FEATURE EXTRACTION, RECURSIVITY AND ATTENTION ARE ALL YOU NEED : URBAN SOUND TAGGING CHALLENGE

Louis Le Dain, Marie Médevielle, Maggy Zhou

Artificial Intelligence and Deep Learning Course - CentraleSupélec

ABSTRACT

Urban sound classification is a challenging task due to the complexity and variability of audio signals in real-world environments. However, accurately detecting and classifying sound events in noisy urban settings remains an open problem, particularly when dealing with diverse sound classes and inconsistent audio quality. In this paper, we outline our solution for the Urban Sound Tagging challenge across the eight coarse classes which consists of a feature extraction model coupled with a Gated Recurrent Unit (GRU) enhanced by an attention mechanism. The results show that our approach improves the Micro AUPRC score from the initial baseline by 0.27 on the test dataset.

***Index Terms*—** SONYC, DCASE, Convolutionnal Neural Networks, Fine-tuning, Attention mechanism

1. INTRODUCTION

Noise pollution is a real-world challenge in most major cities, such as New York City. More than 70 million American people are exposed to sound levels that are considered harmful to human health and may cause aggravations such as hearing loss, sleep disruption or heart tension. Not only are the effects on health critical, but noise pollution also affects the economic well-being of a city, with housing prices dropping with each decibel increase. Being able to implement data-driven solutions in order to identify the biggest contributions to sound pollution in a given area is the first step to curbing this issue. Sounds of New York City (SONYC) aims at developing a data-science method for predictive noise modeling integrated to a physical system to mitigate sound [1].

In this paper, we take on the DCASE 2019 Challenge [2] launched by SONYC which goal is, upon a ten-second-recording of urban environment, to determine whether any of 23 categories of sound sources is present or not.

Our data consist of audio recordings taken in the streets of New York containing elements belonging to one or more categories of the taxonomy described by Figure 1. In all, there are 23 fine-grained tags separated into 8 coarse-grained tags. In the rest of our work, we will consider the 8 coarse-grained tags.

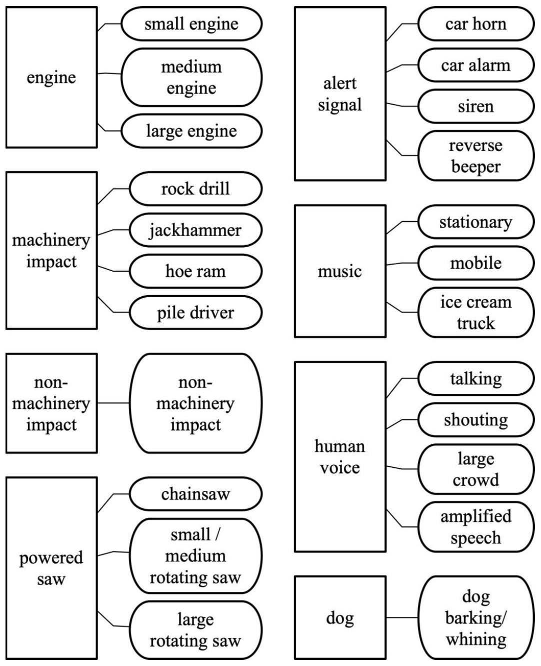

Fig. 1: **Hierarchical taxonomy of tags.** Rectangular and round boxes respectively denote coarse and fine tags.

To make the most of the complexity of audio recordings, we will consider their MEL spectrograms. The MEL spectrogram is a visual representation of frequencies and their energy as a function of time as can be seen in Figure 2.

We use a baseline inspired by the original one implemented by Cartwright et al. (2019) [3]. The global architecture is summarised in Figure 3. It is comprised of a VGGish model pre-trained on AudioSet [4] for feature extraction. The output of these convolutional layers are embedded in order to feed them to the multi-label logistic regression model.

In the current baseline, we identify a point of failure at the embedding step, right before the logistic regression. As it linearly formats the features, it loses all the temporal consistency of the data. In this paper we will showcase the improved performances brought by replacing the embedded layers with a recurrent neural network.

