Environmental sound classification

Michele Coluccelli, 1656336
Emanuele Musumeci, 1653885

I. Introduction

In our project we faced the challenge of *environmental sound classification*, that consists in classifying audio recorded in a specific environment (for example our dataset was entirely recorded in a urban environment), using convolutional neural networks. Being new to this kind of task, we wanted to experiment with various ways to perform data augmentation both on audio clips and on their corresponding log-mel spectrograms. We also experimented with ways of visualizing the ”understanding” the neural network acquires throughout training with both representations of the ”gradient flow” and saliency maps.

II. Dataset

The dataset used is UrbanSound8K [1], which contains 8732 labeled sound excerpts (of approximately 4 seconds in length) of sounds recorded in urban a environment located in 10 folds from 10 classes: air_conditioner, car_horn, children_playing, dog bark, drilling, engine_idling, gun_shot, jackhammer, siren, and street_music. Classes are very heterogeneous, which simplifies classification (on a task that is anyway inherently complex). It is characterized by a limit of 1000 slices per class avoiding large differences in the class distribution. The class labels are drawn from the urban sound taxonomy. The distribution of the dataset in its classes is represented below and in figure 5.

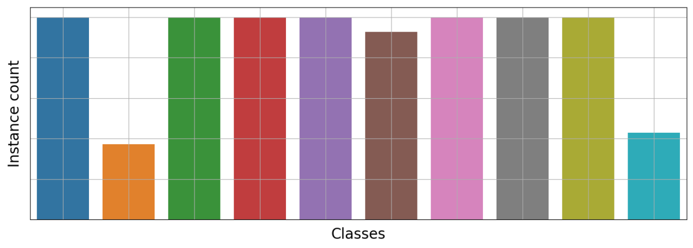


Fig. 1: Class distribution in the dataset

III. Data preprocessing

In our project we performed all the augmentations described in [2] and we also experimented with augmentations applied directly on the spectrograms.

A. Preliminary preprocessing

Clips from the UrbanSound8K dataset were resampled to 22050 Hz. This choice is a reasonable balance between the number of samples, which directly influenced training time, and the loss of quality in the audio clip. All clips were then cut or padded to exactly 4 seconds directly using the *numpy* [3] library. The background noise clips used in the audio augmentation step (see following section) were the same ones used in the official implementation of [2].

B. Audio Augmentation

We experimented with 4 different audio data augmentations. Each deformation is applied to the audio signal, which then is converted in a log-mel-spectrogram that is the input representation for both the networks (see the following section for an in-depth analysis of the spectrogram generation). The deformations and obtained augmentations sets are described below (and can be graphically visualized with the examples in C):

- *Time Stretching (TS)*: without changing the pitch, it slows down or speeds up the sample. Each audio is modified by 4 factors: {0.81, 0.93, 1.07, 1.23}.
- *Pitch Shifting (PS1)*: modifies the pitch of the audio sample without changing the duration. Each sample was pitch shifted by 4 integer values, representing the number of shifted semitones: {-2, -1, 1, 2}.
- *Pitch Shifting (PS2)*: as in the original paper we created a second augmentations set. This time 4 float values were used: {-3-5, -2.5, 2.5, 3.5}.
Doing this we noticed an increment of 5 percent and 3 percent respectively in accuracy and f1-score.
- *Dynamic Range Compression (DRC)*: compress the dynamic range of the sample using 6 parameterizations all taken by the Dolby E standard:
  - *music standard*
  - *film standard*
  - *speech*
  - *radio*
  - *film light*
  - *music light*

  The first 4 parameterizations are the same used in [2].

- *Background Noise (BG)*: mix the sample with 4 acoustic scenes: street-workers, street-traffic, street-people, park. Each mix z is obtained using

$$z = (1 - w) \odot x + w \odot y \tag{1}$$

where x is the original audio sample, y is the background signal (which was previously cut to the same length of the audio clip), and *w* is a weighting parameter choosen randomly from a uniform distribution in the range [0.1, 0.5].

Time Stretching and Pitch Shifting were applied using the *librosa* library [4]. Background Noise uses an implementation based on *numpy* [3] while Dynamic Range Compression uses the *MUDA* library [5].

C. Spectrogram generation

Our classification method uses a *Convolutional Neural Network (CNN)* as a feature extractor. The audio data, augmented as described in the previous section, following [2], is then used to generate *log-mel spectrograms*. An audio clip can be considered as a discretized audio signal, obtained by sampling the original audio signal, which is continuous, at a certain frequency (which in our case was 22050 Hz). The obtained signal is a sequence of discrete values $\mathbf{x}[\mathbf{n}]$ where n is the position of the n-th sample (also called *frame*). If we consider a sliding window $w[n]$, we can subdivide the frames of this clip into shorter (possibly overlapping) segments, obtained by the sliding window, each window being of a fixed length. By computing the *Fourier transform* on each of these shorter clips, we obtain the *Short-time Fourier transform (STFT)*. This process can be summarized by the following equation:

$$X(m, \omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \tag{2}$$

In practice other parameters are taken into account when computing the STFT, such as the *window length* (that is the amplitude of the sliding window) and the *hop length* (that is the distance, in frames, between the centers of two subsequent windows).

In our case, following the same values in the paper, the *hop length* is 512 and the window length is 2048 (samples).

A spectrogram for the clip can then be computed as follows:

$$\text{spectrogram}\{x(n)\}(\tau, \omega) \equiv |X(\tau, \omega)|^2 \tag{3}$$

where n is actually the position of the sample in the clip. Practically, each clip, considered as a sequence of values representing "*air pressure*" depending on the recorded sounds, is therefore turned in a sequence of sequences of values, each sequence being the Fourier transform at a certain instant (or computed from a certain time interval), representing the frequency spectrum (comprising of both magnitude and phase) of the recorded sound at the corresponding instant (or time interval). The spectrogram is therefore derived from the magnitude alone of these frequency spectra.

Given that our task required classifying the environment where a sample was recorded "*as a human would*", another step was taken (according to [2]). The range of frequencies perceived by humans is not distributed linearly. Instead, sounds perceived as equally distant in pitch (therefore frequency) are actually distributed non-linearly in the spectrum of human hearing range (which is variably distributed between 20 Hz and 20000 Hz). This non-linear scale is called the *mel scale* and its distribution is shown below (Fig. 2):

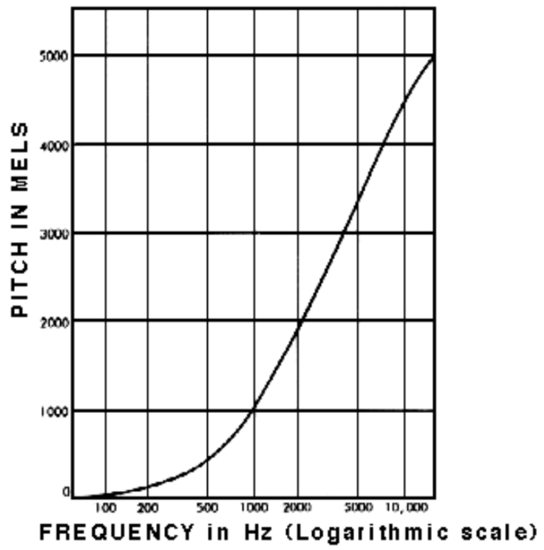


Fig. 2: Representation of the mel scale (taken from [6]). As we can see in this graph, below 500 Hz, sounds are "judged" by human listeners to have the same distance on the mel scale as the one they really have in frequency (so in this range, scales coincide), while above 500 Hz, larger and larger intervals are judged by listeners to produce equal pitch increments.

So it makes sense to redistribute on the mel scale frequencies computed for each frame using the *STFT*. From this scale, the continuous human hearing range can be itself discretized in a number of *bins* (therefore, given a bin, a sound from the previous bin and a sound to the following bin would be "perceived" approximately equidistant in pitch).
As an additional step, the mel scale is then converted to *decibel (dB)*, obtaining the *log-mel spectrogram* of the original audio clip.

All these operations were implemented using the *librosa* [4] library. This library offered a self-contained function for computing the mel-spectrogram, using 128 bins for the mel scale, which was then converted to the logarithmic scale (in decibel).

D. Sub-clip extraction

Given a clip of length T (seconds), sampled with a sampling rate S_r (Hz), the total number of samples N in the clip can be determined as $N = S_r \cdot T$. During training, from each audio clip used for training, a sub-window of W seconds is extracted starting at a random sample of the clip, chosen in the interval $[0, N - WS_r]$, where N is the total number of samples in the clip. Instead, during evaluation the window is always extracted from the beginning of the clip, as most classes of clips feature the sound in the first part of the recording rather than in the last. For reasons that will be more clear by reading section A of our Experimental Analysis, the sub-window might be extracted directly from the spectrogram itself, determining the starting frame in the spectrogram using the equivalence $\lfloor \frac{F}{H} \rfloor$, where F is the starting frame in the audio clip and H is the spectrogram *hop length* .

E. Spectrogram Augmentation

In some of our experiments, also some additional augmentation were applied directly on the generated spectrograms:

- *Gaussian Noise*: with a probability of 0.55, adds zero-mean gaussian noise to the spectrogram, with a *standard deviation* σ determined as:
$$\sigma = |C \min(I)| \tag{4}$$
where I is the spectrogram and C is a constant with value 0.03.
- *Spectrogram Shift*: with a probability of 0.9, shifts the spectrogram to the right of a random number of samples that is extracted from a range $[1, 4]$, distributed uniformly.

These are entirely implemented in *numpy* [3]

F. Additional features

Additionally following a technique seen in the paper [7] we generated up to two additional features (on different channels) of the spectrograms, by computing their *deltas* and *delta-deltas*. In audio recognition tasks, the mel-frequency cepstral coefficients (MFCCs) are a classic input feature describing the instantaneous, spectral envelope shape of the speech signal. In particular, acoustical signal can be described as a sequence of transitions between phonemes. A method to extract information about these transitions is computing the first difference of signal features, called *delta* [8]. Given a feature $\mathrm{f_k}$, at time-instant *k*, the corresponding delta is defined as

$$\Delta_{\mathrm{k}} = f_{\mathrm{k}} - f_{\mathrm{k\text{-}1}} \tag{5}$$

The second difference, known as the *delta-delta*, is correspondingly

$$\Delta\Delta_{\mathrm{k}} = \Delta_{\mathrm{k}}\Delta_{\mathrm{k\text{-}1}} \tag{6}$$

Since these are estimates of the derivatives, they are not particularly accurate and this results in an output noisier than the input, without changing the original signal. Nevertheless, delta and delta-delta features can still provide an improvement in convergence speed.

IV. Model

In this study we did a a comparison between the SB-CNN by Salamon and Bello [2] and our custom CNN based model, which is deeper than the former.

A. SB-CNN

As a baseline, we used the deep convolutional neural network (CNN) proposed by Salamon and Bello composed by 3 convolutional layers interleaved with 2 pooling opearations, followed by 2 dense layers:

- l_1: 24 filters with receptive field of (5,5), s.t *W* has the shape (24,1,5,5) followed by (4,2) strided max-pooling over last 2 dimensions(time and frequency) and a ReLU activation function *h(x) = max(x,0).*
- l_2: 48 filters with receptive field of (5,5), s.t *W* has the shape (48,24,5,5) followed as in l_1 by (4,2) strided max-pooling and ReLU activation function.
- l_3: 48 filters with receptive field of (5,5), s.t *W* has the shape (48,48,5,5) followed by a ReLU activation function without pooling operation.
- l_4: 64 hidden units s.t *W* has the shape (2400,64) followed by a ReLU activation function.
- l_5: 10 output units 1 for each class. The *W* shape in this case is (64,10).

B. Custom Neural Network

This network uses as a convolutional CNN as feature extractor, composed of 4 blocks, each comprising 2 convolutional layers and a max pooling layer, and a classifier composed by 2 fully connected layers.
The proposed CNN is parameterized as follows:

- l_1: 30 kernels with receptive field of (3,3).
- l_2: 30 kernels with receptive field of (3,3).
- p_1: (4,2) strided max-pooling and kernel size (2,2).
- l_3: 60 kernels with receptive field of (3,3).
- l_4: 60 kernels with receptive field of (3,3).
- p_2: (4,2) strided max-pooling and kernel size (2,2).
- l_5: 90 kernels with receptive field of (3,3).
- l_6: 90 kernels with receptive field of (3,3).
- p_3: (4,2) strided max-pooling and kernel size (2,2).
- l_7: 120 kernels with receptive field of (3,3).
- l_8: 120 kernels with receptive field of (3,3).
- p_4: (4,2) strided max-pooling and kernel size (2,2).

with the last two fully-connected layers having respectively an input size of 1920 and 512 features. l_5. Both models were are implemented using PyTorch [9].

C. Regularization

To reduce overfitting, in both models have we applied dropout to the input of the last two layers with a probability of 0.5. As an additional generalization technique, we applied L2-regularization (see the Experimental setup section).

V. Experimental analysis

A. Training time optimization

When performing the first training, we noticed that augmenting samples on-the-fly required too much time, given the number of different experiments we wanted to conduct, the main bottlenecks being the loading of audio clips and the generation of spectrograms (in particular, the most expensive operation was the computation of the Short-time Fourier Transform). Training time reached up to 1h per epoch.
The data augmentation method proposed in [2] uses random values for all the augmentation techniques, extracted randomly from a discrete and small set of optimally performing elements (for example the Pitch Shift augmentations, PS1 and PS2, extract a value randomly from a set of just 4 different pitch shifting amplitudes). Therefore we decided that the best way to optimize training time was to preprocess each sample with all possible augmentations and their values (one at a time, as seen in the paper). To overcome the computational bottleneck of loading and preprocessing sounds, we created a small utility that performed all the required preprocessing, generating the spectrogram of the preprocessed result (therefore a spectrogram for each required random value of each augmentation) and then saved both the preprocessed clip (a WAV file) and the spectrogram in a *memory-mapped* file, that minimized the time required to stream data to the neural-network. To emulate the random extraction of a data augmentation parameter at training time, the pre-processed clip (or spectrogram altogether) is randomly extracted instead.
The resulting speedup reached in some cases a factor of 13x (down to 1:30 minutes per epoch).

B. Model evaluation

When deciding our experimental setup, we noticed that the success of the model in avoiding overfitting on the training set was highly dependent on the number of different samples used for training. For this reason we decided not to use a *dev* set and simply use a *training* set comprising 90% of the dataset and the remaining samples as a *test* set (imitating what is done on the paper [2]), which corresponds to using 9 dataset folds for training and 1 fold for validation. Although the paper [2] performs 10-fold evaluation, testing the model by using each time a different fold as test set and training on the remaining folds, in order to be conservative on time and to be able to perform more experiments, we decided to just train all models on folds 1 to 9 and evaluate them on fold 10. Moreover, the paper [2] performs a very peculiar evaluation procedure, generating a window for each frame, for each testing sample and then choosing the class that had the highest mean activation across all windows. We chose a more naive approach, by performing prediction on the spectrogram generated from the whole 4 seconds window, keeping the evaluation as general as possible, to avoid biasing results, given the heterogeneity of the experiments we wanted to perform.
Although as a main performance metric for model selection we chose the *test accuracy*, as [2] did, two out of the 10 classes (dog_bark and car_horn) have a noticeably lower number of samples, which led us to consider also the *f1-macro* score, a harmonic mean of *recall* and *precision* computed as follows:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{7}$$

This metric gives a more accurate insight in the network ability to correctly predict also samples from less represented classes.

