Low Cost Gunshot Detection using Deep Learning on the Raspberry Pi

1st Alex Morehead
Computer Science, Mathematics, & Physics Department
Missouri Western State University
Saint Joseph, Missouri, USA
amorehead1@missouriwestern.edu

2nd Lauren Ogden
Computer Science Department
Columbia University
New York City, New York, USA
lao2125@columbia.edu

3rd Gabe Magee
Computer Science Department
Pomona College
Claremont, California, USA
glma2016@pomona.edu

4th Ryan Hosler
Computer & Info. Science Department
IUPUI
Indianapolis, Indiana, USA
rjhosler@iu.edu

5th Bruce White
AstroSensor.com
Santa Clara, CA, USA
b@astrosensor.com

6th George Mohler
Computer & Info. Science Department
IUPUI
Indianapolis, Indiana, USA
gmohler@iupui.edu

***Abstract*—Many cities using gunshot detection technology depend on expensive systems that ultimately rely on humans differentiating between gunshots and non-gunshots, such as ShotSpotter. Thus, a scalable gunshot detection system that is low in cost and high in accuracy would be advantageous for a variety of cities across the globe, in that it would favorably promote the delegation of tasks typically worked by humans to machines. A repository of audio data was created from sound clips collected from online audio databases as well as from clips recorded using a USB microphone in residential areas and at a gun range. One-dimensional as well as two-dimensional convolutional neural networks were then trained on this sound data, and spectrograms created from this sound data, to recognize gunshots. These models were deployed to a Raspberry Pi 3 Model B+ with a short message service modem and a USB microphone attached, using a software pipeline to continuously analyze discrete two-second chunks of audio and alert a set of phone numbers if a gunshot is detected in that chunk. Testing found that a majority-rules ensemble of our one-dimensional and two-dimensional models fared best, with an accuracy above 99% on validation data as well as when distinguishing gunshots from fireworks. Besides increasing the safety standards for a city's residents, the findings generated by this research project expand the current state of knowledge regarding sound-based applications of convolutional neural networks.**

***Index Terms*—Machine learning, neural nets, microprocessors and microcomputers, sound and music computing, signal processing**

I. Introduction

Public safety is undoubtedly a growing concern for a number of cities worldwide. According to [1], as it stands, emergency authorities are not made aware of up to 80% of gunshot incidents that occur. As such, gunshot detection technology is an increasingly essential field of research. Many cities utilize companies such as ShotSpotter to identify gunshots on a large scale by combining automated analysis of audio data in a city with human analysis and differentiation of gunshots and other noises.

While this and other technologies have successfully increased the percentage of reported gunshots in a city, these systems are extremely expensive to run and maintain, and they are not fully exempt from human biases. Thus, with elevated local crime rates and an increased workload on law enforcement officers, new, innovative technologies promoting task automation and delegation have the potential to reshape the landscape of smart city ecosystems.

One such technology that we have developed is an open-source pipeline utilizing artificial intelligence to detect gunshots across a city, with an array of microcomputers. The proper implementation of gunshot detection models to be used on a city-wide cluster of microcomputers enables complete automation of what previously required dedicated teams of human operators to achieve, since these types of classification tasks have previously been too difficult for machine learning models to carry out independently with high levels of precision and accuracy. Furthermore, this study demonstrates the capabilities of deployable deep learning architectures in sound classification when providing them with substantial amounts of diverse sound data for training.

A large amount of research [2]–[7] on the uses of convolutional neural networks (CNNs) in sound classification has been conducted before, where some works [8], [9] have compared alternative means of categorizing sounds. However, studies in CNNs have not yet fully explored the particular application of CNNs to gunshot detection in a city, leading to the following questions that we hope to answer through this study: How might machine learning models be used to autonomously identify and alert local authorities of the occurrence of gunfire in a city? What are the advantages and disadvantages of designing this kind of sound detection pipeline for microcomputers? And does training models on professionally recorded sounds translate to their being able to

We gratefully acknowledge the support of NSF grants REU-1659488, SCC-1737585, and ATD-1737996 for funding this summer research project.

