VGG CNN FOR URBAN SOUND TAGGING

Technical Report

Clément Gousseau

Royal Institute of Technology (Stockholm, Sweden) and Orange Labs (Lannion, France)
clement.gousseau@gmail.com

ABSTRACT

A model of urban sound tagging is presented (Task 5 of DCASE 2019 [1][2]). The task is to detect activities from 10-seconds audio segments recorded in the streets of New York City (SONYC dataset). The model is based on the model presented in the book *Hands-On Transfer Learning with Python* [3] which does urban sound classification for the *UrbanSound* dataset. This model has been adapted and optimized to address the task 5 of DCASE2019. It achieved a AUPRC of 82.6 for the coarse-grained model where the baseline achieves an AUPRC of 76.2.

1. INTRODUCTION

The development dataset [2] is composed of 2351 recordings in the training dataset and 443 recordings in the validation dataset. Each recording is a 10-seconds audio segment recorded in the streets of New York City. For the training set, each recording has been annotated by three volunteers on Zooniverse, a web platform for citizen science. For the validation set, each recording has been annotated by the organizers of the DCASE Challenge. The annotations follow a fined-grained or a coarse-grained taxonomy (see figure 1).

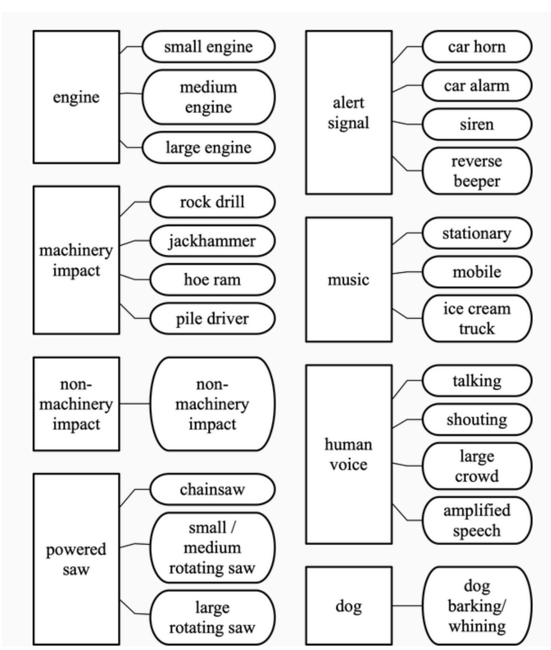

Figure 1: DCASE Task 5 Taxonomy [1]

Labels are non-exclusive: several (or none) classes can be present in each recording. For each recording, the goal is to output a probability of presence for each class. The performance of the model is measured using micro-averaged AUPRC (Area Under Precision Recall Curve [4]).

2. PROPOSED MODELS

2.1. Features

The feature engineering uses the method presented in the book *Hands-On Transfer Learning with Python*. First the recordings are re-sampled using a sampling rate of 22050Hz. Then three features are extracted from these signals:

- The mel-spectrograms using 64 mel-bands and a hop length of 512 thus resulting a 64 rows x 431 columns image.
- The averaged value of the harmonic and percussive components (64 rows x 431 columns image).
- The derivative of the log-mel spectrograms (64 rows x 431 columns image).

These spectrograms have been extracted using the *librosa* library [5]. The figure below represents the three spectrograms extracted from an alert-signal recording.

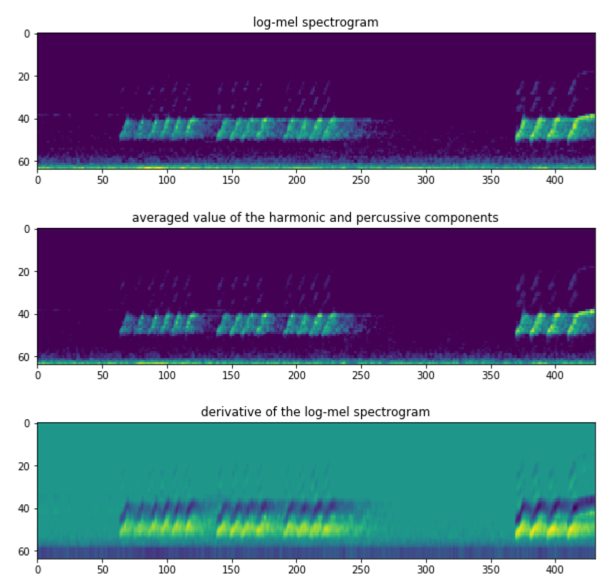

Figure 2: The three input images corresponding to a recording

2.2. Model structure

A VGG-style [6] convolutional neural network is used to detect the classes from the input spectrograms:

