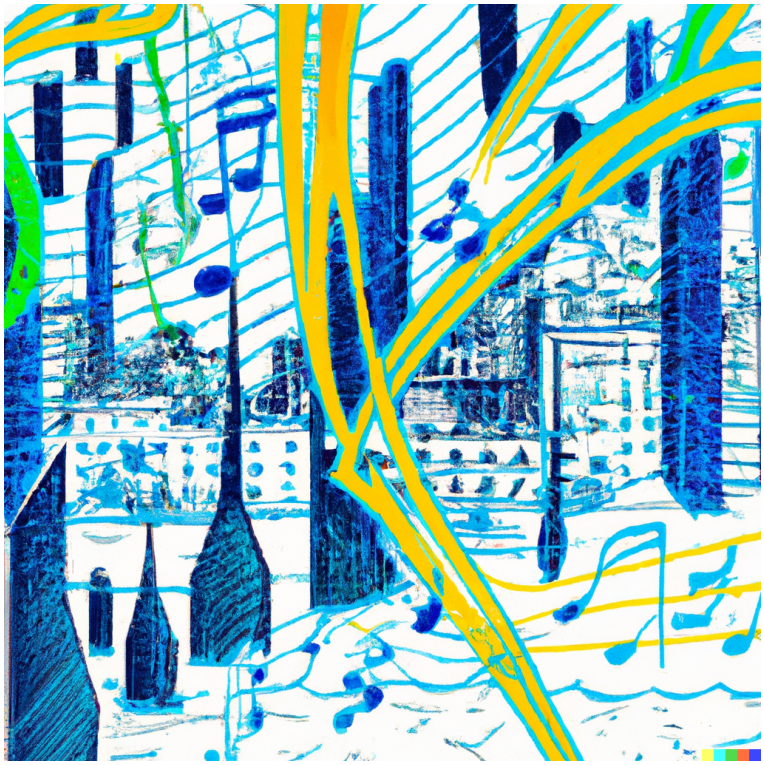
AUDIO EXPLORERS 2023



Team AudioBots

"roll out"

≫ Dennis Chenxi Zhuang

≫ Kasper Niklas Kjær Hansen

≫ Kristoffer Marboe

≫ Kristian Rhindal Møllmann

Sound scene classifier for hearing aids

An audio challenge by Oticon

April, 2023

ABSTRACT

With the need for accurate and effective methods for embedded devices such as hearing aids, interest has been sparked in the field of audio classification. This paper seeks to provide a journal of how audio classification using mel-spectrograms can be done using machine learning. Our CNN-based model proves to get an accuracy of 95.3% using 0.5M parameters, making it feasible to run on low-power devices. We discuss our predictions based on a confusion matrix, consider the further process and propose methods for further research.

1. INTRODUCTION

Hearing aids that can adjust themselves directly improves the quality of life for people who are hard of hearing. They help alleviate the social stigma attached to hearing impairments by effortlessly blending into their daily lives as they move through different sound environments. This challenge truly puts people first, and we are thrilled to have the opportunity to participate in Oticon's audio explorers challenge!

We view this task as an exploration of the possibilities for enhancing hearing aids with machine learning. This report will hence not be written as a classic scientific article but rather as a thorough journal of our process. We will present our general approach to the problem, followed by an exposition of our methods and models. Finally, our results will be presented and discussed with general considerations of the implementation and the project.

2. GENERAL APPROACH

Our first thought was that this was a classic image classification problem - a problem which has definitely been solved before. Since the data provided consists of spectrograms, the problem can be regarded as classifying pictures with size 32×96. The category "other" makes the challenge a bit more tricky, and after looking at the data, we, as the project also suggests, lowered our expectation of approaching 100% accuracy. Furthermore, we zeroed in on using TinyML since the model should run a small embedded device.

Another important consideration was the time constraint. A significant amount of time is used for training and modifying the models, so we had to get to that as soon as possible to arrive at satisfying results. We scheduled our training such that hyperparameter optimisation was done using random search the first week, and then we opted for the settings that performed best, with Bayesian optimisation arriving at the best model Wednesday in the second week. This gave us time to document the results and present the project in accordance with our findings. To optimise the process, we used Thinlinc to access DTU's High-Performance Computing [1]. The code is written in Python, using the frameworks Pytorch for creating the models and Pytorch Lightning to better scale and test different models.

A fundamental value we place great importance on is to deliver a structured and organised project which is straightforward to understand and dissect. To do this, we have tried to be as transparent in our work as possible and make it accessible online. Our repository can be accessed on GitHub[1]. To keep track of the performance and parameters of the models we tested, we used the machine learning operations platform Weight and Biases (WandB)[2] [2].

3. DATA EXPLORATION

One of the first steps was to explore the data we would use. A vital characteristic of the dataset is the distribution of classes. Table 1 shows the amount and distribution of the different sound categories/classes. Here it is quite evident that there is a severe class imbalance, with over half of the sound snippets being music. That is something we need to mind when considering our solution.

