Introduction to Machine Learning - Gunshot Detection Project

Ibrahim Eksheir
Lee College of Engineering
UNC Charlotte
Charlotte, NC 28223
ieksheir@uncc.edu

Carlos Urias
Lee College of Engineering
UNC Charlotte
Charlotte, NC 28223
curias@uncc.edu

Jacinto Martinez
Lee College of Engineering
UNC Charlotte
Charlotte, NC 28223
jmart333@uncc.edu

***Abstract*—The purpose of this project was to utilize a machine learning approach in CNNs (Convolutional Neural Networks) to aid in forming implementable solutions to a modern problem - gunshot detection. This approach consists of a CNN model survey where pre-existing models and datasets are gathered, trained, evaluated, and analyzed to better understand the machine learning theories and processes involved in gunshot detection.**

Disclaimer: The models presented in the Researched Models section were sourced for research purposes only. We do not claim any form of ownership or credit for these models nor their code. All references to their original authors are included in the References section.

I. Introduction

With recent shootings, gun law debates, and gun awareness at the forefront of influence for many conversations, rapidly advancing technologies like machine learning are of great utility. In particular, the application of machine learning in the realm of gunshot safety through detection is one heavily considered and explored. The ability to make use of advanced models such as CNNs to correctly classify gunshot sounds is one that holds many applications within the gun conversation domain. The following report outlines proposed solutions to gunshot detection by presenting and exploring existing audio classifying CNNs and running trainings to analyze gunshot detection solutions.

II. Datasets

A. UrbanSound8k

This dataset consists of 8,372 different labeled audio files. Each file belongs to one of 10 distinct audio classes with a duration of 4 seconds or less. The distribution of the audio classes is shown in Figure 1 below.

Along with the class label, each audio file contains a .wav file of the audio recording itself , start and stop time indexes, and a salience value indicating its foreground or background presence in the audio. These files are shuffled into 10 separate folds, each containing a random mix of different class files. All files are provided in a .csv file [1].

B. SESA

The Sound Events for Surveillance Applications (SESA) dataset is a dataset created from a variation of .wav audio files obtained from Freesound. The dataset consists of four audio classes: gunshots, sirens, explosions, and causal sounds that are not deemed "dangerous". The audio files in the dataset come pre-divided into two folders, one for training and the other for validation, with an 80-20 train and validation split. In each training/validation file, there are four folders each holding audio files belonging to one of the four classes. All files included are mono-channel, 16 kHz, and 8-bit audio files [2].

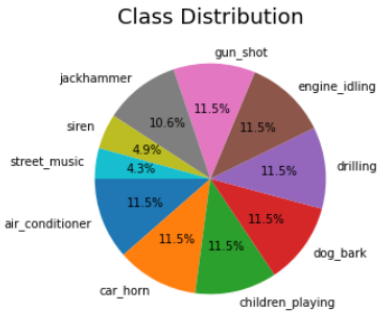

Fig. 1. Audio Classes Distribution

III. Data Pre-Processing

A. Standardization

1) Channel Unification: Stereo audio is recorded using 2 channels, while mono audio is recorded using only 1 channel. Channel unification is the process of converting all audio files of interest into the same type. [3] This process can be accomplished by either discarding a channel in the case of converting to mono or duplicating the audio to a second channel in the case of converting to stereo.

2) Data Resampling: Signal resolution is dependent on the frequency at which the original signal is sampled. Higher sampling rates capture more of the signal values, thus producing higher resolution. Resampling is the process of sampling a signal at a frequency different than its original sampling frequency [4]. This process can aid in scaling lower resolution and higher resolution signals to ensure they contain equal amounts of sample data points.

3) Audio Resizing: Audio resizing is used to scale signals with different duration to a uniform time duration. Signals with shorter duration are padded with values of 0 to increase the duration while conserving all original information [5].

B. Data Augmentation

1) Time Shift: Time shifts are applied to signals to augment the signal shape while retaining all signal information [6]. This is done by introducing a shift in the time axis, similar to a phase shift with respect to the signal. Any portions of the signal shifted outside of the time duration window are then wrapped back to the beginning of the signal's time window, preserving all original signal information.

2) Mel Spectrograms: Mel Spectrograms are a form of spectrogram that represent amplitude variation over time at different frequencies on the Mel Scale in order to capture inconsistencies in human perception of pitch changes at higher frequencies [7]. From these spectrograms Mel Frequency Cepstral Coefficients can be obtained.