Input 64 x 431 x 3
64 x conv 3x3 64 x conv 3x3
MaxPooling 2x2
128 x conv 3x3 128 x conv 3x3
MaxPooling 2x2
256 x conv 3x3 256 x conv 3x3 256 x conv 3x3
MaxPooling 2x2
512 x conv 3x3 512 x conv 3x3 512 x conv 3x3
MaxPooling 2x2
512 x conv 3x3 512 x conv 3x3 512 x conv 3x3
MaxPooling 2x2
Flatten
1024-Fully Connected + L2-regularization ReLu Activation Dropout
1024-Fully Connected + L2-regularization ReLu Activation Dropout
512-Fully Connected + L2-regularization ReLu Activation Dropout
512-Fully Connected + L2-regularization ReLu Activation Dropout
n_classes-Fully Connected + L2-regularization Sigmoid Activation

All convolutional layers are initialized with the weights of VGG16 pre-trained on Imagenet dataset [7] but they remain unfrozen during the training.

This model has 30,188,360 parameters.

n_classes depends on the targets (fine-grained or coarse-grained labels).

The model was implemented using Keras. The code is available at https://github.com/cgousseau/dcase_task5.

2.3. Data augmentation

The training set is quite small (2351 samples). Data augmentation is a way to artificially increase the size of the training set and avoid overfitting. Mixup is a data augmentation method that has been experimented for this task. From two samples $\{input : x_1, target : y_1\}$ and $\{input : x_2, target : y_2\}$ a 'new' sample is created: $\{input : x_3 = \lambda x_1 + (1-\lambda)x_2, target : y_3 = \lambda y_1 + (1-\lambda)y_2\}$ where $\lambda \sim \beta(mixup\_rate)$ [8].

2.4. Model optimization

Hyperparameter tuning is an important but tricky part of machine learning problems. For this task, Particle Swarm Optimization [9] has been used to optimize some hyperparameters of the network. These hyperparameters are: *dropout_rate*, *L2-regularization constant*, *batch size*, *mixup_rate*.

Particle Swarm Optimization is a population-based optimization method, it was inspired from animal social groups like herds, schools and flocks. The swarm is composed of *n* particles which have a position in the *d*-dimensions hyperparameter search space. Particles move in the search space and cooperate according to simple mathematical formulas in order to find an optimal solution. Each particle is driven by three components: an inertia component, an individual component and a collective component.

In the D-dimensional search space:

- the position of the i-th particle at time t is $x_i(t) = (x_{i1}(t), ..., x_{iD}(t))$
- the best position so far of the i-th particle at time t is $pbest_i(t) = (pbest_{i,1}(t), ..., pbest_{i,D}(t))$
- the best position so far of the whole swarm at time t is $gbest(t) = (gbest_1(t), ..., gbest_D(t))$

At each iteration, the position of the i-th particle is updated:

$$x_i(t+1) = x_i(t) + v_i(t) \quad (1)$$

$$\begin{aligned} v_i(t+1) &= \omega v_i(t) + c_1 r_1(pbest_i(t) - x_i(t)) \\ &\quad + c_2 r_2(gbest_i(t) - x_i(t)) \end{aligned} \quad (2)$$

$v_i(t)$ is the velocity of the i-th particle at time t, ω is an inertia weight scaling the previous time step velocity, c_1 and c_2 are two acceleration coefficients that scale the influence of the best personal position of the particle $pbest_i(t)$ and the best global position $gbest(t)$ and r_1 and r_2 are random variables within the range of 0 and 1.

