Acoustic Keylogging via Embedded System Proof of Concept

George Krier
Computer and Electrical Engineering
Colorado State University
Fort Collins, Colorado, USA
George.Krier@colostate.edu

***Abstract*— Acoustic keylogging, a form of side channel attack, represents a growing cybersecurity threat. This project introduces a lightweight, real-time Convolutional Neural Network (CNN) model designed for generalized acoustic keylogging on embedded systems with computational resource constraints. By utilizing a custom dataset, this model can predict keypresses from keyboard acoustic audio with a test accuracy of ~51% and precision of around ~82%. This project also consists of a keylogger, audio recorder, and data preprocessing component that allows for an accurate collection of training data for the acoustic keylogging model. This proof of concept emphasizes the need for further acoustic keylogging research for more robust safeguards against this type of side channel attack.**

Keywords—STFT, CNN, RNN, quantization, inference, ReLU

I. INTRODUCTION

Over the past two decades, the rise in digital reliance has transformed many different facets of our society, where computers are being integrated into every aspect of operations. As this technological dependency deepens within our society, so does cybersecurity threats which can heavily damage not only businesses, but also critical sectors such as healthcare, telecommunications, and national security. In 2024, data breaches cost an average of $4.88 million in business disruption and 70% of affected organizations reported significant disruption [3]. In the ongoing battle against cyber threats and the increasing sophistication of these attacks, businesses and governments have adopted a more rigorous cybersecurity stance. Acoustic keylogging, although a minor but proven threat currently, is representative of a growing form of data exfiltration that can be exploited to steal Personally Identifiable Information (PII) or other sensitive information through digital or physical means.

While many cyber threats focus on network vulnerabilities, acoustic keylogging, classified as a side channel attack, often requires physical proximity to the target system circumventing many digital security measures. Due to the low-profile nature of an acoustic keylogging attack, relying on sound waves instead of software-based methods, acoustic keylogging may be difficult to detect and without a digital trace. For the full magnitude of cyber threats such as acoustic keylogging to be realized, a proof of concept is an effective tool to illustrate potential impacts of these threats and help organizations mitigate risk from such attacks.

This project introduces a CNN model for acoustic keylogging with the optimization for embedded systems as the primary goal. By utilizing a simple 2D convolution network and intense data audio preprocessing, this model can achieve a test accuracy of 51% and a precision of 82% across 54 different keyboard classifications. Utilizing STFT spectrograms, audio is preprocessed before being inferred by the model to receive this accuracy. Additionally, this model is optimized with Quantization and converted to a TFLite model to reduce its size for deployment on embedded systems. Originally this model was proposed to have a Recurrent Neural Network (RNN) included but for the purpose of this project, it seemed unnecessary to classify key press noise recurrently from the audio signature of each key since each keypress is an independent event.

This project presented many difficult and unique challenges that needed to be addressed before a working model could even be synthesized. A major hurdle in this project was collecting and preprocessing data that adequately the problems. This involved answering critical questions: what does the raw data need to look like, how should the audio be labeled, what spectrogram length would be best for the model to extract the keypress audio signature. Additionally, data augmentation techniques and model selection were essential to ensure the model could achieve reasonable accuracy while maintaining computational and memory efficiency.

This paper details the approaches taken to solve these problems including a comprehensive preprocessing pipeline, spectrogram design and model architecture. The results of this project demonstrate the effectiveness of the methods utilized to solve these problems.

II. RELATED WORK

Acoustic keyloggers use machine learning to extract unique audio profiles of individual key presses and identify the keys. Other researchers in this area have been proven to successfully implement an acoustic keylogger on a personal device such as a smartphone with an accuracy of 95% [2]. These models have the downside of having large models that utilize self-attention transformers making these models impractical for embedded deployment and don't dive into real time inference with their models. One model specifically for acoustic keylogging from a remote and physical attack scenario uses the CoAtNet neural network architecture that utilizes convolution and self-attention

[5] leading their model to be large and use significant computational resources to train and infer from [2]. This other project utilizes a similar audio preprocessing pipeline as this project by utilizing STFT spectrograms but additionally utilized keystroke extraction through an energy threshold and performed more comprehensive data augmentation than in this project.

