DCASE 2019 TASK 2: SEMI-SUPERVISED NETWORKS WITH HEAVY DATA AUGMENTATIONS TO BATTLE AGAINST LABEL NOISE IN AUDIO TAGGING TASK

Technical Report

Jihang Zhang

getBIsmart.com
200 Spectrum Center Dr
Irvine, CA, 92618, USA
jihangz2519@gmail.com

Jie Wu

ENGINE — Transformation
60 Great Portland Street, London, England
W1W 7RT, GB
lftuwujie@gmail.com

ABSTRACT

This technical report describes a system used for DCASE 2019 Task 2: Audio tagging with noisy labels and minimal supervision. Building a large-scale multi-label dataset normally requires extensive amount of manual effort, especially for general-purpose audio tagging system. To tackle the problem, we use a semi-supervised teacher-student convolutional neural network (CNN) to leverage substantial noisy labels and small curated labels in dataset. To further regularize the system, we exploit multiple data augmentation methods, including *SpecAugment* [1], *mixup* [2], and an innovative time reversal augmentation approach. Moreover, a combination of binary *Focal* [3] and *ArcFace* [4] losses are used to increase the accuracy of pseudo labels produced by the semi-supervised network, and accelerate the training process. Aadaptive test time augmentation (TTA) based on the lengths of audio samples is used as a final approach to improve the system. We choose a single system that generates the submission file **Zhang_BIsmart_task2_3.output.csv** to be the candidate model considered for the Judges' Award. Other two systems use ensemble approach to furthur improve the performance.

Index Terms— Audio Tagging, Label Noise, Semi-Supervised Learning, Data Augmentation

1. INTRODUCTION

A general-purpose audio tagging system can be applied to various tasks such as real-time sound events recognition and video tagging. However, it is difficult for not only machines, but also human beings to distinguish similar sound events. It means that extensive effort would be required to manually annotate audio events if using supervised approach to build the system. Task 2 of DCASE 2019 challenge expects participants to build an accurate and efficient general-purpose audio tagging system using a combination of small size of manually annotated sound events and a large size of data without verified labels [5].

In this challenge, Freesound provides the dataset **FSDKaggle2019**. It was collected by Music Technology Group at the University Pompeu Fabra (MTG-UPF) organized with the AudioSet Ontology [6]. The duration of the audio clips in the curated set is between 0.3s and 30s. This dataset is split into three subsets: a curated training set with 4,970 audio clips, a public test set with 1,120 clips, and a private test set with 3,361 clips. The labels in FSDKaggle2019 are manually annotated. In addition, the challenge also provides a dataset with 19,815 audio clips gathered from the Yahoo Flickr Creative Commons 100M dataset (YFCC) [7]. Audio clips in YFCC are labeled automatically so that a substantial amount of label noise is expected. The duration of of the audio clips in the noisy ranges between 1s and 15s, with vast majority lasting 15s.

There are three major challenges in this task. First, unlike the task last year [8], participants face a multi-label classification problem instead of multi-class classification. Second, the noisy set is nearly 4 times of the curated training set. A system trained solely on the curated training set would be easily overfit and thus it might have lower performance when applied to audio events of general sources. Third, the source of curated and noisy sets are different. We expect to see a system trained on samples from YFCC have poor performance on samples from FSDKaggle2019 due to the domain mismatch. How to denoise the label, and leverage the use of the noisy set to improve the regularization of the system are the keys to succeed in this challenge.

Inspired by [9], we build a highly generalized system by utilizing the noisy set while keeping the influence of domain mismatch at the minimal level. We first warm up the neural networks with noisy data, then construct a semi-supervised network to automatically generate pseudo-labels for audio clip in the noisy set during the training phase. We use a combination of classification loss and large-margin loss to increase the credibility of pseudo-labels. Moreover, extensive data augmentation approaches are applied to further regularize the network. We introduce a time reversal augmentation approach, which improves the performance with minimal computational cost.

