STACKED CONVOLUTIONAL NEURAL NETWORKS FOR GENERAL-PURPOSE AUDIO TAGGING

Technical Report

Turab Iqbal, Qiuqiang Kong, Mark D. Plumbley, Wenwu Wang

Centre for Vision, Speech and Signal Processing, University of Surrey
{t.iqbal, q.kong, m.plumbley, w.wang}@surrey.ac.uk

ABSTRACT

This technical report describes the methods used for classifying sound events as part of Task 2 of the DCASE 2018 challenge. The data used in this task requires a number of considerations, including how to handle variable-length audio samples and the presence of noisy labels. We propose a number of neural network architectures that learn from mel-spectrogram inputs. These baseline models involve the use of preprocessing techniques, data augmentation, and pseudo-labeling in order to improve their performance. They are then ensembled using a popular technique known as stacking. On the test set used for evaluation, compared to the baseline mean average precision score of 0.694, our system achieved a score of 0.951.

***Index Terms*—** Audio classification, convolutional network, recurrent network, deep learning, data augmentation, stacking

1. INTRODUCTION

Audio classification is the task of identifying sounds of interest in a given audio signal, where the sounds to be detected are typically defined in advance. The Detection and Classification of Acoustic Scenes and Events (DCASE) [1] is a recurring challenge with several tasks pertaining to the classification of audio. This paper describes the system we used to participate in DCASE 2018, Task 2 [2].

In Task 2, participants are provided with a dataset developed by the Freesound initiative [3]. This dataset contains a variety of classes, including domestic sounds, warning sounds, and various musical instruments. It is comprised of 41 classes, and each audio clip is associated with exactly one class; therefore, it is a single-label classification problem. There is a labeled training set of 9473 audio clips and a test set of 9400 audio clips. The availability of labels allows supervised machine learning methods to be used.

In recent times, the state-of-the-art in machine learning has come from research in neural networks [4]. Audio classification is no exception, with many of the top submissions in DCASE challenges utilizing such architectures [5, 6, 7, 8]. We follow this trend and use two types of neural networks: a convolutional neural network (CNN) and a convolutional-recurrent neural network (CRNN). Two variants of each type are used, giving four architectures in total. The purpose of training multiple models is to use an ensembling method to combine their predictions. The rationale is that the strengths of each model can be consolidated. To achieve this, we use a popular technique called stacking [9].

Although the training examples are labeled, only $40\,\%$ of them are manually verified. It is assumed that approximately $30\,\%$ to $35\,\%$ of the unverified labels are incorrect. This problem can be formulated as a form of label noise, for which there is considerable research in the field. These wrongly-labeled examples can negatively affect the performance of the system and should be handled appropriately. Our method is rather simple, and relies on a combination of sample weighting and pseudo-labeling.

Other than the unverified labels, there are a number of other points to note. Firstly, the lengths of the audio clips vary greatly, from $0.3\,\mathrm{s}$ to $30\,\mathrm{s}$. This is a problem because many training models expect a fixed-length input. Another consideration is the saliency of the inputs; does the entire clip contain important information or is it only specific parts? We address both these issues later in the paper and present our findings.

The rest of this paper is organized as follows. In Section 2, the preprocessing and feature extraction methods are described. In Section 3, the neural network architectures, training methodology, and ensembling algorithm are presented. The results are then given in Section 4. Finally, we summarize in Section 5.

2. PREPROCESSING AND FEATURE EXTRACTION

Prior to training, we applied preprocessing to the inputs followed by feature extraction. As mentioned earlier, the audio clips may contain sections that are uninformative. Upon analyzing the clips, although there was very little background noise, some of them contained sections of silence. Long sequences of silence were deemed to be unhelpful, and were therefore detected and removed. More precisely, the non-silent sections of the signal were extracted and considered as separate inputs. Silence was detected by segmenting the audio signal into frames and thresholding the root mean square (RMS) energy of the frames. We required the length of the silence between separate sections to be at least $500\,\mathrm{ms}$. Moreover, $500\,\mathrm{ms}$ of "silence" was retained at the beginning and end of each section to prevent over-cropping of information. Refer to Figure 1.

