A FIRST ATTEMPT AT POLYPHONIC SOUND EVENT DETECTION USING CONNECTIONIST TEMPORAL CLASSIFICATION

Yun Wang and Florian Metze

Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA, U.S.A.
{yunwang, fmetze}@cs.cmu.edu

ABSTRACT

Sound event detection is the task of detecting the type, starting time, and ending time of sound events in audio streams. Recently, recurrent neural networks (RNNs) have become the mainstream solution for sound event detection. Because RNNs make a prediction at every frame, it is necessary to provide exact starting and ending times of the sound events in the training data, making data annotation an extremely time-consuming process. Connectionist temporal classification (CTC), as a sequence-to-sequence model, can relax this constraint, because it suffices to provide ordered sequences of sound events without exact starting and ending times.

This paper presents a first attempt at using CTC for sound event detection. In the polyphonic situation, sound events may overlap with each other, making it hard to define ordered sequences of sound events. We propose to use the boundaries (*i.e.* starts and ends) of the sound events as tokens for CTC. We show that CTC is able to locate the boundaries of sound events on a very noisy corpus of consumer generated content with rough hints about their positions. The CTC approach seems to be particularly suited to detecting short and transient sounds, which have traditionally been hardest to detect.

Index Terms— Sound event detection (SED), recurrent neural networks (RNN), connectionist temporal classification (CTC)

1. INTRODUCTION

Sound event detection (SED) is the task of detecting the type, starting time, and ending time of sound events in audio. Example sound events include car engine, cat meows, and footsteps. They can be produced by difference sources, can be either long-lasting or transient, and can either be stationary or have a temporal structure. In real-life recordings, sound events often overlap with each other ("polyphonic"), which adds to the difficulty of detecting them.

SED can be useful for a number of purposes. It can be used to understand the content of consumer videos without adequate annotation from their uploaders, so they can be indexed and searched [1]. It can also be used to detecting anomalous incidents (such as screaming) in public places and facilities (*e.g.* subway trains) [2].

To solve the task of SED, hidden Markov models (HMMs) have been used to model the temporal structure of sound events [3, 4, 5]. Non-negative matrix factorization (NMF) has also been used to deal with polyphony [6, 7, 8, 9]. With the popularity of deep learning in the past few years, deep neural networks (DNNs) [10, 11, 12] have become the mainstream solution to SED, followed by convolutional neural networks (CNNs) [13, 14, 15, 16] and recurrent neural networks (RNNs) [17, 18, 19, 20]. Equipped with long short-term memory (LSTM) cells [21], RNNs are able to exploit the internal temporal structure of sound events as well as the co-occurrence pattern between them; the strong fitting power of RNNs also makes them more robust to overlapping sound events.

Despite the strengths of RNNs, the predictions they make are still on a frame-by-frame basis. As a result, the training labels must also include the exact starting and ending times of each sound event instance. This makes data annotation a formidably tedious process.

We argue that sequence-to-sequence models, such as connectionist temporal classification (CTC) [22], may offer an elegant solution. In CTC, the supervision is provided as ordered sequences of tokens, instead of frame-wise labels. The objective function is based on the total probability of the token sequence, summed over all possible alignments (*i.e.* starting and ending times of the sound events). This not only reduces the workload of data annotation, but also opens up the possibility of automatically generating sequences of sound events from textual descriptions. Even though the exact timings of sound events may not be provided with the training data, CTC models should be able to figure out these timings and generate a probability peak for each token, thereby completing the SED task.

The temporal structure of sound events is considerably more diverse than that of speech or handwriting data; also, sound data is more noisy than speech data, and available in smaller amounts only. It is not clear if CTC training is practical on such data. This paper presents a first attempt at polyphonic sound event detection using CTC. As sound events often overlap with each other, it is hard to define ordered sequences of sound events themselves. Instead, we propose to predict the *boundaries* of sound events, *i.e.* their starts and ends. In order to speed up the training process, we pre-train the CTC model with a bidirectional LSTM-RNN that performs frame-wise prediction, and clip the gradients while training the CTC model. To guide the CTC model to discover the correct positions of the tokens, we use the timing information in the annotation as rough hints. Experiments show that the CTC model is able to locate the boundaries of most sound events in the training data, but more data is probably needed for good generalization to the test data.