3) MFCCs: Mel Frequency Cepstral Coefficients (MFCCs) are feature coefficients that contain descriptive information of the rate of change in different spectral bands within a signal. These coefficients are calculated by performing the Fast Fourier Transform of the Mel Spectrogram. MFCCs with negative values indicate that the majority of spectral energy is concentrated in the low frequency regions of the spectral band. Likewise, positive MFCCs indicate a concentration of spectral energy in the high frequencies of that spectral band [8].

4) Time and Frequency Masking: Time and frequency masking are data augmentation methods that involve augmenting a set of spectogram data. Frequency masking refers to the random removal of frequencies by masking them with horizontal bars, hiding a portion of the data and thus removing some of the continuity that comes with the data. Time masking is similar to frequency masking, but instead random time ranges in the spectrogram are blocked with vertical lines [9].

IV. Researched Models

A. Background

Research was conducted on existing CNN implementations to approach the problem of gunshot detection. In particular two audio classifying models were studied. The first model [10] takes a binary classification approach to detect the presence or absence of gunshots. The second model [11] is a multi class classifier capable of distinguishing between various audio classes. Both models were researched, trained, and analyzed.

B. Binary Audio Classifier

1) Dataset and Preprocessing: For the training of the model, it was required that audio files be 16-bit, Mono-Channel, 16 kHz, .wav files. Conflicting formats produced errors which would stall training. The obtained gunshot dataset from SESA was converted from 8-bit to 16-bit to fit model training parameters. SESA files were 16kHz and mono-channeled by default. The model converted all audio files to Spectograms then mapped them to Mel Spectograms for feature extraction.

2) Model Structure: YAMNet is a pre-trained audio event classifier that employs the MobileNetV1 depthwise-separable convolution architecture [12]. It can use an audio waveform as input and make independent predictions for each of the 521 audio events from the AudioSet corpus [13]. It consists of 86 layers which include ReLU as its activation function and employs an batch normalization layer which standardizes inputs between layers and drastically aids in training time [14]. Figures 2 and 3 give a visual into the architecture of the model.

YAMNet Body Architecture. Conv: Convolution. dw: Depthwise. pw: Pointwise.

Fig. 2. YAMNet Model Architecture

1	'input_1'	Image Input	96×64×1 images
2	'conv2d'	Convolution	32 3×3×1 convolutions with stride [2 2] and padding 'same'
3	'b'	Batch Normalization	Batch normalization with 32 channels
4	'activation'	ReLU	ReLU
5	'depthwise_conv2d'	Grouped Convolution	32 groups of 1 3×3×1 convolutions with stride [1 1] and padding 'same'
6	'L11'	Batch Normalization	Batch normalization with 32 channels
7	'activation_1'	ReLU	ReLU
8	'conv2d_1'	Convolution	64 1×1×32 convolutions with stride [1 1] and padding 'same'
9	'L12'	Batch Normalization	Batch normalization with 64 channels
10	'activation_2'	ReLU	ReLU
11	'depthwise_conv2d_1'	Grouped Convolution	64 groups of 1 3×3×1 convolutions with stride [2 2] and padding 'same'
12	'L21'	Batch Normalization	Batch normalization with 64 channels
13	'activation_3'	ReLU	ReLU
14	'conv2d_2'	Convolution	128 1×1×64 convolutions with stride [1 1] and padding 'same'
15	'L22'	Batch Normalization	Batch normalization with 128 channels
16	'activation_4'	ReLU	ReLU
17	'depthwise_conv2d_2'	Grouped Convolution	128 groups of 1 3×3×1 convolutions with stride [1 1] and padding 'same'
18	'L31'	Batch Normalization	Batch normalization with 128 channels
19	'activation_5'	ReLU	ReLU
20	'conv2d_3'	Convolution	128 1×1×128 convolutions with stride [1 1] and padding 'same'
21	'L32'	Batch Normalization	Batch normalization with 128 channels
22	'activation_6'	ReLU	ReLU

Fig. 3. YAMNet Layers

3) Training and Results: During training, various parameters were probed to observe effects on accuracy. These parameters include training sample frame steps, training sample frame lengths, batch sizes, and epochs. Running various epochs during training assisted in loss decrease, afterwards, a saturation point was reached. It was found that increasing training sample frame steps decreased accuracy and increased loss, while increasing frame length increased overall accuracy and decreased loss up until it starts increasing again. The following graphs - Figures 4, 5, 6, 7 - illustrate the change in accuracy and loss when varying these parameters.