No.	Attribute name	Amount	Percent
0	Other	14,530	27.47%
1	Music	27,340	51.69%
2	Human voice	4,823	9.12%
3	Engine sounds	4,412	8.34%
4	Alarm	1,785	3.37%
–	Total	52,890	100.00%

Table 1: Overview of the amount and distribution of the different sound types.

Looking at the values of the spectrograms, they are all between -80 dB and 0 dB, with -80 dB being regarded as complete silence in that frequency band at the specific time frame. The data distribution can be seen in Appendix A Figure 5 where it is clear that while it is mostly Gaussian distributed, there is a significant peak at -80 dB.

Now we took a look at the data to get a grasp of the characteristics of the different classes. In Figure 1, three spectrograms of each category can be seen. Looking at the spectrograms, each of the categories possesses some different characteristics. The *music* spectrograms have distinct harmonics and rhythmic patterns and display various intensity levels on the frequency range. The *human voice* spectrograms are characterised by continuous resonant frequencies and glottal stops. Now the *engine sounds* spectrograms have more of a vibrating look, and one can almost hear the "brrrrrrr". The *alarm* is pulsating, especially evident in spectrogram 166, or a single high-pitched, loud tone is present. Lastly, the *other* spectrograms are harder to describe as a group since they are more mutually distinct. However, they tend to look more like engine sounds and human voice spectrograms. We also looked at how the mean spectrogram for each category looked, which

[1] GitHub: https://github.com/rreezN/AudioBots
[2] WandB: https://wandb.ai/audiobots/projects

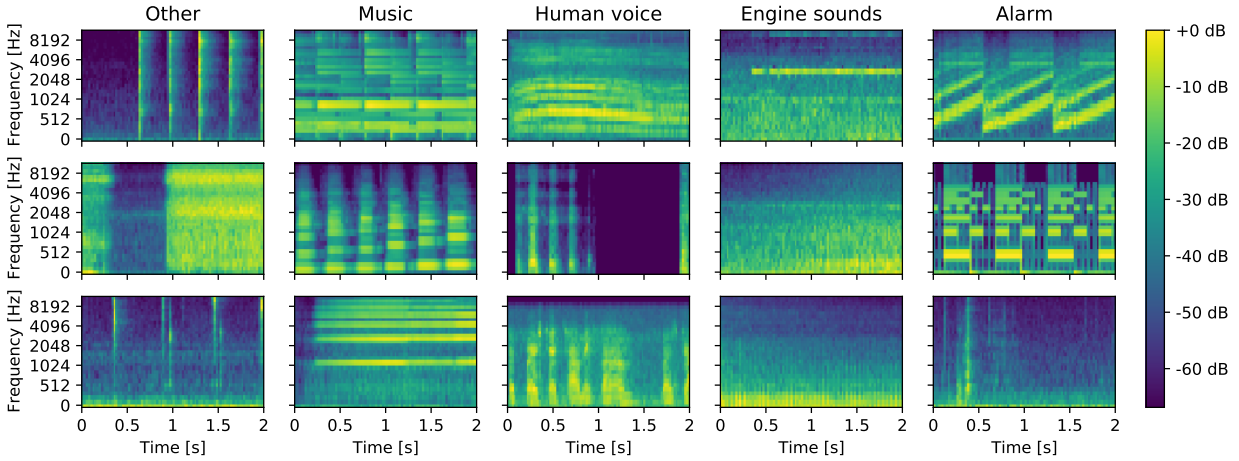


Fig. 1: Three mel-spectrograms of each of the five sound categories.

can be seen in Appendix A Figure 6.

Armed with an insight into how our data looked, we can now look into what methods we will use. This insight will also help us understand the results better.

4. METHOD

The dataset consists of many different spectrograms, each representing a different class, as specified in Table 1. They are essentially 2D matrices that illustrate the power spectral density of a signal. Because the spectrograms can be treated as 2D images, they can be directly fed into a convolutional neural network (CNN). This is no new feat, and literature has shown that treating spectrograms as images for CNNs is beneficial for this type of problem [3, 4, 5, 6, 7]. The convolutional layers of the CNNs serve to project the spectrograms into a high-dimensional many-fold where hidden structures in the spectrograms are separated in a way that enables the fully connected layers to distinguish between the five classes.

When constructing the network, several factors have to be taken into account. With a limit of 500K parameters, methods applied together with the CNN model must mitigate this constriction. In the following four sections, we will elaborate on how we preprocessed data and the three major areas where we focused on optimising the model's performance, namely factors related to the network, the training and data augmentation.

4.1. Preprocessing

Preprocessing the dataset is important to ensure its suitability for neural network training. To achieve this, we divided the dataset into 80% training data, 10% validation data, and 10% testing data to enable model evaluation on unseen data and, thus, ensure accurate generalisation. Additionally, we standardised the data to have a mean of 0 and a standard deviation of 1. This transformation is beneficial for models that rely on gradient-based optimisation methods, and it can also help to speed up training by reducing the scale of the input features. Furthermore, standardisation is more suited for our data than normalisation since it is primarily Gaussian distributed apart from the spike with the value -80; however, standardisation handles this better [8].