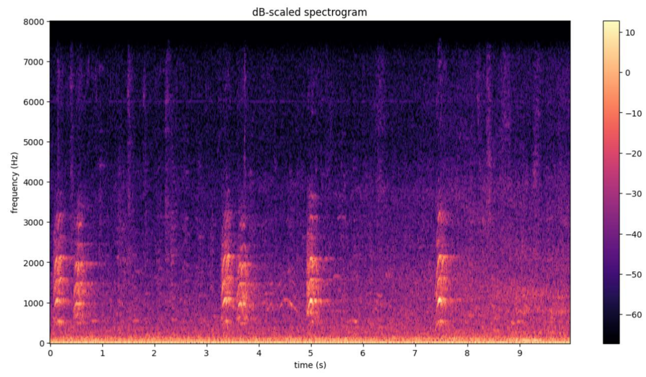

Fig. 2: **A MEL spectrogram from the train dataset.** This type of representation is fed to the baseline model.

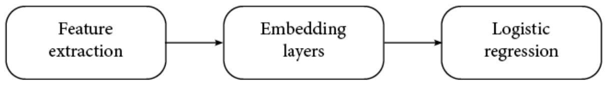

Fig. 3: **Global architecture of the baseline.** The baseline consists of 6 convolutional layers (VGGish model), 3 embedding layers and one layer of logistic regression.

2. IMPROVEMENTS

2.1. A recurrent model for classification

The convolutional layers extract features regarding the patterns of the Mel-spectrogram so the temporal and thus sequential component is preserved. It is therefore possible to apply a recurrent model at the output of the convolutional layers, instead of the embedding layers.

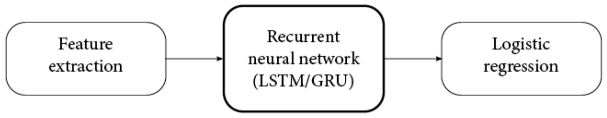

Fig. 4: **Global architecture of the model with a RNN.** We are replacing the embedding layers which flattened the features into a recurrent neural network which will be able to extract information from the temporal consistency of the Mel-spectrograms.

	Micro AUPRC	**Micro F1**	**Macro AUPRC**
Feature Extraction + LSTM	0.739	0.675	0.512
Feature Extraction + GRU	0.769	0.696	0.588

Table 1: **Comparative performances on validation dataset of the model with LSTM or GRU.** Using a GRU model gives slightly better performances.

As we can see in Table 1, we decide on a GRU model [5] to add after the feature extraction layers as it gives better performances and is lighter than LSTM [6].

2.2. Finetuning

Now that we tested different RNN models, the next step is to finetune the convolutional model. Until now, the weights of the VGGish convolutional layers were all frozen in order to benefit from the pre-training. However, in order to specify the model for our specific task of recognizing urban sounds, let us un-freeze some of the layers of the VGGish model.

Feature extraction layer	**# parameters**	**MaxPool**
layer1-conv1	640	True
layer-3-conv2	73,856	True
layer-5-conv3-1	295,168	False
layer-6-conv3-2	590,080	True
layer-8-conv4-1	**1,180,160**	**False**
layer-9-conv4-2	**2,359,808**	**False**

Table 2: **Convolutional layers inside the VGGish model as given by the baseline.** The two last layers in bold are the ones we will try to finetune, either just the last one or both of them.

In this section we highlight the compared performances between un-freezing the last convolutional layer or the last two convolutional layers before applying a GRU model, see Table 2.

	Micro AUPRC	**Micro F1**	**Macro AUPRC**
No finetuning	0.769	0.696	0.588
layer9-conv4-2 finetuned	0.758	0.711	0.576
layer8-conv4-1 and layer9-conv4-2 finetuned	0.774	0.709	0.590

Table 3: **Comparative performances on validation dataset with no finetuning, finetuning of the last VGGish layer, or finetuning of the last two VGGish layers.** Finetuning both of the layers gives the best performances, see section 3.2 for a discussion on learning rates.

2.3. Attention mechanism

The final proposed improvement to our model consists in adding an attention mechanism at the GRU output, before the final classification.

In order to make the most of the sequential aspect of our data, we exploit all GRU output vectors rather than just the last one. So, rather than relying on GRU to contain all the information needed for classification in the last output vector, we will weight the different outputs to make the most of the information.

The proposed attention mechanism is inspired by Bahdanau's additive attention [7]. In it, we compute attention scores using a fully-connected layer from the GRU output. These are then normalized to obtain weights.

These weights are then used to weight our output, which finally passes through a final fully-connected layer to obtain the values relative to the detection of the 8 labels.

	Micro AUPRC	Micro F1	Macro AUPRC
Without attention mechanism	0.774	0.709	0.590
With an attention mechanism	0.809	0.723	0.619

Table 4: **Performance on validation dataset** : the implementation of an attention mechanism significantly improves performance.

