An Experiment on Acoustic Scene Classification with a Deep Learning Model

Luke Zhang (az16408)
Department of Computer Science
University of Bristol
Bristol, United Kingdom

Ben Jones (bj17018)
Department of Computer Science
University of Bristol
Bristol, United Kingdom

Abstract

***Abstract*—This experiment explore, design and replicate an existing work [1] for the acoustic scene classification (ASC) task. The original work was proposed in the "detection and classification of acoustic scenes and events" (DCASE) challenge in 2016. We implement the process of utilising convolutional neural network (CNN) architecture proposed in the paper. The experiment is done using a pre-processed dataset that originate from the public ASC development dataset [2]. The best accuracy that we achieve for evaluating our model on full dataset is 85.13% and 80% for non-full dataset. We find that a further improvement can be obtained by adding one more dense layer into the CNN.**

***Index Terms*—Acoustic scene classification, deep learning, CNN, DCASE, computational audio processing**

I. Introduction

With the drastic growth of computational power and the rise of parallel computing architecture such as GPU, there is an increasing number of algorithms being proposed to optimise machine learning (ML) tasks. ASC is one of the main research fields when it comes to applications such as speech recognition, context detection and audio enhancement [3]. Recognising different acoustic environment is easy for humans, thanks to the complex calculation that our brain is able to perform [1]. Additionally, we learn to associate our experience and memories to a scene that we never seen before. This enable us to be good at recognising sounds in different context. However, this seemingly simple task is not easy for computers, as they need complex models to learn to do this and adapt to the constantly fluctuating environment in order to avoid over-fitting [4] the models. There are many applications by building ML models that are good at recognising acoustic scenes, from making context aware intelligent wearable device [5] to navigation systems in robotics [6]. This experiment aims to implement and potentially improve an existing deep learning (CNN) model in ASC. In terms of the dataset for our experiment, we use same ASC development dataset from the public ASC dataset. At first, we implement the CNN architecture and fine tune it to find the best model and parameters to use. After that, we evaluated the model on both full and non-full dataset, then conclude the best accuracy score we can achieve. Finally, we try adding an additional improvement with one more dense layer [7] to the architecture to generalise the model better.

II. Related Work

The main related work for this experiment can be found in the work of Valenti et al [1], where applying convolutional neural network on acoustic scene classification was first introduced. In the experiment the authors also compared the CNN architecture to two other architectures, multi-layer perceptron (MLP) and gaussian mixture models (GMM). For evaluating the performance of the CNN models, audio with different length of segmentations are used.

In light of recent year's finding, it has been discovered that there is not enough acoustic data we can use to train better models considering the complexity in real world environment [3]. This is particularly to be the case when we want to classify very specific acoustic scenes. Because of that, the trained models are likely prone to over-fitting and lead to incorrect predictions. We can potentially address this problem with techniques like data augmentation on the original data. Additionally, there also exist inconsistent audio properties when using devices with different frequency responses to record audio [8]. This is because different devices are manufactured with various specifications, which means same audio files can have varied frequency components. As a result, acoustic scene classification models also need to factor this to accommodate the situation.

III. Experimental design

A. Dataset

The dataset we use in this experiment originally comes from the DCASE 2016 [2] development dataset. The raw data is stored in a ".npy" file format, with 1560 files in total and each file contains an audio with a 30 seconds length. These files contains data that are equally distributed between 15 different classes. As shown in Figure 1, we set a *full* training mode where the train (development) and test (evaluation) split of the whole DCASE data is 75% and 25%. What's more, the train (development) set is further split into a 70% (819 .npy files) train and 30% (351 .npy files) validation set. Here we use validation set to pick the best parameters for our CNN model to use. We then use the picked parameters and model based on the performance of the validation set to evaluate on the test (evaluation) data. This is the *non-full* training mode which enable us to measure the performance of our

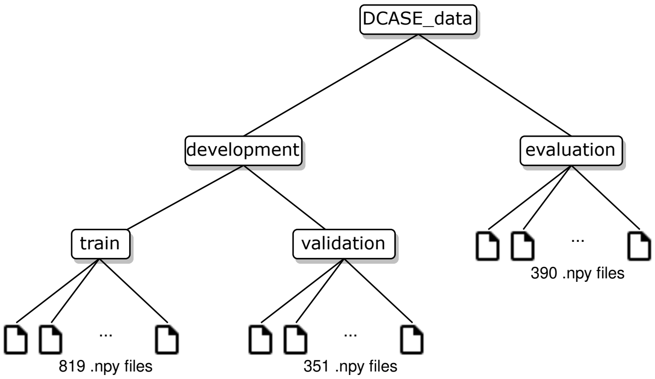


Fig. 1. DCASE dataset split visualisation.

model without potential memorisation from the training set and bias from the validation set. Ideally there will not be too many discrepancies between the performance on test set and performance on validation set when we evaluate our model.