2. APPROACH

In this section, we talk about the approach we used to train our system. To facilitate the description, we introduce some notations. Given a batch V of N_V labeled samples from FSDKaggle2019 with corresponding binary targets y_V and a batch W of N_W samples from YFCC, our Teacher-Student network produces a processed batch of augmented labeled samples $\widehat{V}$ and a batch of augmented samples $\widehat{W}$ with pseudo-labels $\hat{y}_{\widehat{W}}$ guessed by the network. Then we apply *mixup* to create $V' \times y_{V'}$ and $U' \times y_{U'}$ that are used to compute the multi-task loss term described below. We describe each part of the process in the following subsections.

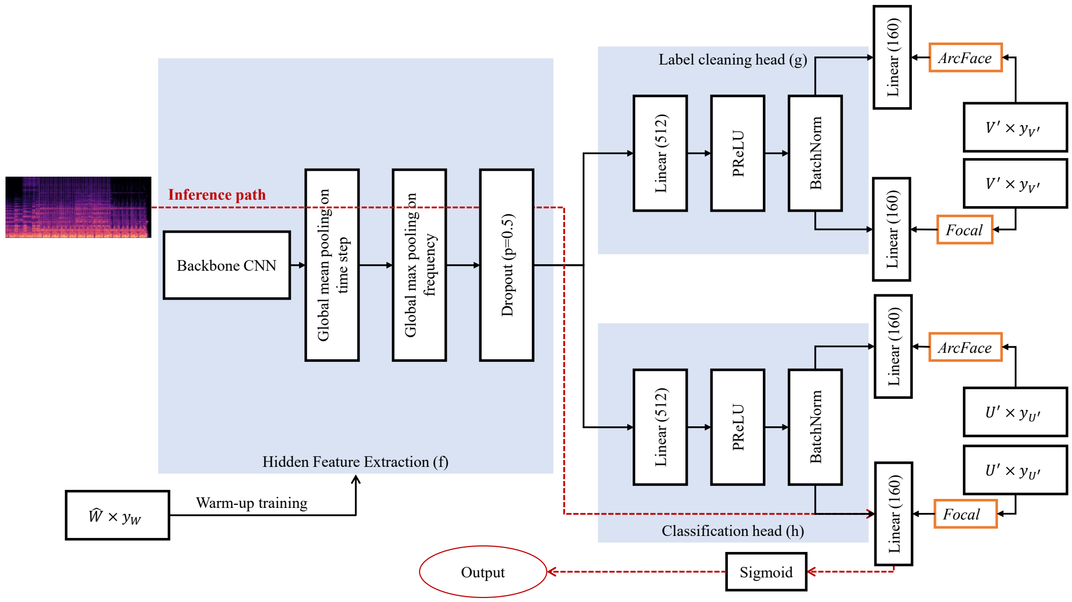

Figure 1: Overview of the system

2.1. Preprocessing and Feature Extraction

We first read the audio files using 32 kHz sampling rate, then trim the beginning and ending silence. A frame is silent if its power is 55 dB below the peak power in the entire signal. Then we covert the signal from time domain to logarithm-scaled mel-spectrogram (log-mel spectrogram) using the following parameters: length-1024 FFTs, hanning window, length-500 hop size, and mel frequency filter bank of size 128. Under such settings, the log-mel spectrogram of an 1-second audio clip consists of 64 frames. After extracting the log-mel spectrogram for all audio clips in both curated set and noisy set, we normalize them using the mean and standard deviation of the extracted log-mel spectrograms in the noisy set. All steps in this section, including silence removal, STFT, mel filtering and log scaling are performed using the *librosa* package.

