Polyphonic Sound Event Detection with Weak Labeling

Yun Wang

October 2017

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

A thesis proposal submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

Sound event detection (SED) is the task of detecting the type and the onset and offset times of sound events in audio streams. It is useful for purposes such as multimedia retrieval and surveillance. Sound event detection is difficult in several aspects when compared with speech recognition: first, sound events are much more variable than phonemes, notably in terms of duration but also in terms of spectral characteristics; second, sound events often overlap with each other, which does not happen with phonemes.

To train a system for sound event detection, it is conventionally necessary to know the type, onset time and offset time of each occurrence of a sound event. We call this type of annotation *strong labeling*. However, such annotation is not available in amounts large enough to support deep learning. This is due to multiple reasons: first, it is tedious to manually label each sound event with exact timing information; second, the onsets and offsets of long-lasting sound events (*e.g.* car passing by) and repeating sound events (*e.g.* footsteps) may not be well-defined.

In reality, annotation of sound events often comes without exact timing information. We call such annotation *weak labeling*. Even though it contains incomplete information compared to strong labeling, weak labeling may come in larger amounts and is well worth exploiting. In this thesis, we propose to train deep learning models for SED using various levels of weak labeling. We start with *sequential labeling*, *i.e.* we know the sequences of sound events occurring in the training data, but without the onset and offset times. We show that the sound events can be learned and localized by a recurrent neural network (RNN) with a connectionist temporal classification (CTC) output layer, which is well suited for sequential supervision. Then we relax the supervision to *presence/absence labeling*, *i.e.* we only know whether each sound event is present or absent in each training recording. We solve SED with presence/absence labeling in the multiple instance learning (MIL) framework, and propose to analyze the network's behavior on transient, continuous and intermittent sound events.

As we explore the possibility of learning to detect sound events with weak labeling, we are often faced with the problem of data scarcity. To overcome this difficulty, we resort to *transfer learning*, in which we train neural networks for out-of-domain tasks on large data, and use the trained networks to extract features for SED. We make special effort to ensure the temporal resolution of such transfer learning feature extractors.

Contents

List of Figures

List of Tables

List of Abbreviations

BPTT	Back-propagation through time
CASA	Computational auditory scene analysis
CLEAR	Classification of events, activities and relationships
CMU	Carnegie Mellon University
CNN	Convolutional neural network
CTC	Connectionist temporal classification
DCASE	Detection and classification of acoustic scenes and events
DNN	Deep neural network
ESC	Environmental sound classification
FA	False alarm
GMM	Gaussian mixture model
GRU	Gated recurrent unit
HMM	Hidden Markov model
KL	Kullback-Leibler (divergence)
LIUM	Laboratoire d'Informatique de l'Université du Maine
LSTM	Long short-term memory
MED	Multimedia event detection
MFCC	Mel-frequency cepstral coefficients
MIC	Multiple instance classification
MIL	Multiple instance learning
mi-SVM	Multiple instance support vector machine
MSE	Mean squared error
NIST	National Institute of Standards and Technology
NMF	Non-negative matrix factorization
PCA	Principal component analysis
ReLU	Rectified linear unit
RNN	Recurrent neural network
SED	Sound event detection
SGD	Stochastic gradient descent
SMI	Standard multiple instance (assumption)
SN-F	SoundNet, fully connected

SN-R	SoundNet, recurrent
SNR	Signal-to-noise ratio
SVM	Support vector machine
TED	Technology, Entertainment, Design
TEDLIUM	(see TED and LIUM)
TER	Token error rate
TREC	Text Retrieval Conference
TUT	Tampere University of Technology
WER	Word error rate
YFCC	Yahoo Flickr Creative Commons

Chapter 1

Introduction

The environment of our daily life is filled with various sound events, such as cars passing by in the streets, or doors opening and closing in the office. These sound events provide us with a tremendous amount of information about the surroundings, and our auditory system is surprisingly good at separating and recognizing them. If an intelligent system aims at interacting with humans and the environment in a natural way, it must be able to recognize and understand sound events.