In terms of the silence threshold, we chose two values: a default value of $-48\,\mathrm{dBFS}$ and a less-aggressive value of $-56\,\mathrm{dBFS}$. We used the latter for transient sounds, as the default was found to over-crop the sounds. Note that neither value is supposed to match the threshold of hearing; it would be more correct to say that it detects quiet parts of the audio that are likely unimportant.

After silence removal, the extracted non-silent sections, which are now considered as separate inputs, were downsampled from $44.1\,\mathrm{kHz}$ to $32\,\mathrm{kHz}$ and transformed into log-scaled mel-frequency (log-mel) spectrograms. Log-mel features are a popular choice for classification using neural networks, as they benefit from the additional information that is retained relative to mel-frequency cepstrum coefficients. As shown in Table 1, two sets of parameters were used to capture multiple resolutions.

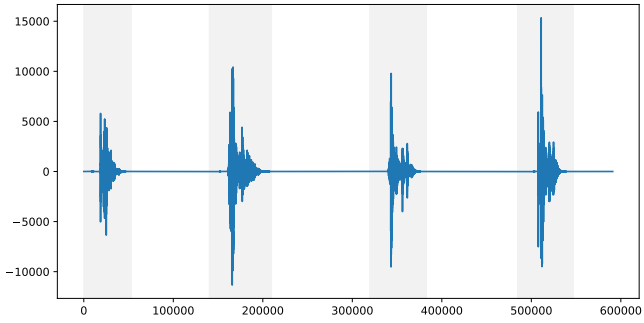

Figure 1: Illustration of the silence removal process on the file "071e836c.wav", which is 13 s in length. The non-silent sections that are extracted are highlighted in gray. The length of silence between sections must be at least 500 ms.

Table 1: Log-mel spectrogram parameters

Parameter	Configuration A	Configuration B
Sample rate	32 000 Hz	32 000 Hz
Window size	1024	512
Hop size	512	256
Mel bands	64	64

Following feature extraction, each feature vector was split into chunks of a fixed size, which resolves the problem that audio clips vary in length. We chose a chunk size of 128×64, where the first axis is the temporal dimension. This corresponds to 2 s chunks and 1 s chunks for configurations A and B, respectively (cf. Table 1). In our system, the chunks were non-overlapping, as this gave better results. When the length of the feature vector was less than the chunk length, it was padded. When it was greater, but not evenly divisible by the chunk length, an additional chunk was added to align with the end of the feature vector to include the remainder.

3. TRAINING AND INFERENCE

In order to utilize the training set that was provided for this task, we used two types of neural networks: CNNs and CRNNs. For each type, we also used two variants: one using standard convolutions and another using gated convolutions. Since there are two log-mel configurations, this gives eight training models in total. In the subsections to follow, we describe the architectures, the training methods, and ensembling algorithm.

3.1. Neural Network Architectures

The neural network architectures are outlined in Table 3. Beginning with the standard CNN, it is essentially equivalent to the VGG13 network proposed in [10], hence the name. Each convolutional block consists of two convolutional layers followed by a max pooling layer that halves each spatial dimension. After each convolution, which uses a rectified (ReLU) activation function [11], batch normalization [12] is applied as a form of regularization. Following the convolutional blocks, each channel is averaged to a scalar value. Finally, a softmax layer is used to generate the predictions.

The CRNN architecture is an extension of VGG13. Instead of averaging across both spatial dimensions after the convolutions, only the frequency dimension is averaged initially. A bidirectional recurrent layer [13] is then applied to output a feature vector for each time step, and these feature vectors are then averaged. By using a recurrent layer, the temporal dynamics of the input can be learned. Indeed, our experiments suggested that it helped.

Table 2: Training parameters

Parameter	Value
Batch size	128
Learning rate (LR)	0.0005
LR decay factor	0.9
LR decay rate	2

The other two architectures are GCNN and GCRNN, which are variants of VGG13 and CRNN, respectively. The difference is that each convolutional layer is replaced with a gated convolutional layer [14]. The idea of a gated layer is inspired by the gating mechanisms found in recurrent neural networks [15, 16], and is used to control the information that is propagated to deeper layers. This mechanism has been shown to produce good results for similar tasks [7], and it has worked well for this task too.