B. Input

Here we use three seconds sequence length as input to our CNN, which means the data in each file will be divided into 10 clips. A spectrogram is a diagram to visualise the spectrum of frequencies of a given signal as it varies in time [9]. We use it to visualise the signal strength (amplitude) over time. The amplitude of the frequency is represented by a third dimension (colour spectrum). The dark blues correspond to low amplitudes, while brighter colours up through red represent larger amplitudes. Two examples of log-mel spectrogram are shown in Fig. 2 and Fig. 3, where Fig. 2 correspond to beach and true label of Fig. 3 is city centre.

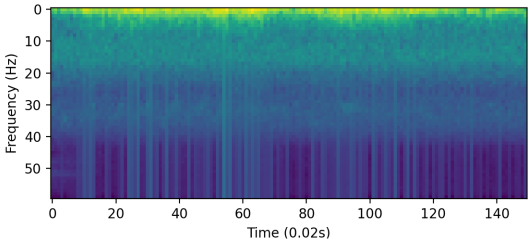


Fig. 2. An example log-mel spectrogram of beach, 3 seconds clip

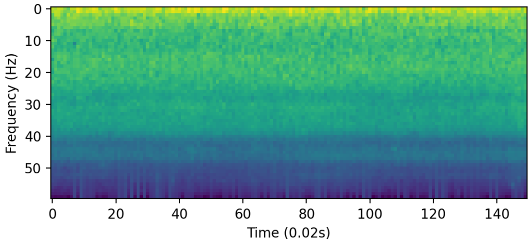


Fig. 3. An example log-mel spectrogram of city centre, 3 seconds clip

By comparing the amplitude of frequencies over time, we see that there is a lot of low frequency (more blue area) sound on a beach, this is in stark contrast to sounds in a city centre where more high frequency (more yellow area) sounds exist.

C. CNN Architecture (Valenti et al)

It is noteworthy that this experiment only produces the CNN architecture part of the original experiment. Hence the MLP and GMM in the original papers are not implemented here for comparison. A diagram of the CNN architecture we are implementing is shown in Figure 4. The layers of the model and their function are explained as follows, presented in the order of how inputs are passed through the network (from left to right):

1) First, there are 128 2-Dimensional convolution kernels applied to input spectrogram clip. The size of the input being 60 x 150 and the kernels being 5x5, padding being applied to the input before convolution so the inner dimensionality of the output remains the same. Each kernel results in a new 2D transformation of the input and there are 128 resulting 60 x 150 images. After the convolution, 2D batch normalisation is applied to all 128 images which aids generalisation by means of reducing internal covariate shift as specified by the target paper [1]. Lastly a rectified linear unit activation function (ReLu) is applied to the output of the batch normalisation.
2) In the next layer, we apply a 2D max pooling (with a kernel size of 5x5 and same size for stride of the window) over the previous output signal, which is made up of several input planes. We use pooling to reduce each feature map dimensions and to enhance the network invariance to input pattern shifts. Max pooling reduces the resolution of the output of a given convolutional layer. Therefore, it reduces the number of parameters in the network, this in turn reduces computational load. Max pooling may also help to reduce overfitting as it can extract and preserve the most important features.
3) Following the previous layer, we utilise another convolutional layer with the same kernel size and padding as the first one. In this layer, we go deeper into the network and extract even more complex features of the acoustic input. Additionally, the number of channels is extended from 128 before convolution to 256 after the convolution. However, the image size stays the same as 12x30.
4) Then, another max pooling layer is placed before we flatten our output. At this point, we further reduce the resolution of the image size from 12x30 down to a 2x6 size whilst the dimension remains the same.
5) Finally, the output resulted from the previous pooling layer is flattened to a one dimensional representation, before we reduce the input feature from 3072 down to 10 features for prediction.