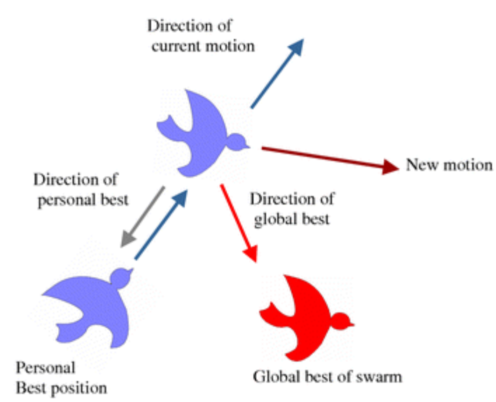

Figure 3: Schematic representation of updating the velocity of a particle [10]

Here, 5 particles were used for the optimization. 6 iterations were done, then 30 hyperparameters settings were evaluated. The hyperparameter setting that gave the best score was:

- batch size: 1

- constant for L2-regularization: 0.1
- mixup rate: 0.85
- dropout rate: 0.3

2.5. Model training

The model is trained using the following hyperparameters:

- loss: binary crossentropy as defined in the baseline
- optimizer: Adam
- learning rate: 1e-5
- batch size: 1
- constant for L2-regularization: 0.1
- mixup rate: 0.85
- dropout rate: 0.3
- number of epochs: 100

The model is trained using a GPU (NVIDIA GeForce GTX 1080), the training takes about 1 hour.

2.6. Model selection

The model is trained during 100 epochs and AUPRC is computed at each epoch. The best 4 models that have the best AUPRC are stored. For each tag the AUPRC is computed and the model that has the best AUPRC is selected to predict this tag.

3. RESULTS

3.1. Coarse-level model

The targets of this model are tags among a list of 8 which are pretty far from each other (e.g. engine, dog, alert signal). The taxononomy is detailed in figure 1.

	best pre-trained VGG	Baseline
Micro AUPRC	82.6	76.2
Micro F1-score	74.3	67.4
Macro AUPRC	61.1	54.2
Coarse tag AUPRC		
engine	86.8	85.5
machinery impact	60.5	36.0
non-machinery impact	56.5	36.1
powered saw	68.9	67.9
alert signal	92.1	81.3
music	18.0	29.9
human voice	94.8	94.5
dog	11.4	2.8

3.2. Fine-level model

The targets of this model are tags among a list of 23 which can be pretty close from each other (e.g. small engine, medium engine, large engine). The taxononomy is detailed in figure 1.

3.2.1. *Fine-level evaluation*

	best pre-trained VGG	Baseline
Micro AUPRC	70.1	67.2
Micro F1-score	61.3	50.2
Macro AUPRC	47.2	42.7
Coarse tag AUPRC		
engine	65.3	71.2
machinery impact	25.1	19.8
non-machinery impact	54.3	36.4
powered saw	31.2	38.6
alert signal	83.3	63.6
music	11.8	21.5
human voice	88.7	88.0
dog	18.2	2.9

3.2.2. *Coarse-level evaluation*

	best pre-trained VGG	Baseline
Micro AUPRC	77.4	74.3
Micro F1-score	63.8	50.7
Macro AUPRC	56.7	53.0
Coarse tag AUPRC		
engine	85.2	85.9
machinery impact	27.0	28.5
non-machinery impact	54.3	36.4
powered saw	64.2	72.0
alert signal	90.4	75.3
music	18.7	28.3
human voice	95.7	94.3
dog	18.2	2.9

4. FUTURE WORK

Binary crossentropy is widely used as a loss function in classification tasks. It is differentiable and therefore it can be used for stochastic gradient descent. However in the DCASE 2019 Task 5, the metrics is AUPRC which is only partially correlated to crossentropy. One can see on the figure below that AUPRC is increasing while crossentropy is decreasing during the first epochs. But quickly AUPRC stops increasing while crossentropy is still decreasing.