4.2. The Network

When constructing a CNN, not only does the size of the layers play a part in the number of parameters in the network, but also the kernel size. To not blow up the number of parameters, kernel sizes were restricted to 3×3 and 5×5 with a zero-padding of (1, 1) and (2, 2), respectively, to maintain the spatial dimension and reduce the risk of information loss during the convolution operations.

Following the convolutional part of the model, the resulting many-fold is collapsed into a lower-dimensional space using adaptive average 2D pooling. This is followed by a few fully connected layers to gradually decrease the dimensionality before downscaling to the five output classes.

Learning using gradient descent methods heavily relies on the starting point. Therefore, all weights in the model are initialised using Kaiming normal initialisation to fit with the activation functions used in this project which all have a mean greater than 0 [9]. Additionally, the outputs are normalised between each convolutional layer using batch normalisation to keep each learning step stable.

4.3. Training

During training, there is a plethora of parameters that can be tuned and engineered to maximise model performance. We will here describe the most critical ones and explain how we optimised the training process to cover as much ground as possible.

As mentioned, we used the machine learning operations platform, Weights and Biases, to allow for easy tracking and visualisation of model performance, hyperparameters, and other relevant information.

One of the key features of WandB is the ability to perform sweeps, which automates tuning hyperparameters. A sweep involves defining a set of hyperparameters, their possible values, and the range of values to be searched. WandB then automatically selects the combination of hyperparameters to train the model, making finding the best combination for optimal performance easier.

Let's explore some of the hyperparameters we tuned in our training process using WandB sweeps.

The activation functions we experimented with were: Rectified Linear Unit (ReLU), Leaky Rectified Linear Unit (LeakyReLU), and Exponential Linear Unit (ELU). ReLU is the most commonly used activation function, while LeakyReLU and ELU can help to address the issue of "dead" neurons in ReLU.

Dropout was used as a technique to promote generalisation. We experimented with different dropout percentages in the range [0.0, 0.6] to find the optimal value for our model.

Two **loss functions** were considered: cross-entropy loss and focal loss. Cross-entropy loss is commonly used in classification tasks, while focal loss is designed to address class imbalance problems in the data [10]. Given the distinct class imbalance in the given dataset, we considered focal loss a favourable choice.

For **the learning rate**, we experimented with different initial values in the range [5e-5, 1e-2] and used a scheduler called ReduceOnPlateau[3] to adjust the learning rate during training once learning stagnates.

batch size scheduling was used to help the model converge more quickly and accurately. We started with a batch size of 8 and gradually increased it in increments of [8, 32] until epoch 48. From then on, we used a batch size of 128 for the remaining training.

Max epochs and **early stopping** are critical aspects of neural network training to prevent overfitting, ensure model convergence and decrease time waste on non-improving models. Thus, we set the maximum number of epochs to 150 and implemented early stopping if the validation loss did not improve for 30 epochs. After each training session, we chose the model with the lowest validation loss as the final candidate.

4.4. Data Augmentation

The dataset used in this study exhibits a significant degree of class imbalance, evident from Table 1. To address this issue, we have implemented three data augmentation techniques well-suited for spectrograms and their inherent patterns and information stored in the time and frequency domain. The use of data augmentation not only mitigates the impact of class imbalance on the training of the model but also facilitates out-of-distribution occurrences.

[3] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

Certain techniques, such as flipping, zooming or rotation, are discouraged as they might impact the underlying information so that it stops representing the actual category. We have opted for the following techniques to preserve the underlying frequency and time information in the spectrograms: shifting along the time and frequency domain, time stretching and noise.

During training, each data-augmentation technique gets a pre-defined probability of being applied. Let p^* denote the probability of a data-augmentation technique being applied. The probability for all methods is pre-defined to $p^* = 0.2$. The low probability of augmentation is chosen to ensure that the model trains on all of the provided data, including some augmented data. The three data augmentation techniques are implemented as follows.

Shifting along the Time and Frequency domain, which randomly shifts the spectrogram along the time axis by a maximum of 10% of the length of the input, or along the frequency axis by a maximum of 10% of the input's frequency dimension.

Time Stretching that either stretches or compresses the spectrogram along the time axis by a random factor between 0.8 and 1.2.

Noise adds Gaussian noise to the spectrogram with a random amplitude between 0.01 and 0.1.

Overall, these augmentation methods are designed to preserve the inherent information in the spectrograms while creating enough variability in the data to improve model generalisation and robustness gradually.