The procedure of a machine turning the ambient sound signal into a meaningful representation is called *computational auditory scene analysis* (CASA) [1]. CASA involves several related tasks, such as *acoustic scene recognition*, *sound event detection*, and *source separation*. These tasks are progressively harder: acoustic scene recognition only requires determining the type of the environment (*e.g.* office, restaurant, train); sound event detection requires the detection and classification of each individual sound event; source separation requires separating sound events in a mixed signal into clean audio streams. These tasks are also closely related and can facilitate each other: knowing the acoustic scene reduces the uncertainty in the distribution of sound events, while the types of the sound events are an important source of information about the acoustic scene; having sound events separated from a mixture makes their recognition easier, while knowing the spectral characteristics of sound events also makes it easier to separate them.

This proposal deals with the task of *sound event detection* (SED). It entails both *classification* and *localization*, *i.e.* we need to recognize the type as well as determine the onset and offset times of each sound event occurrence. We consider a broad range of sound events in this proposal, with some examples shown in Fig. 1.1. Sound events can be categorized from a variety of aspects. They may originate from various sources (*e.g.* human, animals, machinery, nature), and these sources may be either fixed or moving. In terms of spectral characteristics, sound events may be either

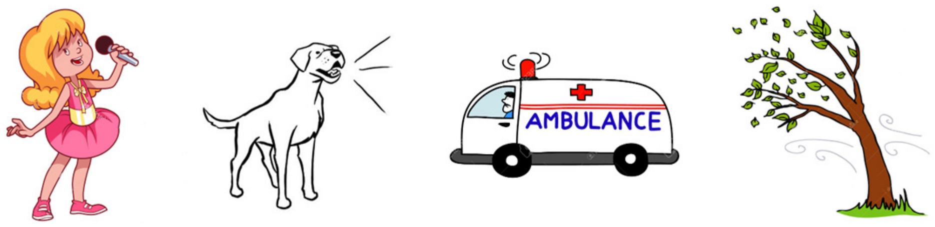


Figure 1.1: Example sound events: singing, dog barking, ambulance siren, wind.

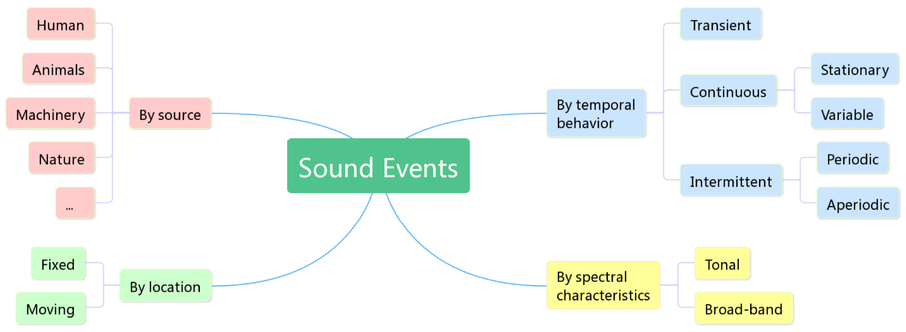


Figure 1.2: A categorization of sound events.

tonal (exhibiting distinct peaks in the spectrum, *e.g.* sirens) or noise-like (with power spanning a broad frequency band in the spectrum, *e.g.* cheering). In terms of temporal behavior, sound events may be transient, continuous or intermittent. Transient sound events (*e.g.* gun shots) last for a very short time, usually less than one second. Continuous sound events last for a long duration, and they may either be stationary (*e.g.* engine noise) or display varying frequency characteristics (*e.g.* music). Intermittent sound events occur repetitively with short intervals in between, and they may either exhibit a periodic pattern (*e.g.* footsteps) or occur at irregular intervals (*e.g.* dog barks). Fig. 1.2 summarizes the categorization of sound events. We are especially interested in the temporal behavior of sound events. At the end of Chapter 4, we propose to automatically classify sound events as transient, continuous or intermittent.

Sound event detection is useful for many applications. For example, in the automatic subtitling of TV dramas for hearing impaired people, it is often necessary to include sound events (*e.g.* telephone ringing), because they can be important for understanding the storyline. SED can also be used for surveillance purposes, such as detecting the noise of a person falling down stairs in hospitals [2], screams in subway trains [3], and gunshots [4]. A more full-fledged application of SED is understanding activities taking place in videos, such as a soccer game or a birthday party. This can be used to generate metadata for the millions of videos uploaded by Internet users every day, so they can be efficiently searched. This has been the goal of the yearly NIST TREC Video Retrieval Evaluation[1] since 2003. A number of systems based on the detection of sound events, either learned in an unsupervised fashion [5, 6] or defined by humans [7–9] have seen success on this task.