C. Experimental setup

The SB-CNN, which is the shallower network proposed in [2], was trained with *Stochastic Gradient Descent*, at a constant learning rate of 0.01, optimizing the Categorical Cross-Entropy loss function, applying L2-regularization to the weights of the last two layers with a penalty factor of 0.001.
The custom network we propose instead, is trained using the *Adam* (Adaptive Moment Estimation) optimization method which is an improvement of Stochastic Gradient Descent, using the default parameters from the *pytorch* framework. Both networks were trained for 50 epochs, without any early stopping policy, to also evaluate their stability during training.

D. Experiments

As a first experiment, we tried to implement and train the same neural network used in [2], called SB-CNN, which is a relatively shallow network (described in section A of the Model section). The network reached, at epoch 43, $75,27\%$ accuracy on the test set and $75,28\%$ F1-macro. In Figure 6 (in appendix A, page 8) we can see plots of the accuracy and F1-macro score over time for this model, a plot of the training loss and also a table containing the hyper-parameters used to train this model, and in Figure 27 in appendix B (page 19) represents the confusion matrix for the SB-CNN computed at the epoch with the best accuracy.
We then tried to implement a deeper network to see how the depth of the convolutional feature extractor influenced the performance of the classifier. The deeper

network reached, at epoch 46, a maximum accuracy of $71,33\%$ and an F1-macro score of $72,16\%$, both considerably (and surprisingly) lower than the SB-CNN. In Figure 7 we can see plots of the accuracy and F1-macro score over time for this model, a plot of the loss and also a table containing the hyper-parameters, and in Figure 28 (page 19) the confusion matrix. We can see a comparative plot of both models in Figure **??**.
Having established a baseline, we noticed a very important overfitting problem: by looking at the train/test comparison plots, we can observe that the orange plot, representing the train accuracy of the model, is in both cases way above the blue curve, representing the test accuracy, for most of the training. This can be interpreted as the inability of the model to generalize, meaning that the model is very good at predicting the very same samples it was trained on but is much less able to do so on a different set of samples, with a different distribution.

At this point we decided that the next step would be to try overcoming the overfitting problem. In order to do so, we followed the same path described in [2]: we first tried adding dropout regularization (that deactivates connections between neurons of two subsequent layers randomly) with a probability of 0.5, to the inputs of the two final fully connected layers of both networks. Then we performed another experiment adding L2-regularization with a weight decay factor of 0.001, as previously described, as also done in [2]. The results of these additions to our deeper model can be seen in the comparative plot in Figure 11. Figure 12 offers a comparison of the difference between training and test accuracy of our custom deeper model, without any regularization, with dropout and dropout plus L2-regularization respectively.

The overfitting problem is usually further overcome by using data augmentation techniques, meant to increase the variance of the data in the training set to prevent the model from overfitting on it: the basic idea is to add some randomness to the data that allows the model to increase its generalization capabilities and at the same time improves the data so that its most important features become more relevant to the classification problem.

The first class of augmentations we tried consists in the audio augmentations described in paragraph B of the Data preprocessing section. Figures 30, 31, 32 for example show a comparison of various diagrams representing an audio clip from the *drilling* class, before and after its preprocessing with the *Time stretching* audio augmentation. The same comparisons are shown for all the other augmentations in the other Figure in appendix C (respectively for Pitch shifting, Background noise addition and Dynamic Range Compression). In Figures 13 and 15 (pages 12 and the following one) we can see a comparison of the scores for all audio augmentations, for both models. In the same way, Figures 14 and 16 represent instead the difference in accuracy of the various trained models from the baseline model of reference (both for the SB-CNN and our deeper model). We can see that the best performing one on our custom model is the Background noise addition, while the best performing one in the SB-CNN model, which also yields the best model overall in the audio augmentations class, is the Dynamic Range Compression. Probably the DRC worked better on the SB-CNN, which is more shallow than our custom CNN, because it is actually the only audio augmentation that, among the others, is less oriented to generalization and more oriented to improving the features of the audio clip, by amplifying sounds that have a lower volume and quietening the louder ones. We also decided to try another class of augmentations by applying image augmentation techniques to the spectrograms generated from non-augmented clips, following an approach inspired by [10]. The three experiments consisted in first applying a gaussian noise to the spectrogram images (with a certain probability), then we tried shifting the spectrogram images to the right of a random amount and finally we tried both augmentations together. The results of this preprocessing can be found in appendix D (page 24). The comparative plots for these experiments are in Figure 17 (page 14) for the SB-CNN and in Figure 18 for our custom model, with also a table containing their scores and training epochs required to reach the max score. In particular, the best overall model is the SB-CNN trained using only the addition of gaussian noise on non-augmented clips, whose scores, including the training loss, are represented in Figure 10 (page 10) and whose confusion matrix at the best epoch is in Figure 29 (page 19).

Similarly to the paper [7], we also tried generating additional features from the spectrograms, by computing their *deltas* and *delta-deltas*, as described in the Data Preprocessing section. The experiment did not lead to better results, as can be seen from the bar plot at pages 15 and 16 respectively, representing the difference in accuracy and F1-macro between the baseline SB-CNN and the SB-CNN models with various audio augmentation that also use the delta and delta-delta features. Anyway, in most cases it sped-up convergence to the maximum score for most models. Figure 23 represents a comparison, for both accuracy and F1-macro score, of the SB-CNN model trained without augmentation but using delta and delta-delta computation, while figures at pages 15 and 16 show a comparison respectively for audio augmentations with deltas and delta-deltas.

It is also interesting to look at the effect that spectrogram deltas and delta-deltas have on the accuracy computed

per-class, by looking at the bar plots at page 18: we can notice how the delta improves the accuracy per-class more than the delta-delta does.
Finally, Figure 24 represents an overall comparison of the best models with the baseline, with a table reporting their best scores and required training epochs. In this plot we can see that we tried combining the best of each class of augmentations, by performing a training on the SB-CNN model with both the best method in the audio augmentations, that is the Dynamic Range Compression, and the best spectrogram augmentation, that is the addition of gaussian noise. Even though we expected to get the highest accuracy, the result wasn't the best one overall. This makes the SB-CNN, with the addition of gaussian noise on the spectrogram, the best achieving model in our experiments, with a peak accuracy of 77.66% and an F1-macro score of 77.88%, at epoch 41.

The provided implementation also features a method to represent computed gradients throughout training with an animated GIF image, where gradient magnitudes for each layer are represented as a bar plot, as in Figure 3 below, representing the gradient "flow" for the baseline SB-CNN model at epoch 4:

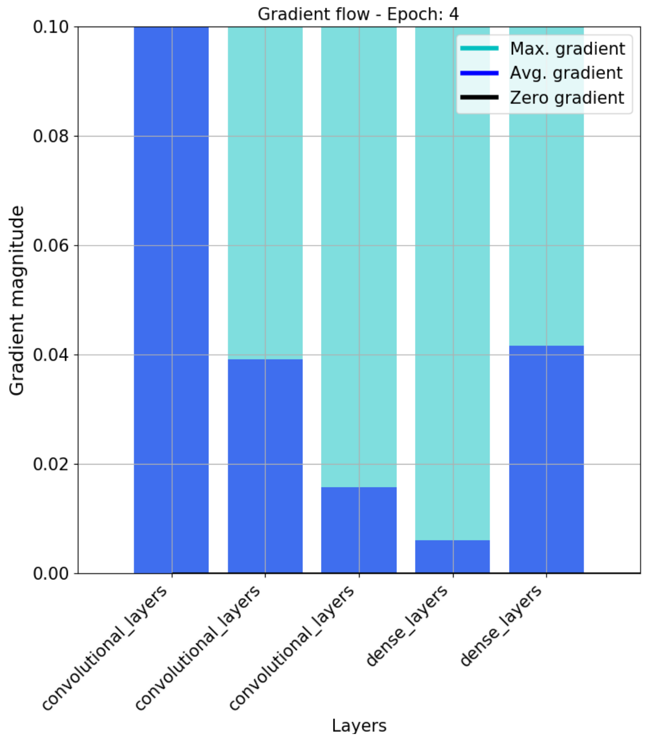


Fig. 3: Example of gradient "flow" bar plot

We also used the PyTorchVisualizations project [11] to generate saliency maps, in order to inspect the attention of the feature extractor throughout prediction, represented as a heatmap on the spectrogram images, that intuitively tell us "where" each layer of the network is "looking at" to determine its activations, constructed by collecting gradients from the back-propagation after having performed the forward-propagation with said spectrogram image. This results in a series of images like in Figure 4.

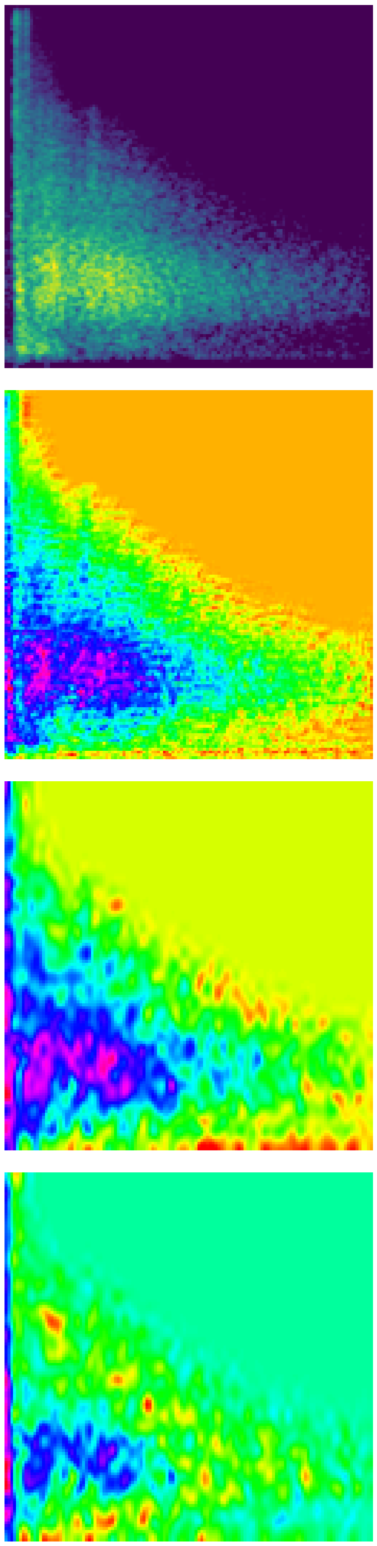


Fig. 4: The image above is the original spectrogram while the following images represent the saliency maps computed for each layer of SB-CNN during the propagation of the spectrogram in the neural network

References

[1] J. Salamon, C. Jacoby, and J. P. Bello, “A dataset and taxonomy for urban sound research,” in *22nd ACM International Conference on Multimedia (ACM-MM’14)*, Orlando, FL, USA, Nov. 2014, pp. 1041–1044.

[2] J. Salamon and J. P. Bello, “Deep convolutional neural networks and data augmentation for environmental sound classification,” *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017.

[3] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R’ıo, M. Wiebe, P. Peterson, P. G’erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[4] McFee, Brian, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, “librosa: Audio and music signal analysis in python.” pp. 18–25, 2015.

[5] B. McFee, J. Salamon, and J. P. Bello, “A software framework for musical data augmentation.” *16th International Society for Music Information Retrival conference (ISMIR)*, 2015.

[6] Mel scale. [Online]. Available: https://www.sfu.ca/sonic-studio-webdav/handbook/Mel.html

[7] K. J. Piczak, “Environmental sound classification with convolutional neural networks,” pp. 1–6, 2015.

[8] T. Bäckström. Deltas and delta-deltas. [Online]. Available: https://wiki.aalto.fi/display/ITSP/Recognition+tasks+in+speech+processing

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[10] T. Bäckström. Github repository that inspired the spectrogram image augmentation techniques. [Online]. Available: https://github.com/mariostrbac/environmental-sound-classification

[11] U. Ozbulak, “Pytorch cnn visualizations,” https://github.com/utkuozbulak/pytorch-cnn-visualizations, 2019.

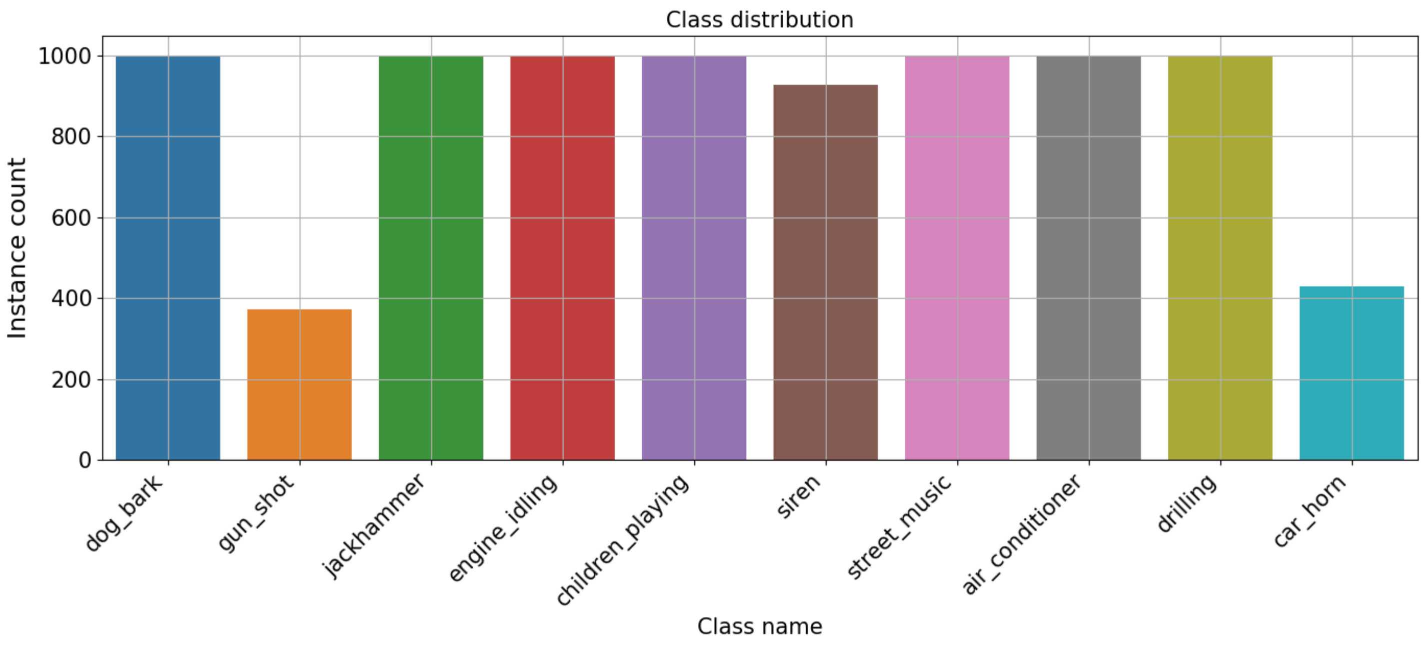


Fig. 5: Distribution of the dataset samples in classes. Notice how the dataset is mainly balanced, except for the two classes gun_shot and *car_horn*, which are less represented

APPENDIX A
TABLES AND PLOTS

Hyperparameter	**Value**
Batch size	128
Batch shuffling	True
Dropout prob.	0.5
L2 reg. weight. decay	0.001
Optimization method	SGD
Learning rate	0.01
Training epochs	43