2.2. Data Generating and Augmentation

In this section, we talk about data generating process and data augmentation during the training phase. As mentioned, each batch contains N_V samples from V and N_W samples from W. Hence there are $N = N_V + N_W$ samples in each batch. One should notice that training CNN requires all samples to have the same dimensionality. However, audio clips in both FSDKaggle2019 and YFCC have various lengths of duration. In our system, we choose each sample to have 192 frames (3 seconds). For audio clips longer than 3 seconds, we randomly extract 192-frame patches from the normalized log-mel spectrogram; for audio clips less than 3 seconds, we pad the spectrogram to 192 frames repeatedly. Thus, each sample in the batch has 192×128 dimensionality.

After getting samples of log-mel spectrograms with the same dimensionalities, we apply *SpecAugment* without time warping [1]. This augmentation approach simply applies m_F frequency masks that each mask sets $f \sim \text{Uniform}(0, p_f)$ consecutive frequency channels to 0, and m_T time masks that each mask sets $t \sim \text{Uniform}(0, p_t)$ time frames to 0.

In addition to *SpecAugment*, we apply a time reverse augmentation. In the batch of augmented log-mel spectrograms, each sample has 50% chance to reverse its entire time steps. Reversed spectrograms no longer represent the original audio signals, therefore we create new classes for them. In this way, our neural network aims to solve a multi-label classification problem with $160 = 2 \times 80$ unique classes. To explain how we create labels with doubled classes, let's imagine mini-version of our task with 3 unique classes only. Suppose the label of an audio clip is $(0, 1, 1)$, then its label in the new system becomes $(0, 1, 1, | \, 0, 0, 0)$, and the label of the time-reversed sample is $(0, 0, 0, | \, 0, 1, 1)$. We concatenate the reversed classes after the original ones in the same order. We will discuss how to make inferences using the system with doubled number of classes in the inference section.

We name the batch with augmented curated samples $\widehat{V}$, and the batch with augmented noisy samples $\widehat{W}$.

2.3. Generating Pseudo-Labels Using Label Cleaning Head

Algorithm 1 shows the procedure to form $V' \times y_{V'}$ and $U' \times y_{U'}$. This directly follows the steps introduced in [9] except the non-linear activation in the multi-task heads. The backbone CNN is a VGGish network with 8 convolutional layers as described in Table 1. As shown in Fig. 1, the backbone extracts hidden features for both label cleaning head g and classification head h. The binary labels of samples in $\widehat{V}$ are fed into the network normally, while the pseudo-labels of samples in $\widehat{W}$ are determined by g using

$$\hat{y}_{\widehat{W}} = \mathbb{1}_{g(f(\widehat{W}))>\eta} \tag{1}$$

where η is a threshold. The label cleaning head g is considered as teacher net that only updates its weights with samples from curated set. On the other hand, the classification head h updates its weights using both curated training set and noisy set. The purpose of having two different heads is to separate the functionalities of the teacher and student nets, so each of them can specialize its own task.

Algorithm 1: algorithm to generate $V' \times y_{V'}$ and $U' \times y_{U'}$ for training the teacher-student network shown in Fig. 1

Input: V, W, y_V, hyperparameters m_f, m_t, p_f and p_t in *SpecAugment*, threshold η to determine the pseudo-labels, Beta distribution parameter α for *mixup*

1. $\widehat{V} = \textit{SpecAugment}(V)$
2. $\widehat{W} = \textit{SpecAugment}(W)$
3. $\hat{y}_{\widehat{W}} = \mathbb{1}_{g(f(\widehat{W}))>\eta}$
4. $\widehat{U} = \text{Concat}(\widehat{V}, \widehat{W})$
5. $y_{\widehat{U}} = \text{Concat}(y_V, \hat{y}_{\widehat{W}})$
6. $V' \times y_{V'} = \textit{mixup}(\widehat{V} \times y_V, \text{Suffle}(\widehat{V} \times y_V))$
7. $U' \times y_{U'} = \textit{mixup}(\widehat{U} \times y_{\widehat{U}}, \text{Suffle}(\widehat{U} \times y_{\widehat{U}}))$