5. MODELS & EXPERIMENTS

The study involved the creation of two CNNs, a large model (AudioBot500) with a parameter size of 499.853 and a small model (AudioBot50) with 49.733. The primary purpose of this investigation was first to see what accuracy we could achieve using the largest possible model and afterwards conduct a comparative analysis between the models' performance metrics, thereby shedding some light on the relationship between model performance and model size.

For the network, inspired by Keton Doshi's blog post [11], we settled on a general architecture that first featured 3-4 convolutional layers in increasing size, followed by an average pooling layer finished off by a few fully connected layers. The specific architecture for AudioBot500 can be seen in Figure 3. Figure 7 in Appendix A illustrates the architecture for AudioBot50.

5.1. Hyperparameter Optimisation

Given the many hyperparameters to tune, we opted for a random search to identify the key factors influencing model performance. This approach gave us an initial indication of the

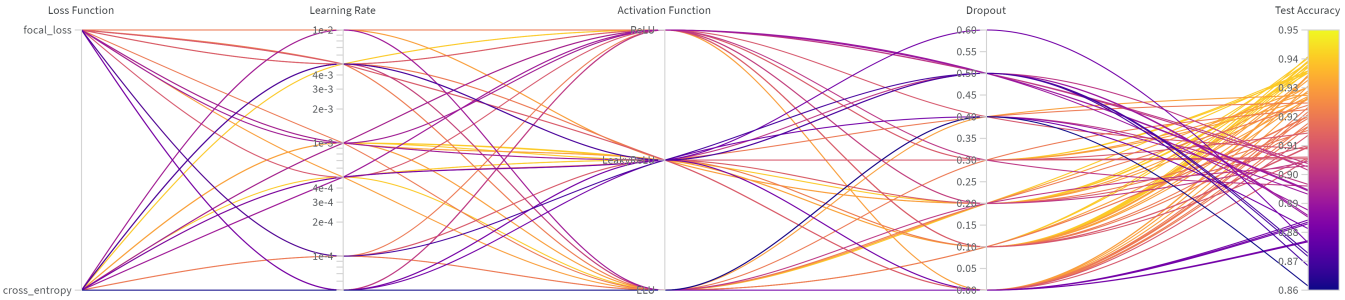


Fig. 2: Results of sweep with 77 runs using random search.

most critical hyperparameters before performing a more focused search.

Figure 2 shows the results from the 77 runs in a sweep with random search, based on the metrics: loss function, activation function, learning rate and dropout, with the response variable: test accuracy. While the figure possesses a lot of information, the most important is that the choice of loss- and activation function had a minor impact on the model performance. At the same time, the learning rate and dropout percentage played a huge role in obtaining maximum accuracy. Specifically, the best performance is seen for a medium learning rate and a relatively low dropout percentage.

Therefore, we decided to exclude ELU while narrowing down the learning rate and dropout percentage to optimise the model performance fully. A second sweep was done again using a random search, and the results hereof can be seen in Appendix A, Figure 8. From these results, we saw cross-entropy loss outperforming focal loss, while we also decided to go with ReLU for the last sweep. For the final sweep, we only investigated dropout percentages in the range [0.0, 0.1] and learning rates in the range [3e-4, 1e-3]. We used WandB's integrated Bayesian optimisation to scrutinise and exploit the most optimal hyperparameters. The results can be seen in Appendix A, Figure 9.

Once the "tuning dust" had been *swept* away, the most promising hyperparameters were chosen. ReLU as the activation function showed the most consistent performance for AudioBot500 combined with a learning rate of 0.00065 and a dropout percentage of 0.08. For AudioBot50, LeakyReLU was used as the activation function, with a learning rate of 0.0021 and a dropout of 0.026. Cross-entropy was used as the loss function for both models.

To validate and compare the final models, 10% of the data was randomly selected as test data. The remaining 90% was then split 9:1 for training and validation across a 10-fold cross-validation.

6. RESULTS

In Table 2, a summary of the models and a generalised estimate of their performance can be seen with their respective standard deviations. The larger model outperforms the smaller model reaching an average accuracy of 95.3% on the 10% unseen test data, whereas the smaller model achieves an accuracy of 91.9%.

To further investigate the performance of AudioBot500, a confusion matrix of its predictions on the 10% unseen test data is shown in Figure 4. Notably, *music* and *alarm*, which supposedly have the most prominent harmonic features, are correctly classified almost all the time. The remaining classes are still correctly classified around 92% of the time; however, the confusion matrix shows that the majority of the misclassifications happen between *other* and *human voice* and *engine sounds* respectively.

