal Waveform and Spectrogram

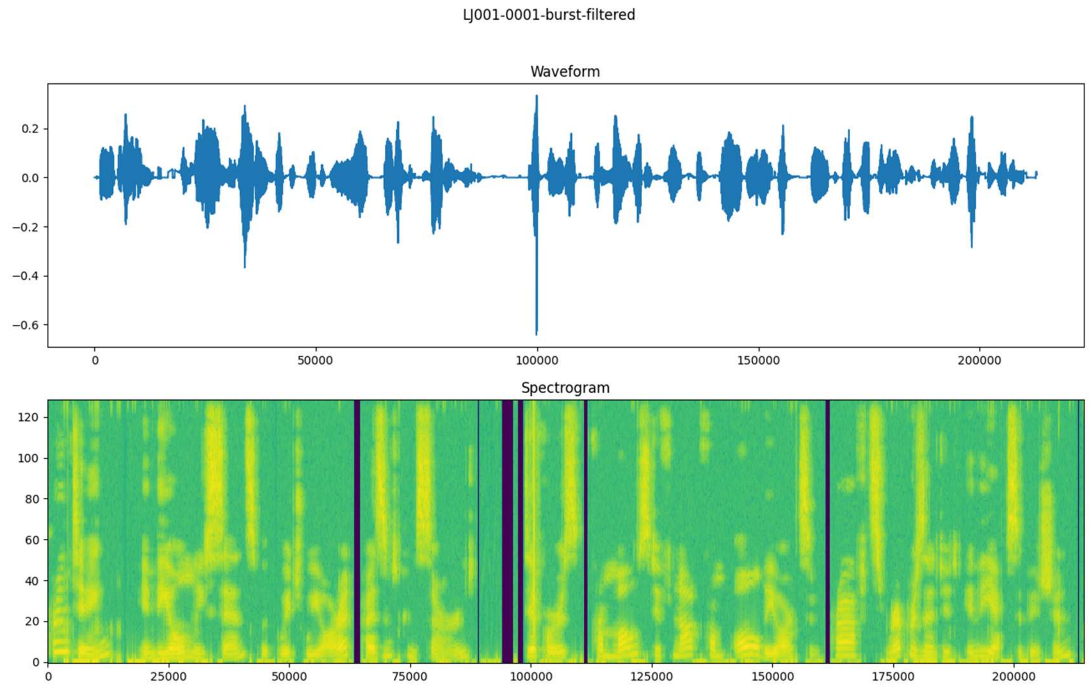

TABLE I
Model Output and Confidence Levels

Output	Classification	Confidence Level
0	real	100% confident that the speech is real
0.3	real	70% confident that the speech is real
0.5	unknown	Unknown if it is real or fake
0.7	fake	70% confident that the speech is fake
1	fake	100% confident that the speech is fake

TABLE II
Final Results

Scenario	Avg Prediction Acc.	EER	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

TABLE III
Acronyms

Acronym	Meaning
AD	Audio Deepfake
ANN	Artifical 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	Siganl-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