Classification of Environmental Sounds with Deep Learning

Coursework for PHAS0056: Practical Machine Learning for Physicists

Department of Physics and Astronomy
University College London
March, 2024

Word Count: 3667

Contents

Abstract

The automatic classification of environmental sounds has considerable potential to innovate in areas such as healthcare, security, and environmental monitoring. This study addresses the challenge of improving the accuracy of environmental sound classification through deep learning techniques. Utilizing the ESC-50 dataset, the efficacy of Mel spectrograms as features and raw audio waveforms in model training were investigated, supplemented by a variety of data augmentation strategies. The study compared the performance of different neural network architectures, including convolutional neural networks (CNNs) optimized for Mel spectrogram inputs and architectures tailored for raw waveform analysis. It was found that data augmentation played a crucial role in enhancing model accuracy and robustness. For Mel spectrograms augmentations that combined pitch shifting, time stretching, and noise addition provided the best classification accuracy 66.4% with the least standar-deviation of 0.3%. For raw audio waveforms it was found that AclNet_SC performed the best at 63.2% with standardeviation of 1.2%. Additionally, the analysis also revealed the importance of addressing class imbalance and feature extraction nuances to improve classification accuracy. The study's findings contribute valuable insights into the optimization of sound classification systems and suggest avenues for future research in the domain of deep learning-based audio analysis.

Methods	DSC↑	HD↓	Aorta	Gallbladder	Kidney(L)	Kidney(R)	Liver	Pancreas	Spleen	Stomach
SAMed Paper	**81.88**	**20.64**	**87.77**	**69.11**	80.45	**79.95**	**94.80**	**72.17**	**88.72**	82.06
PitSAM (Rank5)	80.07	21.32	87.59	58.64	**82.18**	79.66	94.52	68.14	87.20	**82.64**
SAMed Train	78.75	21.44	87.41	63.60	78.25	75.36	94.67	67.38	85.85	77.47
PitSAM	76.61	24.38	86.04	67.80	78.90	78.12	92.78	53.98	80.53	74.75

Chapter 1

Introduction

1.1 Background

Sound classification is the process of analyzing and categorizing audio into different classes or categories. In its most basic form it involves recognizing and distinguishing various types of sounds – such as speech, music, environmental sounds, and machine noises – and labeling them with predefined categories. The task requires sound recording, preprocessing, feature extraction and finally classification.

The general overview of sound classification tasks is as follows. The audio is recorded and labelled - weakly or strongly [1]. Then it is preprocessed using multiple techniques some of which include noise reduction, pitch shifting, time stretching/compression and more [2]. Then comes the feature extraction, where the raw time-series audio data is transformed into a format which is more easily analysable by machine learning algorithms, such as mel spectograms [2]. Then the features and/or the raw waveform is fed into machine learning algorithms, more recently deep learning methods, especially Convolutional Neural Networks (CNNs). Recently, the process has been becoming more complex [3] [4], however for the purposes of this report the above explanation is sufficient.

1.1.1 Applications, Challenges and Trends

Sound classification has a wide range of applications. It's used in environmental monitoring (identifying bird species from their calls [5]), healthcare (analyzing cough sounds [6] or heartbeats [7]), smart homes, security and surveillance, and consumer electronics (song recognition services [8], and etc.).

Despite its advancements, sound classification faces several challenges most of which have to do with deep learning. Dealing with background noise and ensuring robustness across different recording environments is a significant hurdle. Also, the need for large labeled datasets for training deep learning models can be a limitation. Further, distinguishing between similar sounding audio classes and handling the variability in sound due to different recording devices adds to the complexity.

Current trends in sound classification involve the use of more sophisticated neural network architectures, integration with natural language processing for better understanding of context, and efforts towards unsupervised learning approaches. There's also a growing interest in real-time sound classification.

Chapter 2

Methodology

2.1 Data Augmentation

2.1.1 Mel Spectrograms

The study used the ESC-50: Dataset for Environmental Sound Classification. The dataset contains 2000 audio files (`filename.wav`), split into 50 classes and 5 super-categories [9]. The files were loaded and processed in python with the Librosa [10] library. The signal were processed at sampling rate of 44.1 kHz and converted into Mel spectrograms. Since the dataset is small, only 2000 files for 50 classes, a lot of papers with high performing models perform some sort of data augmentation [9] [11] [12] [13]. For the classification of sounds using Mel spectogram feature, 5 different methodologies were implemented. An example of the augmentations to the `5-263501-A-25.wav` ('footstep' category) can be seen in Fig. 2.1. The following augmentations are in the `Sound Data Augmentations.ipynb` file. To ensure that the results were reproducible and the split was done in the same manner for the different datasets `train_test_split` with a `random_state = 42` was used with an 80/20 (training/testing) split.

1. **Method 1 - Applying Random Shift or Stretch Augmentations**: Using soundfile, numpy and librosa packages. For each spectrogram an additional data point with either a pitch shift or an audio stretch was saved. The total number of generated data is 2,000 images, 1,600 of which are for training. So, there are 3,200 training and 800 testing data points. In Fig. 2.1, where (a) has a a pitch shift, larger in value than (d).

2. **Method 2 - Applying both Shift and Stretch Augmentations**: The functions were re-used from above, however they were not applied randomly. For each spectrogram 2 - pitch shifted and time stretched - were generated. So, the total number of generated data is 4,000 images, 3,200 of which are used for training. So, for this augmentation there are 4,800 training and 1,200 testing data points. In Fig. 2.1, it can be seen that the time axis of (c) is compressed relative to (a), the peaks seen in (d) have a higher frequency bandwidth.

3. **Method 3 - Applying White and Pink Noise**: Functions for adding uniform (white) noise as well as (pink) noise lower at higher frequencies, which were scaled to a random signal to noise ratio - between 10dB and 17dB. Similarly to the 2nd augmentation the total number of generated data is 4,000 images, 3,200 of which are used for training. So, for this augmentation there are 4,800 training and 1,200 testing data points. The difference between (e) and (f) can not be seen in Fig. 2.1, due to limitations in resolution.