2. THE BASICS OF CTC

Connectionist temporal classification (CTC) has achieved great success in tasks speech recognition [23, 24]. A main contribution of CTC is eliminating the need of phoneme alignments (*i.e.* the starting and ending times of each phoneme) during training, so the probability of the phoneme sequences could be directly maximized.

In essence, CTC is a new way of defining the objective function for RNNs. An RNN predicts a probability $y_t(k)$ for each token k in the output vocabulary at time t. Let z_t be the ground-truth token at time t, then the traditional objective function for a sequence of length T is $L = -\sum_{t=1}^{T} \log y_t(z_t)$, which is actually the negative

This was was supported in part by a gift award from Robert Bosch LLC.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by NSF grant number OCI-1053575.

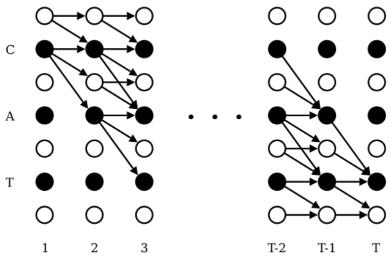

Fig. 1. *Lattice for computing the CTC objective function (taken from [22]). Black circles represent non-blank tokens, and white circles represent blanks. Arrows signify allowed transitions.*

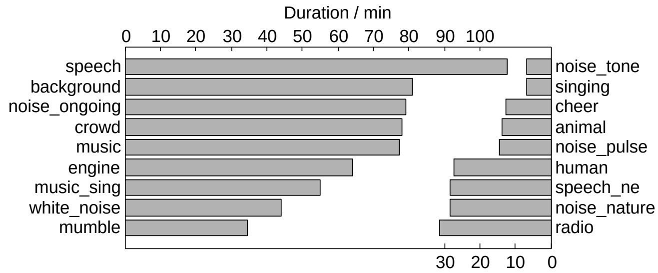

Fig. 2. *Duration of each sound event type in the "noiseme" corpus. "Background" means no sound event is active. "Speech_ne" stands for "non-English speech".*

logarithm of the joint probability of the desired token sequence and the alignment. Often, we are only interested in the token sequence, and a ground-truth alignment may not be available. Therefore we want to marginalize out the alignment.

CTC conducts the marginalization in the following way. First, it adds a "blank" token (denoted by "–") to the output vocabulary. Then, it defines a many-to-one mapping function that transforms an alignment (i.e. the sequence of output tokens at each time step, also called a *path*) to a token sequence. The mapping function first reduces adjacent repeating tokens to a single one, and then removes the "blank" tokens. For example, the paths `CC-A-TT-` and `-CAAA--T` both map to the token sequence `CAT`.

The objective function is defined as the negative logarithm of the total probability of all paths that map to the ground-truth token sequence. This total probability can be found using dynamic programming on the lattice shown in Fig. 1. On the x-axis are time steps, while on the y-axis is a "modified token sequence" – the desired token sequence with blank tokens inserted between every pair of tokens and at both ends. Let L be the length of the modified token sequence, and l_i be its i-th token. A valid path may start at either l_1 or l_2, and may end at either the l_{L-1} or l_L. At each time step, the path may stay at the same token, transition to the next token, or transition to the token after the next provided it is a non-blank token different from the current one. Let $\alpha_t(i)$ be the total probability of partial paths that land on the l_i at time t. Assuming conditional independence between $y_t(k)$ across time steps given the state of the hidden layers, the α's may be computed as follows:

$$\alpha_1(i) = \begin{cases} y_1(l_i) & i \le 2 \\ 0 & i > 2 \end{cases} \tag{1}$$

$$\alpha_t(i) = [\alpha_{t-1}(i) + \alpha_{t-1}(i-1) + \delta_i \alpha_{t-1}(i-2)] y_t(l_i), t > 1 \tag{2}$$

where $\delta_i = 1$ iff $l_i \neq l_{i-2}$, and terms that go past the start of the modified token sequence are zero. The total probability of paths that map to the original token sequence is given by $\alpha_T(L-1) + \alpha_T(L)$, whose negative logarithm is the CTC objective function.

For a derivation of the gradient of the CTC objective function w.r.t. the network outputs, the reader is referred to [22]. However, such derivation by hand is not necessary, given the symbolic derivation functionality of deep learning toolkits such as Theano [25].