Another Acoustic keylogging project [4] is a classification and smart dictionary attack that does not utilize neural networks but instead utilizes feature extraction and performs linear regression to classify keystrokes. This model is based off the acoustic keylogging framework that can classify keystrokes through Skype based on a strong assumption that the proximity of a microphone is close enough to the victim, precise profiling of the victim's typing style, and significant training data on the victim [6]. The model for this project differs in that it has the capability of learning typing data from a victim in ~15 minutes and doesn't rely on a unique typing style of an individual or position of microphone.

III. Data

The dataset that was used initially to work with this project was the Multi-Keyboard Acoustic (MKA) dataset [1] and additional supplemental data collected directly for this model by myself. Half way through the project I deemed the MKA Dataset to have too many issues for splicing in the necessary supplemental data to generalize the model such as: a lack of labeling on individual keystroke fingerprints, audio data that was longer or shorter than the 1 second window that the dataset was supposed to ensure, and not enough noise included into the dataset that could be applied to a real-world scenario. All the issues I found with this dataset I attempted to solve through intense preprocessing, but it brought up more problems than deemed valuable rather than collecting my own noise data. The one downside of deciding to use my own dataset was that my dataset could be imbalanced, I may not have enough audio data to perform machine learning, and a lack of a control environment.

A. Dataset Generation

The dataset generated contains 906 seconds of recording data with each key being pressed around 30 times independently then following a randomly generated typing test at ~150 words per minute. This generated a total of 11,901 keypress data after data augmentation and a total of 54 key press classifications consisting of:

- Letters
  - a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
- Numbers
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Punctuation and Symbols

  apostrophe (’), backslash (), backtick (`), bracketclose (]), bracketopen ([), comma (,), dash (-), equal (=), fullstop (.), semicolon (;), slash (/)

For creating the training dataset, a keylogger and audio recorder were designed to run independently of each other on a “victim” machine. The keylogger uses the python “keyboard” library and checks when a key is pressed “down” before logging the label of the key pressed and the Unix epoch timestamp of the keyboard press into a metadata file. The audio recorder script records microphone input at 44.1 kHz for a total duration of 1 second yielding 44,100 samples before saving the recording as a .wav file named in the format “{timestamp-start}-{timestamp-end}.wav”. The audio recorder repeats this process until it is manually stopped.

The data collected from the keylogger, and audio recorder are then spliced together in the dataset preprocessing pipeline. Once the audio data is matched with the keypress metadata, the raw audio data is normalized to ensure that the max amplitude and min amplitude are between -1 and 1 illustrated by Figure 1 and 2. Then the waveform is augmented with random noise and the volume amplified randomly illustrated by Figure 3 and 4. The normalized raw audio and normalized augmented audio are then merged before generating a spectrogram.