Morehead, A., Ogden, L., Magee, G., Hosler, R., White, B., & Mohler, G. (2019). Low Cost Gunshot Detection using Deep Learning on the Raspberry Pi. 2019 IEEE International Conference on Big Data (Big Data), 3038–3044. https://doi.org/10.1109/BigData47090.2019.9006456

precisely classify real-world data?

II. The Data

A. Sources and Derivatives

We sought to answer these questions by first obtaining audio data from multiple sources, utilizing free internet databases such as Freesound and SoundBible. We also created our own audio clips using a microphone connected to a Raspberry Pi microcomputer, recording audio inside a lab, outside in suburban areas, and at a shooting range. We additionally constructed a generative adversarial network (GAN) to produce supplementary samples of gunfire and to prevent our model from overfitting to our compiled data set.

Besides clips containing audio of actual gunshots, our data set also consisted of a range of clips with audio of potential false positive sounds including fireworks, glass breaking, tapping on the microphone, clapping, and doors slamming. Lastly, we also have clips comprised solely of background noise such as white noise, static, and general urban sounds like dogs barking or people talking.

After acquiring a sizable amount of Waveform Audio (WAV) files, about 14,000 in total, each file was split into one or more two-second segments of audio. For audio clips less than two seconds long (such as the last slice from an audio file with a duration not evenly divisible into two-second partitions), we chose to pad those samples with an appropriate number of integer zeros until the sample's duration reached two seconds. Splitting all audio clips in such a fashion resulted in us having more than 15,000 uniformly-sized samples with which to train our models. The following table, Table I, displays a categorical analysis of our data once split into two-second clips. As seen in Table I, approximately 12% of all audio clips we compiled contain gunshots from an assortment of different firearms.

We then used multiple data augmentation methods in order to increase the size of our data set as well as to mitigate any chances our models might have to overfit to our data set. Detailed on the right in Table II is a list of the data augmentations we applied to each of our sound samples before training our models. For every two-second sample, five altered versions of said sample were created by applying each data augmentation function once to the original audio. The sample and its five derivatives were then added to a new cumulative list of sound samples. This culminated in us having six times our previous number of samples, more than 90,000 total audio segments.

Lastly, we then divided our data into three categories - training data, testing data, and validation data - in order to effectively train and evaluate our Keras models while also reducing our models' propensities to overfit. The ratios of data allocation we used for training data, testing data, and validation data were 64%, 20%, and 16%, respectively. While we experimented with weighting the samples recorded on the device's microphone higher than all other samples, we ultimately chose to weight all samples evenly.

TABLE I
Audio Samples Categorical Listing

Class	Percentage of Samples
Gun Shot	12.36
Jackhammer	4.47
Idle Engine	4.18
Siren	4.06
Fireworks	4.33
Drilling	4.02
Street Music	4.02
Dog Barking	4.02
Children Playing	4.02
Air Conditioner	4.02
Glass Breaking	2.33
Household	2.12
Car Horn	2.05
Foot Steps	0.78
Clapping	0.67
Applause	0.62
Microphone Tap	0.53
Static	0.13
Snapping	0.13
Door Slamming	0.12
Other	41.02

Fig. 1. A categorical breakdown of the types of audio data collected and used to train our models.

TABLE II
Data Augmentations

Time Shift	Pitch Change	Speed Change
Shifts a sound sample to the left or right by a randomly chosen amount less than 50% of the length and then fills in silence as needed.	Changes the pitch of a sample by a randomly chosen factor between 70% and 130%.	Alters the playback speed of a sample by a randomly chosen amount between 70% and 130%.

Volume Change	Background Noise Addition
Decreases the amplitude of a sample according to a uniformly random variable.	Introduces random background noise into a sample while making sure that no gunshots are added into a sample that does not originally contain gunshots.

Fig. 2. The data augmentations used on all aggregated sound samples.

III. Methodology

A. CNNs

Convolutional Neural Networks, or CNNs, are a neural network architecture designed to locate, model, and accurately predict patterns present in input data such as colored images or video. They are able to do so by iteratively sliding over small regions of input data and mapping any inherent properties in a region over to a proceeding layer in the network through the use of filters. Filters, also sometimes referred to as kernels, are matrices typically square in dimensionality that are multiplied by square patches of input data in order to reduce any latent features present in a patch over into a new representation. This process of mapping the features of incoming data into a different feature space is repeated up until the output layer of the network, that produces probability values corresponding to the possible output classes that the input data may belong.