There are several ways to decode the output of CTC model. The simplest method is to select the token with the maximum probability at each frame, reduce adjacent repeating tokens to a single one, and remove the blank tokens. This method, called *best path decoding*, finds the most probable *path*. In order to find the most probable *output sequence*, whose probability can be the sum of the probabilities of multiple paths, *prefix search decoding* can be used. See [22] for more details.

3. CTC EXPERIMENTS FOR SOUND EVENT DETECTION

3.1. Corpus, Feature Extraction, and Network Setup

We conducted sound event detection experiments on the "noiseme" corpus [26]. The original corpus contains the audio tracks of 388 YouTube videos totaling 7.9 hours; more data has been annotated since then, and the corpus now contains 464 recordings totaling 9.6 hours. The data is annotated with the type, starting time, and ending time of each sound event occurrence. The sound events fall into 48 types; we manually merged some rare and semantically close types, ending up with 17 sound event types. The duration of each sound event type in the corpus is shown in Fig. 2. Note that the total length of the bars in Fig. 2 (11.8 h) is longer than 9.6 hours; this is because over a third of the duration is labeled with more than one sound event. The average polyphony (number of sound events occurring simultaneously) in the non-silence part of the corpus is 1.44.

The corpus was divided into training, validation, and test sets with a duration ratio of 3:1:1. Care was taken to make sure that the duration of each sound event type in the three sets also formed a ratio of 3:1:1. It turned out that we didn't use any validation during training, so the validation set was also used as a test set.

We extracted acoustic features using the OpenSMILE toolkit [27]. We first extracted low-level features such as MFCCs and F_0 (fundamental frequency), and then computed a variety of statistics over these raw features using sliding windows of 2 seconds moving 100 ms at a time. This procedure yielded 6,669-dimensional feature vectors, but many of the dimensions were strongly correlated. We conducted principal component analysis (PCA) to decorrelate the features, and retained only the top 50 dimensions. Each dimension was then normalized to span the range $[-0.9, 0.9]$.

We used a bidirectional RNN as the underlying network for CTC. The input layer had 50 units, corresponding to the dimensions of the acoustic features. The network had one hidden layer with two chains running in opposite directions; each chain consisted of 400 LSTM cells [21]. As for the output alphabet, an intuitive idea is to use the repertoire of sound event types. However, the polyphony makes it hard to define ordered sequences of sound events. To solve this problem, we used the *boundaries* of sound events, *i.e.* their starts and ends, as the output tokens. For example, if the content of a recording can be described by *a dog barks while a car passes by*, we use the sequence `engine_start, animal_start, animal_end, engine_end` as the ground truth. As a result, the output layer of our network had 35 output units – two for each sound event and one for the "blank" token – in a softmax group.

The objective function ("training cost") we chose was the per-frame negative log-likelihood, *i.e.* the sum of the negative log-probability of all training sequences divided by the total number

Step	Hidden layer size	Frame accuracy
Baseline [17]	300	46.7
Biasing forget gates	300	48.5
PCA acoustic features	300	49.0
Data augmentation	400	49.7
Additional data	400	52.2
Correcting alignment	400	54.0

Table 1. *Effect of each step to improve the frame-wise BLSTM-RNN, used to initialize the CTC model.*

of training frames. We trained our networks with the stochastic gradient descent algorithm, with an initial learning rate of 0.3 and a Nesterov momentum [28] of 0.9. After 200 epochs, we decayed the learning rate by multiplying 0.99 every epoch, until reaching 500 epochs. The feature sequences were chopped into segments no longer than 500 frames, cutting in the middle of silence segments whenever possible; each minibatch consisted of 5 such segments.

To decode the CTC output, we used the simple *best path decoding*. The output sequences were evaluated against the ground truth with the *token error rate* (TER) metric, computed the same way as *word error rate* (WER) in speech recognition. This metric only cares about the tokens produced by the CTC model, not the temporal position where they are produced.

3.2. Pre-training with a Bidirectional LSTM-RNN

During the initial phase of training, a CTC model often goes through a "warmup" stage, where it only outputs blank tokens. This stage can last for a long time. To shorten the warmup stage, we initialized the weights between the input layer and the hidden layer, as well as the recurrent weights of the hidden layer, with a bidirectional LSTM-RNN trained to perform frame-wise SED. The weights between the hidden layer and the output layer were initialized randomly.