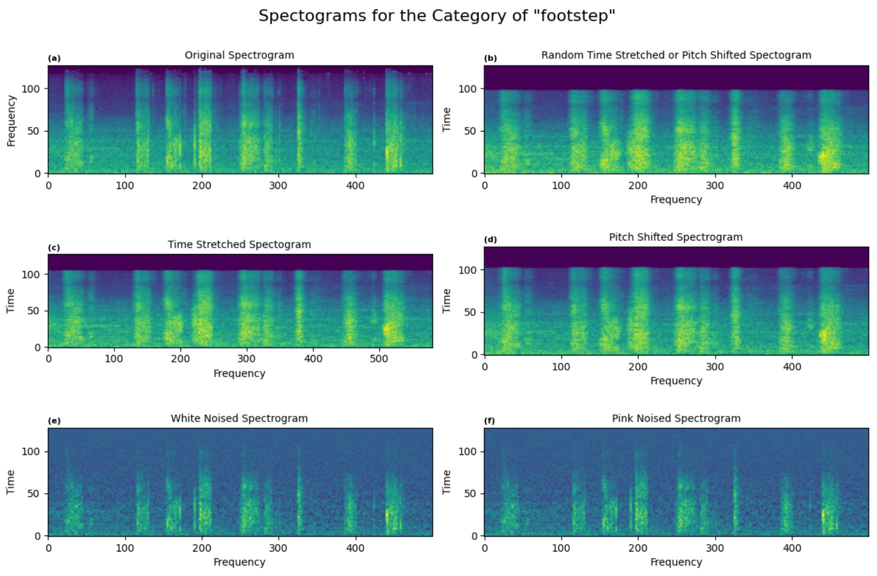


Figure 2.1: An example of data augmentation done to a sound file. Alphabetical labels correspond to specific augmentations: (a) original, (b) shift or stretch, (c, d) shift and stretch, and (e, f) white and pink noise.

Method 4 was implemented by combined the datasets in methods two and three. This means that it had 8,000 training and 2,000 testing images.

2.1.2 Raw Audio Wave-forms

Another set of augmentations were done, where the raw waveform signals were used as the inputs to the deep learning networks. Unlike the previous methodology where the augmentations were generated and stored, these were produce real-time while training. To do this a `class` called `CustomDataGenerator(keras.utils.sequence)` was defined according to specifications in Tokozume's [12] and Huang's [13] papers a was defined. First, a random 2s of audio within a training file was chosen. Then, it was centered to zero mean, and normalized by standard deviation. Then, resampled the waveform by a random factor uniformly chosen in range [0.85, 1.25], after which it was cropped to be from 1.25 to 1.75 seconds — optimal according to Piczak [9]. Finally, the waveform was multiplied by random gain chosen uniformly in range [-6.0, +6.0] dB. An example can be seen in Fig. 2.2.

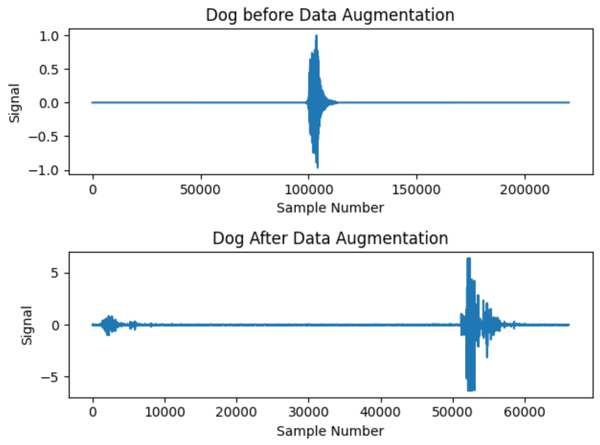


Figure 2.2: (b) is a small snippet of (a), which has been up-sampled and had a positive gain applied.

2.2 Model Architectures and Training

The models used, shared three features. Their final layers have a 'Softmax' activation function, they were compiled with the 'Sparse_Categorical_Entropy' loss function and their training history was evaluated with the metric of 'accuracy'.

2.2.1 Mel Spectrograms

Initially CNN architecture in Table A.1 was defined, however over the course of the study it was optimized into CNN3, in Table A.2. The model has a resizing layer which allows for files of variable input, by cropping all inputs. This allows the model to maintain tensor shape throughout layers and helps it generalize rather than focus on specific pixel locations. The model was compiled with the optimizer 'Adam' at its default learning rate 0.001.

During experiments it was found that CNN2 architecture was insufficient so CNN3 architecture was defined and optimized. It has removed the redundant 4th Conv2D layer, change the filter number of the second Conv2D layer added 'Batch Normalization' after each Conv2D layer stabilizing and speeding up the learning. In addition to this a 'Dropout' to prevent over-fitting, by randomly omitting neuron connections during the training process. Furthermore, 'L2 regularizer' (Ridge Regression) was applied across Conv2D and Dense layers penalizing weights reducing network complexity and improving the its performance for unseen data. Lastly, a learning rate scheduler callback was implemented after experimentation.

2.2.2 Raw Audio Waveforms

For this task model architectures from three papers were used and optimized. The initial approach to classifying sound categories was to take the entire raw audio waveform as the input to the neural network. The paper by Dai [14], tested five very deep convolutional neural networks architectures on the ESC-10 dataset, two of which were used (*m5* and *m11*) and are shown Appendix A in Tables A.3 and A.4. Additionally, a callback `ReduceLROnPlateau` was used to prevent non-convergence towards the minimum, because of large step size i.e. learning rate, and Dropout was added between the most densely connected parts of the networks.

As mentioned previously the data augmentation was done according to Huang's paper. So his proposed model *AclNet*, shown in Table A.5 was tested. AclNets can have layers set up in two different manners, called **Standard Convolution (SC)** and **Depthwise Standard Covolution (DWSC)**. The former is a convolutional layer followed by batch normalization and ReLu activation function, while the latter is a convolution decomposed into depthwise separable convolutions with pointwise layers followed by batch normalizatoin and ReLu activation function [13]. DWSCs have significantly fewer parameters reducing the model size and lessening overfitting. Consequently fewer operations are required, which speeds up the processing and reduces power consumption, making them suitable for mobile devices [15].

There were however issues with implementing his learning setting with the model due to lack of computational resources so 'AclNet' was trained using learning settings used by Tokozume [16]. The paper proposed using the momentum SGD optimizer, training for 150 epochs, and using a lr_scheduler with learning rate of 10^{-2} for the first 80 epochs, 10^{-3} the next 20 epochs, 10^{-4} the next 20 epochs, and 10^{-5} for the last 30 epochs.