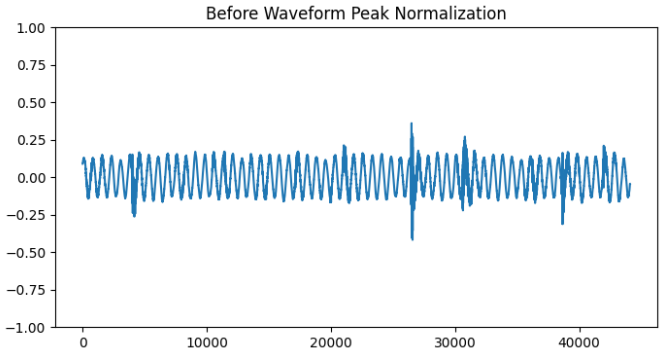

Figure 1: Before Waveform Peak Normalization

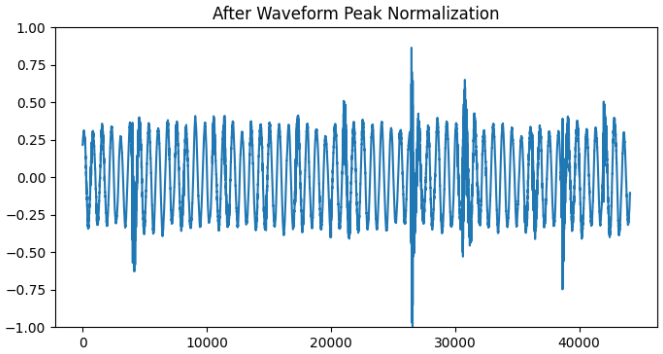

Figure 2: After Waveform Peak Normalization

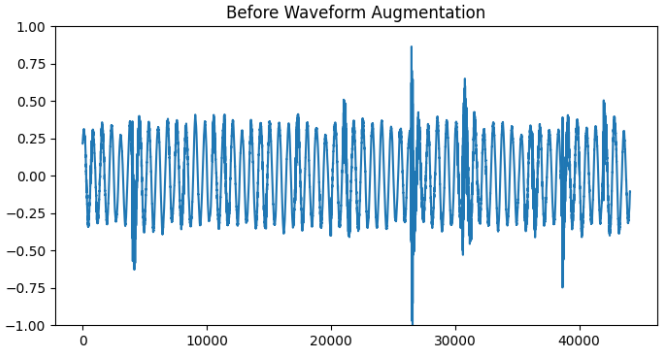

Figure 3: Before Waveform Augmentation

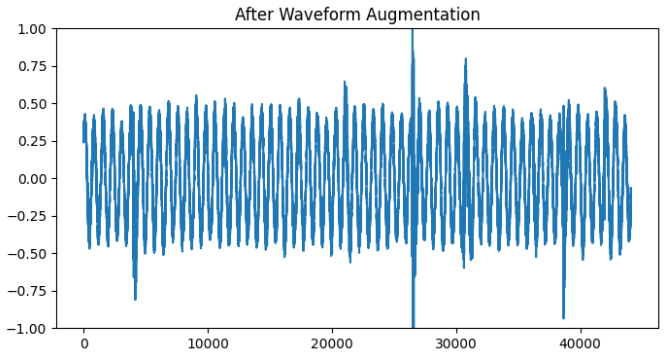

Figure 4: After Waveform Augmentation

The STFT spectrogram is then generated with a frame length of 220 and frame step of 110 which for 44,100 samples in a second of audio results in each frame being 5 milliseconds of audio and each step being 2.5 milliseconds of audio. On average, a keypress audio signature lasts around 25 milliseconds shown in Figure 5, but to be conservative in this dataset, 19 spectrogram frames of length 220 are grouped together to enhance the resolution of the spectrogram leading to a single spectrogram processed in the model to be ~47.4 milliseconds long shown by equation (1) and (2).

$$Group\ Duration = \frac{number\ of\ frames\ x\ frame\ step}{sampling\ rate} \quad (1)$$

$$Group\ Duration = \frac{19\ x\ 110}{44100} = 0.04739\ s \approx 47.4\ ms \quad (2)$$

The grouped spectrograms are then labeled according to the metadata with a time tolerance of 5 milliseconds before and 10 milliseconds after the original keypress to ensure spectrograms are labeled correctly and any latency in data collection is accounted for. An example spectrogram and label in the dataset that would be passed to the model is shown in Figure 6.

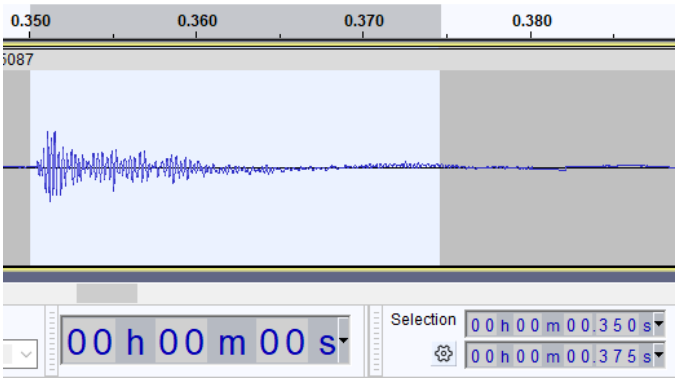

Figure 5: Raw audio of a keypress

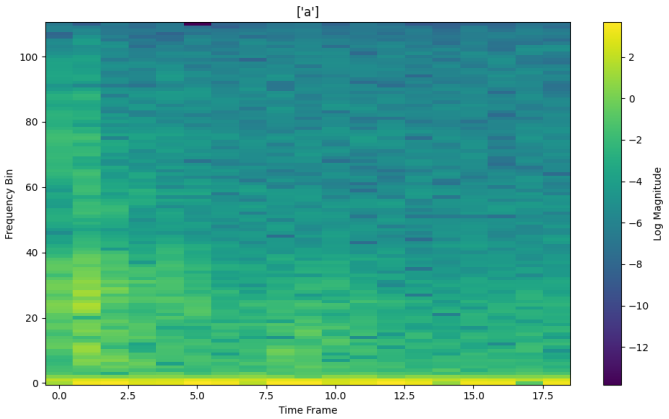

Figure 6: Generated Spectrogram in the Dataset for keypress "a"