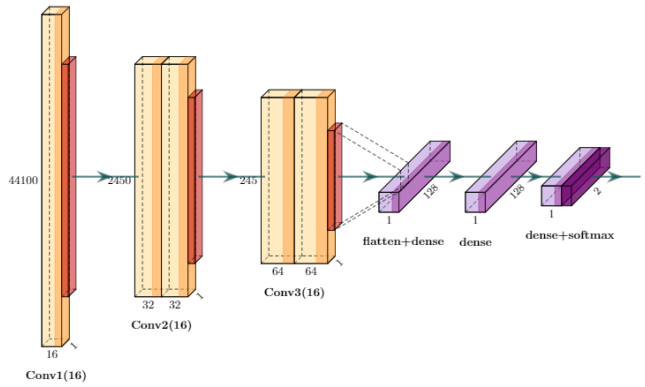

Fig. 3. An illustration of the architecture used by our 1D CNN.

Our one-dimensional (1D) CNN (architecture displayed in Fig. 3) was constructed with an input layer, four hidden layers, two fully-connected layers, and finally an output layer with softmax activation. The shape of the input tensor for the 1D CNN, (1 x 44100 x 1), corresponds to the dimensionality of a two-second audio clip. To follow are some of the parameters we used to train our 1D model. We used a hidden layer dropout rate of 25%, a 50% dropout rate for two fully-connected layers and a flattened layer, an initial learning rate of 0.1%, and a convolution window size of 16 units. We also used a batch size of 32 units and chose Adam as our optimizer function for its alacrity and efficiency during training. Moreover, as with all our models, we set the learning rate to reduce by a factor of 0.6 during plateaus in the learning phase.

Much like our 1D CNN, our two-dimensional (2D) CNNs (Fig. 4 and Fig. 5) were constructed with an input layer, four hidden layers, two fully-connected layers, and finally a softmax activation output layer. Our two 2D CNNs each took in different sized spectrograms of the data; thus the shape of the input tensor for our first 2D CNN was (1 x 128 x 64 x 1), while the input shape for our second 2D CNN was (1 x 128 x 128 x 1). For our two-dimensional models, we used many of

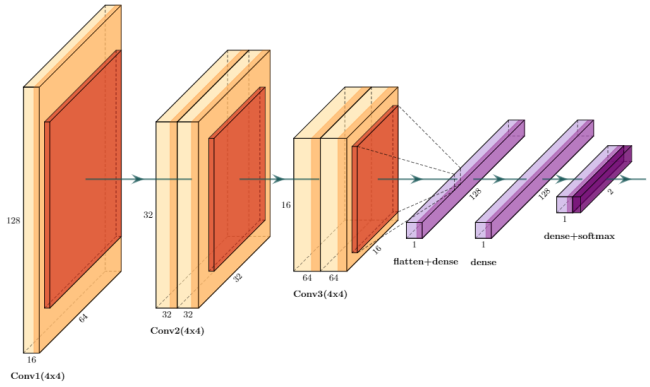

Fig. 4. A visualization of the architecture used by our 128 x 64 2D CNN.

the same parameters as used for the 1D model; we again used a hidden layer dropout rate of 25%, a fully-connected layer drop rate of 50%, an initial learning rate of 0.1%, but we now chose to use a 2D convolution window size of (4 x 4) units. Once again, we used a batch size of 32 units and chose Adam as our optimizer function.

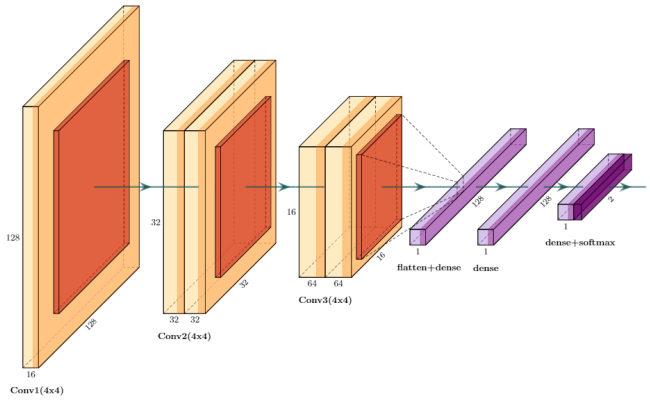

Fig. 5. A representation of the architecture used by our 128 x 128 2D CNN.