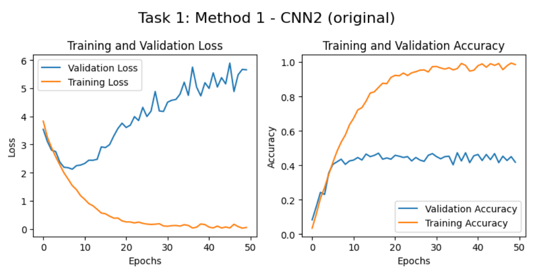
Chapter 3

Results

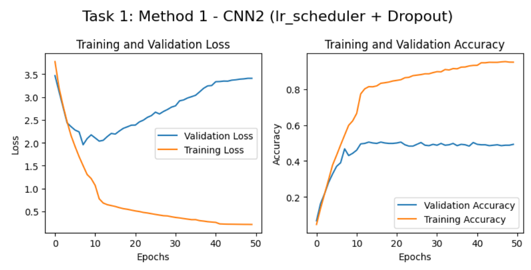
3.1 Mel Spectrograms

To evaluate how the data augmentations performed different metrics were compared. At first, a model was trained and its history was plotted. Then a confusion matrix for the model was constructed and in addition to accuracy, precision and recalls metric were calculated for each category. Finally, five models were trained and their average performance was recorded.

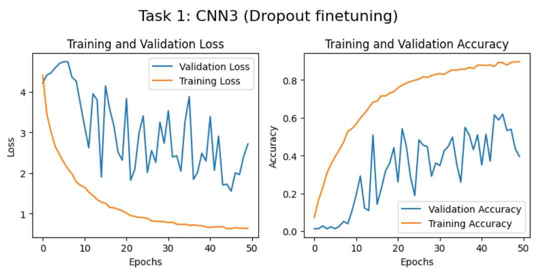
Most of the optimization and parameter tuning took place while training the models on method 1 augmentation, and can be seen in Fig. 3.1. The initial model CNN2, Fig. 3.1 (a), had trouble and its validation accuracy could not reach the 45% mark.



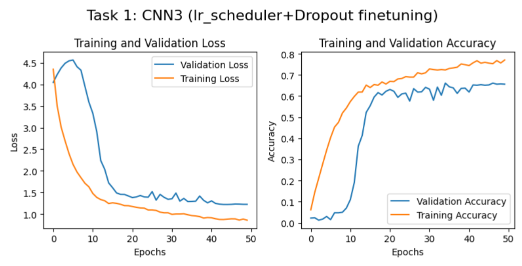
(a) History for the original CNN2



(b) History for CNN2 with lr_scheduler & dropout



(c) History for CNN3 with higher dropout



(d) CNN3 with higher dropout & lr_scheduler

Figure 3.1: Shows the training histories of the model optimization process.

The model was not successful at capturing features. In turn CNN3 was defined, which performed simillarly to CNN2. However, when the dropout rate was increased the CNN3 did not plateau, (c).

A lr_scheduler stabilized the stochasticity and converged the accuracy to around 65%, as seen in (d). These optimizations had no effect on CNN2 as seen in (b). So, CNN3 trained for 50 epochs, batch_size 64 and a lr_scheduler was used for the experiments using Mel spectrograms as input. After training the models for each of the data augmentation methodologies, their average accuracy and standardeviation was found as seen in Table 3.1.

Table 3.1: Scores per Training Session for CNN3 with different inputs

Session	**Original**	**Method 1**	**Method 2**	**Method 3**	**Method 4**
	Accuracy	**Accuracy**	**Accuracy**	**Accuracy**	**Accuracy**
1	56.50 %	62.37 %	63.25 %	60.25 %	66.80 %
2	56.75 %	62.25 %	65.83 %	60.42 %	65.95 %
3	59.00 %	68.12 %	67.67 %	61.33 %	66.35 %
4	59.00 %	63.38 %	68.00 %	60.75 %	66.25 %
5	58.75 %	64.00 %	67.42 %	62.08 %	66.65 %
Average	**58.0±1.1** %	**64.0±2.1** %	**66.4±1.8** %	**61.0±0.7** %	**66.4±0.3** %

There are a few interesting ideas that can be inferred from these results. Firstly, it is important to note that all data augmentation methods have yielded improvements in accuracy over the original dataset. Additionally, it can be seen that method 2 and method 4 have the best performance at 66.4%, however method 4 is much more robust as it performs consistently at a high level, i.e. it has a low standard-deviation. Additionally, it can be seen that method 3 has the lowest accuracy, yet significantly lower standard-deviation than methods 1 and 2, i.e. it is more robust. Method 4 combines the robustness of method 2 (high accuracy) and method 3 (low standard-deviation) and thus models trained on the following methodology have the best generalization. This was followed by the analysis of the specific categories that were misclassified.

The analysis started from looking at confusion matrices and followed by the accuracy, precision, and recall metrics for each category. Firstly, it was found that specific sound categories were constantly misclassified by all of the models. Several conclusions were made after a visual inspection of the confusion matrix, in Appendix B, Fig. 3.3, as well as the performance metric .csv file in Table B.2.

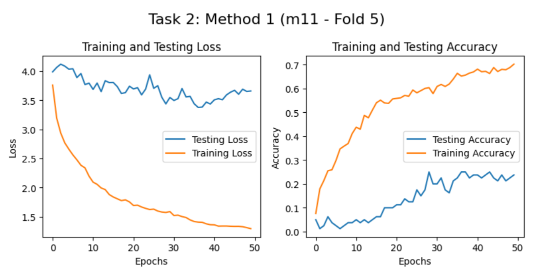
The model shows outstanding performance for classes like 'sheep' and 'door_wood_knock', indicating strong recognition capabilities for these sounds. For 'water drops', the model completely fails, since there are none included in the testing set and for 'glass breaking' and 'hen', it significantly underperforms. Interestingly, the model's performance on 'hen' varies between methods, suggesting that factors such as low signal-to-noise ratio (SNR) can significantly impact the model's effectiveness in discerning certain environmental sounds, The dataset shows a clear class imbalance; for instance, 'rain' is over-represented with 70 samples, while 'water drops' have none. This could have been avoided by using the 5 fold cross validation methodology suggested by the publisher of the dataset [9]. This would have eliminated the class imbalance while training and allow the testing of the optimized model against a validation set - not used during optimization - which would give a more robust conclusion on whether or not the models generalize well to unseen data. Instead, 3 additional models were trained with data split using different `random_state` value. It was found that the model does indeed retain performance over different splits and is thus generalizable, as seen in Table 3.2.