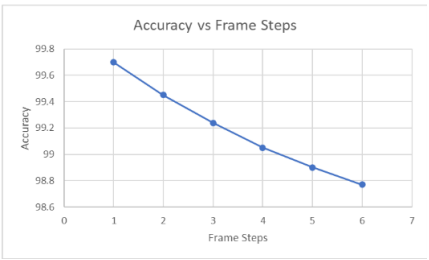

Fig. 4. Training Accuracy with Adjusted Training Frame Steps

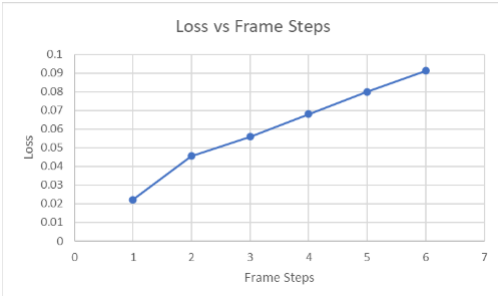

Fig. 5. Training Loss with Adjusted Training Frame Steps

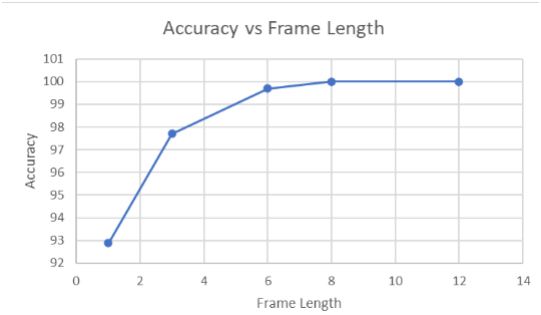

Fig. 6. Training Accuracy with Adjusted Training Frame Length

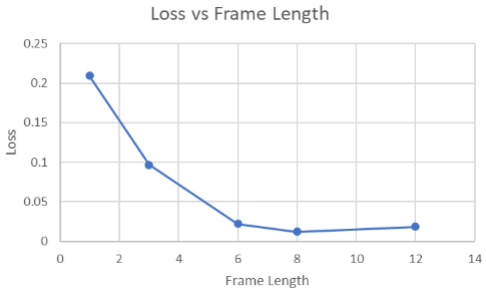

Fig. 7. Training Loss with Adjusted Training Frame Length

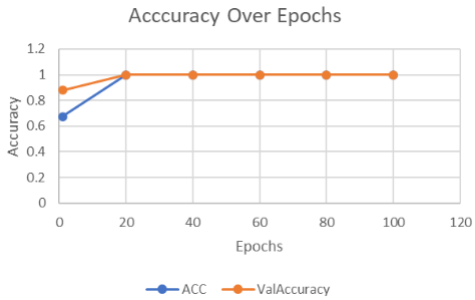

Fig. 8. Accuracy Over Epochs

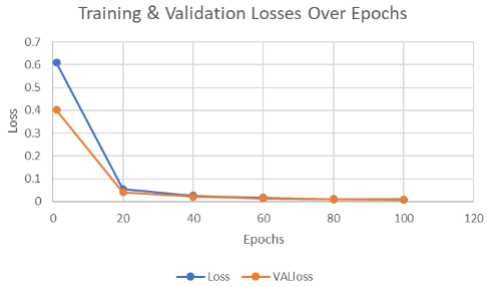

Fig. 9. Training and Validation Loss over Epochs

```
[ ] print('Evaluating the model')
    model.evaluate(test_data)

    Evaluating the model
    9/9 [==============================] - 23s 2s/step - loss: 0.0217 - acc: 1.0000
    [0.021694114431738853, 1.0]
```

Fig. 10. Evaluation Loss and Accuracy

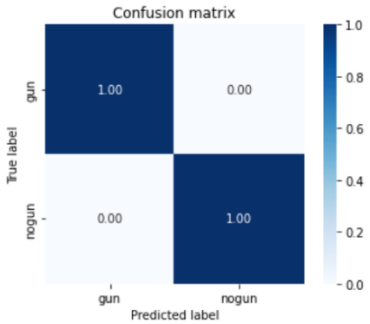

Fig. 11. Evaluation Confusion Matrix

After applying new input (in this case the SESA dataset), the binary classifier finished training in a matter of minutes. The final evaluation and generated confusion matrix results for the trainings are presented below in Figures 10 and 11. The rapid learning of the model has also been plotted using the reached optimized parameters of 1 step per training sample and 6 second frame lengths, and can be visualized in Figures 8 and 9 where both loss and accuracy improve with epoch iterations.