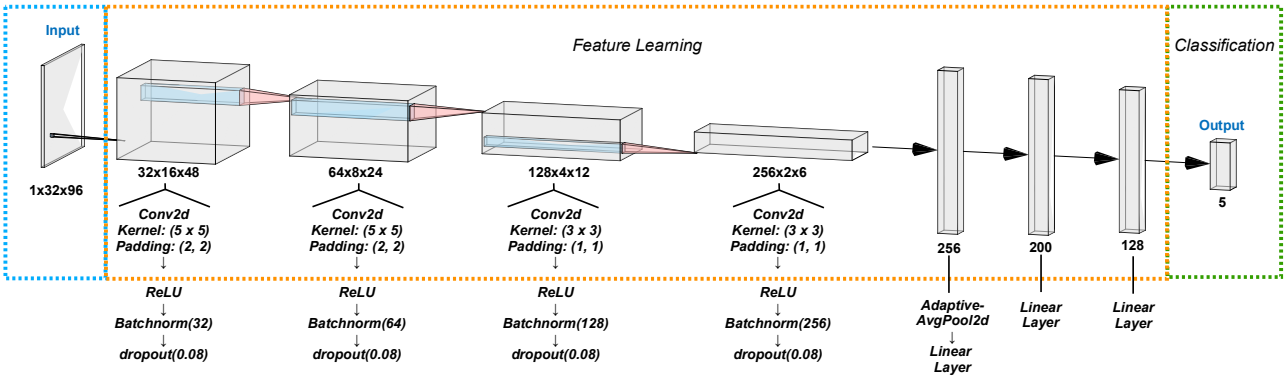


Fig. 3: Architecture of the AudioBot500 CNN.

	(Params, MACs (M))	Test acc	Test loss
AudioBot500	(500K, 17.6)	0.953 ± 0.002	0.153 ± 0.003
AudioBot50	(50K, 3.8)	0.919 ± 0.003	0.228 ± 0.007

Table 2: Cross-validated model performances with standard deviations.

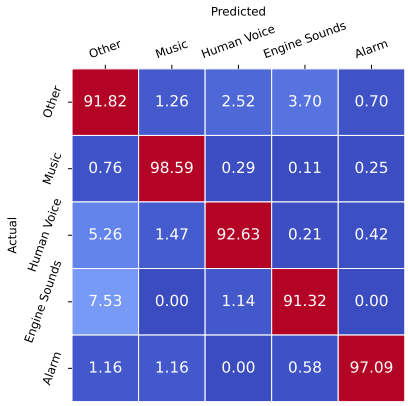


Fig. 4: Confusion matrix of the predictions by AudioBot500 on the 10% unseen test data.

7. DISCUSSION & CONCLUSION

The test accuracy of 95.3 is quite satisfying given the restriction in model size. Additionally, even with class imbalance, it is evident that the model has an acceptable performance on the classes which constitute less than 10% of data.

Even though the *other*-category constitutes approximately 28% of all data, it has the second-to-worst prediction accuracy. However, this is reasonably expected as the soundbites in that category could contain fragments and combinations of sounds in the remaining four categories. Specifically, our model is seen to confuse *engine sounds* and *human voice* for *other*.

A trend across all classes is that most mispredictions are made to the *other*-category. This illustrates the problem of having four distinct classes and then one class that, in theory, could be anything.

Whether some categories are more important to classify correctly can be discussed. Sergei Kochkin, PhD, and executive director of the Better Hearing Institute (BHI), argue that "the essence of a hearing aid is to improve speech intelligibility in listening environments important to the user" [12]. While this proposes that the *human voice* category is the most important for users, one cannot object against the importance of correctly classifying *engine sounds* to the benefit of enhanced safety for the user. The alarm category also holds importance since its sound is an indication of a significant occurrence taking place. Lastly, the *music* category would only be for pleasure, or perhaps work for some, so that might be the least important category to classify correctly. The *other* is difficult to place since its spectrograms are too diverse.

When the model is implemented, the hearing aid's sound settings could potentially be adjusted based on the prediction of several sound snippets. This may increase the accuracy of the settings since several predictions would have to be wrong for them to be adjusted incorrectly. However, it should be balanced with the time needed for the hearing aid to adjust so the user does not experience a wrong setting for too long.

The model used in this study is designed to be applicable on smaller embedded devices, and as such, it is constrained to at most 500K parameters. While it was not applied in this study, it is important to note that it is possible to enlarge the network architecture by a substantial amount while maintaining the parameter size, which consequently could provide a better prediction model. In a paper written by Kaiser et al. [13], it is discussed and empirically shown that *Depthwise Separable Convolutions* (DSC) can obtain better models than previously possible for a given parameter count. Using the same model architectures but applying DSC instead of regular convolutions could make ensemble learning feasible. A potential alternative for ensemble learning could be using smaller models like AudioBot50, which size is one-tenth of the size restriction while maintaining a respectable generalisation accuracy, albeit three percentage points worse than the full model, AudioBot500. Ensemble learning is an excellent method used to account for the large class imbalance evident in the training data and the models' individual noise and false-positive and false-negative predictions.

Transfer learning can be employed as another technique to enhance the model's performance. With transfer learning, a significantly larger and more precise *teacher* network can be established, and its knowledge can be distilled into a *student* network like AudioBot500 or AudioBot50 by using a *Teacher-Student Distillation* technique [14].