Sound event detection is made difficult by several factors. Besides the great variability in their spectral and temporal characteristics, sound events also often overlap in time, which means some of the events need to be recognized in a signal-to-noise ratio (SNR) of less than 0 dB. Depending on how overlapping sound events are dealt with, SED systems may be classified into *monophonic* systems, which can only detect one sound event at a given time, and *polyphonic* systems, which may generate overlapping detections. This proposal is principally concerned with polyphonic SED.

Another difficulty in developing SED systems is the lack of data. This is manifested in many aspects. On one hand, the amount of audio data itself is scarce – most SED systems developed so far do not use more than 20 hours of audio. The amount of audio data also restricts the number of sound event types that can be handled. Some corpora provide annotations of dozens of sound event types, but many of these types are so rare that they suffer from nearly zero recall. On the other hand, SED systems are best trained when the type, onset time and offset time are fully known, but it can be a tedious task to produce such *strong labeling* by hand. In reality, annotation for SED may come in a weaker form, where we only know the sequence of sound events, or even only whether each type of sound event is present or absent. This proposal is interested in how we can utilize such *weak labeling*.

In the remainder of this chapter, I will first review historical and state-of-the-art approaches toward sound event detection, and analyze their advantages and short-comings. Then I will introduce some corpora for SED used in previous work as well as this proposal. At the end of this chapter, I will outline the contributions of the work that I would like to propose for my PhD thesis.

1.1 History and State-of-the-Art of SED

A multitude of models have been used to model sound events. An early example is hidden Markov models (HMMs), *e.g.* [10, 11]. Each type of sound events was modeled by a three-state HMM; left-to-right HMMs were used for sound events with temporal structures, and ergodic HMMs were used for relatively stationary ones. The distribution of acoustic feature vectors belonging to each HMM state was modeled with Gaussian mixture models

[1]http://trecvid.nist.gov/

(GMMs). Viterbi decoding was employed to generate sequences of sound events, and to locate the onset and offset times of each sound event instance. HMMs can take advantage of event priors and "language models" to rule out unlikely sound event sequences, but a drawback of HMMs is the inability to deal with polyphony. A multi-pass decoding procedure was proposed in [12] for polyphonic SED: at each frame in each pass of decoding, the Viterbi path was prohibited from entering HMM states corresponding to sound events that had already been detected at that frame in previous iterations. With multi-pass decoding, HMMs were able to produce polyphonic detections, but they still could not model how overlapping sound events affect the acoustic characteristics of each individual sound event.

To explicitly deal with the overlapping of sound events, researchers have resorted to source separation techniques, such as non-negative matrix factorization (NMF) [13]. In NMF, the non-negative spectrogram X of a mixture signal is decomposed into the product of a basis matrix W and a gain matrix H, both with non-negative elements. The columns of the basis matrix W can be understood as the spectra of stationary sound events or sub-units of sound events with a variable temporal structure, and the elements of the gain matrix H indicate how much each basis is activated at each frame. NMF has been applied to SED in multiple ways. In [14, 15], a test recording was first decomposed into four streams before monophonic SED was conducted on each stream. The training recordings were also decomposed into streams to separate overlapping sound events, in order to train better acoustic models (HMMs) of each sound event type. In contrast, [16] did not attempt to separate the acoustic signals of overlapping sound events. Instead, the authors treated the annotation of a recording as a non-negative matrix similar to a spectrogram, and learnt one basis matrix W_1 for spectrograms and one for annotation matrices W_2 jointly. If a basis in W_1 represented the spectrum of multiple overlapping sounds, then the corresponding basis in W_2 would have multiple entries with large values. For a testing recording with a spectrogram X, a gain matrix H was estimated by solving $X = W_1 H$, then the annotation matrix of the recording was given by thresholding $W_2 H$. NMF is good at dealing with overlapping sounds; however, it handles the spectrum of each frame independently, and fails to model any temporal context.

With the rapid growth of deep learning techniques, neural networks have become the mainstream solution to recognizing sound events. Neural networks overcome many defects of previous approaches: they are no longer restricted by the topology of HMMs, and they can take context into consideration easily. More importantly, neural networks can be regarded as trainable feature extractors, so they eliminate the need for complicated feature engineering often required for HMMs, and their deep structure can analyze the acoustic signal better than the simple matrix multiplication of NMF.

