Multi-label audio classification —6th place solution for Freesound Audio Tagging 2019 Competition

udio tagging is the task of labelling an audio clip with one or more labels or tags. In this post, I describe my solution (code on GitHub) to the Freesound Audio Tagging 2019 competition hosted on Kaggle. In this competition, participants were challenged to build an automating audio tagging system.

Dataset

The dataset provided [1] consists of a curated subset and a noisy subset labelled with **80 different tags**. Moreover, multiple tags can co-occur in the same clip.

Curated subset: 4970 audio clips with duration ranging from 0.3 to 30s.

Noisy subset: 19815 audio clips with duration ranging from 1 to 15s.

Data preprocessing

All waveforms were converted to Mel spectrograms (thanks daisukelab for sharing the code to generate the spectrograms!). In spectrograms, the vertical axis represents the frequency. The bottom part of the image shows lower pitch sounds and moving upwards increasingly higher pitches are represented.

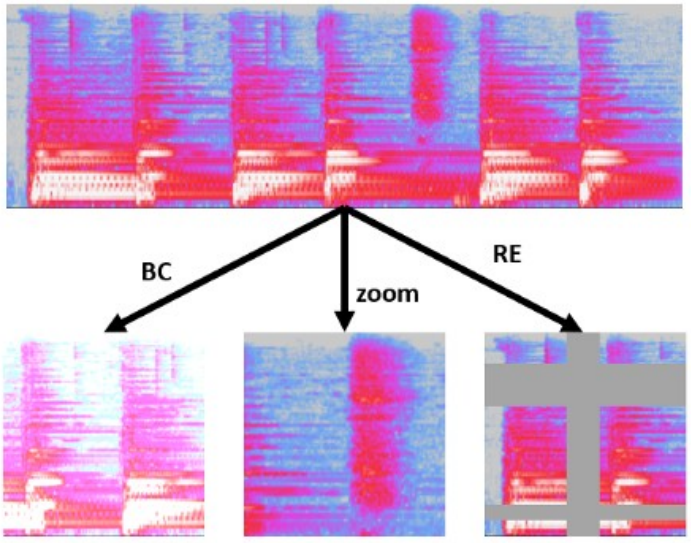
All waveforms were converted to spectrograms with a height of 128 pixels and width depending on the length of the clip.

Augmentations

Data augmentations are quite important in this competition as the dataset is small, particularly the curated subset with less than 5000 clips for 80 classes. Furthermore, no pre-trained models were allowed on this competition; the models need to be trained from scratch!

I've applied brightness and contrast, zoom and random erasing augmentations. For the random erasing vertical and horizontal bars were used, as in Specaugment [2]. As I used square images, I randomly crop 128x128 regions of the clips and rescale to 256x256 as it resulted in better performance.

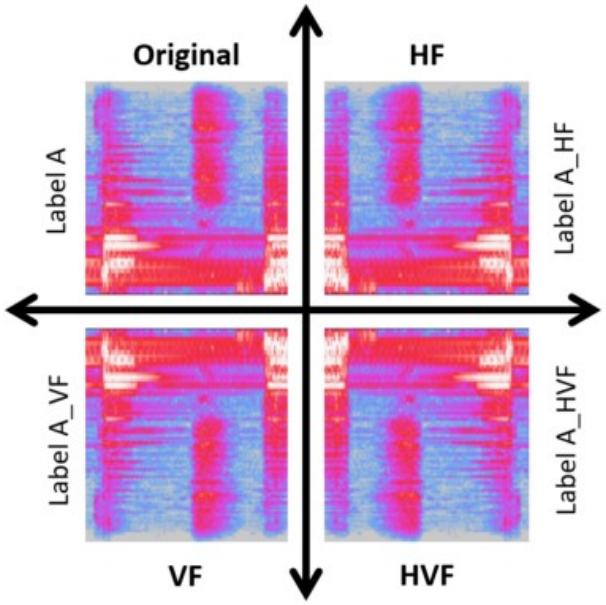
- Brightness and Contrast (BC)
- Max zoom 1.5
- Random erasing (RE)
- Random crop 128x128
- Resize 256x256



Flipping images horizontally and/or vertically can also be a good form of augmentation. However, audio clips change their meaning when applying any of these transformations. If you listen to audio played backwards, it's very weird with a completely different meaning. If you flip the audio vertically, in the frequency domain, it will probably sound even stranger. [Something I should add to my to-do list!] But this "limitation" can be used as an advantage! The strategy I followed was to consider the flipped clips labels as new tags. This way, I get a 4x larger dataset with a total of 320 labels. Moreover, the 4 flips can be ensembled, boosting the performance of the model.