Output: V', U', $y_{V'}$, $y_{U'}$

2.4. *mixup*

In addition to approaches described above, we utilize *mixup* to further generalize our neural network [2]. Since the label cleaning head g only updates its weights from samples of curated set, we do not want *mixup* to "contaminate" its training procedure. We follow [10] to define our *mixup* procedure. For each pair of two samples with their corresponding binary labels $(x_1, y_1), (x_2, y_2)$, *mixup* computes the training pair (x', y') by

$$\begin{aligned} \lambda &\sim \text{Beta}(\alpha, \alpha) \\ \lambda' &= \max(\lambda, 1-\lambda) \\ x' &= \lambda' x_1 + (1-\lambda')x_2 \\ y' &= \lambda' y_1 + (1-\lambda')y_2 \end{aligned} \tag{2}$$

where α is a hyperparameter to determine the Beta distribution. As mentioned previously, the label cleaning head g takes only curated samples while the classification head h takes both curated and nosy samples. To apply *mixup* under such constraints, we first collect augmented samples and their corresponding labels into $\widehat{V}$, $\widehat{W}$, y_V and $\hat{y}_{\widehat{W}}$ as described in section 2.2 and 2.3. Then we combine $\widehat{V}$ and $\widehat{W}$, y_V and $\hat{y}_{\widehat{W}}$ such that $\widehat{U} = \widehat{V} \cup \widehat{W}$ and $y_{\widehat{U}} = y_V \cup \hat{y}_{\widehat{W}}$. After this, we shuffle $\widehat{V} \times y_V$. Together with the original $\widehat{V} \times y_V$, these two sets are served as the sources for *mixup* to form $V' \times y_{V'}$, which is then fed into the label cleaning head g. Similarly, we shuffle $\widehat{U} \times y_{\widehat{U}}$ and form $U' \times y_{U'}$ for the classification head h.

2.5. Loss Function

Instead of cross-entropy loss, *Focal* loss is used to train and make inference for both label cleaning head g and classification head h in our network [3]. Binary *Focal* loss is defined as

$$L_{\text{Focal}} = -\sum_{c=1}^{M}(1-p_c)^{\gamma} y_c \log(p_c) + p_c^{\gamma}(1-y_c)\log(1-p_c) \tag{3}$$

where M is number of unique classes and p_c is the output of the final sigmoid layer of class c. As shown in Fig. 1, we use modified binary *ArcFace* to assist the training of the network in addition to

Stage	Detail
Conv1	3×3, 64, BN, ReLU
Conv2	3×3, 64, BN, ReLU, AvgPool(2)
Conv3	3×3, 128, BN, ReLU
Conv4	3×3, 128, BN, ReLU, AvgPool(2)
Conv5	3×3, 256, BN, ReLU
Conv6	3×3, 256, BN, ReLU, AvgPool(2)
Conv7	3×3, 512, BN, ReLU
Conv8	3×3, 512, BN, ReLU

Table 1: Kernel size, number of neurons, and additional normalization, activation or pooling layers in each stage of the backbone CNN.

Focal loss. The binary *ArcFace* is defined as

$$\begin{aligned} L_{\text{ArcFace}} = &-\sum_{c \in C_{\text{pos}}} \log \frac{1}{1+e^{s(\cos(\theta_{y_c}+m))}} \\ &-\sum_{c' \in C_{\text{neg}}} \log \frac{1}{1+e^{s\cos\theta_{y_{c'}}}} \end{aligned} \tag{4}$$

where C_{pos} and C_{neg} are the sets of positive and negative classes of each sample, respectively. Modifications are made due to the multi-label nature of our task. Please refer [4] for details about other terms such as θ, s and m. Notice that s and m are tunable hyperparameters. Originally, *ArcFace* is a modified cross-entropy loss that aims to enlarge the margins between different classes. The loss is primarily used in face recognition tasks under multi-class classification settings. There are two main reasons we combine *ArcFace* with *Focal* loss in a multi-label classification task.