The frame-wise BLSTM-RNN had the same structure as the CTC model, except that it had only 18 output units (standing for 17 sound event types and a "background" type). It was an improved version of the network introduced in [17]. In that paper, the BLSTM-RNN achieved a frame accuracy[1] of 46.7%, slightly falling short of bidirectional RNNs without LSTM cells (47.0%). We made a number of improvements to the BLSTM-RNN, boosting its frame accuracy to 54.0%. The improvements include:

1. *Biasing the forget gates*. We initialized the bias of the forget gates to 1.0 instead of 0.0, in order to encourage remembering in the early stages of training. This is an effective practice first proposed in [29], and emphasized in [30].
2. *PCA on the acoustic features*. The acoustic features used in [17] were 983 dimensions selected from the 6,669-dimensional OpenSMILE features; we replaced them with the 50-dimensional PCA features introduced in Sec. 3.1.
3. *Data augmentation*. We extracted acoustic features from both channels of the audio files, and used two different versions of OpenSMILE (1.0.1 and 2.1), so each recording had four different copies of features. During training, we still used a minibatch size of 5 streams, but updated the learning rate according to the validation performance after every quarter pass of the augmented training data. During testing, the probabilities predicted on the four copies of features were averaged before selecting the maximum.
4. *Additional data*. The corpus used in [17] had 7.9 hours of data; we added 1.7 hours of newly annotated data.
5. *Correcting the alignment between the features and the annotation*. In [17], the feature vector extracted from the 2 second window $[t, t+2]$ was associated with the annotation at time t. An inspection of the feature, however, revealed that this vector better describes what happens at $t+2$, especially when an abrupt sound event occurs. We corrected this alignment.

[1]The networks in [17] performed monophonic SED; a frame is regarded as correctly classified as long as one of the ground-truth events has the maximum probability. The frame accuracy is the percentage of correctly classified frames in the entire corpus.

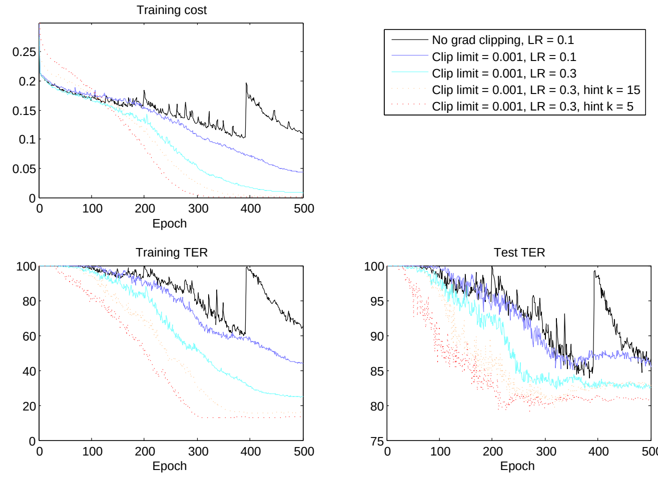

Fig. 3. *The evolution of the training cost, training TER, and test TER during training, with different gradient clipping limits, initial learning rates, and hint tolerances. The test TER is the average on the two testing sets. Note the different scales of the training and test TERs, which indicates severe overfitting. Best viewed in color.*

As we made these improvements, we also changed the hidden layer size from 300 to 400 LSTM cells in each direction. The effect of each improvement is shown in Table 1. The final frame accuracy, 54.0%, was the average of 4 networks trained from different random initializations; the single best model reached a frame accuracy of 55.5%, and was used to initialize the CTC model.

3.3. Gradient Clipping

Although LSTM cells avoid the gradient vanishing problem [31], our model still suffered from gradient explosion in two aspects: in the initial epochs, the magnitudes of gradients were large, forcing us to use a small learning rate, which made later epochs slow; from time to time, one single large value in the gradients would result in an abrupt surge in the training cost, which could take up to 100 epochs to compensate for, or even cause the training to crash. These phenomena highlight the need for gradient clipping during training.

The three solid lines in Fig. 3 demonstrate the effect of gradient clipping. Using a proper clipping limit avoided the surges in the training cost and the TER, allowed for a larger learning rate, and made the training converge faster and to a better result. With a clipping limit of 10^{-3} and a learning rate of 0.3, we achieved a training TER of 25%, and TERs of 84% and 81% on the two testing sets.