To train the various models, the training set was split into five cross-validation folds, ensuring that there was a similar number of verified examples in each fold. The cross-entropy function was used as the training loss and Adam [17] was used as the gradient descent algorithm. Refer to Table 2 for the values of the free parameters. Decay rate is the number of epochs until the learning rate is decayed.

In terms of generating the predictions, the top four epochs were selected based on performance on the validation set. The metric used was the mean average precision (MAP) score. Recalling that the inputs to the neural networks are chunks, and that the chunks are from sections of the original audio clip (cf. Section 2), the chunk predictions need to be merged to produce clip-level predictions. This was achieved using the geometric mean, as this is less sensitive to outliers than the arithmetic mean. With the clip-level predictions, the top four epochs were merged using the arithmetic mean.

3.2. Learning from Noisy Labels

To account for the unverified labels, we experimented with a number of techniques. The first, known as pseudo-labeling, was to relabel the unverified examples automatically using a previously-trained classifier. For pseudo-labeling to be effective, the error rate of the classifier should be lower than the noise rate of the original labels, which was stated to be around 30 %. We used the same system described in this paper as the relabeling classifier, except that it was trained with the original labels.

The second technique was to weight the training loss function for examples that were unverified. If an example is incorrectly labeled, the computed loss will be incorrect and should be disregarded. Of course, we do not know whether it is correct or not if the label is unverified. Therefore, we chose to use a weight, $w \in (0, 1)$, that would at least lower the magnitude of the loss. This was set to 0.7 for the original labels and 0.9 with pseudo-labeling.

The final technique, similar to pseudo-labeling, was to "promote" examples from unverified to verified if a previously-trained classifier agreed with the labels with high confidence. We used a confidence threshold of 0.7 to promote examples. As this has no effect on its own, it needs to be used in tandem with the second technique.

Table 3: Description of the neural network architectures. The parameters used in the convolutional blocks are encapsulated by square brackets. The first two parameters in each line are the kernel size and the number of filters. "BN" refers to batch normalization. "GLU" refers to the use of gated linear units, as described in [14]. "Bi-GRU" refers to using bidirectional gated recurrent units [16].

Feature Size	VGG13	CRNN	GCNN	GCRNN
128×64	Log mel spectrogram			
64×32	$\begin{bmatrix} 3 \times 3, 64, \text{BN}, \text{ReLU} \\ 3 \times 3, 64, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 64, \text{BN}, \text{ReLU} \\ 3 \times 3, 64, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 64, \text{BN}, \text{GLU} \\ 3 \times 3, 64, \text{BN}, \text{GLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 64, \text{BN}, \text{GLU} \\ 3 \times 3, 64, \text{BN}, \text{GLU} \end{bmatrix}$
	2x2 Max Pooling			
32×16	$\begin{bmatrix} 3 \times 3, 128, \text{BN}, \text{ReLU} \\ 3 \times 3, 128, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 128, \text{BN}, \text{ReLU} \\ 3 \times 3, 128, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 128, \text{BN}, \text{GLU} \\ 3 \times 3, 128, \text{BN}, \text{GLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 128, \text{BN}, \text{GLU} \\ 3 \times 3, 128, \text{BN}, \text{GLU} \end{bmatrix}$
	2x2 Max Pooling			
16×8	$\begin{bmatrix} 3 \times 3, 256, \text{BN}, \text{ReLU} \\ 3 \times 3, 256, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 256, \text{BN}, \text{ReLU} \\ 3 \times 3, 256, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 256, \text{BN}, \text{GLU} \\ 3 \times 3, 256, \text{BN}, \text{GLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 256, \text{BN}, \text{GLU} \\ 3 \times 3, 256, \text{BN}, \text{GLU} \end{bmatrix}$
	2x2 Max Pooling			
8×4	$\begin{bmatrix} 3 \times 3, 512, \text{BN}, \text{ReLU} \\ 3 \times 3, 512, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 512, \text{BN}, \text{ReLU} \\ 3 \times 3, 512, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 512, \text{BN}, \text{GLU} \\ 3 \times 3, 512, \text{BN}, \text{GLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 512, \text{BN}, \text{GLU} \\ 3 \times 3, 512, \text{BN}, \text{GLU} \end{bmatrix}$
	2x2 Max Pooling			
4×2	$\begin{bmatrix} 3 \times 3, 512, \text{BN}, \text{ReLU} \\ 3 \times 3, 512, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 512, \text{BN}, \text{ReLU} \\ 3 \times 3, 512, \text{BN}, \text{ReLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 512, \text{BN}, \text{GLU} \\ 3 \times 3, 512, \text{BN}, \text{GLU} \end{bmatrix}$	$\begin{bmatrix} 3 \times 3, 512, \text{BN}, \text{GLU} \\ 3 \times 3, 512, \text{BN}, \text{GLU} \end{bmatrix}$
	-	Bi-GRU, 512, ReLU	-	Bi-GRU, 512, ReLU
	Global Average Pooling			
	Softmax (41 Classes)			