First, the outputs of g range from 0 and 1, not binary labels as the ground truth labels for curated data. [9] chooses 0.4 to be the threshold to convert the outputs into binary labels without detailed explanation. One of the goals of using *ArcFace* is to make the value of the threshold insensitive. We believe that, by enlarging the margin between positive and negative cases in each class, the binary pseudo-labels generated by g is less likely sensitive to the threshold. And thus the pseudo-labels are more trustworthy. Instead of 0.4, we choose the threshold η to be 0.5, but the difference is minimum. Second, [4] suggests that *ArcFace* applies smaller penalty towards hard samples than other large-margin losses. This makes *ArcFace* favorable in our system because we don't want the network to optimize on incorrect pseudo-labels.

The final loss we used to train the neural network is defined as

$$\begin{aligned} &\sum_{(x_v, y_v) \in V' \times y_{V'}} \Big(L_{\text{Focal}}(g(f(x_v)), y_v) + L_{\text{ArcFace}}(g(f(x_v)), y_v) \Big) \\ + &\sum_{(x_u, y_u) \in U' \times y_{U'}} \Big(L_{\text{Focal}}(h(f(x_u)), y_u) + L_{\text{ArcFace}}(h(f(x_u)), y_u) \Big) \end{aligned} \tag{5}$$

2.6. Network Training

Following [9], we first warm up the backbone CNN in our model by a supervised task using samples from the noisy set. The weights of the backbone CNN is updated iteratively using the augmented batch of noisy samples $\widehat{W}$ and the corresponding noisy labels y_W. The batch size is set to 64. Due to the competition baseline, pretraining

Symbol	Detail	Value
m_f	No. of frequency masks in *SpecAugment*	2
m_t	No. of time step masks in *SpecAugment*	2
p_f	Parameter in the uniform distribution that determines the size of each frequency mask	0.1
p_t	Parameter in the uniform distribution that determines the size of each time step mask	0.1
η	Threshold to determine pseudo-labels	0.5
α	Parameter to determine the Beta distribution used in *mixup*	0.4
s	Hyperparameter used in ArcFace to scale up normalized weight matrix	30
m	Hyperparameter used in ArcFace to create large margin	0.5
N_V	Batch size of curated set V	6
N_W	Batch size of noisy set W	58
w_1	No. of epochs that takes the learning rate to ramp up in training stage 1	10
w_2	No. of epochs that takes the learning rate to ramp up in training stage 2	15
l_1	No. of epochs that takes the learning rate to be annealed to 0 in training stage 1	90
l_2	No. of epochs that takes the learning rate to be annealed to 0 in training stage 2	285

Table 2: Symbol, detail explanation and value of each hyperparameter used in both training stages

models using the noisy set only is considered as a way to battle the domain mismatch.

At the second stage of training, we train the backbone, g and h jointly using the loss defined in (5). We set $N_V = 6$ and $N_W = 58$ so the batch size of each iteration is still $N_V + N_W = 64$. In both stages of training, we use Nadam as the gradient optimizer, and Cosine Annealing Learning Rate with Warm Start as the learning rate scheduler [11] [12]. Specifically, we first linearly ramp up the learning rate from 0 to 0.0035 in w epochs, then gradually anneal the learning rate to 0 in l epochs. We set w and l differently in the first and second stage of training. A detailed list of hyperparameters is given in Table 2.

2.7. Inference

The inference step is performed using Kaggle kernel. Data process and inference have to be finished in 1 hour if GPU is used. To accelerate the procedure, we use GPU-enabled *torchaudio* package to remove silence and convert the audio files in test set to log-mel spectrogram. We do not need the label cleaning head g and the weights used to compute *ArcFace* in h. Therefore, the inference system keeps only the backbone and the classification head h with the weights for computing *Focal* loss. It reduces the total number of parameters in the system.