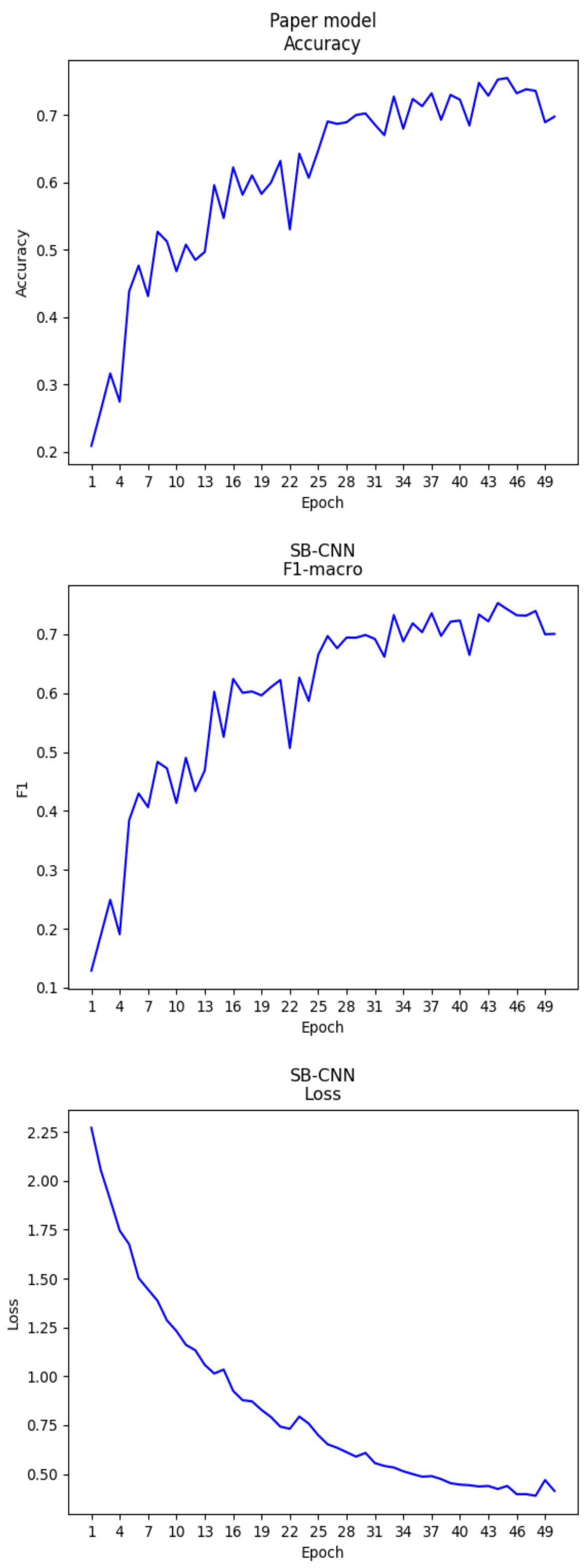
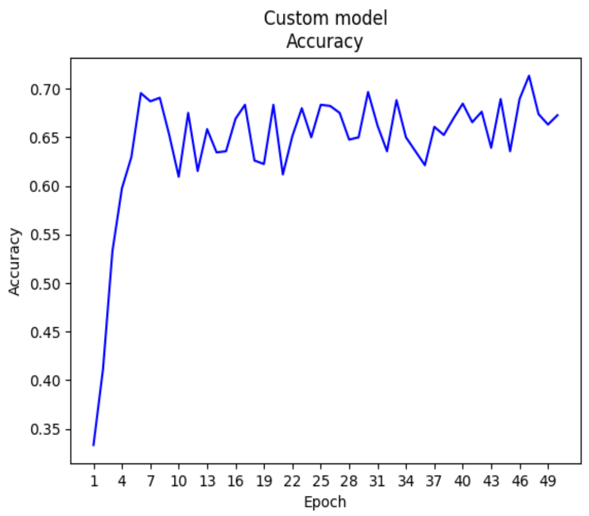
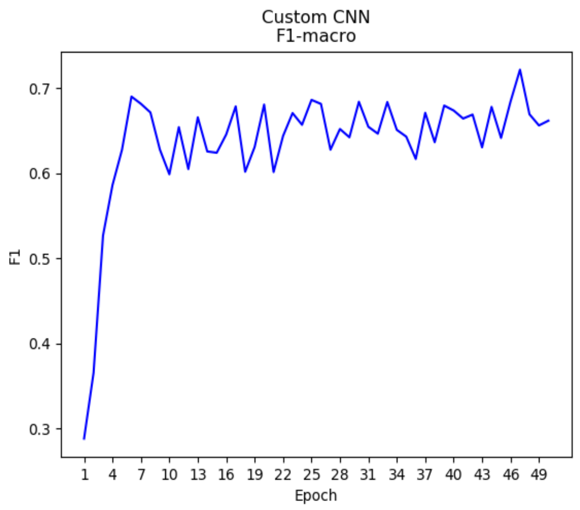


Fig. 6: The table on the left represents the hyper-parameters for the training of the SB-CNN model (no augmentations) while the plots on the right represent (top to bottom) the accuracy, the F1-macro and the loss plot respectively.



Hyperparameter	Value
Batch size	128
Batch shuffling	True
Dropout prob.	0.5
Optimization method	Adam
Training epochs	25



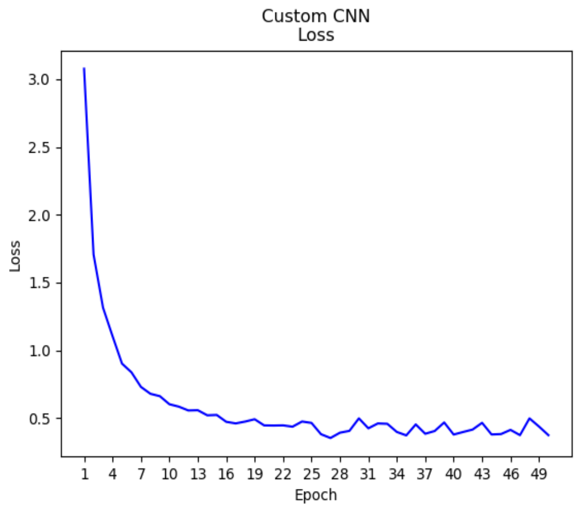


Fig. 7: The table on the left represents the hyper-parameters for the training of the Custom CNN model (no augmentations) while the plots on the right represent (top to bottom) the accuracy, the F1-macro and the loss plot respectively.

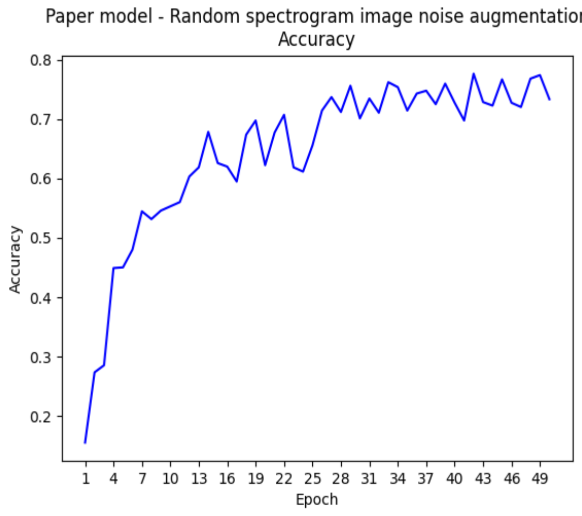


Fig. 8: figure caption

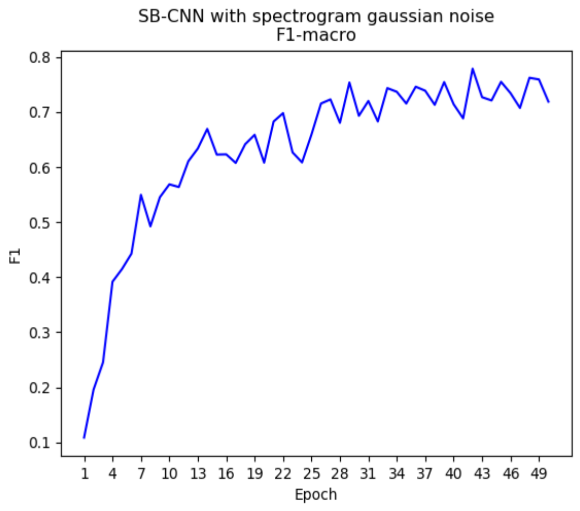


Fig. 9: figure caption

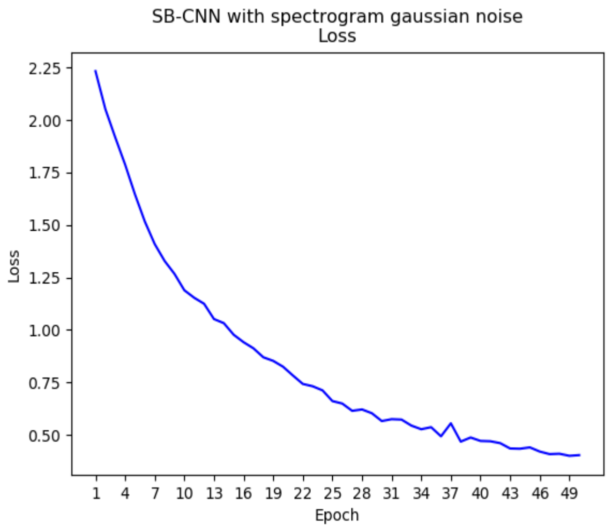
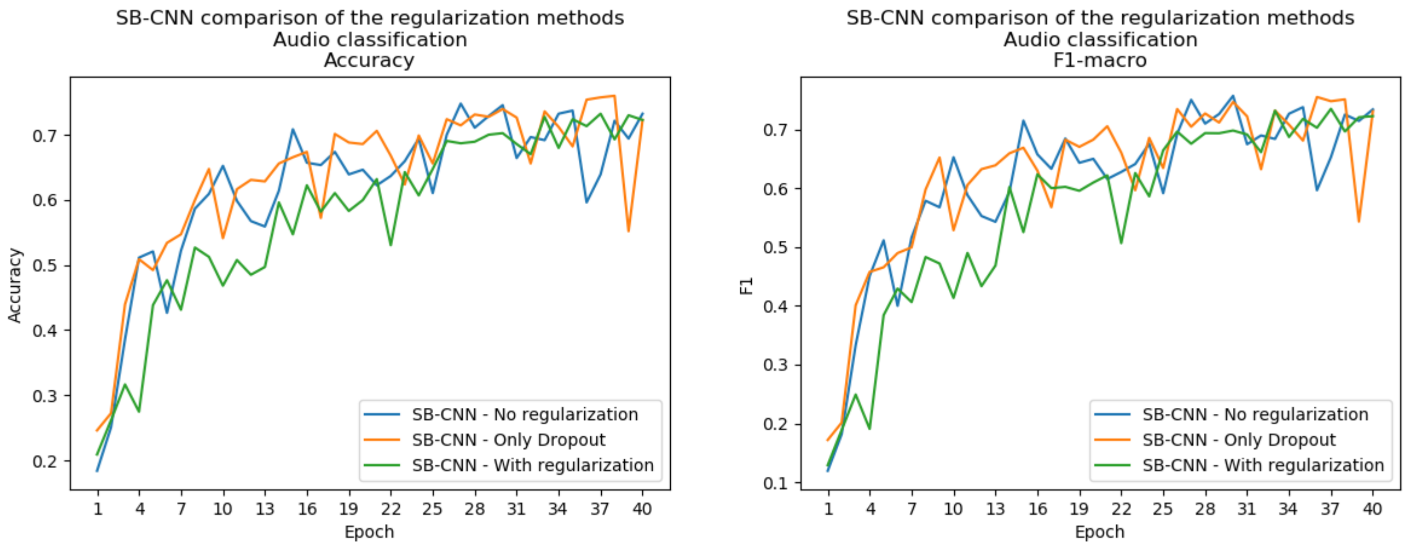


Fig. 10: Accuracy, F1-macro and Loss plot for the SB-CNN model trained with spectrogram augmentation through the application of gaussian noise. This model proved to be the best performing one.



Model	Accuracy	F1-macro	Epochs
SB-CNN - No regularization	74.55%	75.77%	29
SB-CNN - Only Dropout	76.46%	75.83%	47
SB-CNN - With regularization	75.27%	75.28%	43

Fig. 11: Comparison of the accuracy (left) and F1-macro (right) plots for the SB-CNN with increasing levels of regularization

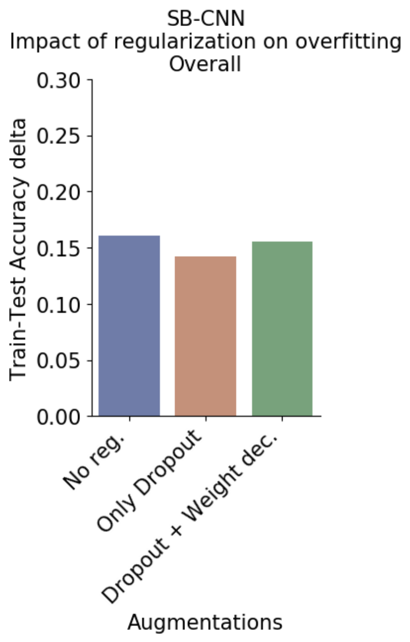
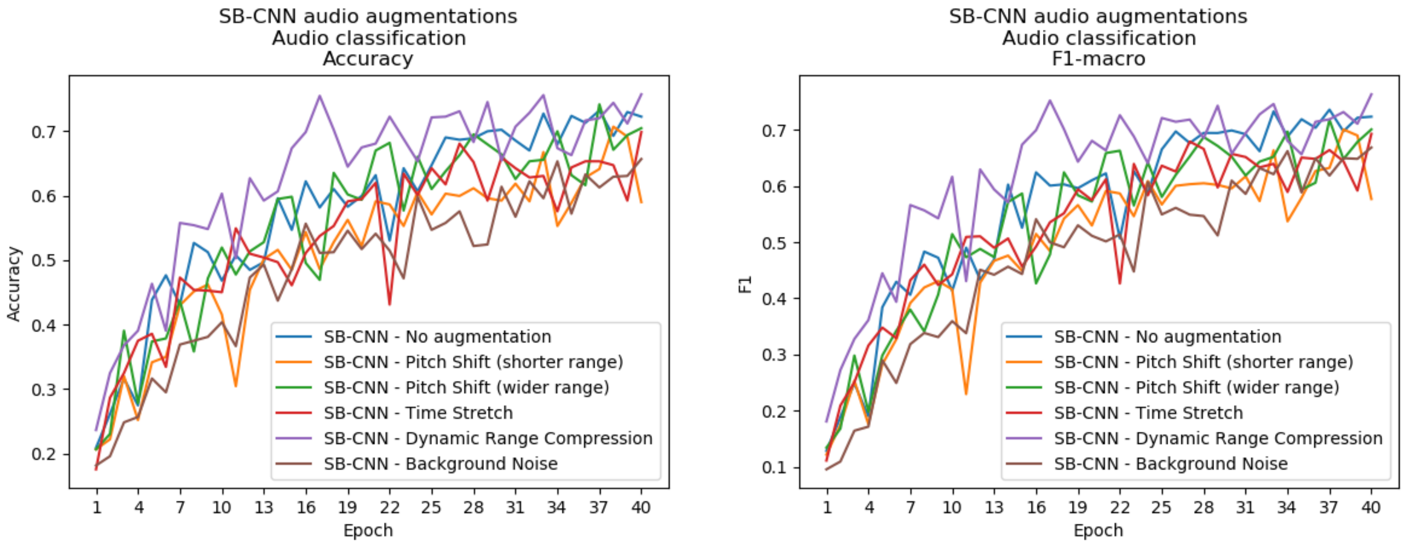