Table 3.2: Each version indicated a `train_test_split`, with a different `random_state`.

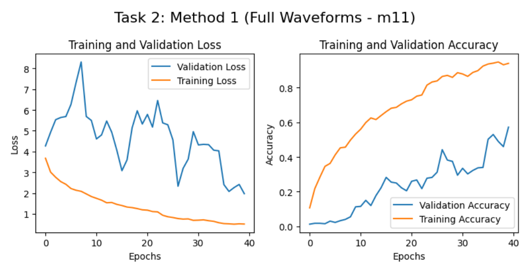
Method 1 Performances				
Split	Original	Version 1	Version 2	Version 3
Average Accuracy	64.0 ± 2.1 %	64.0 ± 1.1 %	63.9 ± 1.1 %	64.5 ± 1.4 %

3.2 Raw Audio Waveforms

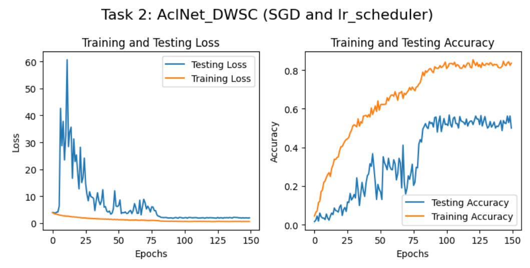
The evaluation methodology for raw audio waveforms was similar to Mel spectrograms, There were 2 major difference. Firstly, two distinct data input techniques were employed, each coupled with a pair of unique network architectures. Secondly, instead of training 5 models and finding their average performance, the dataset standard 5 fold cross validation was used. Fig. 3.2, displays histories from the 5th fold of each model's cross-validation.



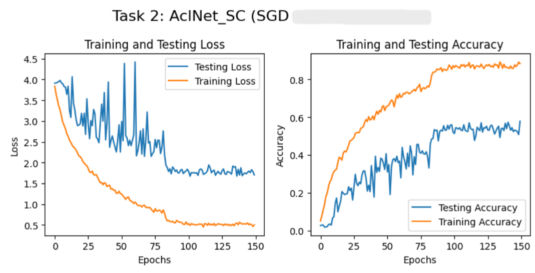
(a) History for the original m5 model



(b) History for the original m11 model



(c) History for AclNet_DWSC with momentum SGD and `lr_scheduler`



(d) History for AclNet_SC with momentum SGD and `lr_scheduler`

Figure 3.2: Training histories of the models. Method 1, (a) and (b), and method 2, (c) and (d).

The cross validation accuracies are displayed in Table 3.3 with their average and standard-deviation.

Fold	**m5**	**m11**	**AclNet_SC**	**AclNet_DWSC**
	Accuracy	**Accuracy**	**Accuracy**	**Accuracy**
1	11.87 %	55.31 %	61.72 %	57.03 %
2	20.31 %	49.38 %	64.84 %	54.69 %
3	17.50 %	53.75 %	61.98 %	57.03 %
4	12.50 %	60.62 %	64.06 %	60.75 %
5	18.44 %	46.56 %	63.54 %	57.81 %
Average	**16.1±3.3** %	**53.1±4.9** %	**63.2±1.2** %	**57±1.3** %

Table 3.3: Scores per fold for each model

The performance comparison of four models—m5, m11, AclNet_SC, and AclNet_DWSC—over five data folds indicates AclNet_SC as the superior model with an average accuracy of 63.2% and minimal variability. AclNet_DWSC follows with an average accuracy of 57.1% but with similar fluctuation in results. The trade-off between computational efficiency should be considered when evaluating the potential applications of the aforementioned networks, as AclNet_DWSC is much better for less powerful devices. The m11 model, with an average accuracy of 53.1%, offers moderate performance and consistency, while the m5 model lags with an average accuracy of only 22.6%, indicating it is not well-suited for the task. Both of these models, take in the whole waveform as the input and they are not able to generalize it away. The best results across models are observed in the fourth fold, except for m5, suggesting particular suitability or an easier dataset in this fold.

3.3 Evaluation Metrics

Before discussing 3.4 Separation of Categories, it is essential to understand the the metrics used to assess the classification performance of a model. Each of them provide unique insights into different aspects of model behavior. The most widely recognized among these are accuracy, recall, precision, and the F1-score.

Accuracy

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observations to the total observations, shown in the equation below.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

The acronyms are defined as follows:

1. TP = True Positives
2. TN = True Negatives
3. FP = False Positives
4. FN = False Negatives

It is a useful metric when the class distribution is similar. However, accuracy alone can be misleading when classes are imbalanced, or the model is biased due to certain features of certain classes being easier to discern, whilst others have similar features. Consequently, accuracy alone may not reflect the effectiveness of a model in identifying the minority class.

Recall

Recall, measures the ratio of correctly predicted observations to the all observations of the actual class. It is calculated using Eq. 3.2 below,

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

It is the measure of a classifier's completeness. High recall indicates the class is correctly recognized (a low number of FNs). For example a machine learning algorithm is trained to predict a tumor in a patient. In this case high recall is more important than high precision, since it is important to not have any FNs and let the patient go home to only find out later that (s)he has a tumour.

Precision

Precision evaluates the ratio of correctly predicted positive observations to the total predicted positives. It is calculated using Eq. 3.3 below,

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

High precision relates to a low rate of false positives, reflecting the model's ability to not label as positive a sample that is negative. For example this metric is useful for spam mail detection. It is okay if any spam mail remains undetected (FN), but what if any critical mail is missed because it is classified as spam (FP). In this situation, False Positive should be as low as possible. Here, precision is more vital as compared to recall.

F1-Score

The F1-score is the harmonic mean of precision and recall, taking both metrics into account in the following equation:

$$F1 = 2 \times \frac{recall \times precision}{recall + precision} \tag{3.4}$$

It is particularly useful when the classes are imbalanced. F1-score is a function of both precision and recall, which makes it a balanced metric for evaluating models which might favor one over the other. For, the purposes of this study the F1-score can give a direct evaluation of whether or not certain classes are miss-categorized.

Together, these metrics provide a robust framework for evaluating the predictive performance of classification models. They allow for a nuanced analysis that can guide the model improvement process and inform decision-making regarding the model's deployment in real-world scenarios.