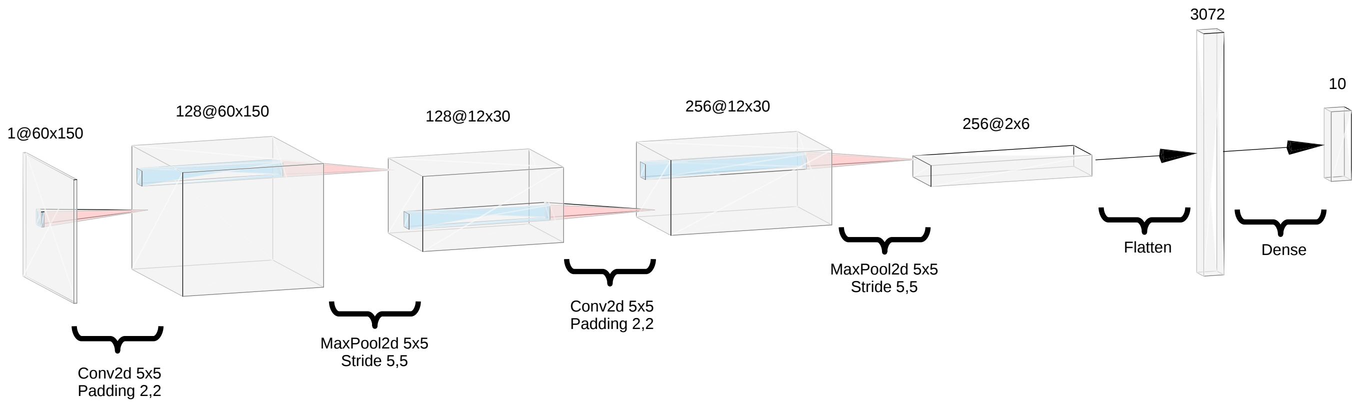


Fig. 4. A diagram of the CNN architecture of the model.

D. Implementation Details

All machine learning functionality, such as model initialisation and training can be accomplished in PyTorch with python. To load the dataset as discussed in section III-A we used the Dataset and Dataloader proved by PyTorch to write a dataset loader for all the DCASE spectrograms, which can be configured to load either the training or testing dataset from file. The data-points are stored as an entire 30 second spectrogram of a single episode with a single label. The goal of inference for this model is to take an entire 30 second spectrogram and guess the label however, the target paper specifies [1] that the CNN model itself should just take a single 3 second clip and that the 30 second original spectrogram should be split into 10 clips. This ensures that the CNN is evaluated separately but for inference the output of each clip CNN result should be averaged before taking the argument maximum to determine the inference result. This means that the model training clips can be arbitrarily or randomly batched providing the clip label matches its source spectrogram. However, for inference clips have to be arranged in batches of 10 and all originating from the same spectrogram for their CNN model outputs to be averaged.

IV. Replicating Quantitative Results

As shown in TABLE I, when the model is evaluated in the *non-full* mode on the test set, the model achieves 80% accuracy. For evaluation in *full* mode on the test set, we obtain a 5.13% improvement which reach a test accuracy of 85.13%. This is likely because the model can generalise well when it is trained with more data.

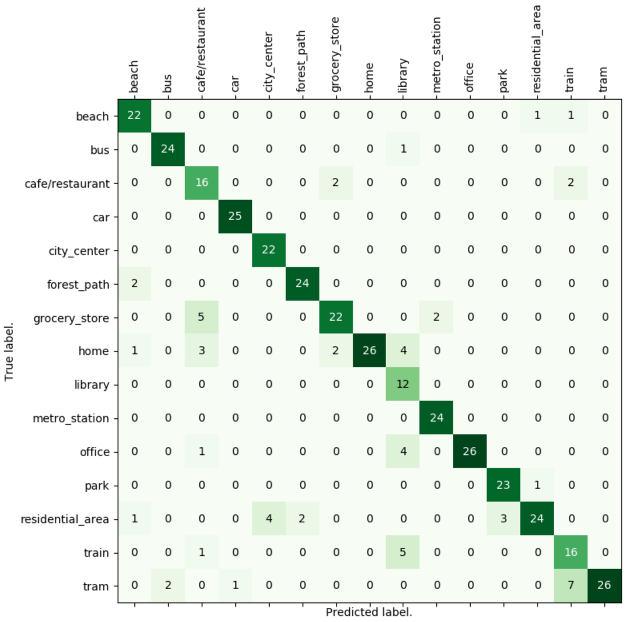
TABLE I
Accuracy of the non-full and full training modes, three seconds sequence

System	Seq len (s)	Accuracy (%)	
		non-full	**full**
Two-layer CNN (log-mel)	3	80%	85.13%

V. Training Curves

The training and test curves for the full dataset can be seen in the following graphs in Figures 6 and 7.