Fig. 12: Difference between train and test accuracy at the best epoch for each level of regularization (without regularization, only dropout, weight decay). As we can see the most effective regularization is dropout: adding weight decay is not as effective



Model	Accuracy	F1-macro	Epochs
SB-CNN - No augmentation	75.27%	75.28%	43
SB-CNN - Pitch Shift (shorter range)	70.73%	70.56%	49
SB-CNN - Pitch Shift (wider range)	75.15%	72.68%	47
SB-CNN - Time Stretch	69.89%	69.25%	39
SB-CNN - Dynamic Range Compression	76.70%	76.87%	47
SB-CNN - Background Noise	66.67%	67.82%	48

Fig. 13: Comparison of the accuracy (left) and F1-macro (right) plots for the SB-CNN with all types of audio augmentation

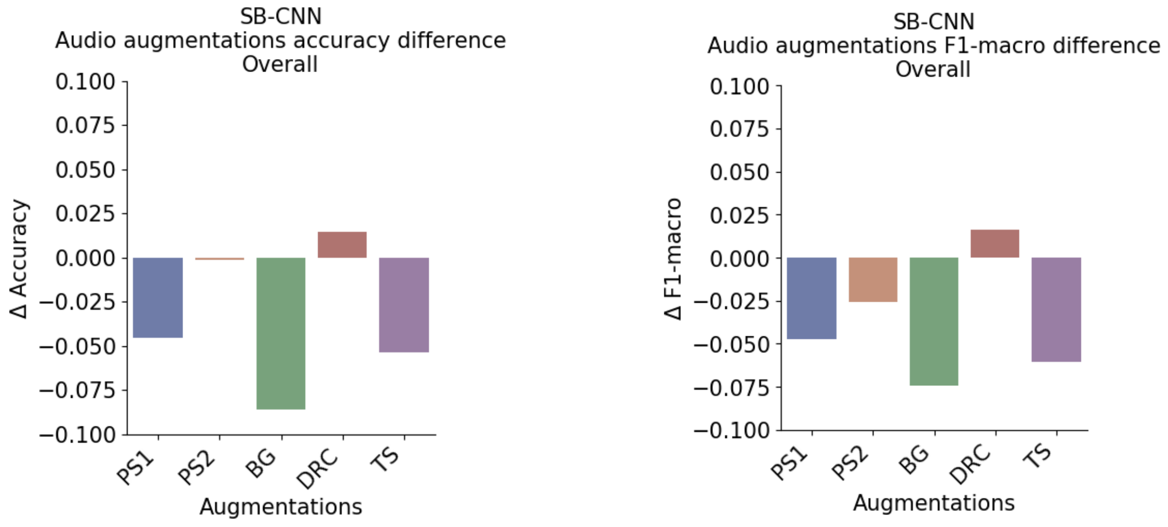
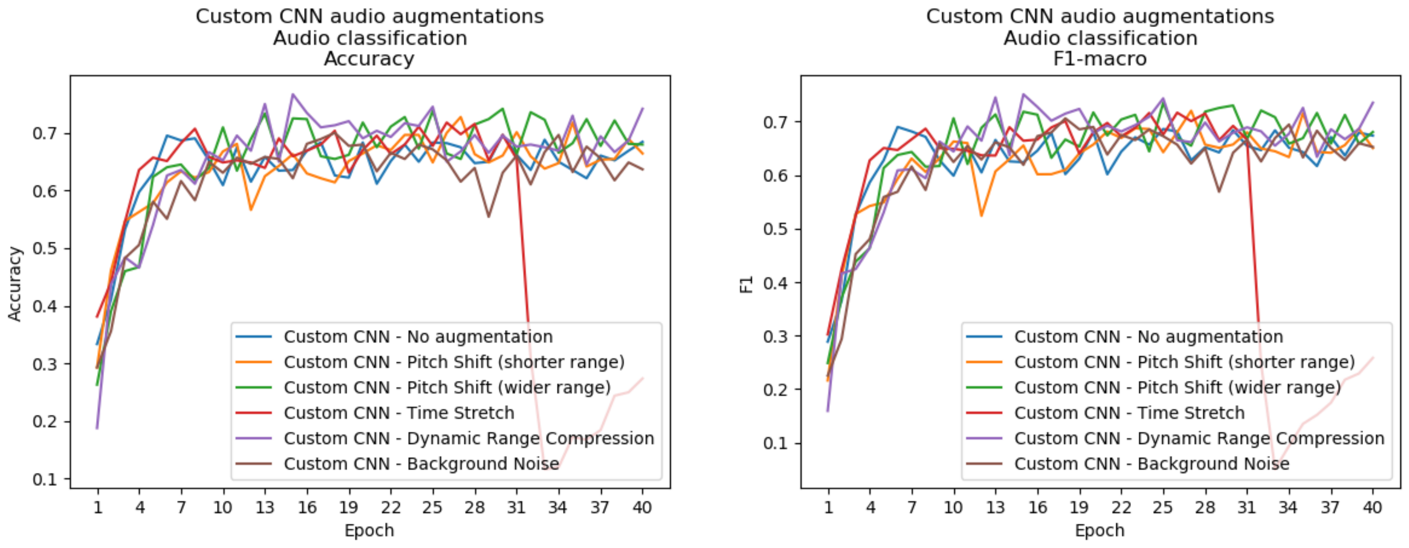


Fig. 14: Difference between the SB-CNN baseline max. score and the max. scores achieved by the same model using different techniques of data augmentation on the audio clip. Accuracy on the left and F1-macro on the right



Model	**Accuracy**	**F1-macro**	**Epochs**
Custom CNN - No augmentation	71.33%	72.16%	46
Custom CNN - Pitch Shift (shorter range)	72.76%	72.03%	26
Custom CNN - Pitch Shift (wider range)	73.84%	73.50%	24
Custom CNN - Time Stretch	71.80%	71.67%	25
Custom CNN - Dynamic Range Compression	76.70%	75.10%	14
Custom CNN - Background Noise	70.01%	70.60%	17

Fig. 15: Comparison of the accuracy (left) and F1-macro (right) plots for the Custom-CNN with all types of audio augmentation

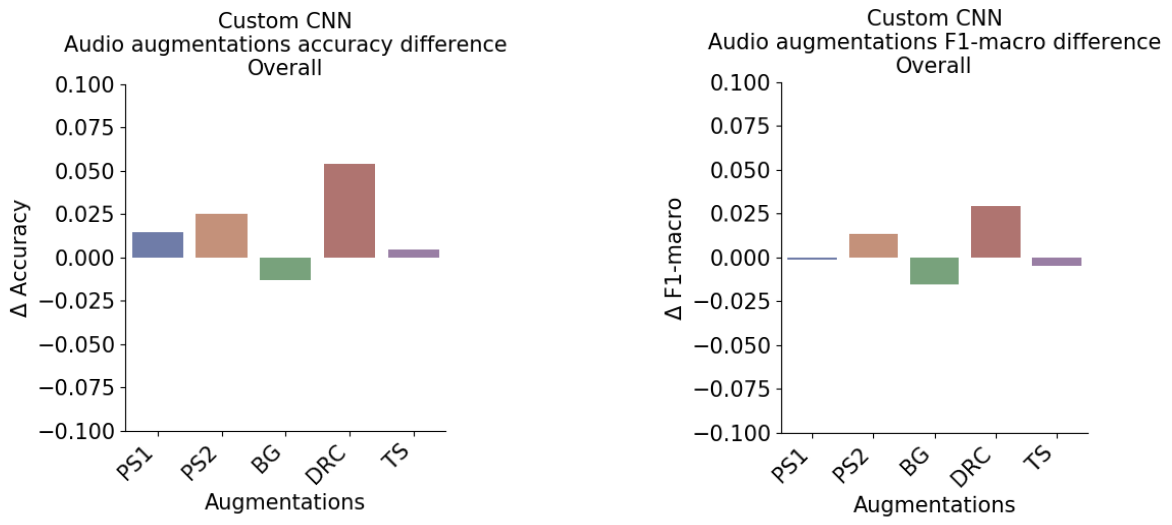
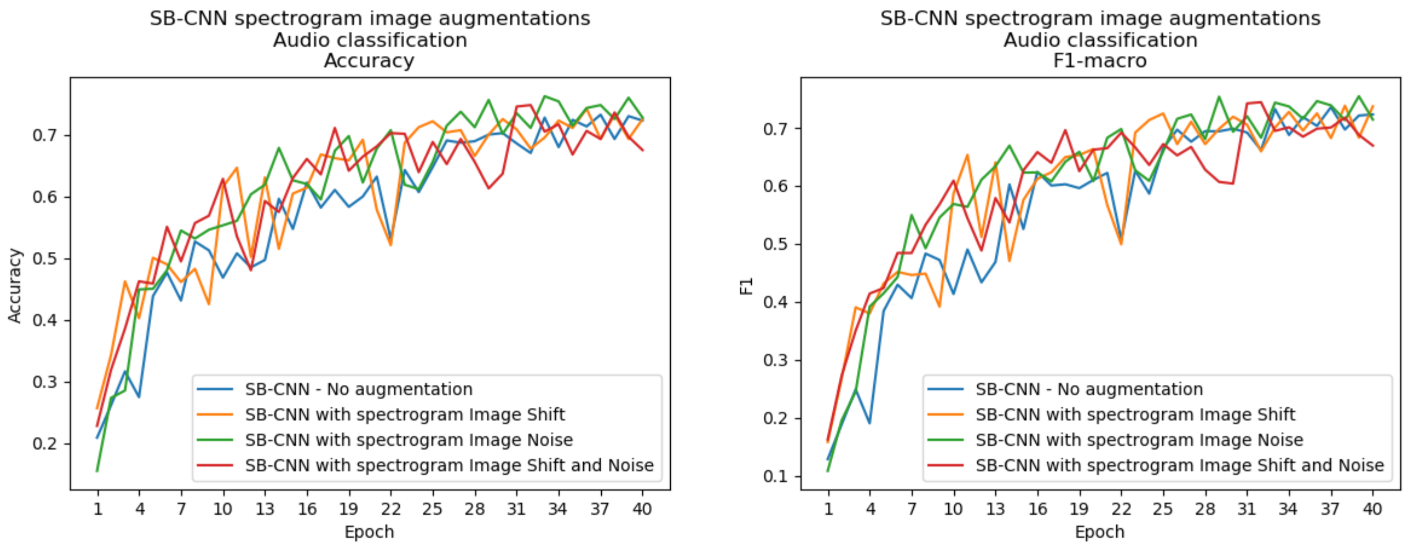
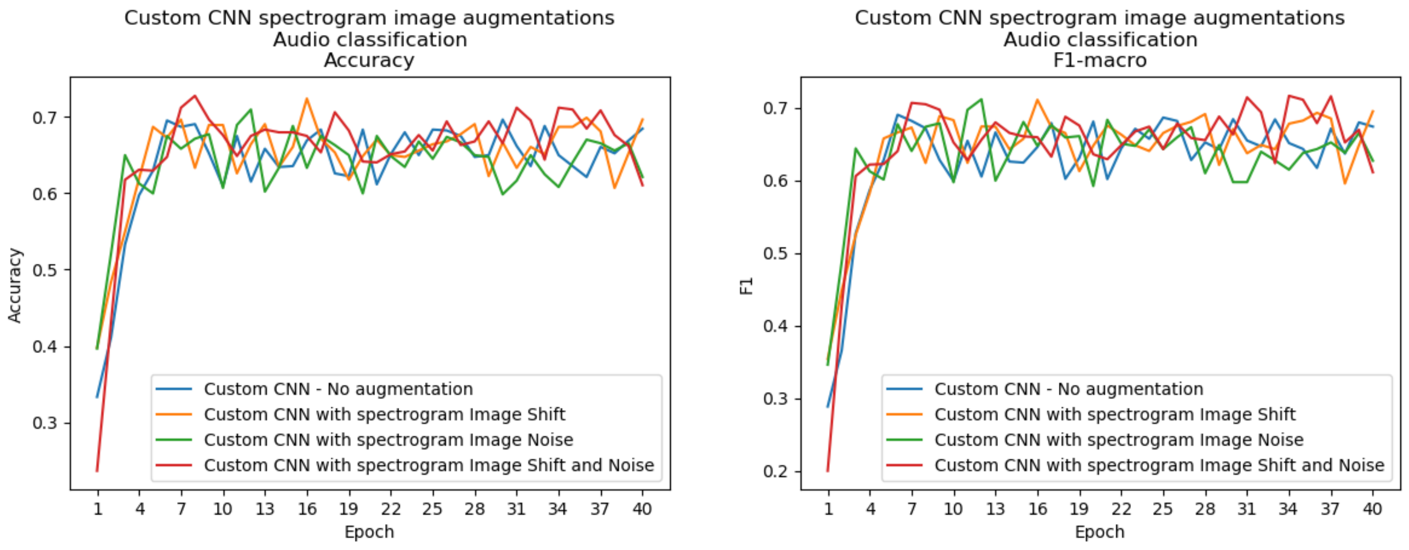


Fig. 16: Difference between the Custom-CNN baseline max. score and the max. scores achieved by the same model using different techniques of data augmentation on the audio clip. Accuracy on the left and F1-macro on the right. As we can see, audio augmentations are much more effective in improving model performance with a deeper model



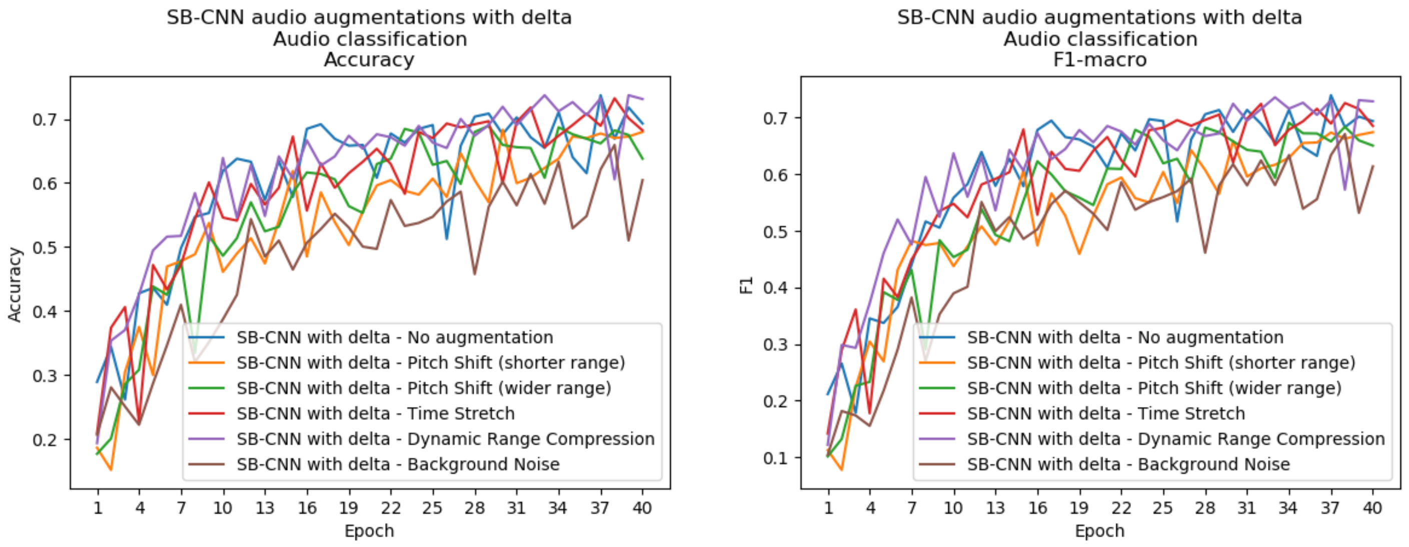
Model	**Accuracy**	**F1-macro**	**Epochs**
SB-CNN - No augmentation	75.27%	75.28%	43
SB-CNN with spectrogram Image Shift	73.24%	73.81%	37
SB-CNN with spectrogram Image Noise	77.66%	77.88%	41
SB-CNN with spectrogram Image Shift and Noise	75.51%	75.55%	43