Hence both ensemble learning and transfer learning with distillation could improve the model's overall performance while accounting for applicability on smaller embedded devices. Furthermore, given more time, we could have tried to handle the peak at -80dB differently, e.g. by using Gaussian Mixture models where we model the distribution as a mixture of two distributions.

In conclusion, we believe that our current model is a good minimum viable product, ready to be taken to the next step in the process. There are several possibilities for further development, which include Depthwise Separable Convolutions and ensemble learning for the current model or transfer learning with distillation if a new alternate method is of interest.

Our work process has felt deliberate and well-executed, and we have enjoyed tackling the problem using our technical toolbox. Finally, it has been purposeful and rewarding to see how technological advancement can improve people's quality of life in the real world.

8. REFERENCES

[1] DTU, "HPC website for DTU," 2023, https://www.hpc.dtu.dk/.

[2] Weights Biases, "Weights & biases website," 2023, https://wandb.ai/site.

[3] Karol J. Piczak, "Environmental sound classification with convolutional neural networks," in *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2015, pp. 1–6.

[4] Nirmal M R and Shajee Mohan B S, "Music genre classification using spectrograms," in *2020 International Conference on Power, Instrumentation, Control and Computing (PICC)*, 2020, pp. 1–5.

[5] Justin Salamon and Juan Pablo Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017.

[6] Zheng Fang, Bo Yin, Zehua Du, and Xianqing Huang, "Fast environmental sound classification based on resource adaptive convolutional neural network," *Scientific Reports*, vol. 12, no. 1, pp. 6599, Apr 2022.

[7] Wenjie Mu, Bo Yin, Xianqing Huang, Jiali Xu, and Zehua Du, "Environmental sound classification using temporal-frequency attention based convolutional neural network," *Scientific Reports*, vol. 11, no. 1, pp. 21552, Nov 2021.

[8] Tashimit, "Normalisation vs. standardisation," 2023, https://www.codingninjas.com/codestudio/library/normalisation-vs-standardisation.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[10] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár, "Focal loss for dense object detection," 2018.

[11] Ketan Doshi, "Audio deep learning made simple: Sound classification, step-by-step," 2021, https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5.

[12] Sergei Kochkin, "Marketrak v," *Hearing Journal*, vol. 53, no. 2, pp. 36, 39–41, 2000.

[13] Łukasz Kaiser, François Chollet, and Aidan N. Gomez, "Depthwise separable convolutions for neural machine translation," *ICLR*, 2018.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," 2015.