3.4 Separation of Categories

The ease or the difficulty of separation of categories can be understood at a detailed level using clustering approaches; however it is sufficient to make such conclusion by observing the confusion matrix, in Fig. 3.3, and the F1-scores, in Appendix B, Table B.2.

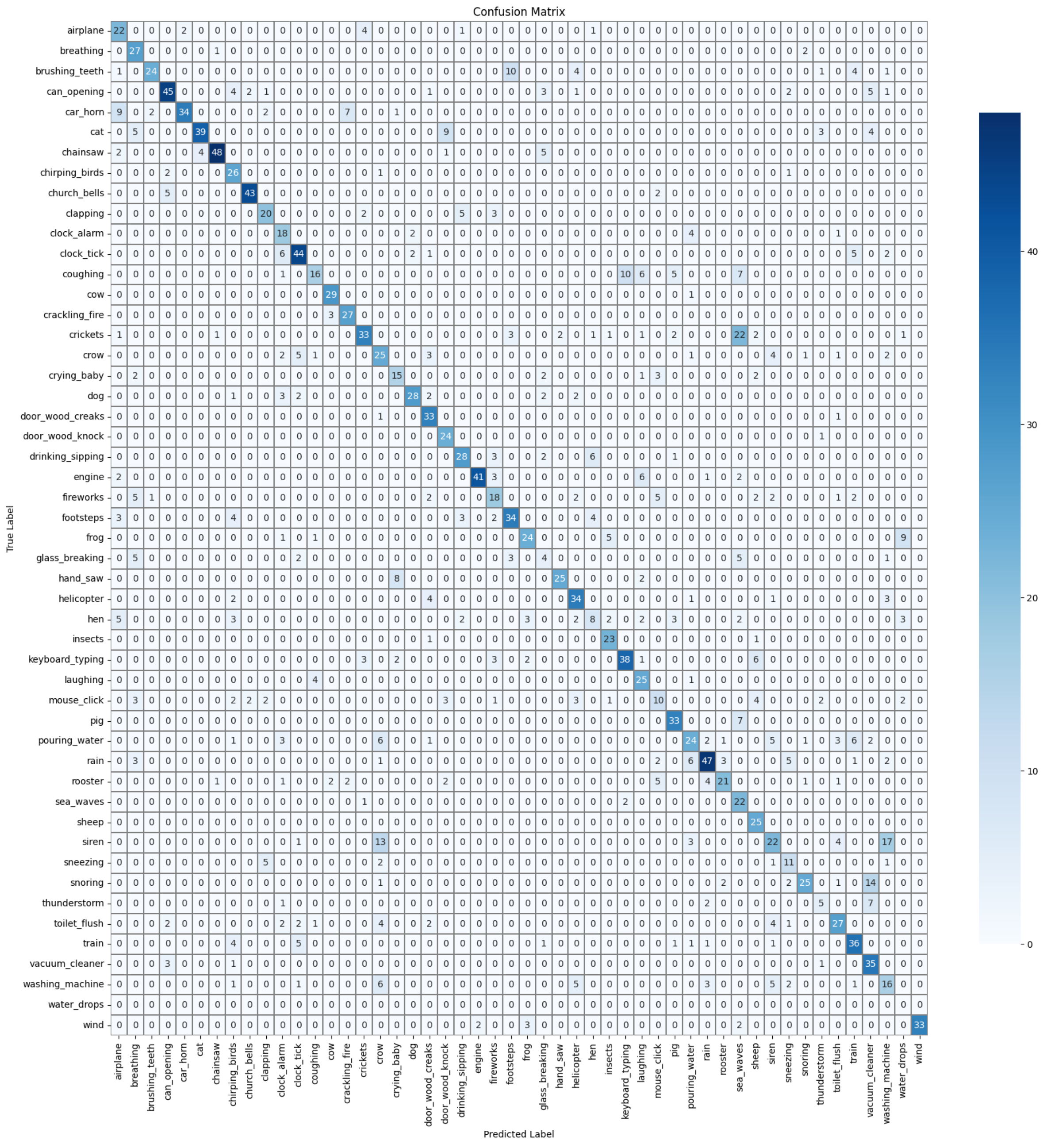


Figure 3.3: Confusion Matrix generated after training CNN3 with augmentation method 4 as input.

In the confusion matrix, several misclassifications are evident. The 'crickets' category is frequently confused with 'sea waves', with 22 instances of mislabeling out of a total of 70. Similarly, 'siren' is often mistaken for 'crow' (13 instances) and 'washing machine' (17 instances) out of a total of 40 occurrences. Likewise, there are 14 out of 45 instances, where 'snoring' is misclassified as 'vacuum_cleaner'. This is not the case vice-versa meaning that the data is biased towards 'sea_waves', 'crow', 'washing_machine' and 'vacuum_cleaner' categories. It can be seen in Table 3.4. that these categories have low precision compared to their accuracy and recall - especially 'sea_waves' - which means that other categories can frequently be classified as one of the aforementioned categories (FPs), which is also supported by the previous observations from the confusion matrix.

Category	Accuracy	Precision	Recall	F-1	Qty
washing_machine	0.40	0.35	0.40	0.37	40
crow	0.56	0.42	0.56	0.48	45
vacuum_cleaner	0.88	0.52	0.88	0.65	40
sea_waves	0.88	0.32	0.88	0.47	25

Table 3.4: An excerpt from Table B.2, with relevant categories to the discussion

The metrics were compared for 2 performance metric datasets which were extracted from training CNN3 on data augmentation method 1 and method 4. They can be seen in the Appendix B. Few categories which stand out will be identified. For example Coughing: The model is correct 70% of the time when it predicts coughing, but it only detects 36% of coughing instances. Additionally, breathing is detected 90% of the time with only a precision of 49% and F1 score of 0.68 in method 4 table, however its is much more robustly predicted by method 1 table where it has an accuracy of 100% and an F1 score of 0.92. This means that method 4 adds extra features which prevents the model from generalizing to certain data. This should be further explored to see how classification of certain categories can be improved.