3.4. Hinting for the Alignment

The huge gap between the training and test TERs indicates severe overfitting. By inspecting the output of the CTC model on the

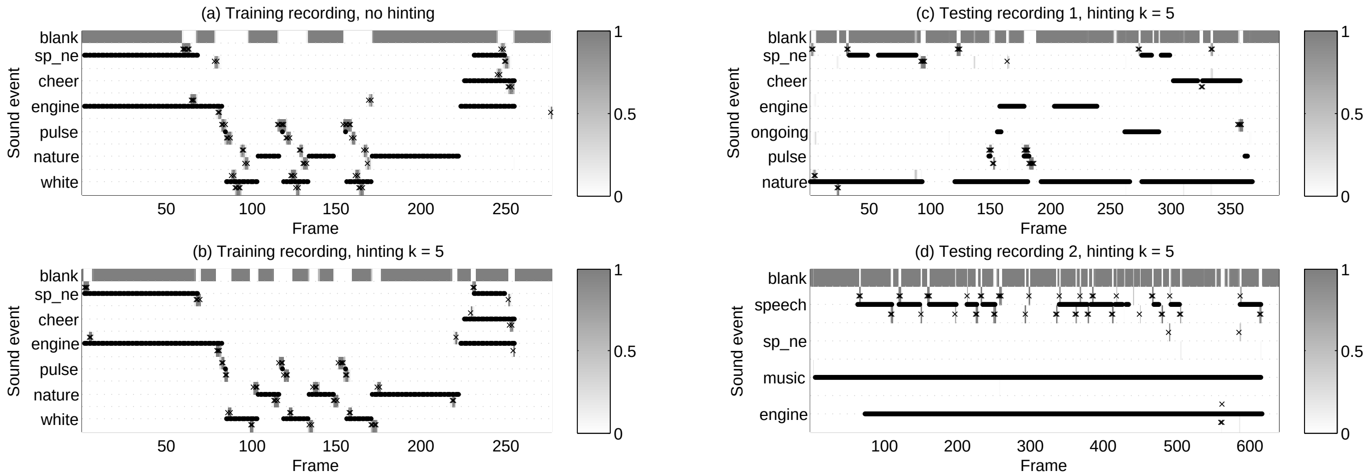

Fig. 4. *The CTC output on some training and test recordings. Each row of the graphs stands for an output token; each sound event is associated with two rows – its start and end tokens. Shades of gray signify the output probability. Crosses mark the most probable token at each frame; black dots (forming strings) mark the true span of sound events. Ideally, a piece of gray (a "peak") with one or more crosses should occur just above the start and below the end of each sound event instance. Unimportant sound events for these examples are omitted.*

training data, we found that it was able to output the correct token sequence most of the time, but the tokens were often far away from the actual time when the boundaries of the sound events occurred (see Sec. 3.5 and Fig. 4 (a)). This means the model picked up spurious patterns from random positions in the feature sequence.

Since we did have annotations about the exact timing of the tokens, we used them as approximate hints for the CTC model to find the correct alignment. We applied the following constraint when computing the alpha trellis during CTC training: all paths must go through a non-blank token within k frames of the moment when the token actually occurs (we call k the *tolerance*). That is, if the i-th token in the modified token sequence, l_i, is not a blank, and occurs at frame t_i according to the annotation, then all $\alpha_t(i)$ with $|t - t_i| > k$ will be set to zero. This constraint still leaves to the model the freedom to find the best alignment within $(2k+1)$-frame windows, as well as allowing annotations to be at most k frames off.

A similar idea was proposed in [32]. Instead of providing a rough range for the position of every token, the authors enforced exact positions for a few selected tokens in the sequence. Both methods provide hints for the CTC model to find a good alignment. We expect that less hints may be required with more data available.

Fig. 3 also shows the effect of alignment hinting with tolerances $k = 15$ and $k = 5$ (dotted lines). At a frame shift of 100 ms, these correspond to windows of 3 seconds and 1 second, respectively. With the hinting, the training cost decreased more slowly at the beginning, but got below the unhinted case after about 120 epochs. Both the training and test TERs decreased faster than the unhinted case. The more precise the hints were, the faster the training converged. With $k = 5$, the final training TER was 13%, and the final testing TER was 81% on both test sets. Although these numbers were lower than the unhinted case, overfitting remained a problem.