Figures 12 and 13 highlight the final accuracy on two .wav audio files, one being a thunder recording and the other a distant gunshot. It is important to note how the two outputs of the model and the splitting of frames of an audio serve in verifying the accuracy of newly defined classes. The actual accuracy roughly 98 percent for the gun audio and 88 percent for the thunder audio. While the thunder resembles a gunshot audio in many ways, the binary classifier was successful in detecting the gunshot audio.

```
Result of the window ith:  your model class -> score,  (spec class -> score)
Result of the window 0:         gun -> 0.739,   (Outside, rural or natural -> 0.200)
Result of the window 1:         nogun -> 1.000,         (Vehicle -> 0.303)
Result of the window 2:         nogun -> 1.000,         (Thunderstorm -> 0.932)
Result of the window 3:         nogun -> 1.000,         (Thunderstorm -> 0.649)
Result of the window 4:         nogun -> 1.000,         (Thunderstorm -> 0.990)
Result of the window 5:         nogun -> 0.999,         (Thunderstorm -> 0.289)
Result of the window 6:         nogun -> 0.999,         (Wind -> 0.206)
Result of the window 7:         nogun -> 0.966,         (Rustle -> 0.454)
Result of the window 8:         nogun -> 0.996,         (Outside, rural or natural -> 0.300)
Result of the window 9:         nogun -> 0.660,         (Silence -> 0.289)
Mean result: nogun -> 0.8881996273994446
```

Fig. 12. Audio Classes Distribution

```
Result of the window ith:  your model class -> score,  (spec class -> score)
Result of the window 0:         gun -> 0.999,   (Outside, rural or natural -> 0.323)
Result of the window 1:         gun -> 0.931,   (Outside, rural or natural -> 0.501)
Result of the window 2:         gun -> 0.997,   (Rustle -> 0.667)
Result of the window 3:         gun -> 0.999,   (Pink noise -> 0.591)
Result of the window 4:         gun -> 0.983,   (White noise -> 0.446)
Result of the window 5:         gun -> 0.978,   (Outside, rural or natural -> 0.326)
Result of the window 6:         gun -> 0.991,   (Outside, rural or natural -> 0.543)
Result of the window 7:         gun -> 0.999,   (Rustle -> 0.618)
Result of the window 8:         gun -> 0.999,   (Rustle -> 0.564)
Result of the window 9:         gun -> 0.970,   (White noise -> 0.382)
Mean result: gun -> 0.9846836924552917
```

Fig. 13. Audio Classes Distribution

C. 10-class Audio Classifier

1) Dataset and Preprocessing: The second model analyzed was a 10-class audio classifier. The dataset used for this particular model was the UrbanSound8k dataset mentioned in Section II-A. The .csv file provided with the data set is used to create the data points X (the path of each audio file), and their target Y (the class the audio file belongs to).

In order to take the 8500+ audio files in this dataset and be able to pass them through a model to learn about them, a series of data pre-processing steps are required. These audio files all hold various lengths, sampling rates, bit sizes, and number of channels [1]. Thus, the first step taken is standardizing and unifying all the various specifications of the audio files.

Firstly, the number of channels each audio file holds is considered. The audio files in the dataset are a mixture of mono-channel audio files and stereo-channel audio files. In order to standardize the channels the channel unifying process mentioned in Section III-A1 is performed. All stereo-channel audio files are preserved as such while the mono-channel audio files are converted to stereo-channel.

Audio files in the dataset come with a different sampling rate depending on its source. This led to the next data pre-processing step to be re-sampling all the audio files in the dataset through the process described in Section III-A2. The sampling rate to which the audio files were re-sampled to is a uniform $44.1kHz$ for all files. The reason the sampling rate used by this model and the one also commonly used when sampling audio files for signal processing purposes is $44.1kHz$ is due to the Nyquist Rate and how the human ear perceives sound [15]. The human ear can perceive sound signal at an average rate of $22.05kHz$, and by Nyquist's theorem stating that the sample rate of a signal should be double in order to prevent aliasing, or loss of data.

The next process that takes place is the resizing all the audio files in the dataset. All audio files are resized to the size of the largest audio file in the dataset. Extra space is added to the audio file by padding the file with zeros, in this case representing pure silence.

The next portion of data pre-processing is data augmentation which aims to add variance to the dataset which helps add robustness to the training process. A time shift process described and explained in Section III-B1 takes place. In this process a random time shift, to the left or to the right, is applied to each audio file in the dataset and any excess information is wrapped around. This process is done with the particular purpose of adding variance to the data.