Chapter 4

Discussion

An alternate preprocessing approach using matplotlib to generate and store image data was initially tried but abandoned due to several drawbacks. This process, documented in `SoundIdentification_V1.ipynb`, was problematic. For example, matplotlib may have downsampled images, leading to resolution loss, as depicted in Fig. 2.1(e) and (f), where visually indistinguishable white and pink noise were actually numerically distinct. Additionally, irrelevant color channels were introduced, potentially causing the model to learn unnecessary features. The data generation was also inefficient, requiring up to an hour per dataset - the current methodology takes less than a minute. Moreover, it likely introduced quantization errors and reduced data fidelity by converting 16-bit spectrogram arrays to 8-bit images. Recognizing these issues early, the methodology was revised and implemented in the `SoundIdentification_Final (V2).ipynb` file.

The initial training results suggested implausibly high testing accuracies using Mel spectrogram augmentations. Initially after method 1 augmentation accuracy surged to 75% which seemed promising and aligned with literature benchmarks. Yet, training with method 2 augmentation yielded over 97% accuracy, seen in Fig. 4.1, surpassing realistic expectations set by state-of-the-art methods such as transformer networks [17] and attention mechanisms [18], which reportedly have accuracies of between 97.00% and 98.10% respectively. The issue arose from feeding all augmented data into training without splitting, causing overlap with the testing set in the training data. Consequently, the model learned the features of the test set, as the augmented data is similar to the original data. This was fixed by using `random_state = 42` in the `train_test_split`, which meant that the augmented data also enhanced the testing set, reduced model noise and increased its performance.

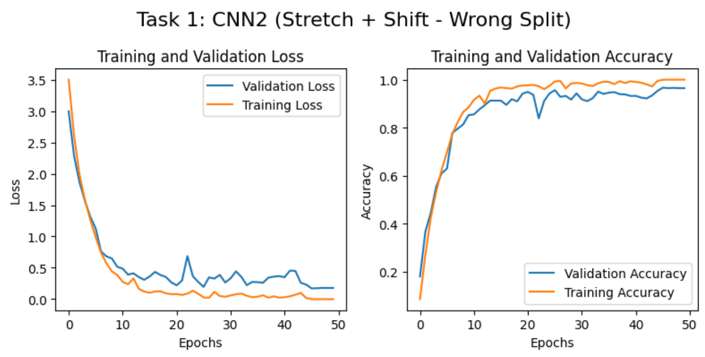


Figure 4.1: Model trained on the wrongly split method 2 data with CNN3

Chapter 5

Conclusion

This study has provided valuable insights into the application of deep learning models for the classification of environmental sounds, a topic of increasing relevance in fields ranging from environmental monitoring to smart home technology. The exploration of various model architectures, including CNN2, CNN3, m5, m11, and AclNet (both SC and DWSC versions), using the ESC-50 dataset, has highlighted the potential and challenges of this rapidly evolving domain.

It was found that among the models and data augmentation methodologies tested, that for Mel spectrogram inputs CNN3 with data augmentation methodology 4 and for raw waveform input, AclNet_SC with emerged as the most effective, i.e. the highest average accuracy and minimal variability across data folds. Additionally, It was outlined that certain models like AclNet_DWSC while less accurate could be applied to specific requirements of the sound classification task. The study also identified key areas where performance varied significantly, particularly in models like m5, which struggled with generalizing across different sound types. This should be further explore to improve classification of certain classes.

The analysis delved into the complexities of sound classification, noting how certain sounds are consistently misclassified across models. This aspect of the research points towards the need for further refining training methodologies and perhaps suggests exploring more advanced neural network architectures, like transformers.

Future research in this area could focus on expanding the dataset, experimenting with different types of data augmentation, or exploring more complex and computationally efficient model architectures, especially for deployment on resource-constrained devices.

Appendices

Appendix A

Model Architectures

Layer (type)	**Output Shape**	**Param #**
InputLayer	Variable	0
Resizing (64x128)	(None, 64, 128, 1)	-
Conv2D (32, (3, 3))	(None, 62, 126, 32)	320
MaxPooling2D (2, 2)	(None, 31, 63, 32)	-
Conv2D (128, (3, 3))	(None, 29, 61, 128)	36992
MaxPooling2D (2, 2)	(None, 14, 30, 128)	-
Conv2D (128, (3, 3))	(None, 12, 28, 128)	147584
MaxPooling2D (2, 2)	(None, 6, 14, 128)	-
Conv2D (128, (3, 3))	(None, 4, 12, 128)	147584
MaxPooling2D (2, 2)	(None, 2, 6, 128)	-
Flatten	(None, 1536)	-
Dense (100)	(None, 100)	153700
Dense (50)	(None, 50)	5050
Total params:		491230 (1.87 MB)
Trainable params:		491230 (1.87 MB)
Non-trainable params:		0 (0.00 Byte)

Table A.1: Architecture of the CNN2 model.

Table A.2: Architecture of the CNN3 model.

Layer (type)	**Output Shape**	**Param #**
Resizing (64x128)	(None, 64, 128, 1)	0
Conv2D (32, (3, 3))	(None, 62, 126, 32)	320
BatchNormalization	(None, 62, 126, 32)	128
MaxPooling2D (2, 2)	(None, 31, 63, 32)	0
Conv2D (64, (3, 3))	(None, 29, 61, 64)	18496
BatchNormalization	(None, 29, 61, 64)	256
MaxPooling2D (2, 2)	(None, 14, 30, 64)	0
Conv2D (128, (3, 3))	(None, 12, 28, 128)	73856
BatchNormalization	(None, 12, 28, 128)	512
MaxPooling2D (2, 2)	(None, 6, 14, 128)	0
Flatten	(None, 10752)	0
Dropout (0.8)	(None, 10752)	0
Dense	(None, 100)	1075300
BatchNormalization	(None, 100)	400
Dropout (0.6)	(None, 100)	0
Dense	(None, 50)	5050
Total params:		1174318 (4.48 MB)
Trainable params:		1173670 (4.48 MB)
Non-trainable params:		648 (2.53 KB)

Layer (type)	**Output Shape**	**Param #**
Input Shape	[220500, 1]	0
Conv1D	[None, 55126, 128]	10368
BatchNormalization	[None, 55126, 128]	512
Activation (ReLU)	[None, 55126, 128]	0
MaxPooling1D	[None, 13781, 128]	0
Conv1D	[None, 13781, 128]	49280
BatchNormalization	[None, 13781, 128]	512
Activation (ReLU)	[None, 13781, 128]	0
MaxPooling1D	[None, 3445, 128]	0
Conv1D	[None, 3445, 256]	98560
BatchNormalization	[None, 3445, 256]	1024
Activation (ReLU)	[None, 3445, 256]	0
Dropout (0.5)	[None, 3445, 256]	0
MaxPooling1D	[None, 861, 256]	0
Conv1D	[None, 861, 512]	393728
BatchNormalization	[None, 861, 512]	2048
Activation (ReLU)	[None, 861, 512]	0
MaxPooling1D	[None, 215, 512]	0
Global Average Pooling1D	[None, 512]	0
Dense (Softmax)	[None, 50]	25650
Total params:		581682 (2.22 MB)
Trainable params:		579634 (2.21 MB)
Non-trainable params:		2048 (8.00 KB)

Table A.3: Architecture of the m5 model.