B. Spectrograms

While audio data can be represented as a time series of frequency values, it can also be represented visually as a spectrogram. Spectrograms are visual representations of the frequency and amplitude of sound over a specified span of time, and are constructed from an array of frequencies. In a spectrogram, the entry for each specific frequency-time Cartesian coordinate is an amplitude value. As there are numerous ways to interpret audio data, all with their own benefits and drawbacks, we created our three models as discussed above, each uniquely interpreting sound data in one of three formats: a 1D architecture that convolves on sound represented as a time series of frequency values; a 2D architecture that analyzes sound depicted as spectrograms with a length of 64 units; and a

2D architecture that takes input as spectrograms with a length of 128 units.

To acquire these spectrogram representations of our sound samples, we opted to use Librosa [10], an open-source Python library known for its convenience and versatility in audio analysis and manipulation. Librosa allowed us to pass our samples to a function that would then compute the appropriate Fourier transforms needed to compose a valid spectrogram. The following figures are graphical plots of the kinds of spectrograms we created using Librosa and then input into our models, with the first plot (Fig. 6) exhibiting the kind of wavelength patterns indicative of gunfire, and the second plot (Fig. 7) demonstrating the frequency and amplitude distributions of audio from a firework, a sound commonly mistaken for that of a gunshot.

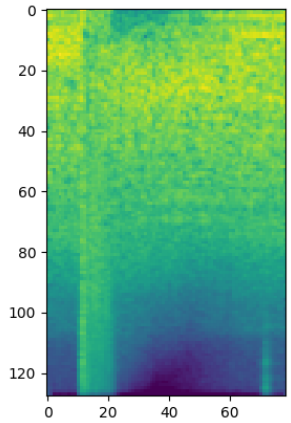

Fig. 6. An image of a spectrogram derived from a gunshot sound sample.

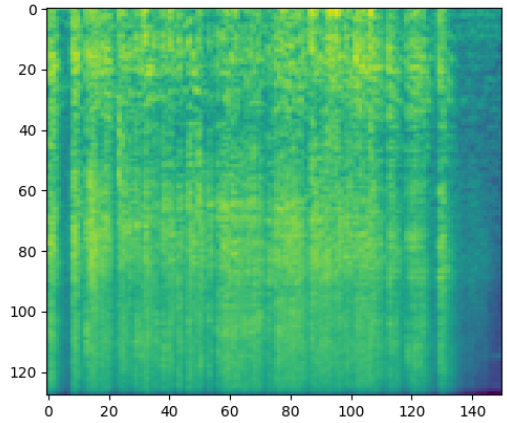

Fig. 7. An example of a spectrogram derived from a fireworks sound sample.

C. Training

As already alluded to, multiple CNNs were trained on our compiled data set, attained as described in the preceding section. While more descriptive labels for sounds other than gunshots could have been made available to our models for training (see Fig. 1 for an itemization of all available labels and their frequency in our data set), we chose to organize all non-gunshot audio clips into a single category, ”other”, in order to accommodate binary classification with our models. This resulted in a breakdown of 12.36% gunshot samples and 87.64% ”other” samples.

While no intensive sound preprocessing was requisite to train our 1D model, for our 2D models, each input sample was preprocessed to form a spectrogram before training. All three models were trained for 100 epochs or until the target metric for a training session did not improve for fifteen consecutive epochs. For our model architectures, the target metric being optimized for during training was validation accuracy. Other metrics like area under curve (AUC) and intersection over union (IoU) were also recorded and monitored during training and validation sessions, however they were not the primary objective statistics concerning the optimizer.

D. Hardware

Our short message service (SMS) pipeline for detecting gunshots was deployed on a Raspberry Pi 3 Model B+ connected to an AT&T USBConnect Lightning Quickstart SMS modem as well as a Sizheng omnidirectional USB microphone. In the interest of stability and software support for the advanced RISC machine (ARM) processor used by the Raspberry Pi, the operating system we set up for the device is Raspbian Stretch. While we originally hoped to deploy to an Arduino Uno or a comparable microcontroller, we chose to use a Raspberry Pi instead due to easier integration of Tensorflow and Keras on the Raspberry Pi as compared to the Arduino ecosystem, as well as the Raspberry Pi’s increased computing power and memory storage.