Generally speaking, we want to make sure that the gap between our training and validation loss curves decreases drastically at first, and then stabilise to a certain our training data. Conversely, if the gap is too big, we are likely underfitting our model, which is not what we want in the first place. In Fig. 6 we see the training loss decreases rapidly at the first 1000 steps for all dataset in a *non-full* setting. The loss gap between the test and the train set seems to stabilise after that between a level of 0 to 0.3. However, this does not seem to be the case when we observe the gap between test and train set as it appears to be slowly increasing over more steps. According to the accuracy comparison for the same setting in Fig. 8, the



	beach	bus	cafe/restaurant	car	city_center	forest_path	grocery_store	home	library	metro_station	office	park	residential_area	train	tram
beach	22	0	0	0	0	0	0	0	0	0	0	0	1	1	0
bus	0	24	0	0	0	0	0	0	1	0	0	0	0	0	0
cafe/restaurant	0	0	16	0	0	0	2	0	0	0	0	0	0	2	0
car	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0
city_center	0	0	0	0	22	0	0	0	0	0	0	0	0	0	0
forest_path	2	0	0	0	0	24	0	0	0	0	0	0	0	0	0
grocery_store	0	0	5	0	0	0	22	0	0	2	0	0	0	0	0
home	1	0	3	0	0	0	2	26	4	0	0	0	0	0	0
library	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0
metro_station	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0
office	0	0	1	0	0	0	0	0	4	0	26	0	0	0	0
park	0	0	0	0	0	0	0	0	0	0	0	23	1	0	0
residential_area	1	0	0	0	4	2	0	0	0	0	0	3	24	0	0
train	0	0	1	0	0	0	0	0	5	0	0	0	0	16	0
tram	0	2	0	1	0	0	0	0	0	0	0	0	0	7	26

Fig. 5. Confusion matrix for the proposed CNN evaluated on the full evaluation set. Three second sequences are used.

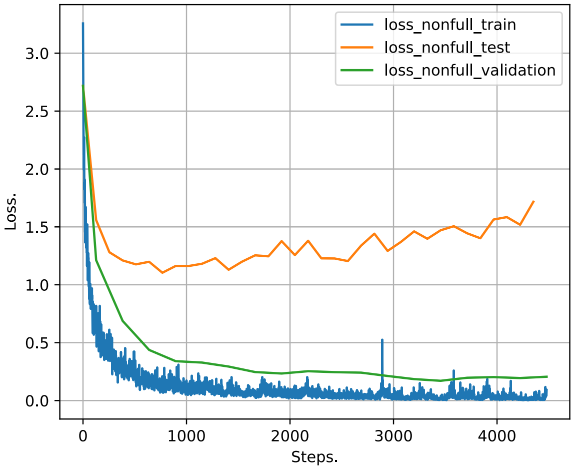


Fig. 6. Loss of the model during non-full training.

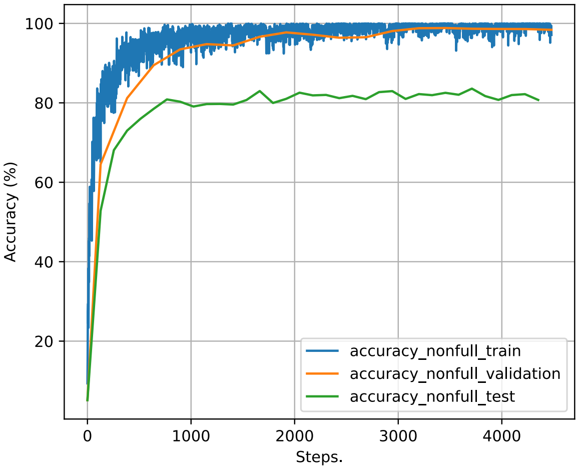


Fig. 8. Accuracy of the model during non-full training.

training and validation accuracy remain steadily just below 100% after 4000 steps, while the test accuracy fluctuating around 80%. The gap between test and train set stays steady later on in training which is exactly what we want to see.