Unlike the data generating procedure during the training phase, we extract patches using a sliding window with hop size of 16 frames (0.25s). For audio clips less than 3s, we again pad them to 192 frames repeatedly. The typical way to make the final prediction for each audio clip is to take the average (arithmetic or geometric) of all patches extracted from the clip. However, we could only extract small number of patches from video clips with short duration. This leads to less generalized predictions of short video clips. To deal with this, we apply test-time augmentations using *SpecAugment* to audio clips with no more than 7 patches extracted using sliding window so that at least 10 patches are generated for each short clip. This is implemented by repeat the sliding window extraction k more times with *SpecAugment*, where k is defined as

$$k = \begin{cases} 9 & \text{if no. of patches} = 1 \\ 5 & \text{if no. of patches} = 2 \\ 3 & \text{if no. of patches} = 3 \\ 2 & \text{if no. of patches} = 4 \\ 2 & \text{if no. of patches} = 5 \\ 1 & \text{if no. of patches} = 6 \\ 1 & \text{if no. of patches} = 7 \end{cases} \tag{6}$$

After collecting the augmented batch $\widehat{Z}$, we reverse the time steps of every log-mel spectrogram patch in it, and name the time-reversed batch $\tilde{Z}$. The prediction of each patch $x_z \in \widehat{Z}$ is made by the arithmetic average of the x_z's sigmoid output of the original 80 classes and $\tilde{x}_z$'s sigmoid output of the reversed 80 classes, where $\tilde{x}_z \in \tilde{Z}$.

3. ENSEMBLE AND RESULT

In section 2 we describe the approach used to build a single system. The single system generates the submission file reaches 0.712 lwlrap on public test set. Two more systems use ensemble to further improve the performance. The details about each system are shown below

- **Zhang_BIsmart_task2_1.output.csv:** 5 CV averaging + 7 fine-tuned systems based on public Kaggle kernels. The hop size of sliding window for inference is 0.4s. The ensemble system reaches 0.730 lwlrap on public test set.
- **Zhang_BIsmart_task2_2.output.csv:** The system above plus a single system trained on the whole training set. The hop size of sliding window for inference is 0.4s. The ensemble system reaches 0.729 lwlrap on public test set.
- **Zhang_BIsmart_task2_3.output.csv**: Single system trained on the entire training set. The hop size of sliding window for inference is 0.25s. The system reaches 0.712 lwlrap on public test set. This submission is used as the candidate for the Judges' Award.

4. ACKNOWLEDGEMENT

I'd like to give special thanks to researchers at CVSSP, University of Surrey, UK. Thanks to their well-written baseline system [13], I could easily make modifications and experiment with any kind of hypotheses. Also thank Freesound for hosting the competition and providing such interesting dataset.

5. REFERENCES

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

[2] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[3] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[4] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," *arXiv preprint arXiv:1801.07698*, 2018.

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

[6] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio set: An ontology and human-labeled dataset for audio events," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 776–780.

[7] "The ins and outs of the yahoo flickr creative commons 100 million dataset," https://code.flickr.net/2014/10/15/the-ins-and-outs-of-the-yahoo-flickr-100-million-creative-commons-dataset/.

[8] "General-purpose audio tagging of freesound content with audioset labels," http://dcase.community/challenge2018/task-general-purpose-audio-tagging-results/.

[9] M. Hu, H. Han, S. Shan, and X. Chen, "Multi-label learning from noisy labels with non-linear feature transformation," in *Asian Conference on Computer Vision*. Springer, 2018, pp. 404–419.

[10] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. Raffel, "Mixmatch: A holistic approach to semi-supervised learning," *arXiv preprint arXiv:1905.02249*, 2019.

[11] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[12] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[13] Q. Kong, Y. Cao, T. Iqbal, Y. Xu, W. Wang, and M. D. Plumbley, "Cross-task learning for audio tagging, sound event detection and spatial localization: Dcase 2019 baseline systems," *arXiv preprint arXiv:1904.03476*, 2019.