Fig. 8. A picture of our proposed Raspberry Pi hardware configuration.

We also experimented with a variety of USB microphones, including a Tanbin Super Mini USB Microphone and a Cyber

Acoustics CVL-1084 USB Desktop Microphone. We ultimately chose to employ the Sizheng USB microphone because of its improved audio quality and range, and still low cost relative to both the Tanbin microphone and the Cyber Acoustics microphone. The combined cost of purchasing all hardware necessary to run our software is approximately $150 USD, making it a cost-friendly unit and a much more affordable alternative to contemporary gunshot detection systems like ShotSpotter, which can annually run up to $90,000 USD per square mile of coverage.

E. Deployment

Each model was then deployed to the hardware described above. Our pipeline, orchestrated with Python, operates with three concurrent threads: one to continuously capture audio received from an attached microphone and put two seconds worth of said audio onto an audio analysis queue; one to analyze sound samples retrieved from the audio analysis queue and verify whether or not a gunshot occurred in a given sample; and finally one to dispatch an SMS alert message to a predetermined list of phone numbers if a gunshot was detected in the segment of audio most recently analyzed. Fig. 8 depicts the interaction and the flow of information between these threads in our pipeline.

Raspbian Stretch, a Linux distribution designed specifically for the Raspberry Pi, was chosen as the base operating system for our pipeline. However, as a result of diminished support for later versions of Python 3 in Raspbian Stretch, we chose to use Python 3.5 for the entirety of our project, including all project dependencies, as opposed to using what is currently the latest Python version, Python 3.7. This decision was made because the only official software repositories available in Raspbian Stretch do not at present house the updated packages necessary to run a newer version of Python 3 on the Raspberry Pi 3 Model B+. Similarly, because of the ARM architecture with which the Raspberry Pi 3 is equipped, update releases for Python libraries in Raspbian are quite behind their x86 architecture counterparts.

Besides having to use an older version of Python for development of the pipeline, we also needed to find and implement a reputable Python library for interfacing with the USB microphone connected to the Raspberry Pi. By virtue of its stability, wealth of features, and total possible configurations, the library we chose to use for this task was PyAudio [11]. Using this library, the data received from attached microphones is a stream of 32-bit float values. This data type works well with our models since the samples they were trained on are of the same type and format.

IV. Results

We found that all three of our Keras models performed well when forming predictions on our validation data set. This data set was intentionally separated from the rest of the training and testing data sets before training began. The best model configuration for achieving high validation metrics, as shown in Table III, was found to be an ensemble of all three models