Appendix A

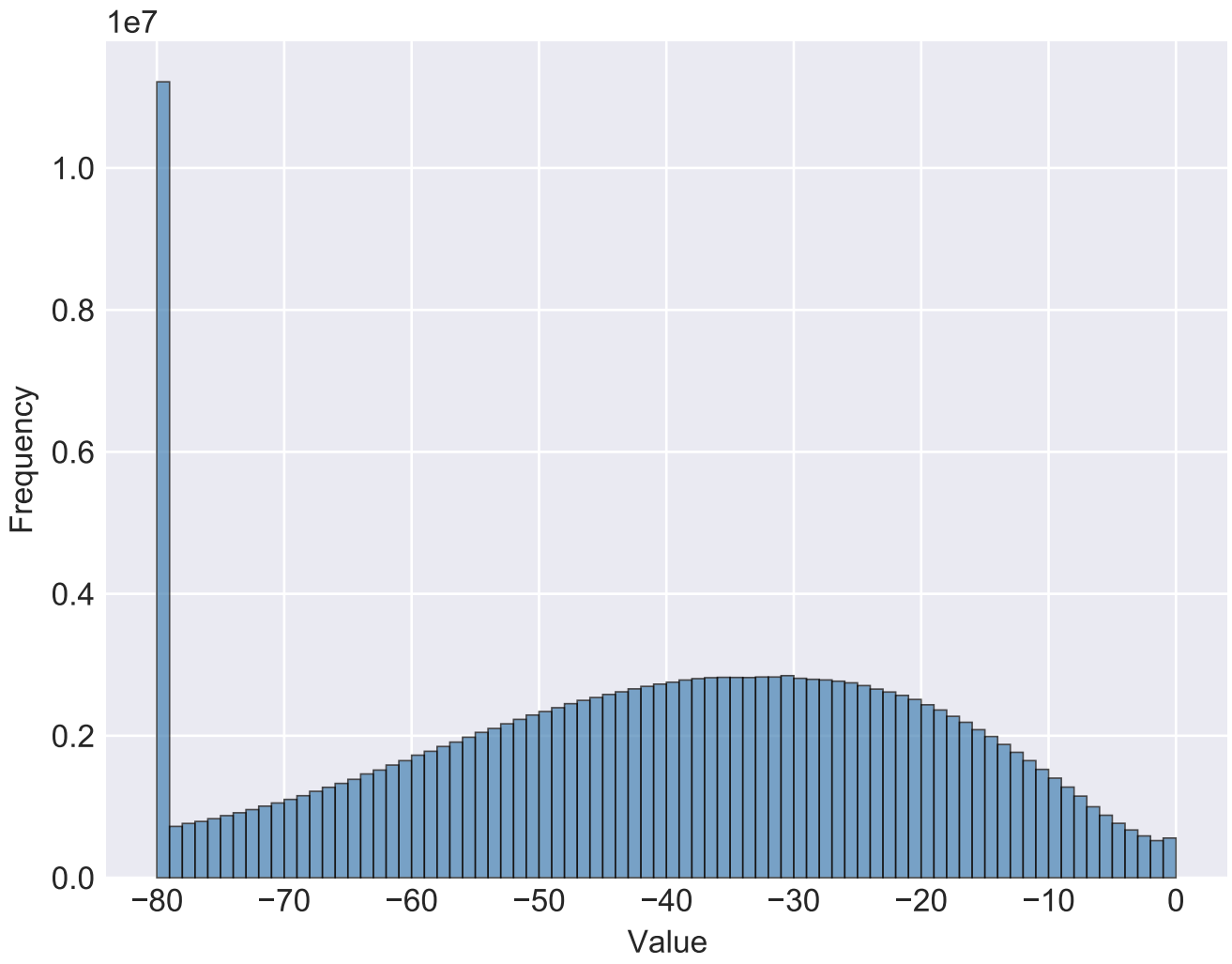


Fig. 5: The distribution of values in the mel-spectrograms. It is mostly Gaussian however with a significant peak at -80 dB.

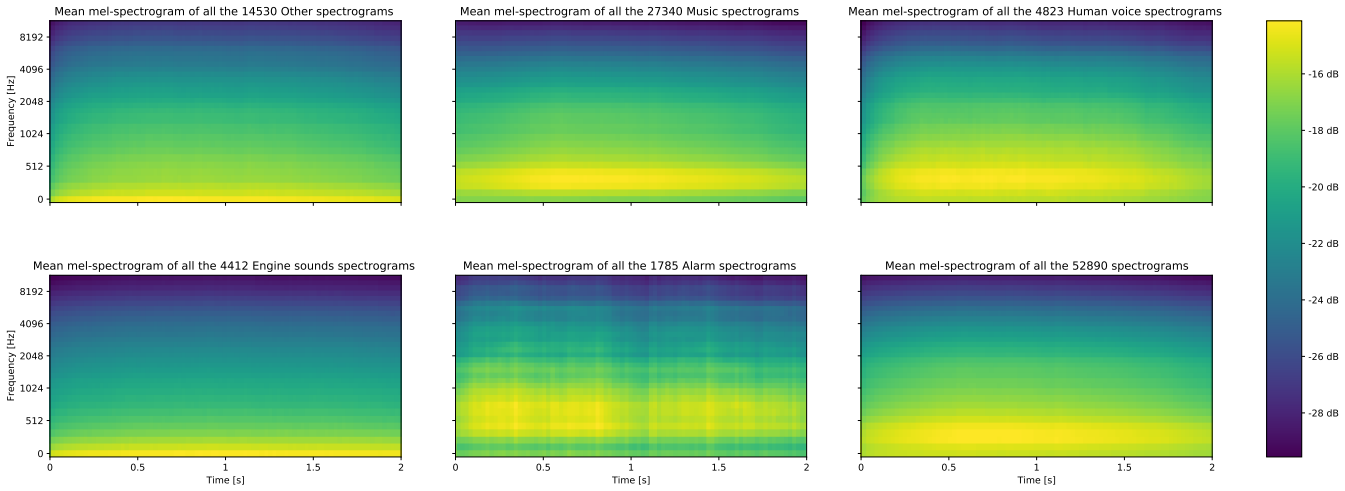


Fig. 6: The mean spectrograms for the 5 categories.

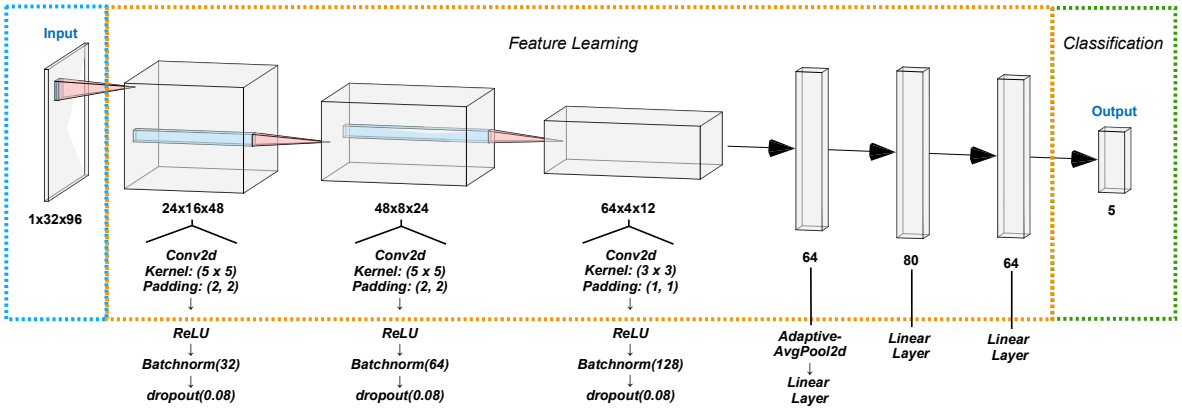


Fig. 7: Architecture of the AudioBot50 CNN.