The application of neural networks to SED started with feed-forward neural networks. Feed-forward neural networks were used in [17, 18] to classify isolated instances of sound events, taking the acoustic features of many consecutive frames as input. This could be easily extended to detecting sound events in audio streams by applying the network to sliding windows of acoustic features and smoothing the decisions with a simple median filter (*e.g.* [19]) or an HMM (*e.g.* [20]). The simple feed-forward neural network in [19] yielded a significantly better polyphonic SED performance than the HMM system with NMF pre-processing in [15].

Feed-forward networks treat all the input neurons independently, while the spectrograms of sounds exhibit high correlation between neighboring time-frequency units, just like images. To make better use of the spectro-temporal locality of spectrograms, convolutional neural networks (CNNs) were used to classify isolated events [21–26] as well as to detect sound events in mixtures [27, 28]. The CNN in [28] again yielded superior performance compared with the feed-forward network in [20].

CNNs make a decision for each frame based on the signal within a temporal window around this frame, thereby making use of limited context. Taking this a step further, recurrent neural network (RNNs) make frame-wise decisions based on unlimited context. Even though some sound events only last for a short period of time, this unlimited context can provide information about the background they occur in, and make them easier to recognize. In addition, the recurrent connections in the hidden layers can function as an implicit "language model" of sound events, making it unnecessary to smooth the frame-wise decisions. RNNs were successfully applied to SED in [8, 29–31]. A more recent study [32] proposed a network with convolutional layers followed by recurrent layers. Combining the ability of CNNs to learn locally invariant filters and the power of RNNs to model both long and short temporal dependencies, this network achieved better polyphonic SED performances than either a CNN or an RNN alone on four datasets.

All the methods presented above depend on *strong labeling*, *i.e.* the exact onset and offset times of sound event instances. Such labeling can take a formidable amount of effort to create. As a result, state-of-the-art research on SED focus on *weak labeling*. This proposal deals with two types of weak labeling: *sequential labeling* and *presence/absence labeling*.

In *sequential labeling*, we know the order of sound events happening in each recording, but do not know the exact onset and offset times of each sound event occurrence. This is the type of labeling most commonly found for speech recognition, if we regard phonemes as equivalents of sound events, and connectionist temporal classification (CTC) is the state-of-the-art technique of exploiting such supervision. However, this type of supervision has not been explored much in other tasks. One example is [33], where the authors applied CTC to the detection of actions in video recordings, given only the order of actions in each recording. For sound event detection, we have seen preliminary success with CTC, which has been published in [34, 35], and will be presented in detail in Chapter 3.

Another common form of weak labeling is *presence/absence labeling*: it is only known whether each type of sound event occurs in each recording or not. Because it is expensive to generate stronger forms of labeling, recent large corpora (*e.g.* Google Audio Set) are often annotated with the presence/absence of sound events, and SED with presence/absence labeling is becoming a hot focus of research. Learning to detect sound events from presence/absence labeling can be formulated as a special case of multiple instance learning (MIL) [36]: instead of knowing the label for each instance, the instances are grouped into *bags*, and labels are known for the bags only. In the case of presence/absence labeling for SED, each recording may be treated as a bag, and the frames (or segments) of the recording as instances in the bag. The labels for the instances and those for the bags conform to the *standard multiple instance assumption*: a bag is labeled as positive if it contains at least one positive instance, and negative if it contains only negative instances. One solution to SED with presence/absence labeling is to train a recording-level classifier by aggregating (or "pooling") frame-level decisions. The "max pooling" function has been used in [37] and [38]; in Chapter 4 of this work, we also study a "noisy-or" pooling function which has been applied to object detection in images [39–41]. More complicated MIL methods have been applied to SED as well, such as multiple instance support vector machines (mi-SVM) [38] and manifold regularization on graphs [42].

1.2 Corpora for Sound Event Detection

Some works on sound event detection were first validated with the task of classifying isolated sound events in clean environments. The ESC-50[2] corpus [43] comprises 2,000 short clips of 5 seconds long, each clip containing an instance of one of 40 sound event types. However, the controlled clean environments are drastically different from the complex acoustic scenes encountered in real life, so this corpus is of limited use.

A corpus of sound events recorded in real-life environments is Urban-Sound [44]. It contains 1,302 recordings totaling 27 hours, and includes 3,075 instances of 10 types of sound events frequently occurring in cities, such as car horns, sirens, and street music. A weakness of this corpus, however, is that each recording is only annotated for one type of sound events. No overlapping sound events are annotated, and therefore the corpus cannot be used for studies on polyphonic sound event detection.