B. *Problems Encountered*

Early in the model training after completing the dataset generation, the simple CNN model had an accuracy of ~4% with a precision of nearly 90%. After evaluating the model, it was apparent that the model was not at fault for this lopsided accuracy and precision, this led me to investigate the dataset further which uncovered two large issues.

The largest issue that was identified was that the keylogger that I created was saving keypress timestamps nearly 80 milliseconds from the actual audio signature recorded. This meant that all the data that had been collected up to that point was labeled incorrectly with classifications not being labeled at all. It became apparent that the model was incredibly good at identifying noise as no classification leading to a high precision but low accuracy since the model couldn't learn actual keypress signatures. The issue was identified to be due to the early approach used to record the audio through a Raspberry Pi and recording key logged keystrokes through the victim machine. Since data collection was happening on two separate devices, the clocks on the devices deviated around 40-80 milliseconds and once the data was spliced it caused these inconsistencies. After switching to do data collection on 1 device for both keylogging and audio recording, the key presses and audio signatures were matched correctly, and it raised training accuracy to ~14% with a high prediction.

While evaluating why the model was still performing poorly, I evaluated the amount of collected data for each label and found that the maximum occurrence of labels for keypresses was unlabeled at 26,336 occurrences while the median for all other labels was 178 occurrences illustrated in Figure 7. This implied that the dataset generated was heavily unbalanced and the model was biased more towards unimportant unlabeled data rather than actual key press data. The solution was to drop most of the unlabeled data from the dataset to balance the dataset for training. This was done by randomly dropping the unlabeled data from using a percentage generated by 1 minus the median of all label occurrences divided by the number of occurrences of unlabeled data. After dropping a large portion of the unlabeled data, the dataset was more balanced with the next highest occurring label being "space" illustrated in Figure 8. This improved the model accuracy drastically but further improvements in data balancing may increase the accuracy and precision further.

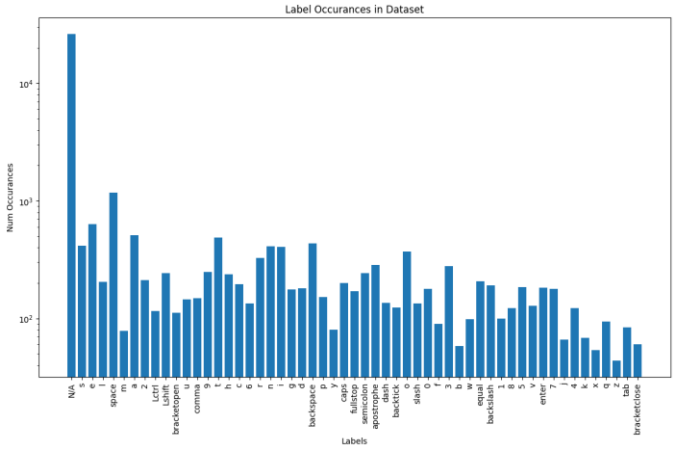

Figure 7: Dataset Label Composition before balancing no label (Log scale)

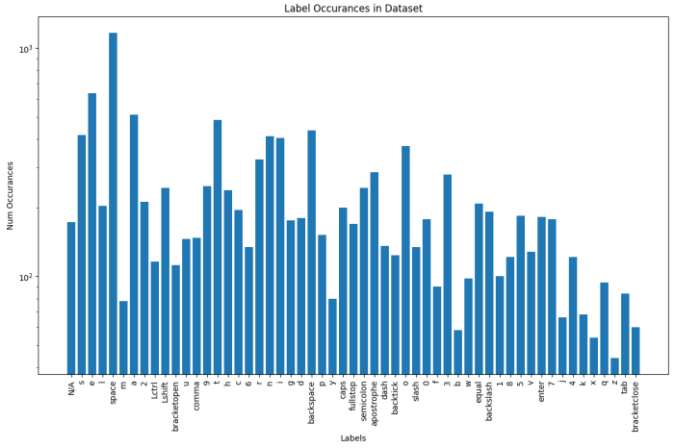

Figure 8: Dataset Composition after balancing no label (Log scale)