3.5. A Qualitative Analysis of the Network Output

The TER metric reflects how good a network is at recovering the correct sequence of sound events. But this is not enough for sound event detection; we also expect the peaks in the network output to occur at the right positions, *i.e.* at the starts and ends of sound event instances. In Fig. 4, we plot the output of the networks with and without hinting on some training and test recordings.

Graph (a) shows the output of the unhinted network on a training recording. At the middle of this recording are three cannon shots, signified by the `pulse, white, nature` sequence repeated three times. The unhinted network is able to recover this sequence correctly, but most tokens are placed far away from the positions where they really occur, and the tokens tend to cluster together. Graph (b) shows the output of a hinted network (with tolerance $k = 5$) on the same recording. Now we see the peaks occurring near the actual starts and ends of the sound events. We can also see that the network tends to generate peaks wider than 1 frame.

Graphs (c) and (d) show the output of the same hinted network on two test recordings. We find that the CTC model is able to detect the span of some sound event instances (*e.g.* the speech segments in graph (d)), notably short and transient ones (*e.g.* the pulses in graph (c)). However, it still makes many errors, such as predicting two false speech segments between frames 260 and 340, and confusing between English and non-English speech at frames 490 and 590 in graph (d). In addition, many sound events are missed (*e.g.* the cheering, music and engine noise). More data is probably needed for the network to learn to detect these sound events.

4. CONCLUSION AND FUTURE WORK

We conducted sound event detection using a CTC network, in order to relax the need for exact annotations of the starting and ending times of sound events. To deal with polyphony, we used the boundaries of sound events as output tokens. We demonstrated the importance of gradient clipping, and the helpfulness of providing rough hints about the positions of the event boundaries.

Although the CTC model achieved a low token error rate on the training data, and was thus able to learn the temporal characteristics of sound events successfully, its generalization was still poor. We believe this is mainly due to insufficient amounts of training data, but the advantages of CTC – the ability to detect short sounds and work with fuzzy labels – can still be leveraged for sound event detection. In the future, we will increase the amount of training data both by incorporating other corpora (*e.g.* the TUT Sound Events 2016 corpus [33]) and by data augmentation. We will also try out regularization techniques such as Dropout [34].

5. REFERENCES

[1] Y. Wang, S. Rawat, and F. Metze, “Exploring audio semantic concepts for event-based video retrieval”, in *Proc. of ICASSP*, pp. 1360-1364, 2014.

[2] P. Laffitte, D. Sodoyer, C. Tatkeu, and L. Girin, “Deep neural networks for automatic detection of screams and shouted speech in subway trains”, in *Proc. of ICASSP*, pp. 6460-6464, 2016.

[3] X. Zhou, *et al.*, “HMM-based acoustic event detection with AdaBoost feature selection”, in *Multimodal technologies for perception of humans*, pp. 345-353, Springer, 2008.

[4] T. Heittola, A. Mesaros, A. Eronen, and T. Virtanen, “Context-dependent sound event detection”, in *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2013, no. 1, pp. 1-3, 2013.

[5] A. Mesaros, T. Heittola, A. Eronen, and T. Virtanen, “Acoustic event detection in real life recordings”, in *Proc. of EUSIPCO*, pp. 1267-1271, 2010.

[6] S. Innami and H. Kasai, “NMF-based environmental sound source separation using time-variant gain features”, in *Computers & Mathematics with Applications*, vol. 64, no. 5, pp. 1333-1342, 2012.

[7] A. Dessein, A. Cont, and G. Lemaitre, “Real-time detection of overlapping sound events with non-negative matrix factorization”, in *Matrix Information Geometry*, pp. 341-371, Springer, 2013.

[8] T. Heittola, A. Mesaros, T. Virtanen, and M. Gabbouj, “Supervised model training for overlapping sound events based on unsupervised source separation”, in *Proc. of ICASSP*, pp. 8677-8681, 2013.

[9] O. Dikmen and A. Mesaros, “Sound event detection using non-negative dictionaries learned from annotated overlapping events”, in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2013.