Several global competitions of sound event detection have been organi-

[2] "ESC" stands for "environmental sound classification".

zed, including the CLEAR[3] evaluation in 2006 and 2007 [45, 46], and the DCASE[4] challenge in 2013[5], 2016[6] and 2017[7]. These competitions provide standard corpora for SED, as well as benchmark results to compare with. The audio data used in the CLEAR evaluations was seminars recorded in meeting rooms. The corpus contained 3 hours of development data and 2 hours of evaluation data[8]. Twelve types of sound events commonly found in meeting rooms were annotated; speech was also annotated but not evaluated. The evaluation data contained 1,454 target sound event instances. The DCASE 2016 SED challenge [49] used 78 minutes (22 recordings) of development data and 36 minutes (10 recordings) of evaluation data, broken down into two environments: home and residential area. In the development part of the data, 11 and 7 types of sound events were annotated in the two environments respectively, with a total of 1,465 instances.

Even though these corpora are accepted as standard data for evaluation, they are rather small for training SED systems, especially when the systems use weak labeling. As a complement, researchers have also collected private corpora of sound events. At the Tampere University of Technology (TUT) in Finland, a corpus has been collected with a total duration of 1,133 minutes. The corpus contains 103 recordings collected from ten real-life environments, such as offices, restaurants, and buses. 61 types of sound events are annotated with exact onset and offset times. The average number of events active simultaneously, called the *average polyphony level*, is 2.53[9] [29]. This corpus was first described in [50] and named "TUT-SED 2009" in [32], and it has been used in a number of studies at TUT [11, 12, 15, 17, 19, 29, 32].

At Carnegie Mellon University (CMU), we have also been collecting and annotating data for sound event detection. The corpus we have collected is called the Noiseme corpus – the word "noiseme" was coined imitating "phoneme" and "grapheme" since sound events are the basic units that make up noises in an acoustic scene. This is the corpus that we use in most of our experiments in Chapter 3. Since its first documentation in [51], the corpus has had three versions, but most of our experiments used Version 2. The size of the corpus has grown from 7.9 hours (388 recordings) in Version 1 to 12.9 hours (587 recordings) in Version 3. Most of the recordings are around 1 minute long; a detailed histogram of the duration of the recordings is

[3] "CLEAR" stands for "classification of events, activities and relationships".

[4] "DCASE" stands for "detection and classification of acoustic scenes and events".

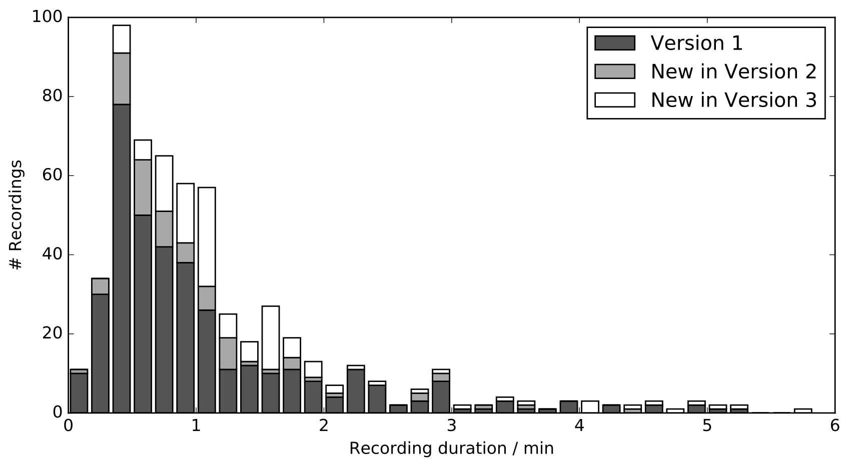
[5] `http://c4dm.eecs.qmul.ac.uk/sceneseventschallenge/`