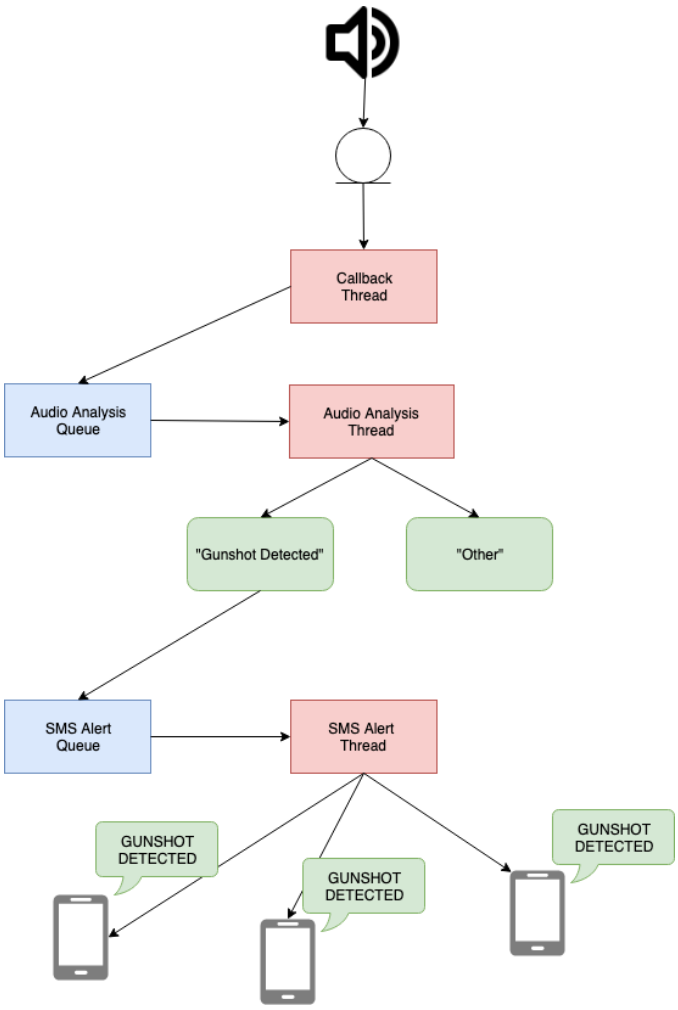

Fig. 9. A high-level overview of our gunshot detection pipeline.

using a majority-rules algorithm, in which an SMS alert is sent out if at least two of the three models positively identify the sound of a gunshot.

TABLE III
Keras Model Results

	1D CNN	2D CNN (64)	2D CNN (128)	CNN Ensemble
Accuracy	99.4%	99.4%	99.4%	99.5%
Precision	98.0%	97.1%	97.4%	97.9%
Recall	96.6%	97.6%	97.6%	98.0%
F1 Score	97.3%	97.4%	97.5%	97.9%
AUC	98.2%	98.6%	98.6%	98.9%
IoU	94.7%	94.9%	95.1%	96.0%

Fig. 10. The findings of a cross-evaluation technique applied to our original Keras models.

As presented in Table III, the 2D spectrogram models generally fared better than the 1D time series model in terms of recall, F1 score, AUC, and IoU. However, with a high

degree of accuracy and precision, our 1D model is able to correctly classify a plethora of gunshot noises in the midst of background noise and other auditory disturbances. Thus, we decided to continue incorporating the 1D model in our ensemble.

We also experimented with Tensorflow Lite (TFLite) conversions by converting each Keras model to its TFLite counterpart. Converting a model to TFLite takes advantage of quantization techniques to decrease the size of the model while it is stored in a persistent format (.tflite) and likewise while it is loaded into system random access memory (RAM). For a period of testing on one of our Raspberry Pi units, each model was loaded into system RAM from both its original hierarchical data format (H5) file as well as its converted TFLite counterpart file in order to compare the performances and behaviors of the two formats.

By running our Keras models and TFLite models through a suite of inference tests with our standardized validation set, we found marked differences in inference time measurements. While it takes significantly longer to load H5 models into system RAM as opposed to models in a TFLite format, employing the original Keras models in our pipeline did allow us to perform predictions with the Raspberry Pi more quickly. Nonetheless, the predictions from the Keras models and their TFLite counterparts rarely differed, leading to extremely similar, if not identical, metrics for the two models, as seen in the statistics listed in Table III and Table IV. Additionally, due to our project's adoption of the Raspberry Pi, a microcomputer with modest processing speeds and a relatively small amount of accessible RAM, compressing our models into the minimal amount of memory space, as TensorFlow Lite aims to do, was beneficial for our purposes. Thus, we chose to utilize the converted TFLite models in our final pipeline.

TABLE IV
TFLITE MODEL RESULTS

	1D CNN	**2D CNN (64)**	**2D CNN (128)**	**CNN Ensemble**
Accuracy	99.4%	99.4%	99.4%	99.5%
Precision	98.0%	97.1%	97.4%	97.9%
Recall	96.6%	97.6%	97.6%	98.0%
F1 Score	97.3%	97.4%	97.5%	97.9%
AUC	98.2%	98.6%	98.6%	98.9%
IoU	94.7%	94.9%	95.1%	96.0%

Fig. 11. The outcomes of a cross-evaluation technique applied to our converted TFLite models.