The next process converts the unified and standardized audio files from the datset into a form which can be fed into 10-class classifier and provide feature information that can be learned by the model. This is the process that takes place in Section III-B2 where the audio files of the dataset are converted into Mel Spectograms. The result of this step is files with three dimensions: time x frequency x channels, which are in a proper shape that can be passed to the model. An example of one of the spectograms generated from the dataset can be seen below in Figure 14.

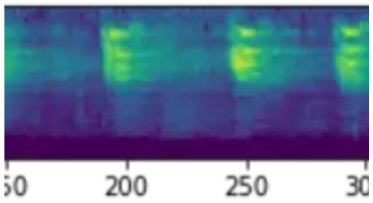

Fig. 14. Spectogram

The last data pre-processing that takes place is the time and frequency masking described in Section III-B4 which adds even more variance to the dataset by adding random vertical and horizontal masks on the spectograms. An example of time and frequency masking applied to one of the spectograms generated from the dataset can be seen below in Figure 15.

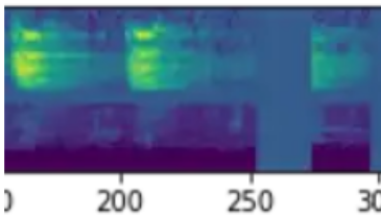

Fig. 15. Time and Frequency Masking

2) Model Structure: After pre-processing the audio files through standardization, data augmentation, and made into an acceptable form, they are passed onto the various layers of the model. The model consisted of four convolutional layers who's input is the three dimensional spectograms explained in IV-C1. The four convolutional layers apply various filter to then produce a feature map consisting of the learned features pertaining to the data. Each of the convolutional layers utilize a set of functions and filters, some of which are: 2D Batch

normalization, ReLU activation functions, Adam optimizers, and Cross Entropy Loss. To briefly elaborate, the Adam optimizer is used to optimize the model's parameters based on gradients of the loss function [16] and is considered to be a reliable optimizer in deep learning. The loss function in this case is the Cross Entropy Loss which is a function that calculates the loss used in back propagation. This function in specific pertains to multi-class classifier models [17]. Batch normalization in general is used to standardize the input to each neural network layer by applying this standardization to batches of the data instead of as a whole, which plays a significant role when storage and computation limitations are at present [14]. The feature map output from the four layers is then passed into a linear classifier. This model's structure holds 11 layers and produces around a 138 million parameters

3) Training and Results: The model was then ran through a training loop. The number of epochs was initially set to 20, however the results did not show convergence so the model was trained again with the number of epochs increased to 50. Convergence was seen starting at around 40 epochs and remained almost constant past 45 epochs. The training loss was analyzed per epoch and the average was calculated accordingly as the iterations increased. The training accuracy was calculated by taking the ratio of correct predictions over the total number of predictions for the specified epoch running. Final training loss was 0.49 and final training accuracy was 0.84. Loss plots over the epoch iterations can be seen in Figure 16 and 17 below. A validation function was ran in order to also record the validation accuracy for how accurate the model can correctly label data it had not seen during training. The final validation accuracy was seen to be 0.83.