As for the training and testing of the model in the *full* setting in Fig. 9, we recognise that there exist little improvement in training and testing accuracy as they fluctuate within the same range throughout the steps. The loss curve in Fig. 7 paints a similar picture, with the loss gap between train and test continue increasing slowly. Nonetheless, the model performs better than when training and testing in the *non-full* setting. It is also worth mentioning that the training and testing of the model in the *full* setting act almost like a continuation of the *non-full* setting process. In a way that the training accuracy continue to rise as unnoticeable as it gets, and the loss gap between train and testing keeps tilting upwards slowly as well. The reason for this might be the fact that additional data can help generalise the model well, hence we see a slight improvement in accuracy.

VI. Qualitative Results

One metric we can use to see how our model performs in *full* training mode is by looking at the confusion matrix. As confusion matrix reveals how many times a model mistakenly classify one class as another, we can speculate the weakness of our model on a much granular level. The result for testing the proposed CNN architecture in the *full* mode can be shown in Fig. 5. Looking at the diagonal results we can see that our model have most of the predictions right. It is especially good at predicting cars related acoustic scenes since it only has 1

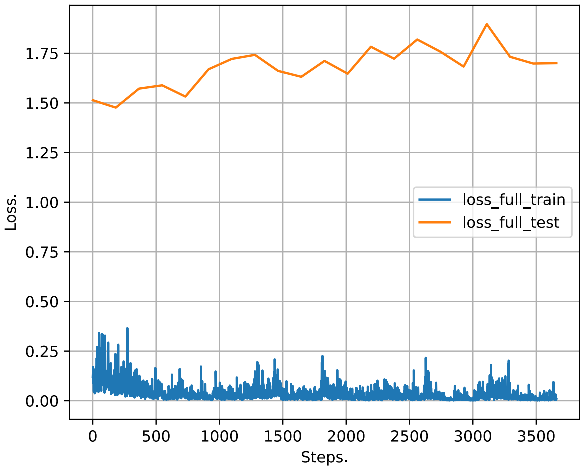


Fig. 7. Loss of the model during full training.

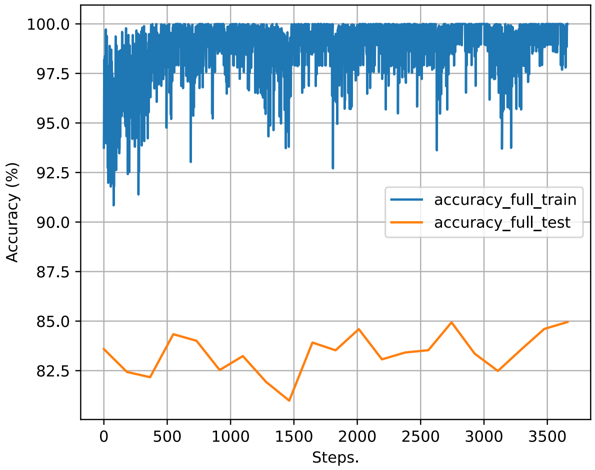


Fig. 9. Accuracy of the model during full training.

incorrect prediction, this is crucial because it means it generalise that class well. There are 7 times when the model misclassified a tram scene as a train scene. This is likely due to the fact that our model is slightly over-fitting the parameters related to the train scene. Moreover, train and tram likely share some common features. As a result, it is not generalising very well when it has to distinguish train from trams. Another weakness can also be observed when looking at the predicted column of the library class. It can be calculated that there is over half predictions are incorrect ones out of all 26 classification tasks it is given. This tells us that our model may not be good at classifying library related acoustic scenes.

VII. Improvements

To improve the current CNN performance we propose an extra dense layer in between the flatten operation and current dense connection. The proposed dense connection also has 1-D batch normalization and a ReLu activation function. Furthermore, the dense layer have 3,072 input neurons and 1,000 output neurons. We hypothesise that more dense layer will be able to make more accurate predictions when the input clip contains more complicated audio patterns that last for a longer duration because the convolutional layers are best suited for analysis close spatial data while dense layers consider all of the input dimension for calculating each output of the layer. Looking at the accuracy from extra dense layer we can observe that the model has a higher non-full test accuracy of 84.1% and a higher maximum full test accuracy of 87% before dropping back down to 84.1%.