- Horizontal Flip (HF)
- Vertical Flip (VF)
- Horizontal + Vertical Flip (HVF)

Different labels for each flip!
80 x 4 = 320 unique labels
4x larger dataset



Finally, another useful type of augmentation that can be applied in several domains is Mixup [3]. In **Mixup**, two images are combined as follows:

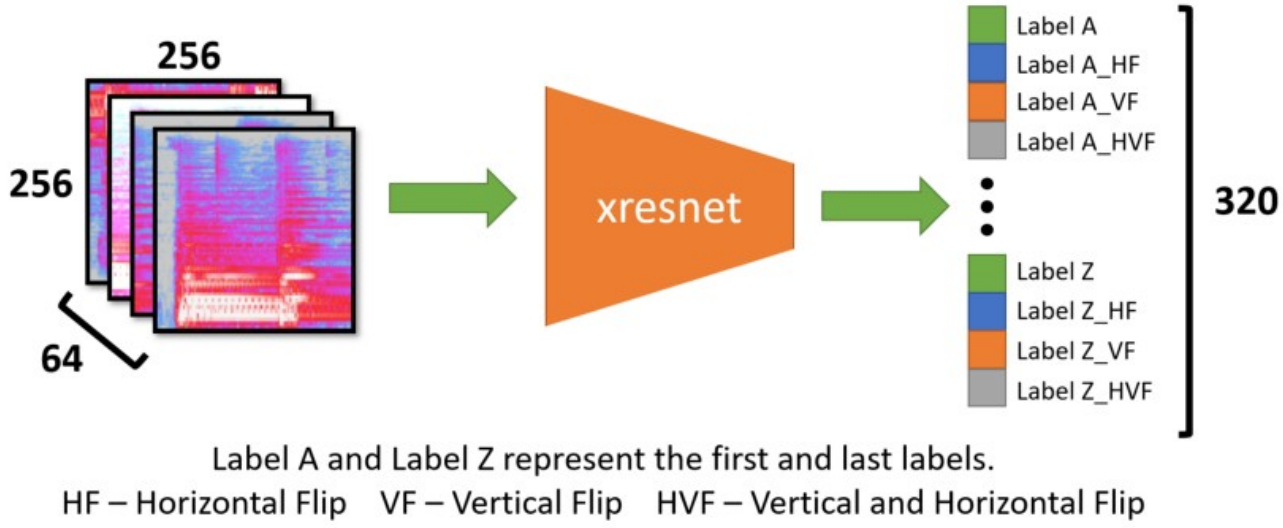
Mixed image = a * Image 1 + (1-a) * Image 2

where "a" is randomly selected between 0 and 1. The same operation is applied to the labels to obtain a combined label. Usually "a" is generated from a beta distribution but I opted to use a uniform

distribution as summing the spectrograms seems more natural than combining let's say a picture of a cat with a picture of a dog. [I still need to check in the final models if using a uniform distribution is actually improving the score. I may update this section later.]

Model

For most of my experiments, I used an **xresnet18** (resnet18 with modifications suggested by [4]) as implemented in the fastai library with the addition of *simple self-attention* [5] that resulted in about +0.002 improvement on the public leaderboard.



Loss function — BCE++

For the loss function, I tried Binary Cross Entropy (BCE) and Focal Loss. BCE gave about +0.004 comparing with FocalLoss.

I was thinking about curriculum learning and decided to try something very simple: computing the loss only with samples with an F2 score (with a threshold of 0.2) lower than 1, this turned out to give me an improvement of about +0.005 on the public LB. The idea is that as the train progresses the easy samples (the ones the model get right first) are not used to compute the loss, making the task progressively more difficult. [I've tried this idea on Imagenette but without success so far. I may update this section if I find further insights about this.]