IV. Technical Approach

To tackle the problem of keypress audio classification and effectively extract features from raw data, two different approaches were explored and evaluated. This project initially focused on a CNN and RNN approach given their efficacy in audio signal processing and sequence learning, a model with these two architectures were believed to be promising due to their ability to handle spatial and temporal patterns in data that could prove useful for this application. However, during the application of this model, limitations were identified, and the model was unable to learn the patterns effectively. Additionally, the dataset prepared by the preprocessing pipeline was not aligned with the goal of acoustic keypress classification since each keypress should be treated as an independent event. This first approach also utilized the MKA Dataset and supplemental data which created issues when trying to splice the data together and have a model that could extract features and sequences effectively.

The secondary and final approach taken was to remove the MKA Dataset from the project and utilize only CNNs. This approach was more in line with the goals of the project of independent keypress classification. Along with this secondary approach the preprocessing pipeline was rewritten to rework how the spectrograms, labeling, and dataset specification was designed before being passed to the model. This approach simplifies the model and results in significant improvements to the project's goals.

This section will detail the approaches taken in the project and discuss the shortcomings encountered along with the approach taken for model optimization and inference. Additionally, challenges such as preprocessing and data collection will be examined further to highlight how these challenges were mitigated.

A. CNN and RNN Project Approach

The dataset used for this approach was a combination of the MKA dataset and supplemental collected data which caused a few issues. The MKA dataset contained extremely clean audio signatures from 5 different keyboards but contained issues with consistency. For instance, the dataset contained labeled 1 second audio recordings of a certain key being pressed but it failed to label when the audio signature began. Additionally, the quantity of keypresses in a recording were inconsistent, usually between 2-3 keypresses in a 1 second recording. Attempting to work around this problem required a threshold-based approach of labeling and splitting the 1 second audio recording into frames of 2205 samples with a 1102 sliding window and labeling the entire window as a keypress signature since it was unfeasible to find the exact start and end of the keypress audio signature. This caused audio signatures to be a very large size and lead to a higher computational cost of processing.

Since the MKA dataset also consisted of 5 different heavily preprocessed keyboard keypress signatures, it did not align with the project's goal of classifying keyboard noise from a real-world environment. Noise could have been introduced to the dataset but to make the signatures more realistic, it made more sense to collect other data leading to the use of the supplemental dataset.

The preprocessing for this merged MKA dataset and supplemental dataset further introduced unnecessary complexity into the project. The 1 second audio recordings from both datasets were converted to STFT spectrograms with a frame of 2205 and 1102 sliding window and then processed into a sequence of 3 spectrograms for processing with an RNN. This approach aimed to influence the CNN/RNN model to learn sequentially, enabling it to perform a single iteration of training and inference per second of audio input and produce a list of outputs for each spectrogram every iteration.

In retrospect, the rationale for pursuing this sequential approach did not align with the project goals of real-time processing since a second long audio sample would need to be buffered and then inferred upon before predictions would be returned. Additionally, training the model sequentially would require significantly more data, and this approach is unnecessarily complex as keypresses are independent events with no sequential dependency. The model in this approach consisted of over 36 million parameters which furthermore made it infeasible to run on an embedded system in real-time if at all due to memory constraints. The RNN component itself increased the number of parameters on the model by over 16 million parameters. The CNN/RNN model layer composition is illustrated in Figure 9, note that this approach has a final dense layer of 63 neurons for 63 labels which were reduced in the next

approach. Many of the layers in the model were created to force ensure output aligns with the requirements and provide meaningful results but did not promote learning in the model, but instead increased complexity such as with the time distributed layers.