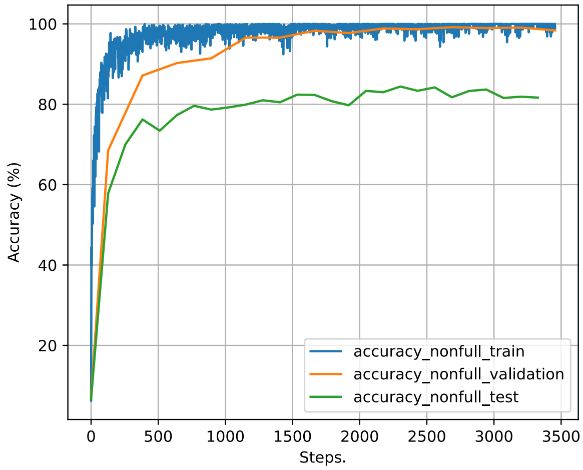


Fig. 10. Accuracy of the modified model during non-full training.

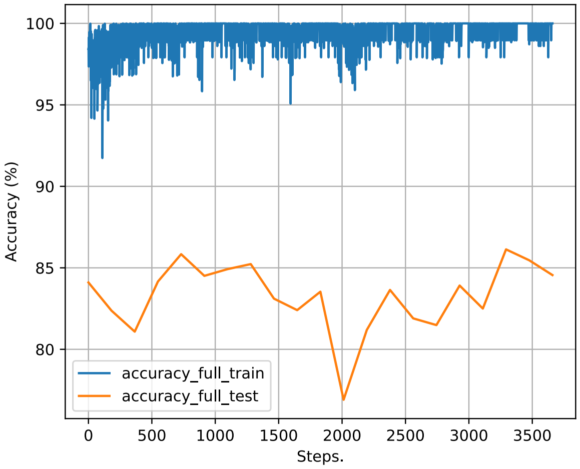


Fig. 11. Accuracy of the modified model during full training.

VIII. Conclusion and Future Work

To conclude, this experiment explored and implemented an existing CNN architecture on scene detection and classification. We included the two modes of training from the original paper (with more dataset), namely *non-full* and *full*. From the experimental results, we find that the best test accuracy (85.13%) that we achieved in a *full* training setting is rather close to the target specified (87.2%). Finally, we made an attempt to improve the architecture by adding one more dense layer between the existing dense layer and flatten layer. The improvement in terms of accuracy is small but noticeable. For taking this experiment further, we can consider augmenting the input data or change our learning algorithm to make more extensive comparisons with the same dataset.

References

[1] M. Valenti, A. Diment, G. Parascandolo, S. Squartini, and T. Virtanen, “Dcase 2016 acoustic scene classification using convolutional neural networks,” 2016.

[2] A. Mesaros, T. Heittola, and T. Virtanen, “Tut database for acoustic scene classification and sound event detection,” *2016 24th European Signal Processing Conference (EUSIPCO)*, pp. 1128–1132, 2016.

[3] L. Pham, H. Tang, A. Jalali, A. Schindler, and R. King, “A low-compexity deep learning framework for acoustic scene classification,” 2021.

[4] D. Zou and Q. Gu, “An improved analysis of training over-parameterized deep neural networks,” 2019.

[5] A. Jati, A. Nadarajan, K. Mundnich, and S. Narayanan, “Characterizing dynamically varying acoustic scenes from egocentric audio recordings in workplace setting,” 2019.

[6] S. Aziz, M. Awais, T. Akram, M. U. Khan, K. Khursheed, and M. Alhussein, “Automatic scene recognition through acoustic classification for behavioral robotics,” *Electronics*, vol. 8, 04 2019.

[7] H. Wang, Y. Zou, and W. Wang, “Specaugment++: A hidden space data augmentation method for acoustic scene classification,” 2021.

[8] M. Kośmider, “Spectrum correction: Acoustic scene classification with mismatched recording devices,” *Interspeech 2020*, Oct 2020. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2020-3088

[9] “What is a spectrogram?” 2021. [Online]. Available: https://pnsn.org/spectrograms/what-is-a-spectrogram