Implementation details

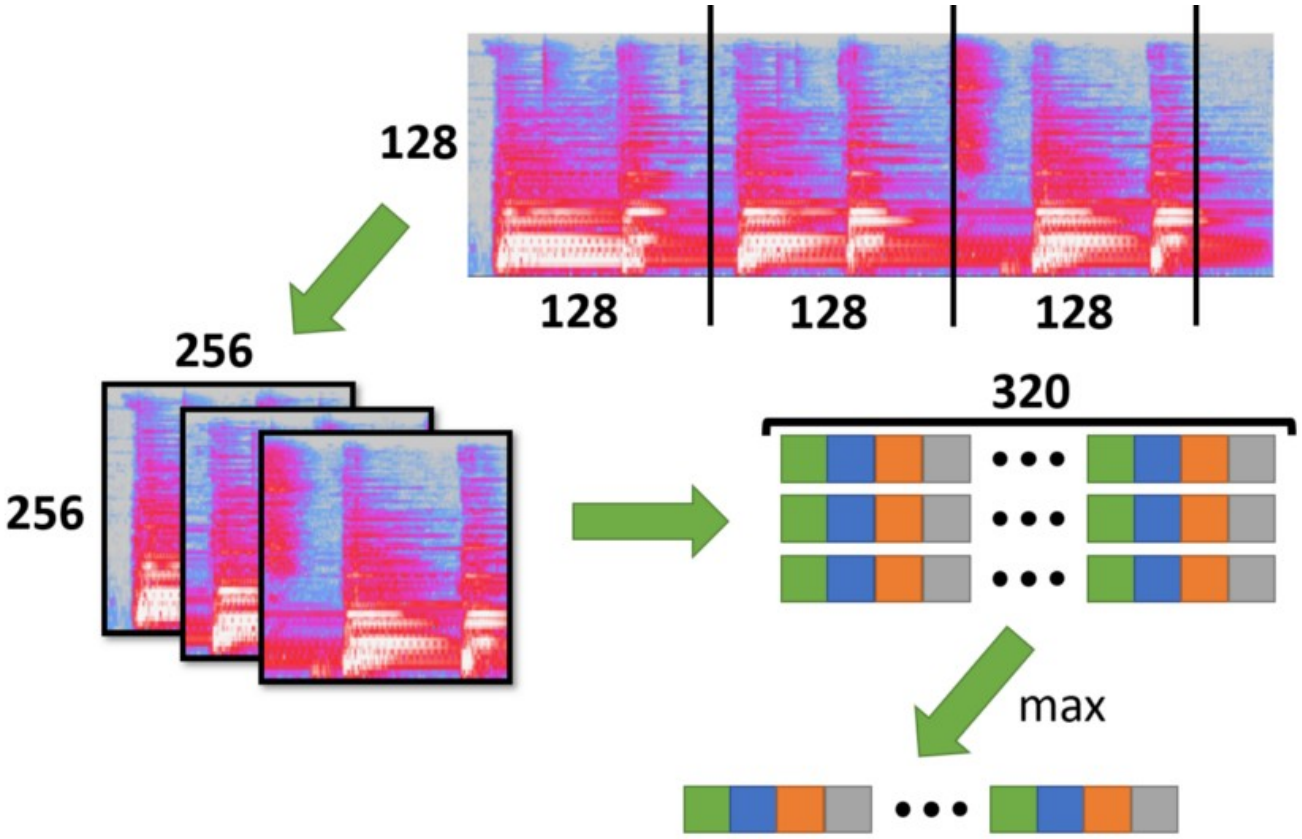
I used the fastai library, as it makes it very easy to implement new ideas together with all the state-of-the-art features already included out-of-the-box. I trained with *one_cycle_learning* with a maximum learning rate of 0.01 for close to 9h (the Kaggle kernel time limit) and adjusting the number of epochs accordingly, depending on the model being used.

Noisy data

I have tried several approaches to use noisy data, but at the end, the only that gave me an improvement on the final score was: to use a trained model to predict the noisy labels; select the noisy clips that were correctly predicted; and use them again for training the model from scratch together with the curated data. I selected about 3500 noisy clips to add to the 4970 curated clips.