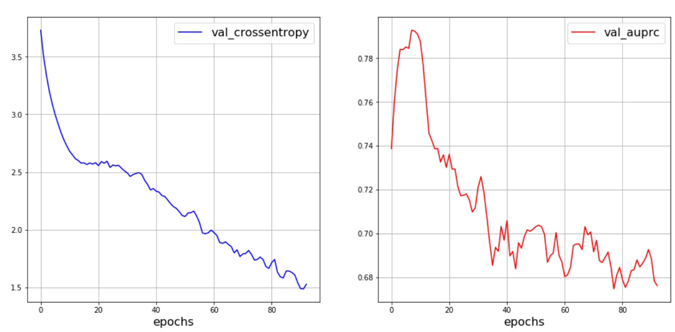

Figure 4: Evolution of crossentropy and AUPRC during a training (evaluation on the validation set). Recall: we want to minimize crossentropy whereas we want to optimize AUPRC.

We can explain this by the fact that crossentropy depends on the distance between the predicted probabilities and the true probabilities (targets) whereas AUPRC depends on the ranking of predicted probabilities.

An illustration: we have a problem where the true probabilities (targets) are $y_{true} = [0, 0, 1, 1]$ and two diffrent predicted probabilities ($y_{pred_1} = [0.01, 0.8, 0.2, 0.99]$ for case 1 and $y_{pred_2} = [0.01, 0.6, 0.4, 0.99]$ for case 2).

	case 1	case 2
predicted probabilities	$[0.01, 0.8, 0.2, 0.99]$	$[0.01, 0.6, 0.4, 0.99]$
true probabilities	$[0, 0, 1, 1]$	$[0, 0, 1, 1]$
crossentropy	3.24	1.85
AUPRC	0.79	0.79

We have $cross\_entropy_2 < cross\_entropy_1$ because the predicted probabilities are closer to true probabilities in case 2 than in case 1. However the probabilities are ranked in the same order so they will have the same precision/recall curve, and therefore the same AUPRC.

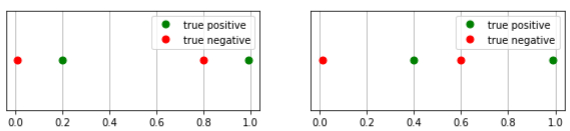

Figure 5: Predicted probabilities for case 1 (left) and 2 (right) (green dots correspond to a target 1, red dots correspond to a target 0).

Another loss function should be used to optimize AUPRC. AUPRC cannot be used as a loss function since it is not differentiable. Alternatives have been tested (e.g. pair-wise mean squared error) but it showed convergence problems.

5. CONCLUSION

Tagging of urban sounds was investigated. A model based on pre-trained VGG-style network was developped and submitted to the challenge DCASE 2019 (task 5). The best model achieves an AUPRC of 82.6 for the coarse-level prediction and 70.1 for the fine-level prediction.

6. REFERENCES

[1] http://dcase.community/challenge2019/.

[2] J. P. Bello, C. Silva, O. Nov, R. L. Dubois, A. Arora, J. Salamon, C. Mydlarz, and H. Doraiswamy, "Sonyc: A system for monitoring, analyzing, and mitigating urban noise pollution," *Communications of the ACM*, vol. 62, no. 2, pp. 68–77, Feb 2019.

[3] D. Sarkar, R. Bali, and T. Ghosh, *Hands-On Transfer Learning with Python: Implement advanced deep learning and neural network models using TensorFlow and Keras*. Packt Publishing, 2018. [Online]. Available: https://books.google.fr/books?id=aPFsDwAAQBAJ

[4] https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html.

[5] B. McFee, M. McVicar, S. Balke, V. Lostanlen, C. Thom, C. Raffel, D. Lee, K. Lee, O. Nieto, F. Zalkow, D. Ellis, E. Battenberg, R. Yamamoto, J. Moore, Z. Wei, R. Bittner, K. Choi, nullmightybofo, P. Friesch, F.-R. Stter, Thassilo, M. Vollrath, S. K. Golu, nehz, S. Waloschek, Seth, R. Naktinis, D. Repetto, C. F. Hawthorne, and C. Carr, "librosa/librosa: 0.6.3," Feb. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2564164

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[7] https://keras.io/applications/#vgg16.

[8] https://en.wikipedia.org/wiki/Beta_distribution.

[9] Y. Shi and E. RC, "A modified particle swarm optimizer," vol. 6, 06 1998, pp. 69 – 73.

[10] A. Ahmadi, F. Karray, and M. S. Kamel, "Flocking based approach for data clustering," *Natural Computing*, vol. 9, pp. 767–791, 09 2010.