Model: "sequential"

Layer (type)	Output Shape	Param #
time_distributed (TimeDistributed)	(None, 3, 1101, 32)	128
time_distributed_1 (TimeDistributed)	(None, 3, 1099, 128)	12,416
time_distributed_2 (TimeDistributed)	(None, 3, 549, 128)	0
time_distributed_3 (TimeDistributed)	(None, 3, 70272)	0
time_distributed_4 (TimeDistributed)	(None, 3, 70272)	0
bidirectional (Bidirectional)	(None, 3, 128)	36,012,544
dense (Dense)	(None, 3, 128)	16,512
dense_1 (Dense)	(None, 3, 63)	8,127

Figure 9: CNN/RNN Model Approach Composition

B. *CNN Project Approach*

After assessing the limitations of the previous approach and concluding that it was unsuitable for the project, I restructured and rewrote significant portions of the preprocessing pipeline and the model to implement a CNN based approach. This approach utilizes a different STFT spectrogram window approach in the preprocessing pipeline that utilizes much shorter frame lengths and frame steps yielding in spectrograms being ~50 milliseconds of audio frequency data. These STFT spectrogram frames are then labeled allowing for the model to learn from a very high-resolution spectrogram preserving the brief keypresses acoustic signature frequency information. The dataset is then loaded from the preprocessing pipeline, shuffled, and split into 80:10:10 train, validation, and test datasets for optimal model training.

The CNN model is then built, illustrated in Figure 10, with an input shape of (19, 111, 1) for 19 spectrogram windows with 111 frequency bins and a dimension of 1 to ensure the features of the spectrograms are appropriately modeled. The model then consists of two 2-dimensional convolution layers with filter sizes of 32 and 128 to extract low-level and high-level patterns from the spectrograms, while the 3 x 3 kernel size effectively captures localized acoustic features. Additionally, the convolution layers utilize the ReLU activation function for efficient and robust learning of complex patterns to enhance the ability for the model to distinguish subtle variations in the keypress audio signatures. The model then consists of a max pooling 2D layer to reduce the spatial dimensions of the feature maps, therefore improving the computational efficiency and mitigating overfitting. The max pooling layer retains the significant audio signature features and discards irrelevant noise or variations in the data which is essential for the relatively small dataset that this project generates. A dropout of 40% is applied to the model to prevent overfitting which is a considerable concern with the dataset. The layers are then flattened into one dimension and a dense layer of 128 neurons is then applied to capture the complex patterns from the keypress spectrograms extracted features followed by an output dense layer with 54 neurons representing the 54 labels with a sigmoid activation function to output probabilities for each target class. This approach yields a much smaller model consisting of just over 8 million parameters in contrast to the CNN/RNN approach that has over 36 million parameters. This model is much more efficient and less complex than the CNN/RNN approach and leads to more promising results.

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 19, 111, 32)	320
conv2d_13 (Conv2D)	(None, 19, 111, 128)	36,992
max_pooling2d_4 (MaxPooling2D)	(None, 9, 55, 128)	0
dropout_4 (Dropout)	(None, 9, 55, 128)	0
flatten_4 (Flatten)	(None, 63360)	0
dense_8 (Dense)	(None, 128)	8,110,208
dense_9 (Dense)	(None, 54)	6,966

Figure 10: CNN Model Approach Composition

The model is then trained with a learning rate of 0.001, a loss function of binary cross-entropy, and uses accuracy and precision as training metrics for a total of 15 epochs. Accuracy and precision are crucial metrics for evaluating keypress acoustic classification. The precision metric ensures the model reliably identifies actual keypresses and minimizes false positives which is important for distinguishing keypresses from noise or other acoustic events such as mouse clicks. Utilizing only accuracy, the model could achieve a high accuracy by classifying the most prominent class in the dataset most of the time. Without training with the accuracy metric, the model could have a high precision for when it did predict a keypress, but it may only predict very few keypresses out of the actual key presses. Utilizing both accuracy and precision metrics in contrast to the CNN/RNN approach leads to an optimal balance and ensures the model is robust in a variety of acoustic environments.

A significant issue that presented itself in the model was that the dataset being used to train with had some underrepresented classes which caused model bias towards more prominent classes. An optimal solution was determined to be calculating the weights of the classes in the dataset to encourage learning for the minority classes by modifying the loss function for each class based on the weight of each class in the dataset. This balances the overall accuracy of the model to be a more balanced evaluation of all classes in the dataset. This class weighting approach balances the model further increasing generalizability.