[10] O. Gencoglu, T. Virtanen, and H. Huttunen, “Recognition of acoustic events using deep neural networks”, in *Proc. of EUSIPCO*, pp. 506-510, 2014.

[11] E. Cakir, T. Heittola, H. Huttunen, and T. Virtanen, “Polyphonic sound event detection using multi-label deep neural networks”, in *IEEE International Joint Conference on Neural Networks*, 2015.

[12] M. Ravanelli, B. Elizalde, K. Ni, and G. Friedland, “Audio concept classification with hierarchical deep neural networks”, in *Proc. of EUSIPCO*, pp. 606-610, 2014.

[13] H. Zhang, I. McLoughlin, and Y. Song, “Robust sound event recognition using convolutional neural networks”, in *Proc. of ICASSP*, pp. 559-563, 2015.

[14] H. Phan, L. Hertel, M. Maass, and A. Mertins, “Robust audio event recognition with 1-max pooling convolutional neural networks”, arXiv:1604.06338, 2016.

[15] K. J. Piczak, “Environmental sound classification with convolutional neural networks”, in *IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1-6, 2015.

[16] J. Salamon and J. P. Bello, “Deep convolutional neural networks and data augmentation for environmental sound classification”, arXiv:1608.04363, 2016.

[17] Y. Wang, L. Neves, and F. Metze, “Audio-Based Multimedia Event Detection Using Deep Recurrent Neural Networks”, in *Proc. of ICASSP*, pp. 2742-2746, 2016.

[18] G. Parascandolo, H. Huttunen, and T. Virtanen, “Recurrent neural networks for polyphonic sound event detection in real life recordings”, in *Proc. of ICASSP*, pp. 6440-6444, 2016.

[19] S. Adavanne, *et al.*, “Sound event detection in multichannel audio using spatial and harmonic features”, in *Workshop on Detection and Classification of Acoustic Scenes and Events*, 2016.

[20] T. Hayashi, *et al.*, “Bidirectional LSTM-HMM hybrid system for polyphonic sound event detection”, in *Workshop on Detection and Classification of Acoustic Scenes and Events*, 2016.

[21] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, in *Neural Computation*, vol. 9, pp. 1735-1780, 1997.

[22] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, “Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks”, in *Proc. of ICML*, 2006.

[23] A. Graves and N. Jaitly, “Towards end-to-end speech recognition with recurrent neural networks”, in *Proc. of ICML*, 2014.

[24] Y. Miao, M. Gowayyed, and F. Metze, “EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding”, in *Proc. of ASRU*, 2015.

[25] J. Bergstra, *et al.*, “Theano: a CPU and GPU math expression compiler”, in *Proc. of the 9th Python for Scientific Computing Conference*, pp. 1-7, 2010.

[26] S. Burger, Q. Jin, P. F. Schulam, and F. Metze, “Noisemes: manual annotation of environmental noise in audio streams”, technical report CMU-LTI-12-07, Carnegie Mellon University, 2012.

[27] F. Eyben, F. Weninger, F. Gross, B. Schuller, “Recent developments in OpenSMILE, the Munich open-source multimedia feature extractor”, in *Proc. of ACM Multimedia*, pp. 835-838, 2013.

[28] Y. Nesterov, “A method of solving a convex programming problem with convergence rate O(1/sqr(k))”, in *Soviet Mathematics Doklady*, vol. 27, pp. 372-376, 1983.

[29] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with LSTM”, in *Neural Computation*, vol. 12, no. 10, pp. 2451-2471, 2000.

[30] R. Jozefowicz, W. Zaremba, and I. Sutskever, “An empirical exploration of recurrent network architectures”, in *Proc. of ICML*, 2015.

[31] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies”, in *A Field Guide to Dynamical Recurrent Neural Networks*, IEEE Press, 2001.

[32] D.-A. Huang, F.-F. Li, and J. C. Niebles, “Connectionist temporal modeling for weakly supervised action labeling”, arXiv:1607.08584, 2016.

[33] A. Mesaros, T. Heittola, and T. Virtanen, “TUT database for acoustic scene classification and sound event detection”, in *Proc. of EUSIPCO*, 2016.

[34] N. Srivastava, *et al.*, “Dropout: a simple way to prevent neural networks from overfitting”, in *Journal of Machine Learning Research*, vol. 15, pp. 1929-1958, 2014.