[6] `http://www.cs.tut.fi/sgn/arg/dcase2016/`

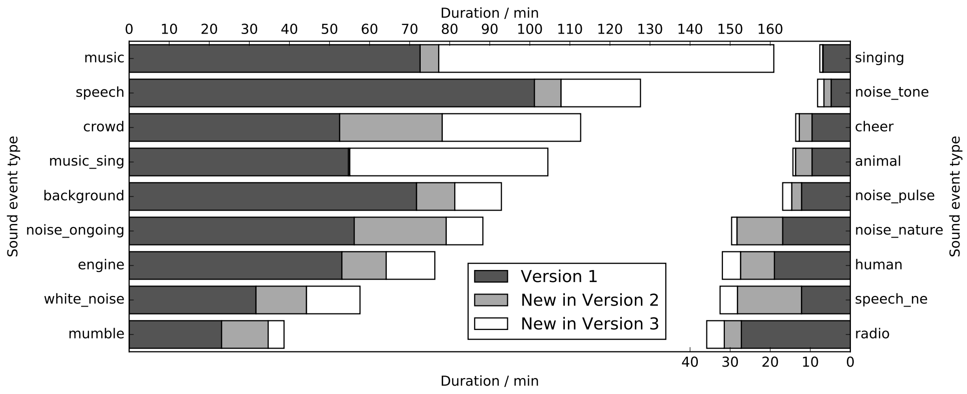
[7] `http://www.cs.tut.fi/sgn/arg/dcase2017/`

[8] These numbers follow the description of the CLEAR 2007 evaluation in [10] and [47]. Accounts differ in other references: [48] says the CLEAR 2006 evaluation used 5 seminars each lasting $10 \sim 20$ minutes; [46] says the CLEAR 2007 evaluation used 100 minutes of seminar data for development and 200 minutes for evaluation.

[9] It is not clear whether background segments (where no sound events occur) are included when this number is calculated.



(a) The distribution of the duration of each recording. A small number of recordings longer than 6 minutes are omitted.



(b) The distribution of the total duration of each sound event type. “Speech_ne” stands for “non-English speech”. “Background” refers to segments when no sound events occur.

Figure 1.3: Some statistics of the Noiseme corpus.

shown in Fig. 1.3 (a). 48 types of sound events are annotated with exact onset and offset times; because some event types are rare, we merged them down to 17 types in our experiments. The total duration of each sound event type, as well as the “background”, is plotted in Fig. 1.3 (b). The average polyphony level ranges between 1.40 and 1.43 in the non-background region.

In March 2017, Google released Audio Set [52], which is hundreds of times larger than all the aforementioned corpora. Audio Set contains 2.1 million 10-second excerpts from YouTube videos, which sum up to 5,800 hours (8 months). The data is annotated with 527 types of sound events. Unlike other corpora, Audio Set only comes with *presence/absence labeling*: the annotation of Audio Set does not specify the exact onset and

Name		No. of Recordings	Avg. Rec. Duration	Total Duration	No. SE Types	No. SE Instances	Average Polyphony
ESC-50 [43]		2,000	5 s	2.8 h	50	2,000	
UrbanSound [44]		1,302	75 s	27.0 h	10	3,075	
CLEAR 2007 [10, 47]				5.0 h	12	1,454	
DCASE 2016 [49]		32	214 s	1.9 h	18	1,465	
TUT-SED 2009 [50]		103	660 s	18.9 h	61	10,278	2.53
Noiseme [51]	v1	388	73 s	7.9 h	48	9,237	1.40
	v2	464	75 s	9.6 h	(merged	12,163	1.43
	v3	587	79 s	12.9 h	to 17)	14,382	1.40
Google Audio Set [52]		2.1 million	10 s	8 months	527	N/A	
SoundNet [53]		2.1 million	20 s	1 year	1,000 obj. + 401 scenes	N/A	

Table 1.1: A summary of corpora available for SED.

offset times of the sound events, but only indicates whether each type of sound event is present or absent in each 10-second excerpt. Google Audio Set is a valuable resource for studying sound event detection with weak labeling.

We have also used the training data of SoundNet [53], which has a similar size to the Google Audio Set. SoundNet is a deep convolutional neural network for transfer learning; it takes the audio tracks of videos as input, and tries to predict the distribution of objects and scenes in the video. As such, its training data is not annotated with sound events, but with distributions of visual objects and scenes. Nevertheless, the enormous size of the corpus still makes it useful for the study of sound event detection. The corpus consists of 2 millions videos for training and 147 thousand videos for validation; the videos come from either the YFCC100M[10] corpus [54] or the Flickr website. The first 20 seconds of the audio tracks are used for training the network, and these sum up to 1 year worth of audio. The annotation, generated by the VGG16 image recognition network [55], contains the distributions of 1,000 objects and 401 visual scenes at keyframes. Keyframes are selected at 3 s, 8 s, 13 s and 18 s of each video; the total number of keyframes is 7 million for training and 0.5 million for validation. All the audio tracks, keyframe images, and object and scene distributions can be downloaded from the demo page[11].