C. *Model Deployment*

The approach taken to optimize and deploy the model was initially to perform weight pruning, weight clustering, and quantization before converting the model to a TFLite model for embedded deployment. Unfortunately, performing weight

pruning and weight clustering proved to be challenging in the development environment since the "tensorflow_model_optimization" library conflicted heavily with the base Tensorflow installation even with proper versions being installed. This caused the project to pivot in optimization to only perform Quantization and conversion to a TFLite model.

Although the CNN model functions as intended, this project fell short of the goal of deployment on a Raspberry Pi 5 due to time constraints with the project deadline and the necessary work needed to buffer audio, preprocess the waveforms, convert the waveforms to STFT spectrograms, and to run model inference in parallel took longer than expected to complete before the deadline.

V. Results and Experiments

This section evaluates the performance of the CNN-based approach on the given problem and discusses an experimental configurations that may improve accuracy and precision. The CNN model results indicate overfitting in the model and a lack of data in the dataset which may be solved with different model layers or different hyperparameters. The model may need more collected data to learn the complex acoustic signatures of which was an oversight on this project.

A. Baseline Results

The baseline CNN model evaluated on the training dataset produces a 51% accuracy with 82% precision which is misleading. Looking at the per class true positive, true negative, false positive, and false negative evaluations illustrated by Figure 11, the model had very few false negatives in comparison to true negatives and had more true positives than false positives. Looking further into specific class predictions, for example the key label "g" shown in figure 12 has a true positive rate of 27.2% which is less than ideal. This shows that while the aggregate TP TN FP FN readings do not realistically imply that each key classification is equally represented by the model implying that the model may have overfitting problems due to the lack of data in the generated dataset.

These results indicate that more data should be collected for the model to learn complex patterns in key acoustic key data or have a model that more effectively learns these patterns.

Model Accuracy	51%
Model Precision	82%
Aggregate True Positives	465
Aggregate True Negatives	62,890
Aggregate False Positives	190
Aggregate False Negatives	715

Figure 11: Baseline Model Metrics

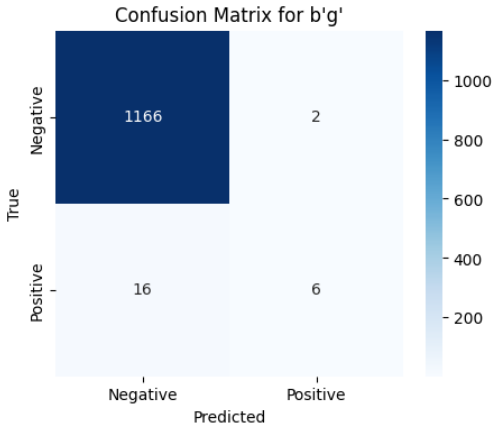

Figure 12: Baseline Model Confusion Matrix for label "g"

B. Experiment: Hyperparameter Tuning

An alternative mode approach that was taken to possibly enhance the strength of the model resulted in an exacerbated overfitting problem. In contrast to the baseline model, this model, illustrated in Figure 13, I modified the second convolution layer to have a filter size of 64 and added a dropout layer for 0.2, then a max pooling layer is added along with another convolution layer of filter size 128, a max pooling layer, and a dropout is applied of 0.4 before being flattened. Additionally, this model adds a 64-neuron dense layer before the output layer with a dropout of 0.4. The intention with this model change is to increase the strength of the model while attempting to solve the overfitting problem with dropout layers to influence the model to learn the complex audio signatures better.

The results of this approach were not promising, leading to an accuracy of 40.7% and a precision of 79.6%. The metrics of this approach were also less than ideal illustrated in Figure 14. The true positives in comparison to the true positives in the baseline model dropped by around 50% while the false negatives grew by ~200. Additionally for the same label "g" illustrated in Figure 15, the number of positive predictions drastically dropped to 9% which indicates that the overfitting problem was exacerbated by this approach. The solution to the problems in this project regarding accuracy and precision seem to be directly linked to a lack of acoustic keypress data collected.