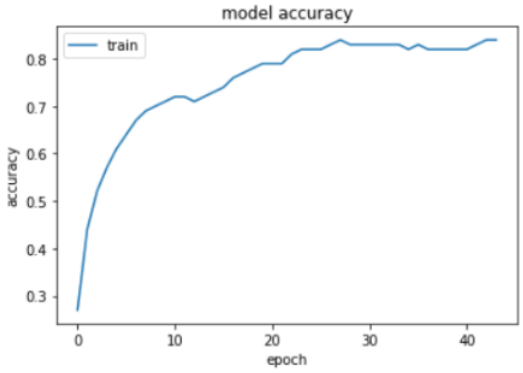

Fig. 16. Training Accuracy

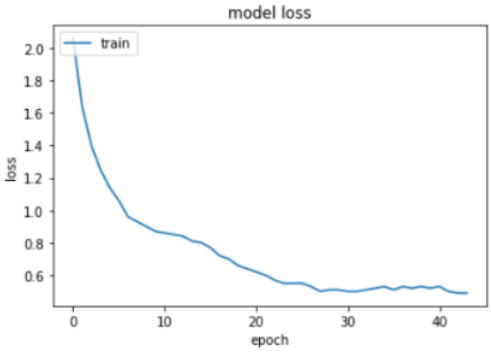

Fig. 17. Training Loss

V. Developed Model

Attempts were made to build a functioning audio classifier model from scratch on the PyTorch framework that would utilize pre-processing techniques similar to those of the 10-class classifier, namely MFCC calculation and re-sampling. However, this endeavor was not achieved due to time constraints and management. Ideally, the proposed model would have been trained with the UrbanSound8k dataset and made use of 2D convolution, ReLU activation, and max 2D pooling layers for its architecture. The code for the developed model, along with the two researched models, is included in the GitHub repository link provided in this report. More time and testing will need to be invested in the future to obtain a functioning model.

VI. Results and Applications

One key aspect that was noted with respect to the researched models was the frameworks on which they were constructed. The 10-class classifier was built on PyTorch, while the binary classifier was built on TensorFlow. TensorFlow has the capability to export models to TensorFlow Lite versions, producing lightweight models that can operate with fewer computational resources. Incorporating such models onto compatible MCU edge devices would be of great benefit in helping reduce emergency response times by quickly and accurately detecting gunshots and alerting authorities through wireless connection. Although such an outcome was not produced in this project, it remains an application of interest to both the team and public safety.

VII. Conclusion

In this project two audio classifiers were explored, trained, and analyzed. Although the original project proposal goal of producing a functioning model was not met, gaining an understanding of theories as they apply to audio classification throughout this project was immensely beneficial. There are many possibilities in integrating machine learning to aid in community safety. Much was learned from the research presented in this report. New insight was gained on more advanced data processing techniques as well as the theory behind them and a deeper understanding on sophisticated model layers and their functions was also attained. For future

iterations of this project, dedicating more time to constructing an original functioning model would be a major improvement as well as conducting more in depth research on deployment of models for in-field applications.

VIII. GitHub Repository

https://github.com/jacintomart/4105/tree/main/Project

References

[1] J. Salamon, C. Jacoby, and J. P. Bello, "Urbansound8k: A dataset for research on soundscapes and sound-based classification." Available at https://urbansounddataset.weebly.com/urbansound8k.html, 2017.

[2] W. Wong, S. Chen, and Y. Lo, "Learning sound event detection from weakly labeled data," 2019.

[3] M. Idrayana, "Why the difference between mono and stereo audio files is important for your game." Available at https://medium.com/double-shot-audio/ (2020/04/10).

[4] G. Brown, "Digital audio basics: Sample rate and bit depth." Available at https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html (2021/05/10).

[5] N. Instruments, "Zero padding." Available at https://www.ni.com/docs/en-US/bundle/labview/page/lvanlsconcepts/lvac_zero_padding.html.

[6] M. T. NTNU, Department of Music, "Distortion: Wrapping." Available at http://gdsp.hf.ntnu.no/lessons/3/, 2014.

[7] P. Fuller, "Librosa: A python audio library." Available at https://medium.com/@patrickbfuller/librosa-a-python-audio-libary-60014eeaccfb (2019-05-28).

[8] J. Hui, "Speech recognition: Feature extraction (mfcc, plp)." Available at https://jonathan-hui.medium.com/speech-recognition-feature-extraction-mfcc-plp-5455f5a69dd9 (2019/08/28).

[9] N. Messitte, "What is frequency masking?." Available at https://www.izotope.com/en/learn/what-is-frequency-masking.html (2022/12/17).

[10] TensorFlow, "Model maker: Audio classification." Available at https://colab.research.google.com/github/googlecodelabs/odml-pathways/blob/main/audio_classification/colab/model_maker_audio_colab.ipynb, 2022.

[11] K. Doshi, "Audio deep learning made simple: Sound classification step by step." Available at https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5 (2021/03/18).

[12] MathWorks, "Yamnet: Deep neural network for sound event classification." Available at https://www.mathworks.com/help/audio/ref/yamnet.html.

[13] T. Spadini, "Sound events for surveillance applications." Available at https://zenodo.org/record/3519845#.Y50i1ezMJpR (2019/10/20), 2019.

[14] J. Brownlee, "A gentle introduction to batch normalization for machine learning." Available at https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/ (2019/01/16).

[15] H. Pierce, "Using a sampling rate of 44.1 kHz." Available at https://www.cardinalpeak.com/blog/why-do-cds-use-a-sampling-rate-of-44-1-khz.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[17] J. Brownlee, "A gentle introduction to cross-entropy for machine learning," *Probability*, October 2019.