A summary of the corpora available for SED can be found in Table 1.1. Put in a nutshell, these corpora mainly fall into two categories. Some of the corpora are fully annotated, but they are not large enough to support deep learning; others are sufficiently large, but only come with weak labeling. This proposal addresses the challenge of exploiting both the quality of the former and the volume of the latter.

[10] "YFCC" stands for "Yahoo Flickr Creative Commons".
[11] https://projects.csail.mit.edu/soundnet/

1.3 Contributions of This Proposal

This proposal studies how we can make use of two types of weak labeling for sound event detection: *sequential labeling* and *presence/absence labeling.* I specially focus on the temporal localization of sound events, *i.e.* determining the onset and offset times of each sound event occurrence even though such information is not present in the labeling. I also study how *transfer learning* can be employed to deal with the data scarcity problem.

SED with sequential labeling: I approach SED with sequential labeling using the connectionist temporal classification (CTC) framework. I will show that CTC can be used to predict sequences of sound event boundaries, and it is better than frame-wise models at detecting transient events such as pulse noises. Bottlenecks in current research on SED with sequential labeling include the insufficient amount of training data, and inadequate exploitation of long events. For the first problem, I propose a solution of semi-supervised training, in which reliable training instances are automatically discovered from Google Audio Set and added to the training data. For the second problem, I propose to add tokens to the CTC vocabulary that signify sound events are ongoing, besides tokens that signify the onset and offsets. Temporal localization is also a weakness of CTC. I propose to train a feed-forward neural network in parallel with the recurrent neural network underlying the CTC output layer, in order to emphasize local information and improve the temporal localization of CTC.

SED with presence/absence labeling: I study SED with presence/ absence labeling in the framework of multiple instance learning (MIL). I compare two types of pooling functions used in MIL: max pooling and noisy-or pooling. With a proof-of-concept experiment on a speech recognition task, I demonstrate that max pooling is better suited for SED. I propose to apply MIL to the Google Audio Set, especially focusing on the temporal localization of sound events. I am also interested in the discovery of temporal structure of sound events, and propose methods to automatically classify sound events as transient, continuous or intermittent.

Transfer learning: Transfer learning can be used to extract features for SED. Trained on large corpora, transfer learning feature extractors can overcome the problem of insufficient training data, and it has been shown to improve the performance of a CTC network [35]. However, current transfer learning feature extractors (*e.g.* SoundNet [53]) have a poor temporal resolution. In this proposal, I will study how to train transfer learning feature extractors with enough temporal resolution to support the temporal localization of sound events.

The rest of this proposal is organized as follows: Chapter 2 gives a review of common models and techniques for deep learning, including feed-forward neural networks, recurrent neural networks (RNNs), convolutional neural networks (CNNs), and connectionist temporal classification (CTC).

Chapter 3 presents my current progress and proposes future work on SED with sequential labeling, as well as feature extraction with transfer learning. Chapter 4 is concerned with SED with sequential labeling, and the automatic discovery of the temporal structure of sound events. Finally, Chapter 5 summarizes the contributions of this proposal, and gives a tentative timeline for the work leading up to my thesis defense.

Chapter 2

Review of Deep Learning Techniques

2.1 Feed-Forward Neural Networks

Neural networks have become the most popular machine learning model in the past decade. With their tremendous power of approximating functions, they can be used for a variety of pattern recognition tasks, including image recognition, speech recognition, sound event detection, *etc.*

A neural network can be regarded as a complex function. The simplest form of neural networks is *feed-forward* neural networks. Feed-forward neural networks usually consist of many *layers* stacked on top of each other; for this reason, they are also called *deep* neural networks (DNN). Each layer consists of many *neurons*; collectively, they take a vector of a fixed size as input, and generate a vector of a fixed size as output. Let $\boldsymbol{h}^{(l-1)} \in \mathbb{R}^m$ be the input to the l-th layer, and $\boldsymbol{h}^{(l)} \in \mathbb{R}^n$ its output, then the behavior of the layer can be expressed as:

$$\boldsymbol{h}^{(l)} = \sigma(W^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}) \tag{2.1}$$