Table A.4: Architecture of the m11 Model. (x2) and (x3) indicate layers repetition.

Layer (type)	**Output Shape**	**Param #**
Input Shape	[220500, 1]	0
Conv1D	[None, 55126, 64]	5184
BatchNormalization	[None, 55126, 64]	256
Activation (ReLU)	[None, 55126, 64]	0
MaxPooling1D	[None, 13781, 64]	0
Conv1D (x2)	[None, 13781, 64]	12352 (each)
BatchNormalization (x2)	[None, 13781, 64]	256 (each)
Activation (ReLU) (x2)	[None, 13781, 64]	0 (each)
MaxPooling1D	[None, 3445, 64]	0
Conv1D (x2)	[None, 3445, 128]	24704 (each)
BatchNormalization (x2)	[None, 3445, 128]	512 (each)
Activation (ReLU) (x2)	[None, 3445, 128]	0 (each)
MaxPooling1D	[None, 861, 128]	0
Conv1D (x3)	[None, 861, 256]	98560 (each)
BatchNormalization (x3)	[None, 861, 256]	1024 (each)
Activation (ReLU) (x3)	[None, 861, 256]	0 (each)
MaxPooling1D	[None, 215, 256]	0
Conv1D (x2)	[None, 215, 512]	393728 (each)
BatchNormalization (x2)	[None, 215, 512]	2048 (each)
Activation (ReLU) (x2)	[None, 215, 512]	0 (each)
Dropout (0.4)	[None, 215, 512]	0
Global Average Pooling1D	[None, 512]	0
Dense (Softmax)	[None, 50]	25650
Total params:		1811442 (6.91 MB)
Trainable params:		1806962 (6.89 MB)
Non-trainable params:		4480 (17.50 KB)

Layer	**Stride**	**Out dim**	**Out Chans**	**Kernel Size**
Conv1	S1	C1,1,20480/S1	C1	9
Conv2	S2	64,1,20480/(S1S2)	64	5
Maxpool1	1	64,1,128	64	160/(S1S2)
Conv3	1	32, 64, 128	32	3 x 3
Maxpool2	1	32, 32, 64	32	2 x 2
Conv4	1	64, 32, 64	64	3 x 3
Conv5	1	64, 32, 64	64	3 x 3
Maxpool3	1	64, 16, 64	64	2 x 2
Conv6	1	128, 16, 32	128	3 x 3
Conv7	1	128, 16, 32	128	3 x 3
Maxpool4	1	128, 8, 16	128	2 x 2
Conv8	1	256, 8, 16	256	3 x 3
Conv9	1	256, 8, 16	256	3 x 3
Maxpool5	1	256, 4, 8	256	2 x 2
Conv10	1	512, 4, 8	512	3 x 3
Conv11	1	512, 4, 8	512	3 x 3
Maxpool6	1	512, 2, 4	512	2 x 2
Conv12	1	50, 2, 4	50	1 x 1
Avgpool1	1	50	50	2 x 4

Table A.5: AclNet for an example input - 1.28s 16kHz waveform. Input dimension (1, 1, 20480)

Appendix B

Classification Metrics

Category	Accuracy	Precision	Recall	F-1	Qty
water_drops	0	0	0	0.00	0
glass_breaking	0.2	0.21	0.2	0.20	20
hen	0.23	0.4	0.23	0.29	35
mouse_click	0.29	0.37	0.29	0.33	35
thunderstorm	0.33	0.38	0.33	0.35	15
coughing	0.36	0.7	0.36	0.48	45
siren	0.37	0.49	0.37	0.42	60
washing_machine	0.4	0.35	0.4	0.37	40
pouring_water	0.44	0.57	0.44	0.50	55
fireworks	0.45	0.55	0.45	0.50	40
crickets	0.47	0.77	0.47	0.58	70
rooster	0.52	0.78	0.52	0.62	40
brushing_teeth	0.53	0.89	0.53	0.66	45
sneezing	0.55	0.46	0.55	0.50	20
crow	0.56	0.42	0.56	0.48	45
snoring	0.56	0.83	0.56	0.67	45
toilet_flush	0.6	0.68	0.6	0.64	45
crying_baby	0.6	0.58	0.6	0.59	25
frog	0.6	0.75	0.6	0.67	40
car_horn	0.62	0.94	0.62	0.75	55
cat	0.65	0.91	0.65	0.76	60
clapping	0.67	0.67	0.67	0.67	30
rain	0.67	0.78	0.67	0.72	70
footsteps	0.68	0.68	0.68	0.68	50
can_opening	0.69	0.79	0.69	0.74	65
keyboard_typing	0.69	0.76	0.69	0.72	55
dog	0.7	0.88	0.7	0.78	40
drinking_sipping	0.7	0.72	0.7	0.71	40
hand_saw	0.71	0.93	0.71	0.81	35
clock_alarm	0.72	0.47	0.72	0.57	25
train	0.72	0.65	0.72	0.68	50
airplane	0.73	0.49	0.73	0.59	30
clock_tick	0.73	0.71	0.73	0.72	60
engine	0.75	0.95	0.75	0.84	55
helicopter	0.76	0.64	0.76	0.69	45
chainsaw	0.8	0.94	0.8	0.86	60
wind	0.82	1	0.82	0.90	40
pig	0.82	0.73	0.82	0.77	40
laughing	0.83	0.57	0.83	0.68	30
church_bells	0.86	0.91	0.86	0.88	50
chirping_birds	0.87	0.53	0.87	0.66	30
vacuum_cleaner	0.88	0.52	0.88	0.65	40
sea_waves	0.88	0.32	0.88	0.47	25
breathing	0.9	0.54	0.9	0.68	30
crackling_fire	0.9	0.75	0.9	0.82	30
insects	0.92	0.72	0.92	0.81	25
door_wood_creaks	0.94	0.66	0.94	0.78	35
door_wood_knock	0.96	0.62	0.96	0.75	25
cow	0.97	0.85	0.97	0.91	30
sheep	1	0.6	1	0.75	25