Fig. 17: Comparison of the accuracy (left) and F1-macro (right) plots for the SB-CNN with spectrogram augmentations



Model	**Accuracy**	**F1-macro**	**Epochs**
Custom CNN - No augmentation	71.33%	72.16%	46
Custom CNN with spectrogram Image Shift	71.56%	71.21%	46
Custom CNN with spectrogram Image Noise	70.97%	71.13%	11
Custom CNN with spectrogram Image Shift and Noise	71.21%	71.60%	33

Fig. 18: Comparison of the accuracy (left) and F1-macro (right) plots for the Custom-CNN with spectrogram augmentations



Model	Accuracy	F1-macro	Epochs
SB-CNN with delta - No augmentation	73.71%	73.95%	36
SB-CNN with delta - Pitch Shift (shorter range)	72.16%	71.59%	46
SB-CNN with delta - Pitch Shift (wider range)	71.80%	71.10%	46
SB-CNN with delta - Time Stretch	73.24%	72.57%	37
SB-CNN with delta - Dynamic Range Compression	74.55%	74.67%	41
SB-CNN with delta - Background Noise	65.95%	67.10%	37

Fig. 19: Comparison of the accuracy (left) and F1-macro (right) plots for the SB-CNN with all types of audio augmentation, using spectrogram deltas

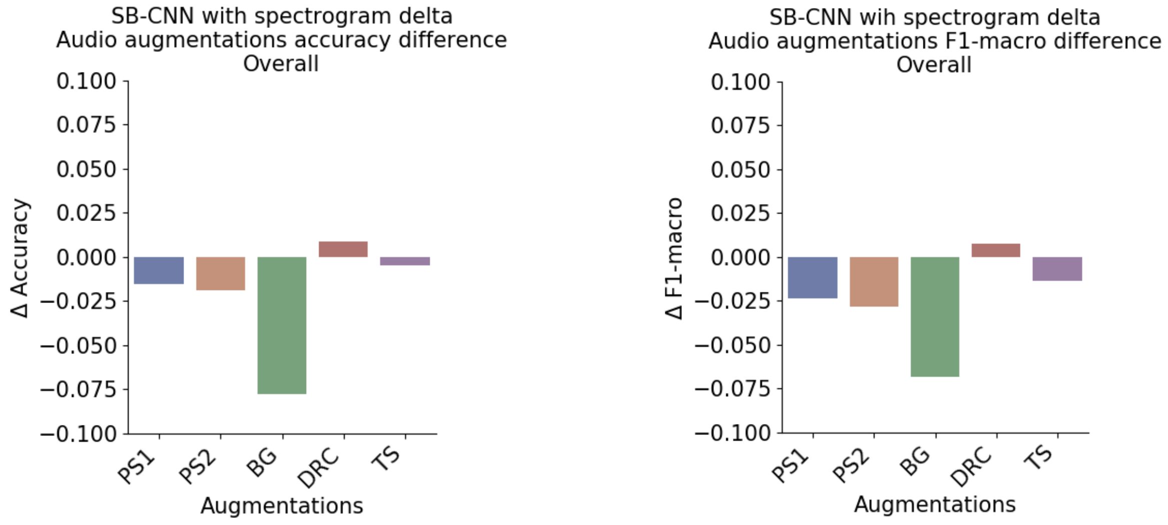
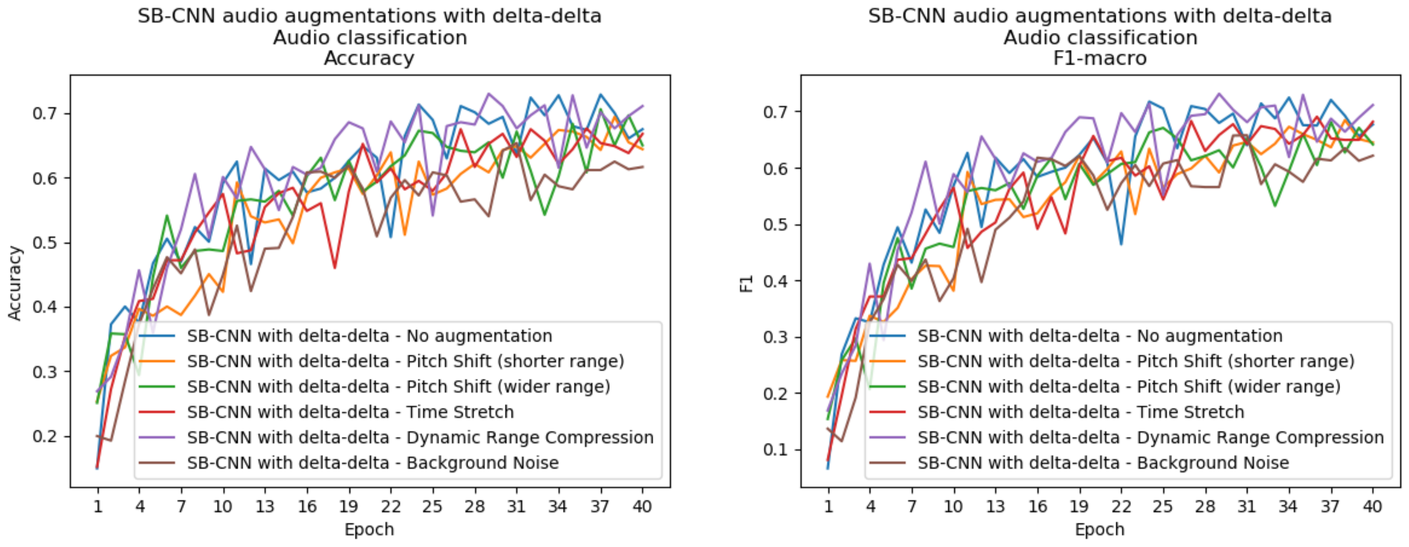


Fig. 20: Difference between the SB-CNN baseline max. score and the max. scores achieved by the same model using different techniques of data augmentation on the audio clip, with the addition of spectrogram delta features. Accuracy on the left and F1-macro on the right.



Model	Accuracy	F1-macro	Epochs
SB-CNN with delta-delta - No augmentation	74.43%	73.55%	43
SB-CNN with delta-delta - Pitch Shift (shorter range)	73.71%	72.69%	46
SB-CNN with delta-delta - Pitch Shift (wider range)	71.80%	70.42%	49
SB-CNN with delta-delta - Time Stretch	70.01%	70.94%	46
SB-CNN with delta-delta - Dynamic Range Compression	73.0%	73.28%	40
SB-CNN with delta-delta - Background Noise	65.71%	66.55%	47

Fig. 21: Comparison of the accuracy (left) and F1-macro (right) plots for the SB-CNN with all types of audio augmentation, using spectrogram delta-deltas

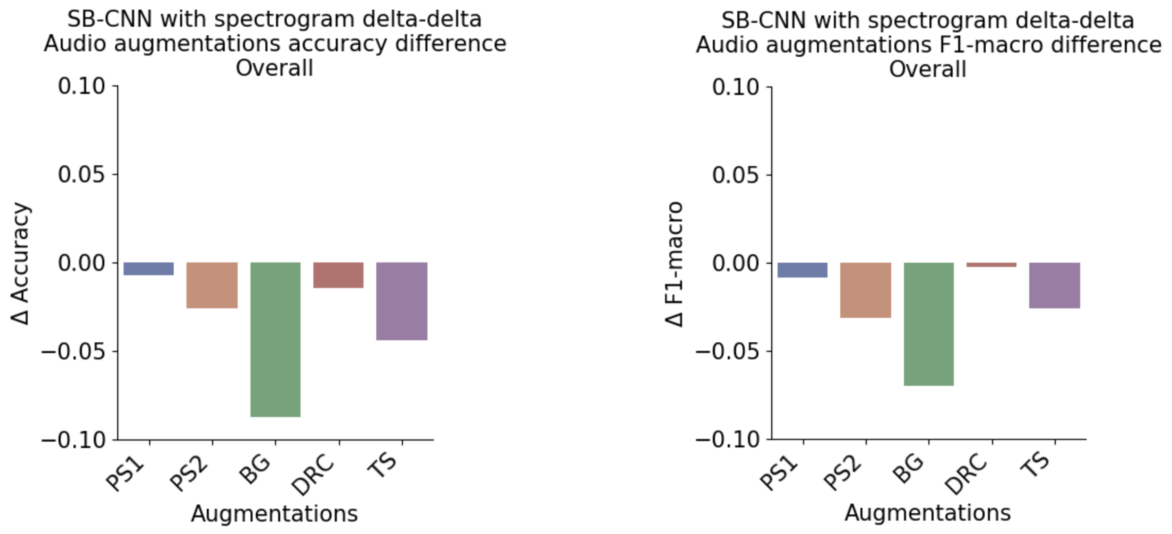
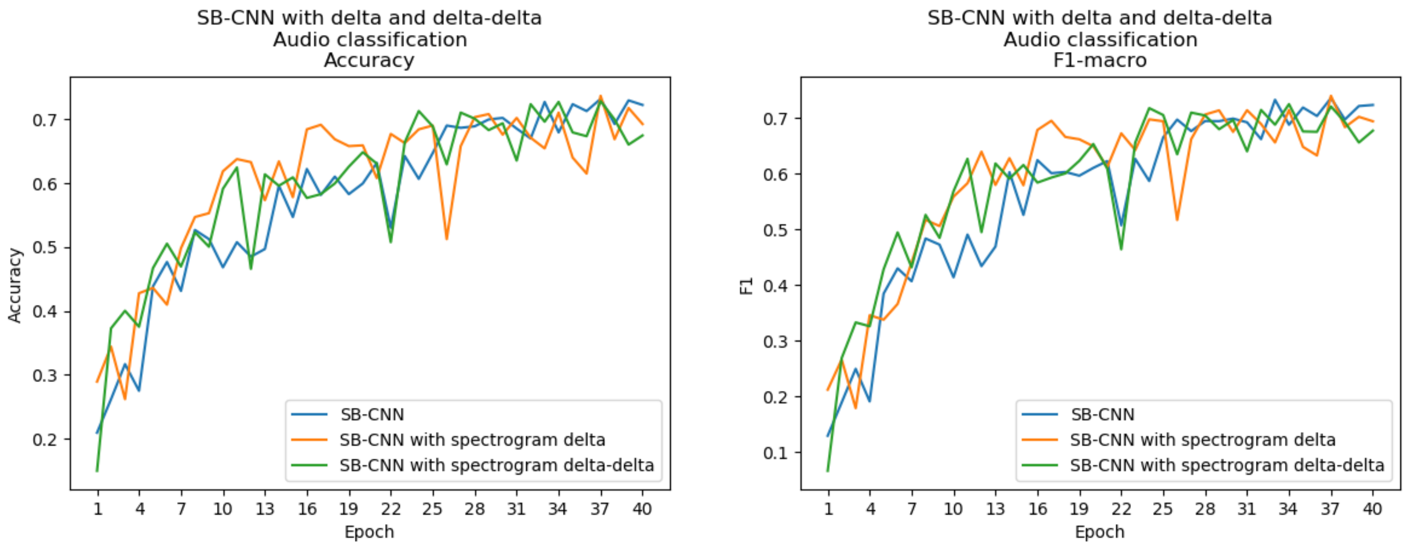
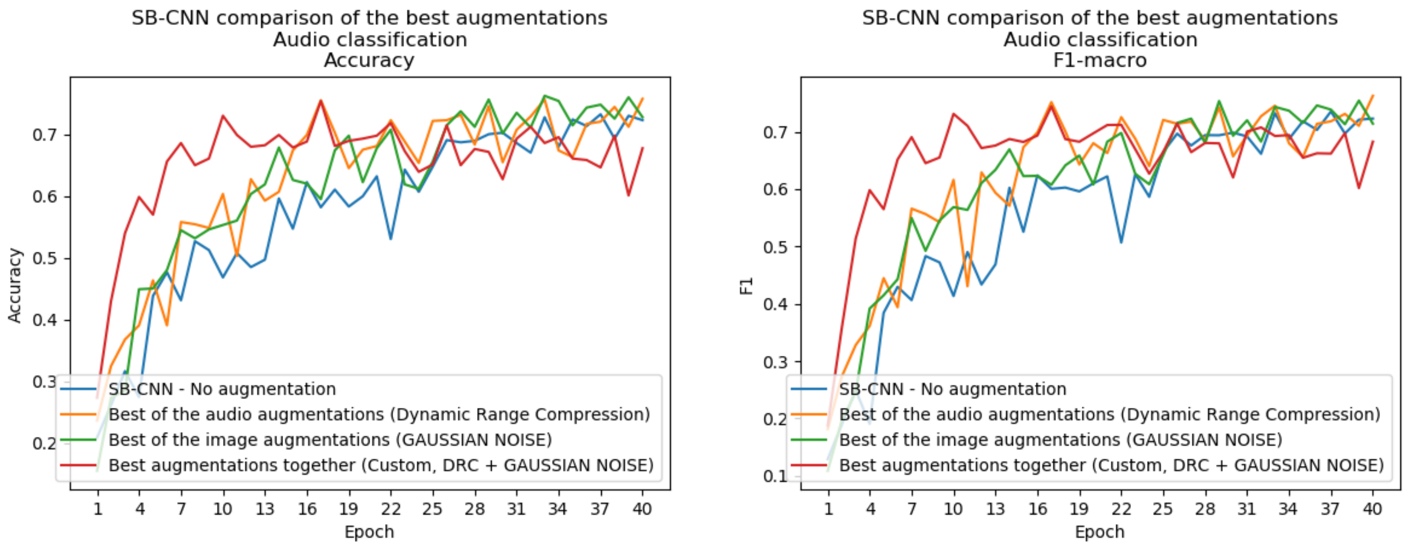


Fig. 22: Difference between the SB-CNN baseline max. score and the max. scores achieved by the same model using different techniques of data augmentation on the audio clip, with the addition of spectrogram delta-delta features. Accuracy on the left and F1-macro on the right.



