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 tagging system. To facilitate the description, we introduce some notations.

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 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. Adaptive test time augmentation (TTA) based on the lengths of audio samples is used as a final approach to improve the performance. We choose a single system that generates the submission file ZhangBIsmart_task2_3.output.csv to be the candidate model considered for the Judges’ Award. Other two systems use ensemble approach to further improve the performance.

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 Nv labeled samples from FSDKaggle2019 with corresponding binary targets yV and a batch W of NW samples from YFCC, our Teacher-Student network produces a processed batch of augmented labeled samples ˆV and a batch of augmented samples ˆW with pseudo-labels ˆyW guessed by the network. Then we apply mixup to create V′ × yV′ and U′ × yU′ that are used to compute the multi-task loss term described below. We describe each part of the process in the following subsections.
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 \( 192 \times 128 \) 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 \((0, 1, 1) \oplus (0, 0, 0)\), and the label of the time-reversed sample is \((0, 0, 0) \oplus (0, 1, 1)\). 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 \( \hat{V} \), and the batch with augmented noisy samples \( \hat{W} \).

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 \( \hat{V} \) are fed into the network normally, while the pseudo-labels of samples in \( \hat{W} \) are determined by \( g \) using

\[
\hat{y}_{\hat{W}} = \mathbb{1}_{g(f(\hat{W})) > \eta}
\]

where \( \eta \) 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

<table>
<thead>
<tr>
<th>Stage</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv1</td>
<td>$3 \times 3, 64$, BN, ReLU</td>
</tr>
<tr>
<td>Conv2</td>
<td>$3 \times 3, 64$, BN, ReLU, AvgPool(2)</td>
</tr>
<tr>
<td>Conv3</td>
<td>$3 \times 3, 128$, BN, ReLU</td>
</tr>
<tr>
<td>Conv4</td>
<td>$3 \times 3, 128$, BN, ReLU, AvgPool(2)</td>
</tr>
<tr>
<td>Conv5</td>
<td>$3 \times 3, 256$, BN, ReLU</td>
</tr>
<tr>
<td>Conv6</td>
<td>$3 \times 3, 256$, BN, ReLU, AvgPool(2)</td>
</tr>
<tr>
<td>Conv7</td>
<td>$3 \times 3, 512$, BN, ReLU</td>
</tr>
<tr>
<td>Conv8</td>
<td>$3 \times 3, 512$, BN, ReLU</td>
</tr>
</tbody>
</table>

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

$$L_{\text{ArcFace}} = - \sum_{c \in C_{\text{pos}}} \log \frac{1}{1 + e^{\gamma (c_{\text{y}_c} + m_c)}}$$

(4)

where $C_{\text{pos}}$ and $C_{\text{neg}}$ 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 $\theta$, $s$, 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 $\eta$ 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

$$\sum_{(x_u, y_u) \in V' \times y_{V'}} \left( L_{\text{Focal}}(g(f(x_u)), y_u) + L_{\text{ArcFace}}(g(f(x_u)), y_u) \right)$$

$$+ \sum_{(x_u, y_u) \in U' \times y_{U'}} \left( L_{\text{Focal}}(h(f(x_u)), y_u) + L_{\text{ArcFace}}(h(f(x_u)), y_u) \right)$$

(5)

### 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 corresponding binary labels $(x_1, y_1), (x_2, y_2)$, mixup computes the training pair $(x', y')$ by

$$\lambda \sim \text{Beta}(\alpha, \alpha)$$

$$\lambda' = \max(\lambda, 1 - \lambda)$$

$$x' = \lambda x_1 + (1 - \lambda') x_2$$

$$y' = \lambda y_1 + (1 - \lambda') y_2$$

(2)

where $\alpha$ is a hyperparameter to determine the Beta distribution. As mentioned previously, the label cleaning head $h$ takes only curated samples while the classification head $h$ takes both curated and noisy samples. To apply mixup under such constraints, we first collect augmented samples and their corresponding labels into $V$, $W$, $y_V$ and $y_W$ as described in section 2.2 and 2.3. Then we combine $V$ and $W$, $y_V$ and $y_W$ such that $U = V \cup W$ and $y_U = y_V \cup y_W$. After this, we shuffle $V \times y_V$. Together with the original $V \times y_V$, 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 $U \times y_U$ 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

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

(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...
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 \texttt{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 \texttt{ArcFace} in \( h \). Therefore, the inference system keeps only the backbone and the classification head \( h \) with the weights for computing \texttt{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 of 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 \texttt{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 \( m \) times with \texttt{SpecAugment}, where \( m \) is defined as

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

After collecting the augmented batch \( \hat{Z} \), 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 \( x_z \in \tilde{Z} \) 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_1output.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_2output.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_3output.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