Table B.1: Table of metrics for model training with Mel spectrograms with data input as Method 4. Sorted by 'Accuracy'

Category	Accuracy	Precision	Recall	F-1 score	Quantity
water drops	0	0	0	0	0
glass breaking	0.12	0.17	0.12	0.2	8
mouse click	0.14	0.33	0.14	0.29	14
crickets	0.21	1	0.21	0.33	28
washing machine	0.31	0.71	0.31	0.35	16
fireworks	0.31	0.36	0.31	0.48	16
thunderstorm	0.33	0.29	0.33	0.42	6
toilet flush	0.33	0.5	0.33	0.37	18
keyboard typing	0.36	0.8	0.36	0.5	22
siren	0.38	0.53	0.38	0.5	24
coughing	0.39	0.5	0.39	0.58	18
hen	0.43	0.5	0.43	0.62	14
crying baby	0.5	0.38	0.5	0.66	10
hand saw	0.57	0.5	0.57	0.5	14
car horn	0.59	0.81	0.59	0.48	22
footsteps	0.6	0.6	0.6	0.67	20
crow	0.61	0.44	0.61	0.64	18
rooster	0.62	1	0.62	0.59	16
rain	0.64	1	0.64	0.67	28
engine	0.64	0.93	0.64	0.75	22
pouring water	0.64	0.61	0.64	0.76	22
crackling fire	0.67	0.62	0.67	0.67	12
laughing	0.67	0.62	0.67	0.72	12
clock tick	0.67	0.55	0.67	0.68	24
clapping	0.67	0.73	0.67	0.74	12
drinking sipping	0.69	0.65	0.69	0.72	16
train	0.7	0.78	0.7	0.78	20
cat	0.71	0.85	0.71	0.71	24
helicopter	0.72	0.59	0.72	0.81	18
snoring	0.72	0.93	0.72	0.57	18
sneezing	0.75	0.43	0.75	0.68	8
brushing teeth	0.78	0.88	0.78	0.59	18
insects	0.8	0.53	0.8	0.72	10
church bells	0.8	0.94	0.8	0.84	20
clock alarm	0.8	0.57	0.8	0.69	10
vacuum cleaner	0.81	0.68	0.81	0.86	16
can opening	0.81	0.91	0.81	0.9	26
dog	0.81	0.93	0.81	0.77	16
frog	0.81	0.81	0.81	0.68	16
airplane	0.83	0.59	0.83	0.88	12
chainsaw	0.83	0.95	0.83	0.66	24
wind	0.88	0.88	0.88	0.65	16
sea waves	0.9	0.36	0.9	0.47	10
chirping birds	0.92	0.61	0.92	0.68	12
pig	0.94	0.79	0.94	0.82	16
door wood knock	1	0.5	1	0.81	10
door wood creaks	1	0.58	1	0.78	14
cow	1	0.5	1	0.75	12
breathing	1	0.86	1	0.91	12
sheep	1	0.5	1	0.75	10

Table B.2: Table of metrics for model training with Mel spectrograms with data input as Method 1. Sorted by 'Accuracy'

Bibliography

[1] A. Kumar and V. K. Ithapu, "A sequential self teaching approach for improving generalization in sound event recognition," 2020.

[2] S. Wei, S. Zou, F. Liao, and weimin lang, "A comparison on data augmentation methods based on deep learning for audio classification," *Journal of Physics: Conference Series*, vol. 1453, p. 012085, jan 2020.

[3] M. Mohaimenuzzaman, C. Bergmeir, I. West, and B. Meyer, "Environmental sound classification on the edge: A pipeline for deep acoustic networks on extremely resource-constrained devices," *Pattern Recognition*, vol. 133, p. 109025, 2023.

[4] B. Elizalde, S. Deshmukh, and H. Wang, "Natural language supervision for general-purpose audio representations," 2024.

[5] F. Zhang, L. Zhang, H. Chen, and J. Xie, "Bird species identification using spectrogram based on Multi-Channel fusion of DCNNs," *Entropy (Basel)*, vol. 23, Nov. 2021.

[6] K. S. Alqudaihi, N. Aslam, I. U. Khan, A. M. Almuhaideb, S. J. Alsunaidi, N. M. A. R. Ibrahim, F. A. Alhaidari, F. S. Shaikh, Y. M. Alsenbel, D. M. Alalharith, H. M. Alharthi, W. M. Alghamdi, and M. S. Alshahrani, "Cough sound detection and diagnosis using artificial intelligence techniques: Challenges and opportunities," *IEEE Access*, vol. 9, pp. 102327–102344, July 2021.

[7] A. Raza, A. Mehmood, S. Ullah, M. Ahmad, G. S. Choi, and B.-W. On, "Heartbeat sound signal classification using deep learning," *Sensors*, vol. 19, no. 21, 2019.

[8] A. Wang, "An industrial strength audio search algorithm," Jan 2003.

[9] K. J. Piczak, "ESC: Dataset for Environmental Sound Classification," in *Proceedings of the 23rd Annual ACM Conference on Multimedia*, pp. 1015–1018, ACM Press, Oct 2015.

[10] B. McFee *et al.*, "librosa/librosa: 0.10.1," Aug. 2023.

[11] Y. Tokozume and T. Harada, "Learning environmental sounds with end-to-end convolutional neural network," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2721–2725, 2017.

[12] Y. Tokozume, Y. Ushiku, and T. Harada, "Learning from between-class examples for deep sound recognition," 2018.

[13] J. J. Huang and J. J. A. Leanos, "Aclnet: efficient end-to-end audio classification cnn," 2018.

[14] W. Dai, C. Dai, S. Qu, J. Li, and S. Das, "Very deep convolutional neural networks for raw waveforms," 2016.

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[16] Y. Tokozume and T. Harada, "Learning environmental sounds with end-to-end convolutional neural network," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2721–2725, 2017.

[17] Y. Gong, Y.-A. Chung, and J. Glass, "Ast: Audio spectrogram transformer," 2021.

[18] K. Chen, X. Du, B. Zhu, Z. Ma, T. Berg-Kirkpatrick, and S. Dubnov, "Hts-at: A hierarchical token-semantic audio transformer for sound classification and detection," 2022.