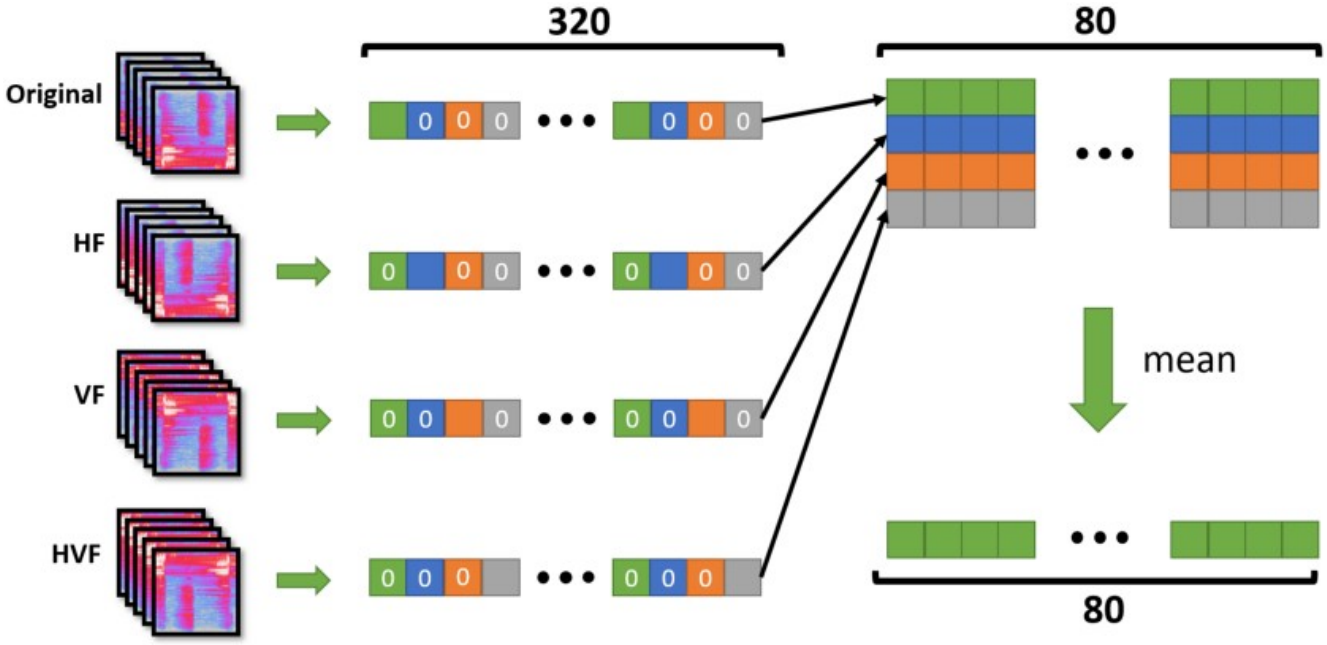
Test-time augmentations (TTA)

For the TTA I split each clip in 128x128 images without overlap, I then generated predictions for all slices of the clip and computed the *max* of those. Using the *max* gave me an improvement of about +0.001 comparing with the *mean*. The rationale is that some sounds may only occur in part of the clip and the model may predict them with high confidence, however, if they are not in the remaining parts of the clip, the average will decrease the probability for that label, and the outcome may be different.



Test predictions and ensembling

For the test set, I generate predictions for the 4 image flips (original, horizontal flip (HF), vertical flip (VF), horizontal and vertical flips (HVF) and average the probabilities of the 4 corresponding labels as exemplified in the next figure. Labels from a different flip are set to 0.



For multiple model ensembling, I average the probabilities resulting from the above procedure.

Final submissions

1. **Average of two models:**

   Model 1 — xresnet18 with simple self-attention and BCE loss.

   Model 2 — xresnet34 with simple self-attention and BCE loss.

   Result: 0.742 public LB, 0.74620 private LB.

2. Average of six models:

   Model 1 — xresnet18, simple self-attention, Focal Loss.

   Model 2 — xresnet34, simple self-attention, Focal Loss.

   Model 3 and Model 4 — as the first two but using BCE loss.

   Model 5 — xresnet34, Focal Loss

   Model 6 — xresnet50, BCE loss.

   Result: 0.742 public LB, **0.75421 private LB.**

Both ensembles scored 0.742 on public LB, but the ensemble of 6 models got +0.008 on the private LB!

Code

https://github.com/mnpinto/audiotagging2019

References

[1] Fonseca, E., Plakal, M., Font, F., Ellis, D. P., & Serra, X. (2019). Audio tagging with noisy labels and minimal supervision. *arXiv preprint arXiv:1906.02975*.

[2] Park, D. S., Chan, W., Zhang, Y., Chiu, C. C., Zoph, B., Cubuk, E. D., & Le, Q. V. (2019). Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*.

[3] Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*.

[4] Xie, J., He, T., Zhang, Z., Zhang, H., Zhang, Z., & Li, M. (2018). Bag of tricks for image classification with convolutional neural networks. *arXiv preprint arXiv:1812.01187*.

[5] https://github.com/sdoria/SimpleSelfAttention