3.3. Data Augmentation

To reduce overfitting during training, data augmentation is a popular approach. We used a method called mixup [18] to achieve this. Mixup operates on the fly by randomly mixing a pair of inputs and their associated target values. Consider a pair of inputs, x_1 and x_2, and their one-hot-encoded target values, y_1 and y_2. To mix these, a parameter, $\lambda \in (0, 1)$, is used to create convex combinations.

$$x = \lambda x_1 + (1 - \lambda) x_2. \tag{1}$$

$$y = \lambda y_1 + (1 - \lambda) y_2. \tag{2}$$

The output, x, y, is then used as the training example rather than the original examples. In our system, the parameter λ was a random variable from the Beta distribution $\mathrm{B}(1.0, 1.0)$, and a different value was used for each mixing pair. Note that because we used sample weights (cf. Section 3.2), they were also mixed in the same manner.

3.4. Ensembling

To combine the predictions of the different models, we used a method known as stacking [9, 19, 20]. In this method, the base model predictions are used as features to train a second-level classifier. The output of a base model is an $N \times K$ vector of probabilities, where N is the number of data samples and $K = 41$ is the number of classes. By concatenating the outputs of the models, the result is an $N \times 8K$ vector; this is the input of the new classifier.

As the validation sets constitute the training set, the validation set predictions were used to generate the features for the training set. Similarly, the test set predictions were used as the test set features. We used logistic regression with an L_2 penalty as the second-level classifier. It was configured to use class weights to compensate for class imbalance and sample weights as described in Section 3.2.

4. RESULTS

To assess the performance of our system, we evaluated the training set and test set predictions. The former is possible because we used cross-validation folds. We also only evaluated the manually-verified training examples. The metric used to assess the performance is the mean average precision ($\mathrm{MAP@3}$) score, which is defined as

$$\mathrm{MAP@3} = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{\min\{K,3\}} P(i), \tag{3}$$

where N is the number of data samples, $K = 41$ is the number of classes, and $P(i)$ is the precision at cutoff i.

The results for the training set are shown in Table 4. The systems that are compared are the single models (log-mel configuration A only), an arithmetic-mean ensemble of the models, and the stacked ensemble described in the previous section. It can be seen that the mean ensemble performs much better than all of the single models – by almost $2\,\%$. However, it is the stacked ensemble that performs the best, with a $\mathrm{MAP@3}$ score of 0.972. The weight-learning capability of stacking, with respect to model and class, appears to help.

In Table 5, the results for the test set are presented. This is for the private subset of the dataset. We look at the 8-model stacked ensemble compared to a smaller 4-model version. In the latter, the VGG13 and CRNN architectures are omitted. The results of both

Table 4: Training set results. Only the results of configuration A models are given to highlight the difference in architectures.

Model	MAP@3
VGG13	0.950
GCNN	0.951
CRNN	0.952
GCRNN	0.958
Arithmetic Mean	0.968
Stacking	0.972

Table 5: Test set results comparing the 8-model stacked ensemble with a 4-model version that excludes VGG13 and CRNN models.