In our first round of testing at a live outdoor gun range in Indianapolis, we found that a modified version of our pipeline, one in which an alert was triggered if either of our two 2D models recognized a gunshot, functioned fairly competently. Nevertheless, of the approximately 280 two-second segments of audio that contained gunshots fired at this gun range, this particular version of our ensemble only positively identified 35 instances of gunfire, resulting in a very high precision metric, but a low recall score. After retraining all models on this data, upon a second trip to the same gun range, 158 out of 342 two-second audio clips containing gunshots were positively identified by our revised ensemble. All three models were again retrained on this new audio data, and we were once again successfully able to improve the recall rate of our models when predicting on our validation data set.

Experimenting at a live gun range also gave us the opportunity to test for edge cases and the effect of distance on our models' predictions. During our second round of testing at the gun range, we found that performance began to suffer at a distance of 225 meters away from the site of gunfire, and when stationed more than 300 meters away, our pipeline utilizing the ensemble method with all three models was no longer able to distinguish any true positive samples. This was most likely due to the ensemble being subjected to both an increased distance from the gunshots as well as a heightened level of background noise being intercepted by the Sizheng microphone as we moved farther away from the gun range.

Gunshot detection often suffers from a high false positive rate, particularly with mistaking the sound of a firework as the sound of a gun being fired. To test our TFLite models' certainties in the particular case of distinguishing between gunshots and fireworks, we ran inference on a collection of data containing only clips of audio from fireworks or gunshots, with the results of this measure displayed in Table V. As seen in the high precision metrics for all models including our ensemble method, our TFLite models had a very low false positive rate and thus are effective at distinguishing between gunshots and fireworks. Running inference on the same data set using our Keras models resulted in nearly identical performance metrics. We also were able to test the performance of our pipeline and models with actual fireworks; of 44 clips containing firework audio, our ensemble method produced only three false positives, confirming our ensemble's general ability to distinguish between the two.

TABLE V
TFLITE MODEL RESULTS FOR GUNSHOTS & FIREWORKS

	1D CNN	**2D CNN (64)**	**2D CNN (128)**	**CNN Ensemble**
Accuracy	98.4%	98.8%	98.8%	99.1%
Precision	99.8%	99.6%	99.7%	99.8%
Recall	96.6%	97.6%	97.6%	98.0%
F1 Score	98.2%	98.6%	98.6%	98.9%
AUC	98.2%	98.7%	98.7%	99.0%
IoU	96.4%	97.3%	97.3%	97.9%

Fig. 12. A survey of our TFLite models' effectiveness in distinguishing gunshots from fireworks.

V. SIGNIFICANCE & FUTURE WORK

The findings generated by this research project add to the current base of knowledge regarding sound-based applications of CNNs, and also provide insight into a compelling, new avenue of public safety measures. As stated before, while the use of these devices may reduce the amount of jobs that require

human input, we hope that they can make gunshot detection technology more affordable and accurate for a number of cities, thus increasing safety standards.

Most of our research was centered on creating models that can accurately identify gunshots, and deploying said models to a single Raspberry Pi microcomputer. In the future, in order to apply this new technology in a city, additional research must be done on the integration and interaction of multiple devices in a microcomputer array. Ideally, a feature we would next like to implement in our pipeline would allow for a robust localization of gunshots using a group of Raspberry Pi units. By having three or more devices positioned in relatively close proximity to each other, a server, by use of triangulation or a similar method, could utilize differences in the time of occurrence of the devices' alerts to determine where a gunshot might have occurred; previous research on localization of gunshots includes [12], [13]. Furthermore, a novel approach to localization using deep learning could be taken, in which an artificial intelligence model uses the time and volume differences from multiple units as input in order to predict the location of the gunfire. As the Raspberry Pi has no internal clock, in order to achieve the precise time measurements needed for localization, an internal time-tracking peripheral like the Adafruit DS3231 Precision RTC Breakout would need to be installed for each device.

Undoubtedly, more real-world evaluation of these models is required before officially deploying our pipeline into production in a city. While we had the opportunity to test the performance of our pipeline at a live gun range using different models at varying distances away from gunfire, further testing is needed in outdoor areas more similar to where such a unit would realistically be deployed, such as a high-crime urban neighborhood; gunshots in such an area will have different distributions, as opposed the highly-concentrated bursts found at a gun range, and will also have different echo patterns due to the new environment. What is more, the use case outlined for our pipeline necessitates the deployment of sensors that are self-contained and powered by a self-sustainable energy source, very likely solar energy. This question of how to best architect a functional microcomputer array will need to be explored in depth in a subsequent study.