Model Accuracy	40.7%
Model Precision	79.6%
Aggregate True Positives	251
Aggregate True Negatives	63018
Aggregate False Positives	64
Aggregate False Negatives	927

Figure 13: Hyperparameter Tuned Model Metrics

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 19, 111, 32)	320
conv2d_10 (Conv2D)	(None, 19, 111, 64)	18,496
dropout_9 (Dropout)	(None, 19, 111, 64)	0
max_pooling2d_6 (MaxPooling2D)	(None, 9, 55, 64)	0
conv2d_11 (Conv2D)	(None, 9, 55, 128)	73,856
max_pooling2d_7 (MaxPooling2D)	(None, 4, 27, 128)	0
dropout_10 (Dropout)	(None, 4, 27, 128)	0
flatten_3 (Flatten)	(None, 13824)	0
dense_9 (Dense)	(None, 128)	1,769,600
dense_10 (Dense)	(None, 64)	8,256
dropout_11 (Dropout)	(None, 64)	0
dense_11 (Dense)	(None, 54)	3,510

Figure 14: Hyperparameter Tuned Model Layer Composition

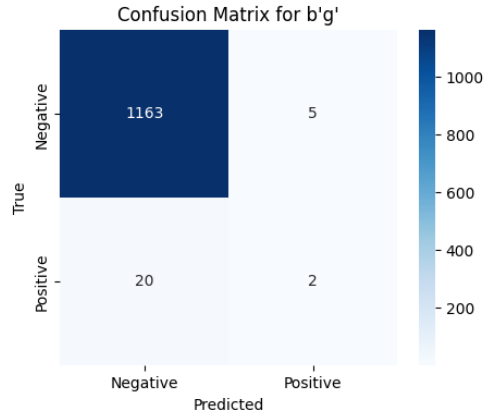

Figure 15: Hyperparameter Tuned Model Confusion Matrix for label "g"

VI. Conclusion

Demonstrated by the Experiments section, the project was a moderate success at classifying keypresses through their acoustic fingerprint. The approach of utilizing a solely a CNN instead of an RNN/CNN hybrid model drastically improved the model's ability to learn many complex auditory fingerprints of keypresses while not classifying abstract noise as keypresses leading to the success of this project. The project resulted in less than desirable results in accuracy and precision believed to be due to a lack of data in the collected dataset and as a result, an overfitting problem in the model since there was not enough data examples for the model to learn the very fine-grained details in the audio waveform. A large fallacy of the project was my approach to complete components such as the dataset labeling and preprocessing pipeline without comprehensive testing leading to issues that set the project backwards considerably. If I were to restart the project, I would ensure that each component of the project was tested, and a visual representation of the dataset would be available to understand the key problems that I would encounter. The project also fell short of its goal of deploying an embedded system due to time constraints with the project deadline. These key issues identified earlier lead to this time constraint and the model's deployment plan and compression would have permitted it to run inference on the Raspberry Pi. In the future, a better understanding of the data in the project and a more set forth plan would allow for much higher accuracy and precision while still maintaining an optimized workflow for deployment on embedded systems.

References

[1] K. M. Hama Rawf, A. O. Abdulrahman, H. O. Kamel, L. M. Hassan, and A. O. Ali, "Multi-datasets for different keyboard key sound recognition," Sci. Direct. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340924009090.

[2] J. Harrison, E. Toreini, and M. Mehrnezhad, "A Practical Deep Learning-Based Acoustic Side Channel Attack on Keyboards," arXiv, 2023. [Online]. Available: http://arxiv.org/pdf/2308.01074.

[3] IBM, "IBM Report: Escalating Data Breach Disruption Pushes Costs to New Highs," IBM Newsroom, 2024. [Online]. Available: https://newsroom.ibm.com/2024-07-30-ibm-report-escalating-data-breach-disruption-pushes-costs-to-new-highs.

[4] SPRITZ Research Group, Skype-Type: A Keyboard Acoustic Eavesdropping Tool. GitHub repository. [Online]. Available: https://github.com/SPRITZ-Research-Group/Skype-Type.

[5] Z. Dai, H. Liu, Q. V. Le, and M. Tan, "CoAtNet: Marrying Convolution and Attention for All Data Sizes," arXiv, 2021. [Online]. Available: https://arxiv.org/pdf/2106.04803.

[6] A. Compagno, M. Conti, D. Lain, and G. Tsudik, "Don't Skype & Type! Acoustic Eavesdropping in Voice-Over-IP," arXiv preprint arXiv:1609.09359, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1609.09359