Model	MAP@3
4-Model Stacking	0.948
8-Model Stacking	0.951

systems are far superior to the competition's baseline system, which scored 0.694. Although the additional models in the 8-model version help, the difference is minor. This can be explained by the lack of diversity that the omitted models have to offer.

5. CONCLUSION

This report described a system used to participate in Task 2 of the DCASE 2018 challenge. A number of steps were involved in feature extraction, including silence removal, computing mel-spectrograms, and splitting the feature vectors into chunks. We used eight neural network models and combined their predictions using stacking. To ameliorate the effects of label noise, a combination of loss function weighting and pseudo-labeling was used. To reduce overfitting, we used a data augmentation technique called mixup. On the test set, the system achieved a mean average precision score of 0.951, which surpassed the baseline by a large margin.

6. ACKNOWLEDGMENT

This research was supported by the EPSRC grant EP/N014111/1, "Making Sense of Sounds", and a Research Scholarship from the China Scholarship Council (CSC), No. 201406150082. We would also like to thank Yong Xu for his contributions in the early stages of the competition and the relevant work he did in previous challenges.

7. REFERENCES

[1] D. Giannoulis, E. Benetos, D. Stowell, M. Rossignol, M. Lagrange, and M. D. Plumbley, "Detection and classification of acoustic scenes and events: An IEEE AASP challenge," in *2013 IEEE Workshop Appl. Signal Process. Audio, Acoustics (WASPAA)*, New Paltz, NY, 2013, pp. 1–4.

[2] E. Fonseca, M. Plakal, F. Font, D. P. W. Ellis, X. Favory, J. Pons, and X. Serra, "General-purpose tagging of Freesound audio with AudioSet labels: Task description, dataset, and baseline," *arXiv preprint arXiv:1807.09902*, 2018.

[3] E. Fonseca, J. Pons, X. Favory, F. Font, D. Bogdanov, A. Ferraro, S. Oramas, A. Porter, and X. Serra, "Freesound datasets: A platform for the creation of open audio datasets," in *Proc. 18th Int. Soc. Music Inf. Retrieval Conf. (ISMIR)*, Suzhou, China, 2017, pp. 486–493.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] S. Mun, S. Park, D. K. Han, and H. Ko, "Generative adversarial network based acoustic scene training set augmentation and selection using SVM hyper-plane," in *Proc. Detection Classification Acoust. Scenes Events 2017 Workshop (DCASE)*, Munich, Germany, Sep. 2017, pp. 93–102.

[6] H. Lim, J. Park, and Y. Han, "Rare sound event detection using 1D convolutional recurrent neural networks," in *Proc. Detection Classification Acoust. Scenes Events 2017 Workshop (DCASE)*, Munich, Germany, Sep. 2017, pp. 80–84.

[7] Y. Xu, Q. Kong, W. Wang, and M. D. Plumbley, "Large-scale weakly supervised audio classification using gated convolutional neural network," *arXiv preprint arXiv:1710.00343*, 2017.

[8] D. Lee, S. Lee, Y. Han, and K. Lee, "Ensemble of convolutional neural networks for weakly-supervised sound event detection using multiple scale input," in *Proc. Detection Classification Acoust. Scenes Events 2017 Workshop (DCASE)*, Munich, Germany, Sep. 2017, pp. 74–79.

[9] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, pp. 241–259, 1992.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd Int. Conf. Learn. Repr. (ICLR)*, San Diego, CA, 2015.

[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, Haifa, Israel, 2010, pp. 807–814.

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, Lille, France, 2015, pp. 448–456.

[13] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[14] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, vol. 70, Sydney, Australia, 2017, pp. 933–941.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd Int. Conf. Learn. Repr. (ICLR)*, San Diego, CA, 2015.

[18] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *6th Int. Conf. Learn. Repr. (ICLR)*, Vancouver, Canada, 2015.

[19] K. M. Ting and I. H. Witten, "Issues in stacked generalization," *J. Artif. Int. Res.*, vol. 10, no. 1, pp. 271–289, 1999.

[20] M. J. van der Laan, E. C. Polley, and A. E. Hubbard, "Super learner," *Statistical applications in genetics and molecular biology*, vol. 6, no. 1, 2007.