Acknowledgment

We gratefully acknowledge the support of NSF grant REU-1659488 which provided research stipends, travel funds, and supply money for this summer research project. Thank you also to all faculty, staff, and personnel at Indiana University-Purdue University Indianapolis (IUPUI) who helped coordinate the 2019 Data Science Research Experience for Undergraduates (REU). We thank Tammy Kaser and the Indianapolis Metropolitan Police Department for hosting us at the IMPD firearms training range. This research was also supported by NSF grants SCC-1737585 and ATD-1737996.

References

[1] Irvin-Erickson, Yasemin, Bing Bai, Annie Gurvis, and Edward Mohr. "The effect of gun violence on local economies." *Washington, DC: Urban Institute* (2016).

[2] K. J. Piczak, "Environmental sound classification with convolutional neural networks," *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, Boston, MA, 2015, pp. 1-6. doi: 10.1109/MLSP.2015.7324337

[3] J. Salamon and J. P. Bello, "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification," in *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279-283, March 2017. doi: 10.1109/LSP.2017.2657381

[4] Takahashi, Naoya, Michael Gygli, Beat Pfister, and Luc Van Gool. "Deep convolutional neural networks and data augmentation for acoustic event detection." *arXiv preprint* arXiv:1604.07160 (2016).

[5] Lim, Hyungui, Jeongsoo Park, and Y. Han. "Rare sound event detection using 1D convolutional recurrent neural networks." In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*, pp. 80-84. 2017.

[6] K. Jaiswal and D. Kalpeshbhai Patel, "Sound Classification Using Convolutional Neural Networks," *2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Bangalore, India, 2018, pp. 81-84. doi: 10.1109/CCEM.2018.00021

[7] DAmiriparian, Shahin, N. Cummins, S. Julka, and B. W. Schuller. "Deep convolutional recurrent neural network for rare acoustic event detection." In *Proc. DAGA*, pp. 1522-1525. 2018.

[8] Prince, Peter, Andrew Hill, Evelyn Pia Covarrubias, Patrick Doncaster, Jake L. Snaddon, and Alex Rogers. "Deploying acoustic detection algorithms on low-cost, open-source acoustic sensors for environmental monitoring." *Sensors 19*, no. 3 (2019): 553.

[9] Shi, Bowen, Ming Sun, Chieh-Chi Kao, Viktor Rozgic, Spyros Matsoukas, and Chao Wang. "Compression of acoustic event detection models with low-rank matrix factorization and quantization training." *arXiv preprint* arXiv:1905.00855 (2019).

[10] Brian McFee, Vincent Lostanlen, Matt McVicar, Alexandros Metsai, Stefan Balke, Carl Thom, Colin Raffel, Dana Lee, Kyungyun Lee, Oriol Nieto, Jack Mason, Frank Zalkow, Dan Ellis, Eric Battenberg, , Ryuichi Yamamoto, Rachel Bittner, Keunwoo Choi, Josh Moore, Ziyao Wei, nullmightybofo, Pius Friesch, Fabian-Robert Stter, Daro Here, Thassilo, Taewoon Kim, Matt Vollrath, Adam Weiss, CJ Carr, and ajweiss-dd, librosa/librosa: 0.7.0. Zenodo, 08-Jul-2019. doi: 10.5281/zenodo.591533

[11] People.csail.mit.edu. PyAudio documentation PyAudio 0.2.9 documentation. (2016). Retrieved August 2, 2019 from https://people.csail.mit.edu/hubert/pyaudio/docs/

[12] M. A. Khalid, M. I. K. Babar, M. H. Zafar and M. F. Zuhairi, "Gunshot detection and localization using sensor networks," *2013 IEEE International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, Kuala Lumpur, 2013, pp. 1-6. doi: 10.1109/ICSIMA.2013.6717917

[13] A. K. Bandi, M. Rizkalla and P. Salama, "A novel approach for the detection of gunshot events using sound source localization techniques," *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boise, ID, 2012, pp. 494-497. doi: 10.1109/MWSCAS.2012.6292065