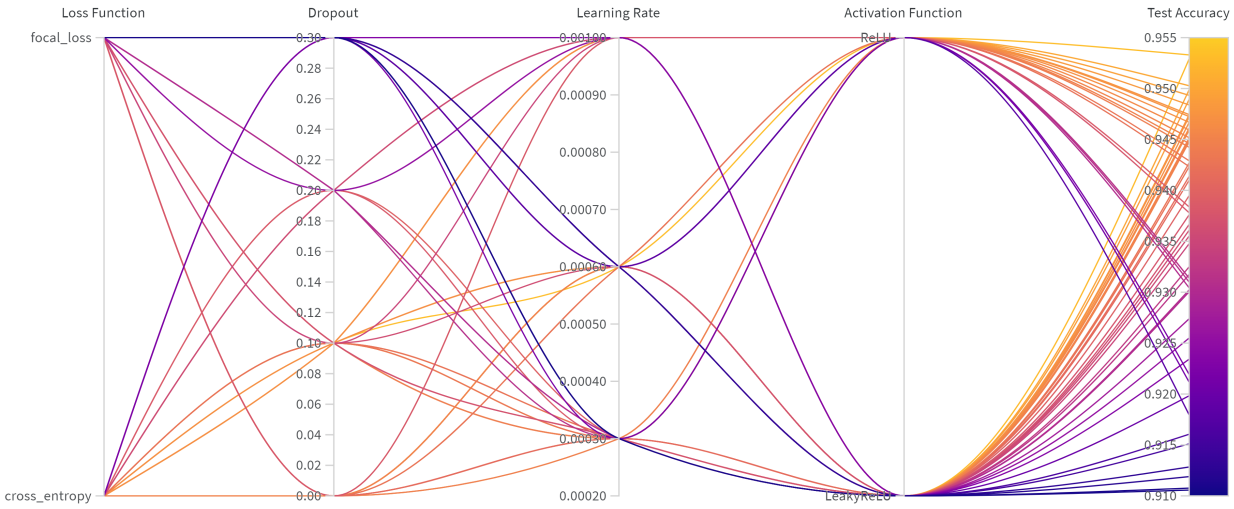


Fig. 8: Results of the second sweep with 58 runs using random search.

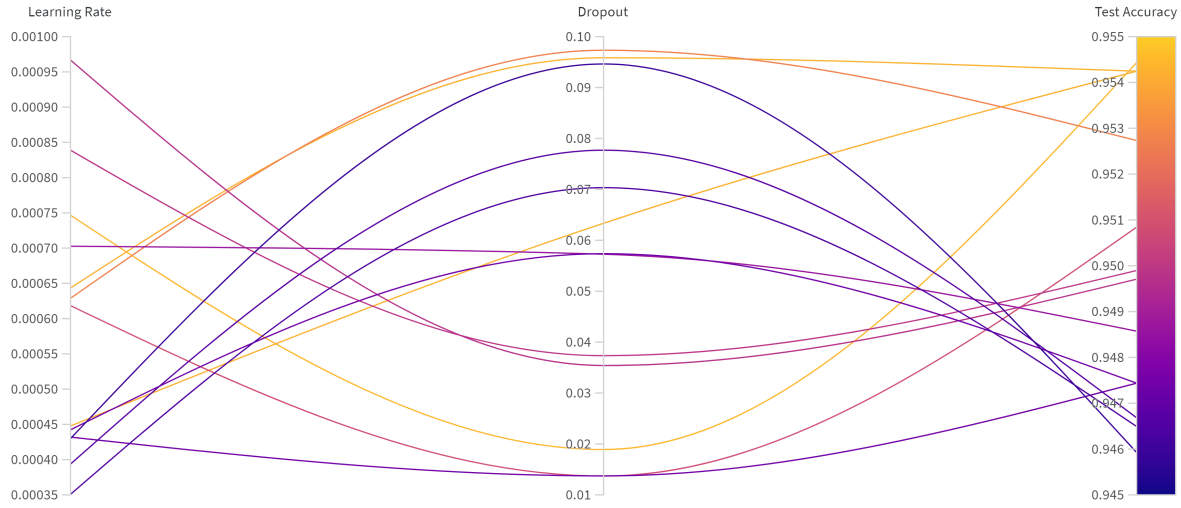


Fig. 9: Results of the third sweep with 13 runs using Bayes search.