Model	Accuracy	F1-macro	Epochs
SB-CNN - No augmentation	75.27%	75.28%	43
SB-CNN with spectrogram delta	73.71%	73.95%	36
SB-CNN with spectrogram delta delta	74.43%	73.55%	43

Fig. 23: Here we can see a comparison of the accuracy (left) and f1-macro (right) of the base model against the models that use spectrogram deltas and delta-deltas. We notice a decrease in training time required to reach the best epoch but also a decrease in accuracy. Probably a deeper model would have benefited more by this additional features



Model	Accuracy	F1-macro	Epochs
SB-CNN - No augmentation	75.27%	75.28%	43
Best of the audio augmentations (Dynamic Range Compression)	76.70%	76.87%	47
Best of the image augmentations (GAUSSIAN NOISE)	77.66%	77.87%	41
Best of the image augmentations (Custom DRC + GAUSSIAN NOISE)	75.39%	74.42%	16

Fig. 24: Comparison of the accuracy (left) and F1-macro (right) plots of the best models overall in our experimentation process. Notice our last experiment that combined the best audio augmentation and the best spectrogram augmentation, which is represented here in red: it has the steepest learning curve but fails to reach the max score, which instead is reached by performing gaussian noise addition alone

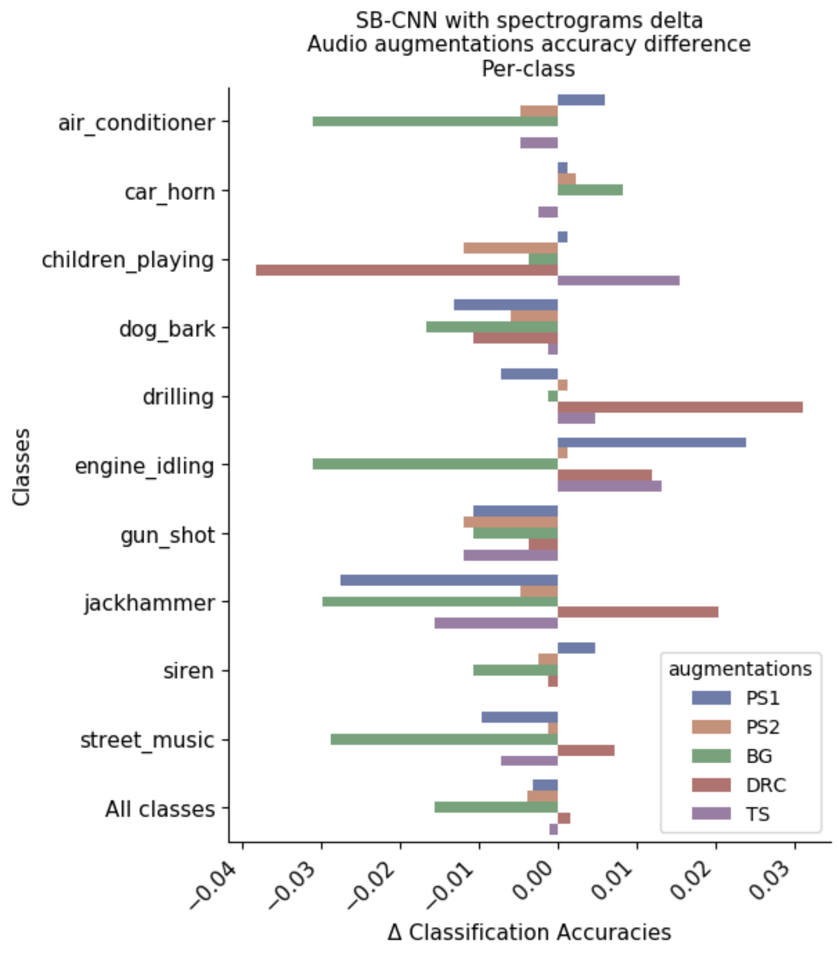


Fig. 25: Difference of the per-class accuracy computed on the SB-CNN model using spectrogram deltas, for each audio augmentation, with respect to the baseline SB-CNN using spectrogram deltas as well

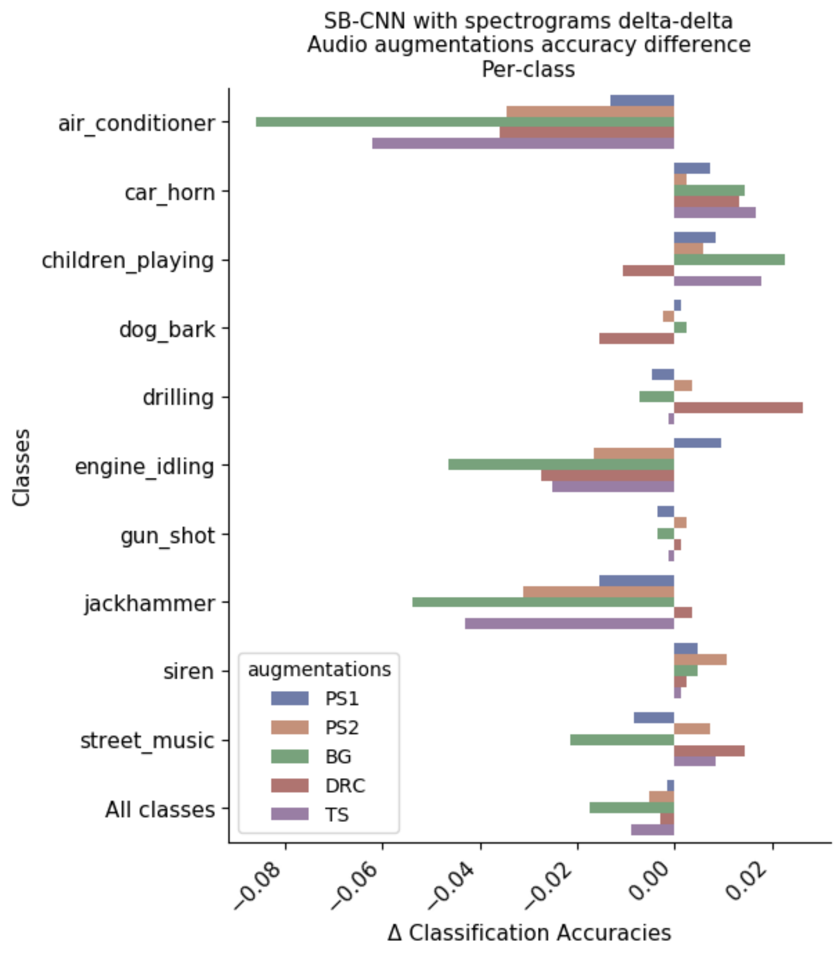


Fig. 26: Difference of the per-class accuracy computed on the SB-CNN model using spectrogram delta-deltas, for each audio augmentation, with respect to the baseline SB-CNN using spectrogram delta-deltas as well

APPENDIX B
CONFUSION MATRICES

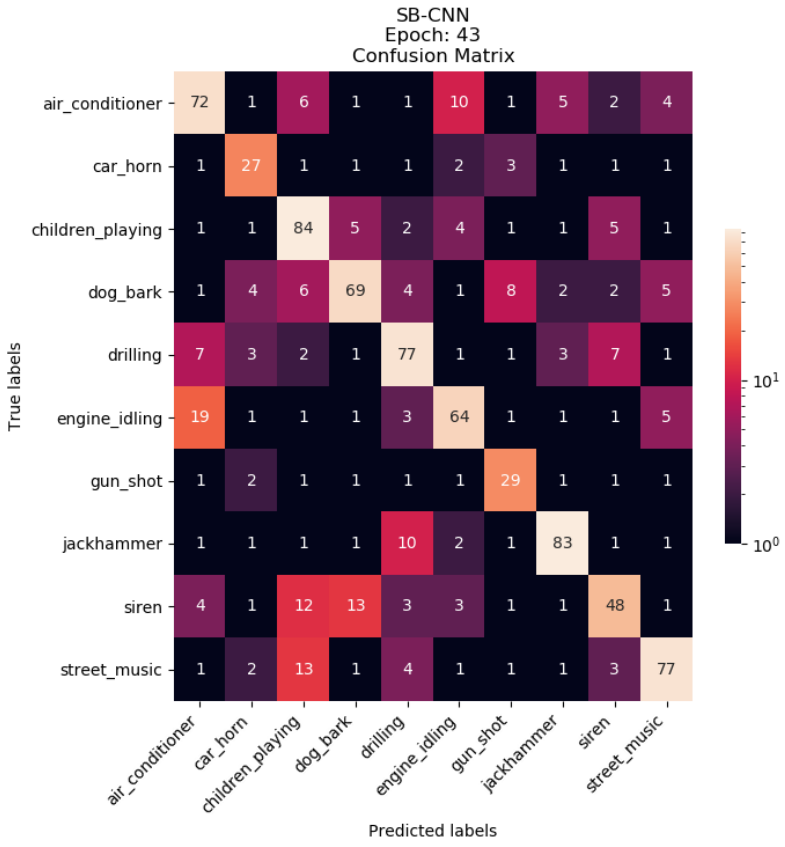


Fig. 27: SB-CNN confusion matrix (epoch 43)

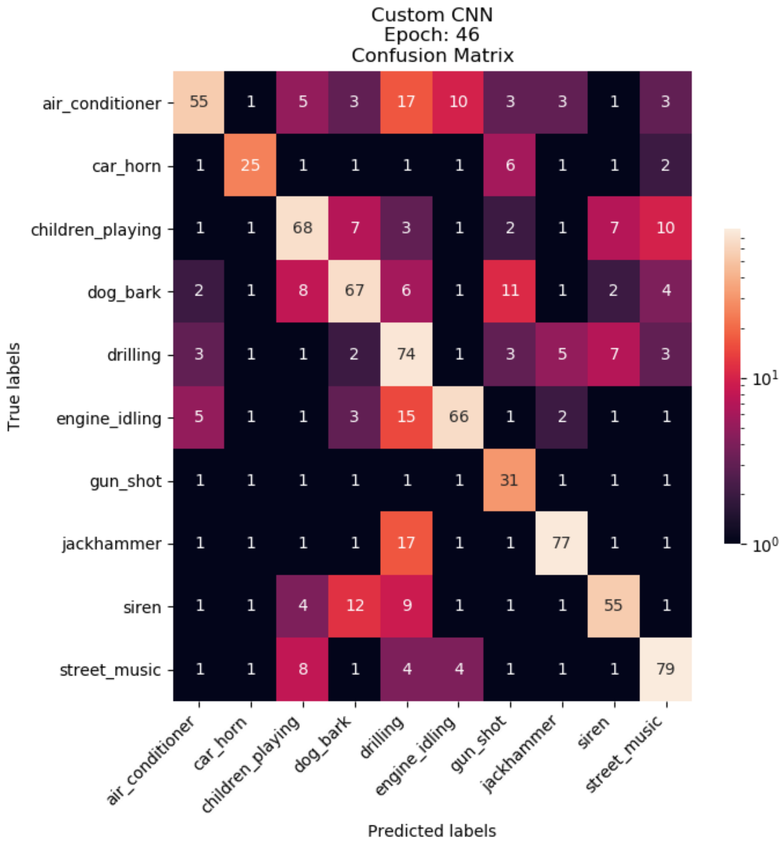


Fig. 28: Custom CNN confusion matrix (epoch 43)

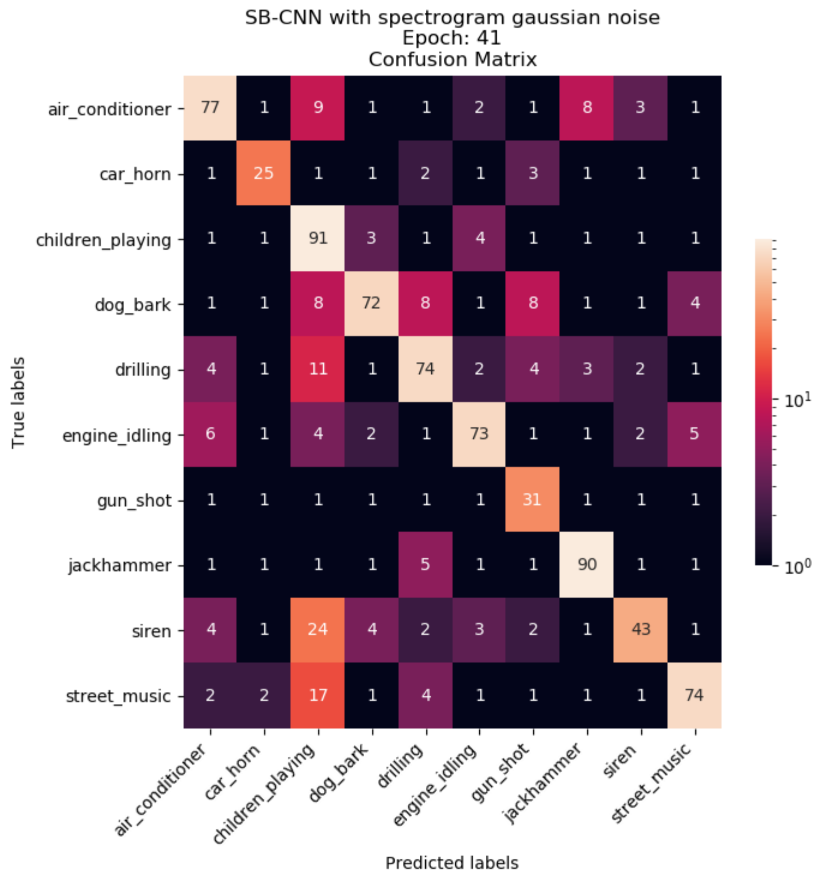


Fig. 29: Confusion matrix for the SB-CNN trained with spectrogram augmentation with gaussian noise (epoch 41) which proved to be the best model