3. FINAL FRAMEWORK

3.1. Architecture

3.1.1. Feature extraction

We keep the feature extraction part of the VGGish model almost as is apart from the fact that the last MaxPool layer is removed and the weights of the two last convolutionnal layers are now trainable.

3.1.2. Recurrent model for classification

The feature extraction part is then followed by a recurrent model with the architecture of a GRU model, in order to take advantage of the sequential aspect of our data.

3.1.3. Attention mechanism

An attention mechanism is added at the end of the GRU model. All GRU output vectors are considered and weighted with attention scores to finely balance their contributions. Finally, a linear layer gives us the desired shape of output data.

3.2. Training

Our proposed model is trained with only 10 epochs, using BCE loss and Adam optimizer [8]. One remarkable point is that the finetuned feature extraction layers and the GRU model are not trained at the same speed. The two last extraction layers are respectively trained with a learning rate of 0.00008 and 0.0005 whereas the GRU model, as well as the weights of the attention mechanism, have a learning rate of 0.005. We use the method of differential learning rates [9] because the feature extraction is already pre-trained on more general data.

	Micro AUPRC	Micro F1	Macro AUPRC
GRU added	0.769	0.696	0.588
Finetuned version	0.774	0.709	0.590
Attention mechanism added	0.809	0.723	0.619

Table 5: **Results on the validation dataset through the various stages.** A look back at our performance improvements.

4. RESULTS COMPARISON AND ANALYSIS

Table 5 shows the relevance and effectiveness of the various improvement decisions taken.

Now we need to evaluate the performance of our final model on the test dataset.

	Micro AUPRC	Micro F1	Macro AUPRC
Baseline	0.531	0.486	0.348
Final Framework	0.801	0.721	0.671

Table 6: **Performance on test dataset.** All suggestions combined together lead to an overall boost of performances.

Performance on the test dataset as shown by Table 6 demonstrates the value of our final framework. Our final model improves the Micro AUPRC score by 0.27 compared to the baseline.

However, as Table 7 underlines, the results remain highly disparate across the different classes, due to the imbalance in the representations of the different coarse-grained tags in the training dataset.

	Micro AUPRC
engine	0.827
machinery-impact	0.769
non-machinery-impact	0.269
powered-saw	0.668
alert-signal	0.811
music	0.193
human-voice	0.909
dog	0.919

Table 7: **Micro AUPRC on coarse-grained tags from test dataset.** A look back at our performance improvements.

Thus, a further discussion on improving our model could focus on balancing performance on the different tags, for example through data augmentation.

5. CONCLUSION

Our results show how harnessing the sequential consistency of the data is key to improving the baseline model of urban sound recognition. In addition to replacing the embedding layers by a GRU model, a few steps help further enhance performances:

Firstly, finetuning the last two layers of the feature extraction part of the baseline allows to leverage the advantages of the pre-trained VGGish model while specializing it to recognize urban sounds. A study on hyperparameters such as the learning rates for each part allows for an adaptive training.

Secondly, using an attention mechanism at the output of the GRU model enables a refined usage of all the output vectors.

Further work would be to focus on balancing performances over the different tags, for instance through data augmentation.

6. REFERENCES

[1] Juan P. Bello, Claudio Silva, Oded Nov, R. Luke Dubois, Anish Arora, Justin Salamon, Charles Mydlarz, and Harish Doraiswamy, "SONYC: A system for monitoring, analyzing, and mitigating urban noise pollution," *Communications of the ACM*, vol. 62, no. 2, jan 2019.

[2] "DCASE 2019 Challenge," `https://github.com/sonyc-project/urban-sound-tagging-baseline`.

[3] M. Cartwright, J. Cramer, A. E. Mendez, Y. Wang, H. Wu, V. Lostanlen, M. Fuentes, G. Dove, C. Mydlarz, J. Salamon, O. Nov, and J-P. Bello, "Sonyc-ust-v2: An urban sound tagging dataset with spatiotemporal context," *arXiv preprint arXiv:2009.05188*, 2020.

[4] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "Youtube-8m: A large-scale video classification benchmark," *arXiv preprint arXiv:1609.08675*, 2016.

[5] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[6] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[7] B. Yoshua D. Bahdanau, C. Kyunghyun, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[8] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," 2017.

[9] Dhruv Matani, "A practical guide to using PyTorch," `https://www.kdnuggets.com/2023/06/practical-guide-transfer-learning-pytorch.html`, June 2023.