APPENDIX C
AUDIO PREPROCESSING

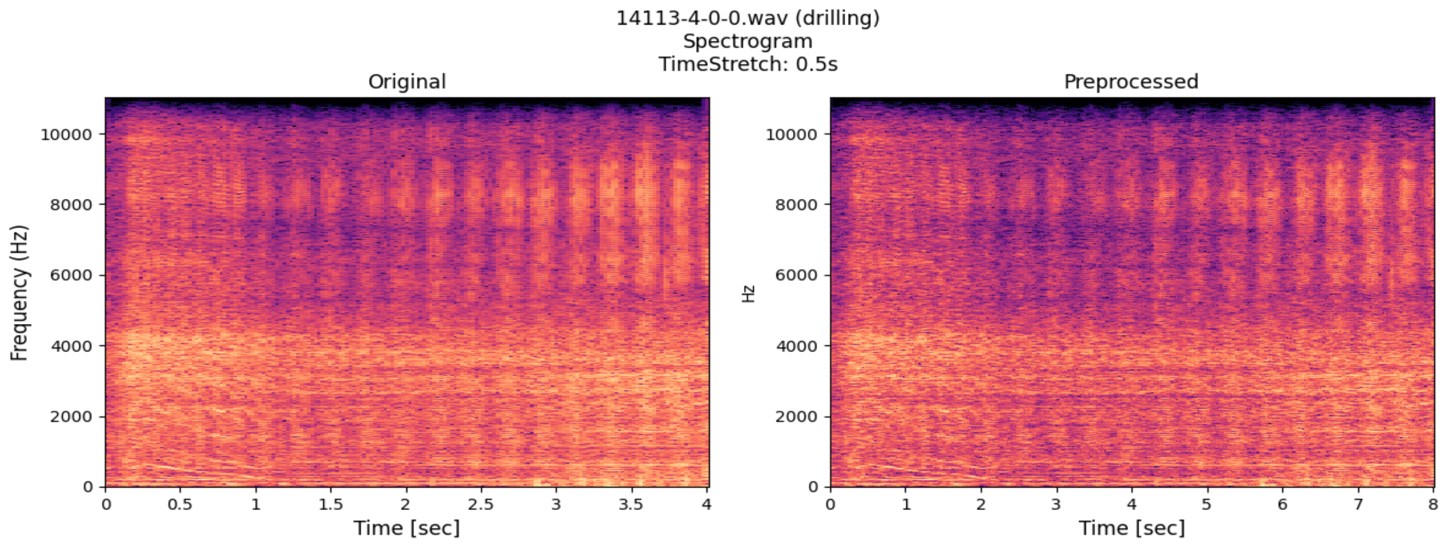


Fig. 30: Time Stretching spectrogram on class drilling

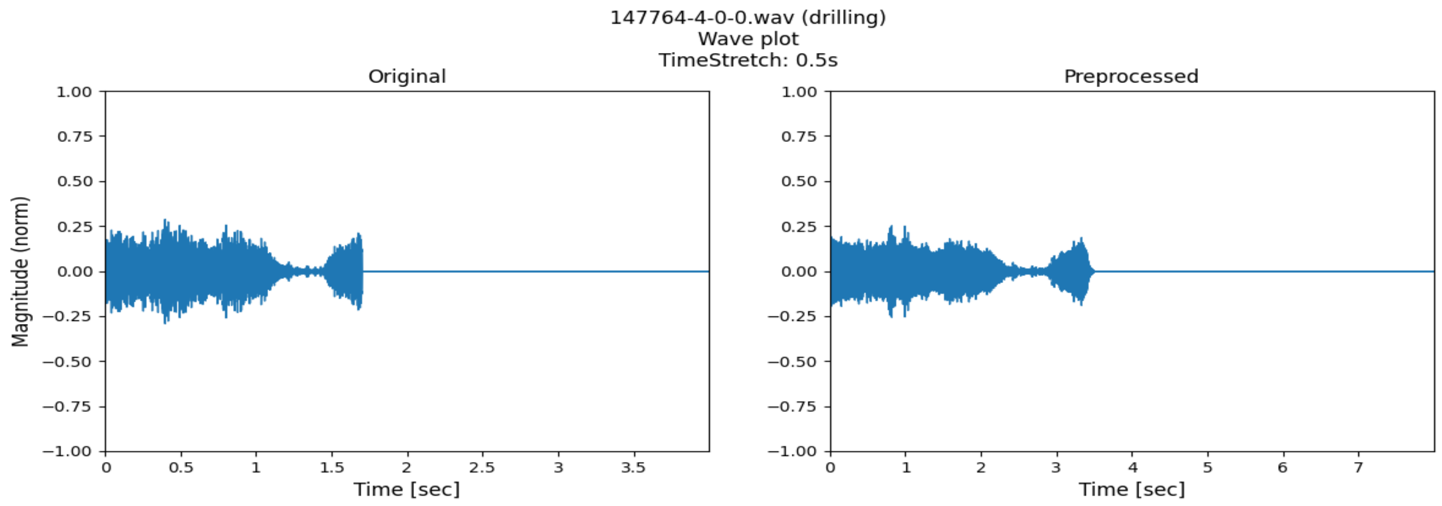


Fig. 31: Time Stretching waveplot on class drilling

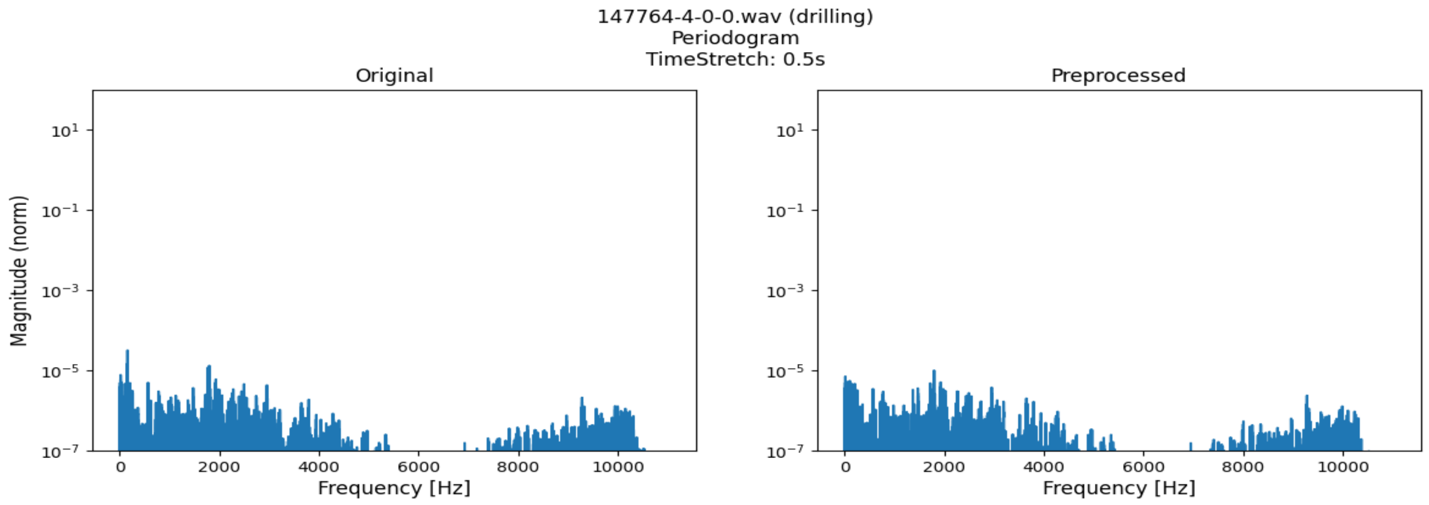


Fig. 32: Time Stretching period gram on class drilling

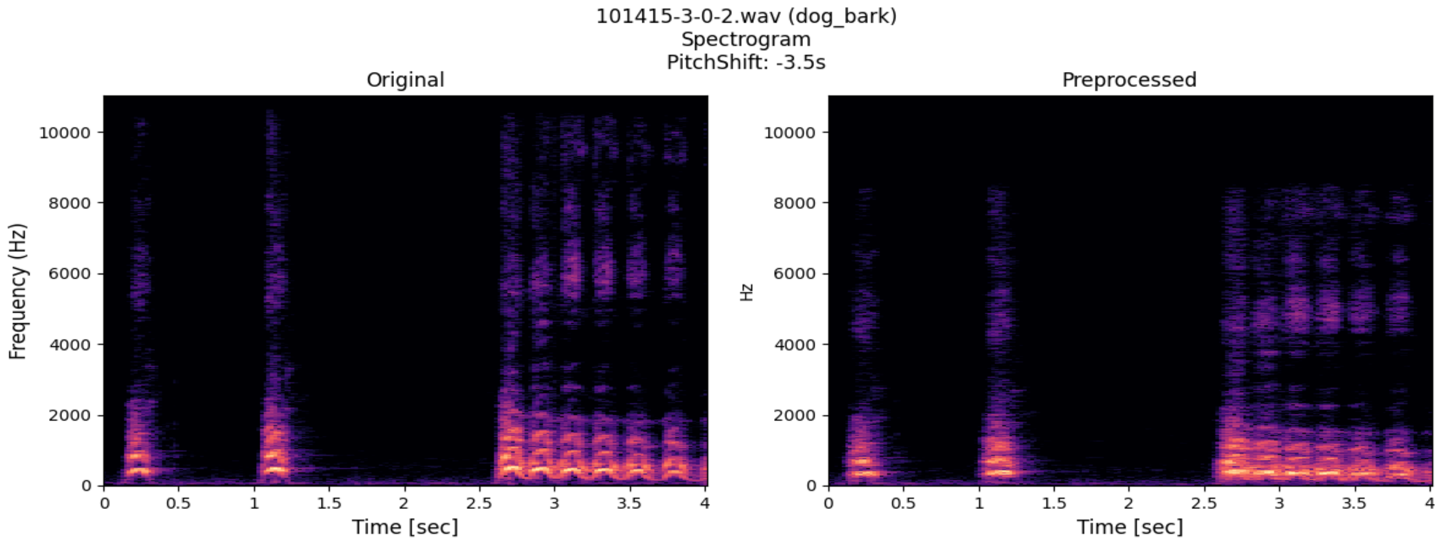


Fig. 33: Pitch Shift on class dog_bark

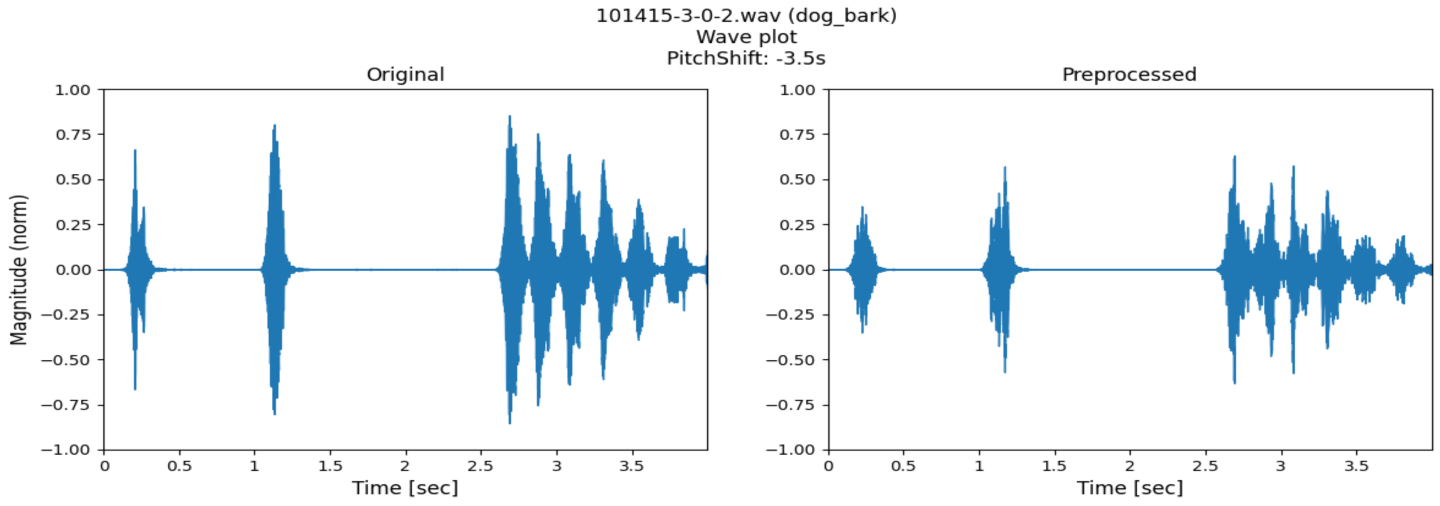


Fig. 34: Pitch Shift waveplot on class dog_bark

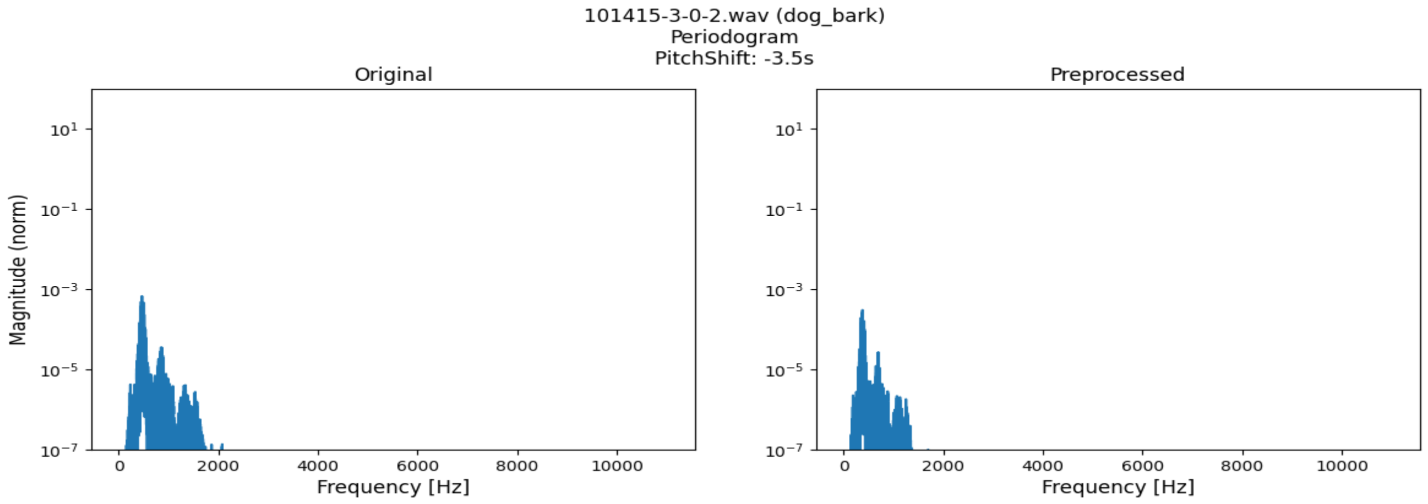


Fig. 35: Pitch Shift periodogram on class dog_bark

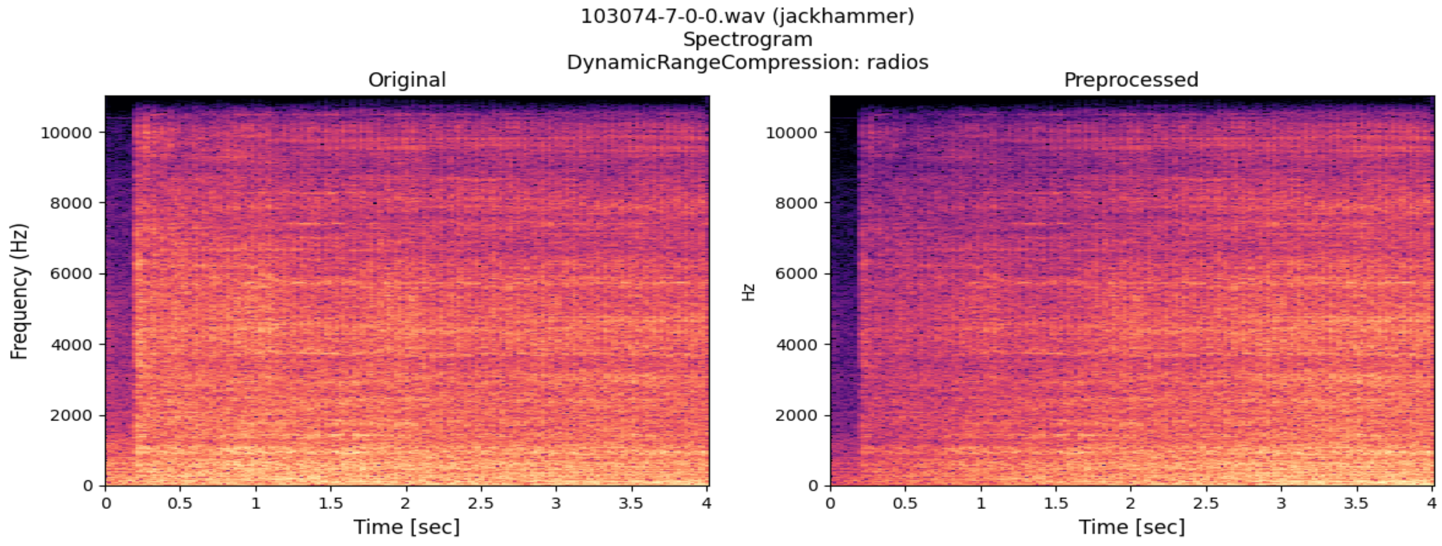


Fig. 36: Dynamic Range Compression using preset ”radio” on one instance of class drilling.This technique is an audio signal processing operation that reduces the volume of loud sounds or amplifies quiet sounds. In this particular case on the right we can see that quiet sounds have been amplified (blue areas depict a lower amplitude)

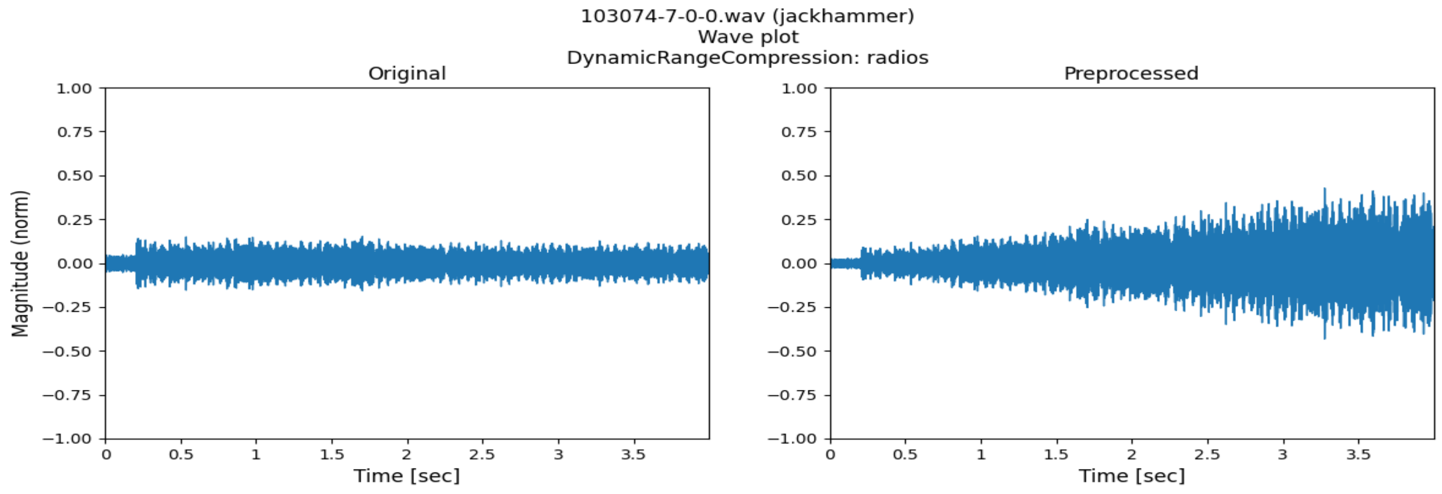


Fig. 37: Dynamic Range Compression wave plot on class jackhammer

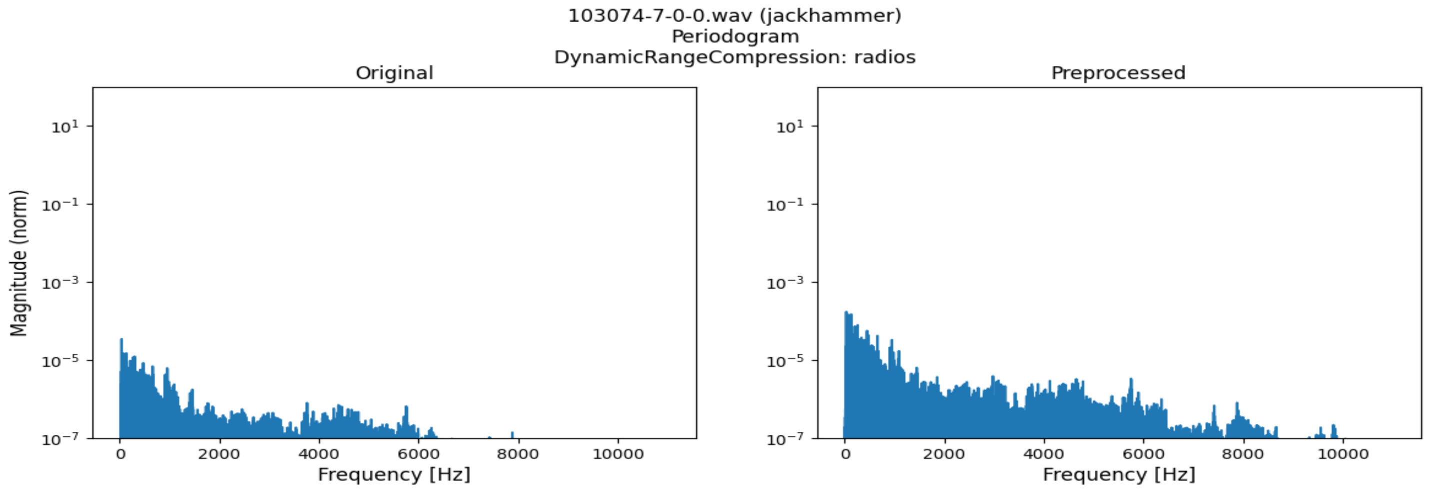


Fig. 38: Dynamic Range Compression periodogram on class jackhammer

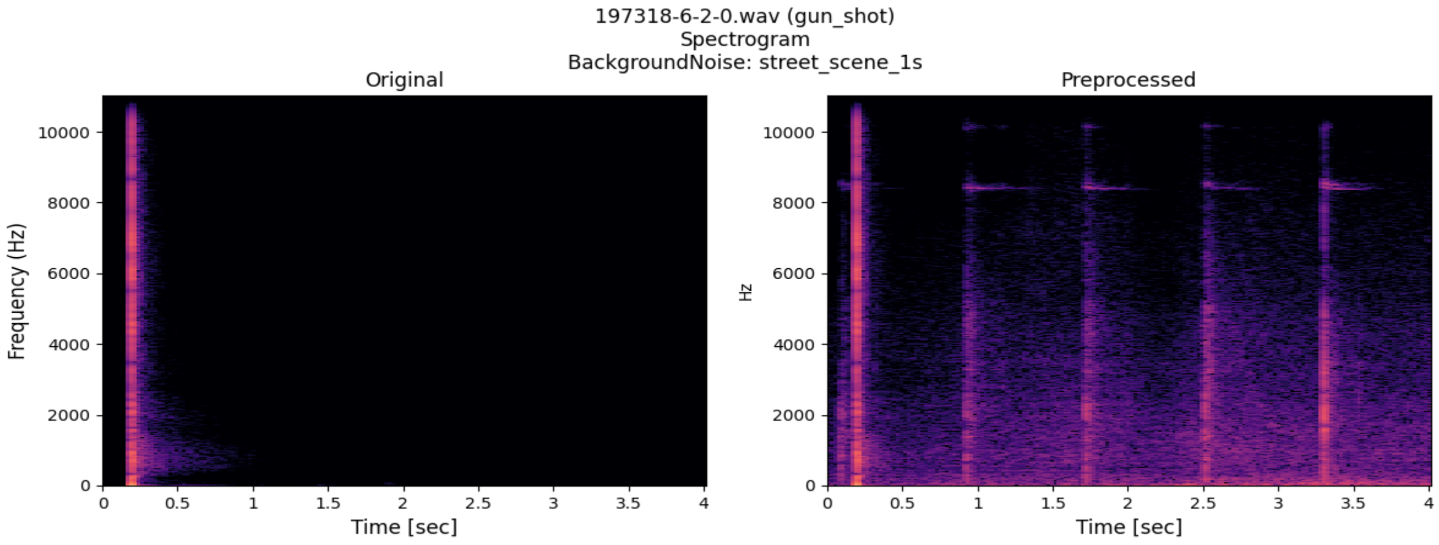


Fig. 39: Background Noise spectrogram on class gun_shot

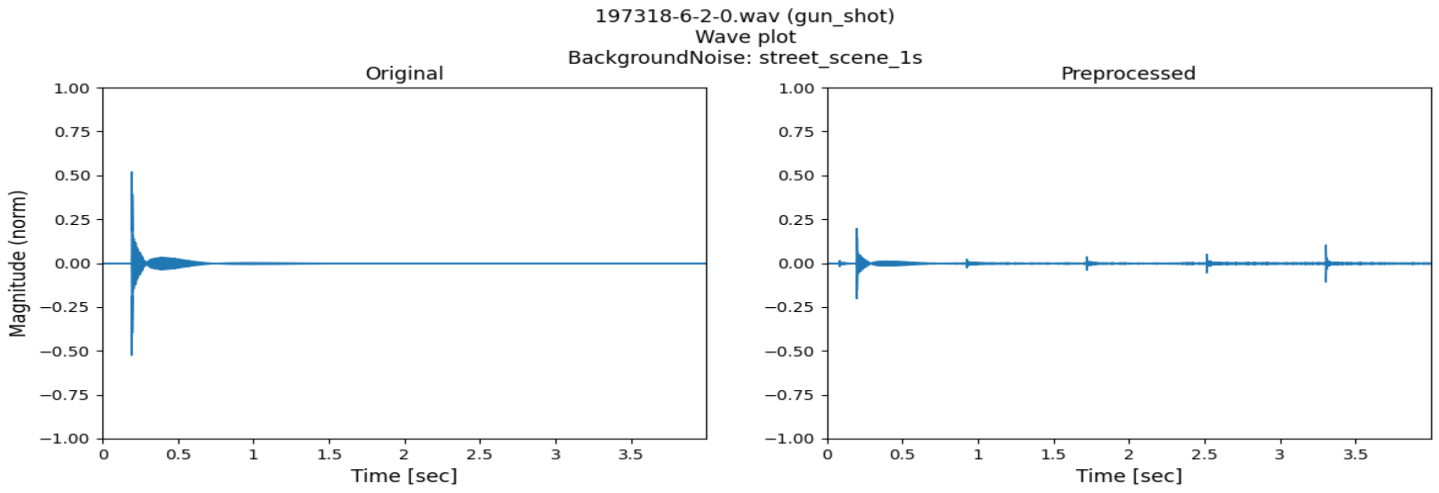


Fig. 40: Background Noise waveplot on class gun_shot

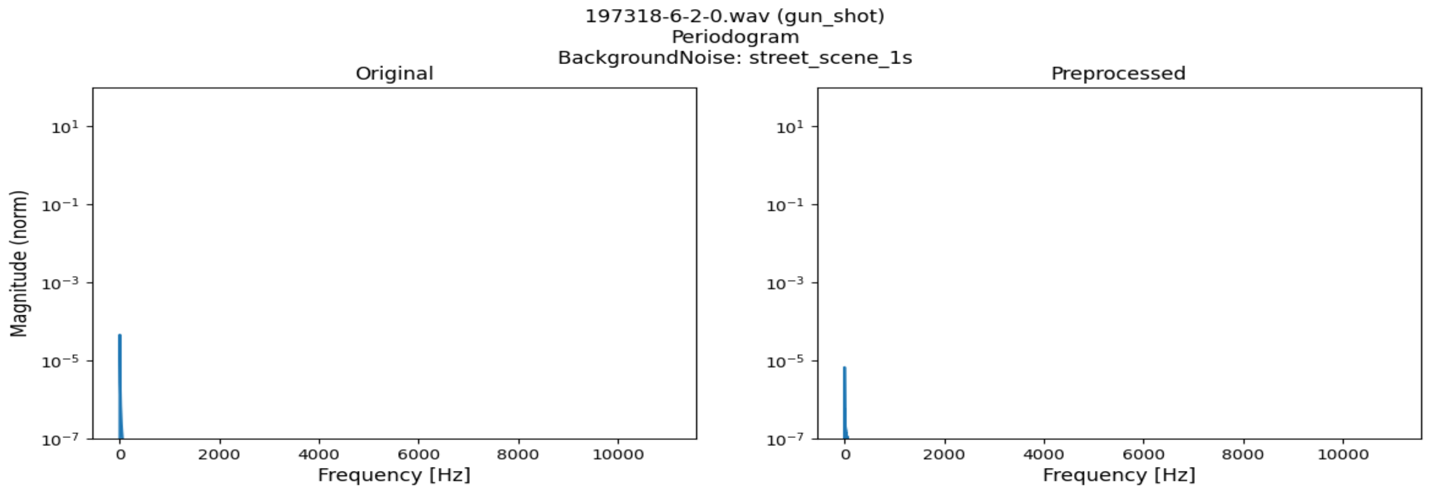
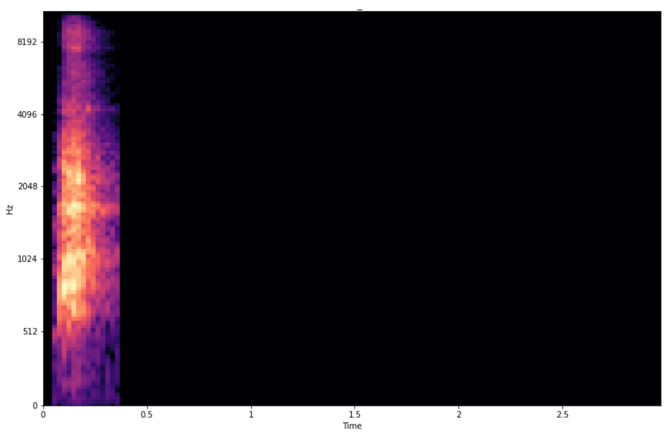
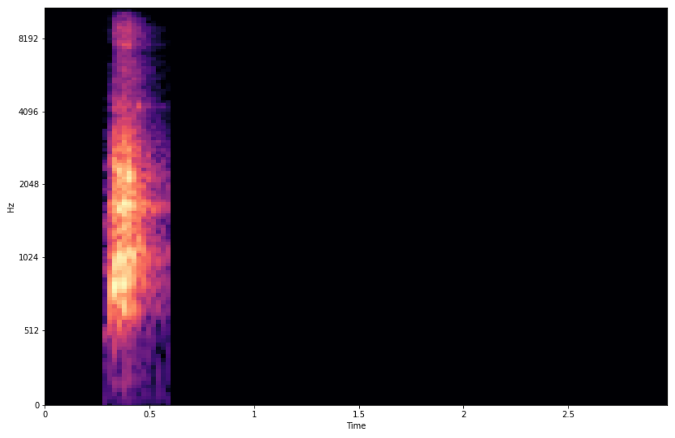


Fig. 41: Background Noise waveplot on class gun_shot

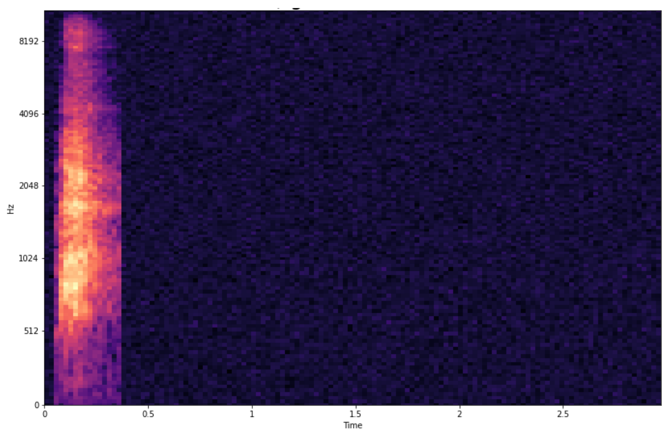
APPENDIX D
SPECTROGRAM PREPROCESSING



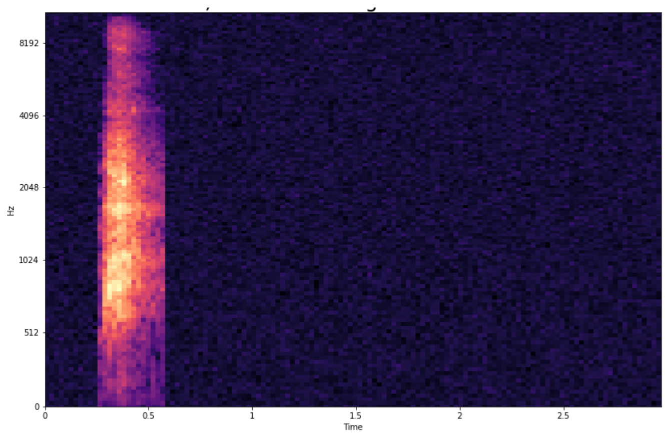
(a) Original spectrogram



(b) Right shift of the original spectrogram



(c) Gaussian noise applied to the original spectrogram



(d) Gaussian noise applied to the shifted spectrogram

Fig. 42: Image augmentations applied to the spectrograms. Notice that the order of application of these augmentations matters