In this equation, $W^{(l)} \in \mathbb{R}^{n \times m}$ and $\boldsymbol{b}^{(l)} \in \mathbb{R}^n$ are called the *weight matrix* and the *bias vector*, and they are the parameters of the l-th layer. σ is a non-linear function called the *activation function*, and it is this non-linearity that gives neural networks the power to approximate functions. Commonly used non-linear functions include element-wise function such as the logistic sigmoid function (sigm), the hyperbolic tangent function (tanh), and the rectified linear unit function (ReLU). The equations of these functions are

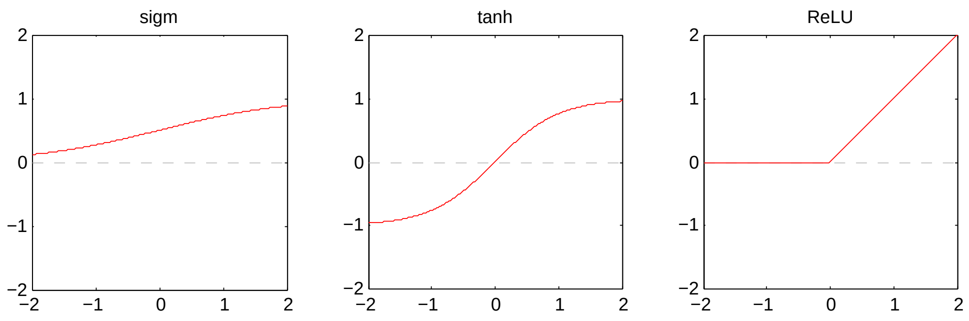


Figure 2.1: Common activation functions used in neural networks.

given below, and their graphs are shown in Fig. 2.1.

$$\mathrm{sigm}(x) = \frac{1}{1+\mathrm{e}^{-x}} \tag{2.2a}$$

$$\tanh(x) = \frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}} \tag{2.2b}$$

$$\mathrm{ReLU}(x) = \max(x, 0) \tag{2.2c}$$

The choice of the activation function is arbitrary for all but the output layer. For the output layer, the activation function depends on the type of the task that the network is trying to solve: for regression, the output layer uses the identify function; for binary classification, the logistic sigmoid function is used to generate values between 0 and 1, which can be interpreted as probabilities; for multi-class classification, a non-element-wise softmax function is used to generate a probability distribution. Let $x_1, \ldots, x_n$ be the elements of the vector input to the softmax function, then the i-th element of its output will be

$$y_i = \frac{\mathrm{e}^{x_i}}{\sum_{j=1}^{n} \mathrm{e}^{x_j}}. \tag{2.3}$$

It can be easily verified that $y_i > 0, \forall\, i$ and that $\sum_{i=1}^{n} y_i = 1$, which makes the vector $\boldsymbol{y}$ a valid probability distribution. The output layer activation functions suitable for different types of machine learning tasks are summarized in Table 2.1.

The training of a neural network is the procedure of learning the parameters of its layers in order to minimize a scalar loss function. The loss function is usually a sum or average of the contribution from each instance of the training data. Denote by $\boldsymbol{x}$ the input of an instance, and $\boldsymbol{t}$ its target output, and let $\boldsymbol{y}$ be the actual output of the network when $\boldsymbol{x}$ is fed into it. The form of the contribution $L(\boldsymbol{y}, \boldsymbol{t})$ of this instance to the loss function depends on the type of the task; the most common forms are also listed in Table 2.1.

Given the training data, the loss function L on the entire training corpus can be regarded as a function of the network parameters $\boldsymbol{\theta}$. There

Task	**Output layer activation**	**Loss function**	**Expression of loss function**
Regression	Linear	Mean squared error (MSE)	$L(\boldsymbol{y}, \boldsymbol{t}) = \|\|\boldsymbol{y} - \boldsymbol{t}\|\|_2^2$
Binary classification	Sigmoid	Binary cross-entropy	$L(\boldsymbol{y}, \boldsymbol{t}) = -\sum_{i=1}^n t_i \log y_i - \sum_{i=1}^n (1-t_i) \log(1-y_i)$
Multi-class classification	Softmax	Categorical cross-entropy	$L(\boldsymbol{y}, \boldsymbol{t}) = -\sum_{i=1}^n t_i \log y_i$, or $L(\boldsymbol{y}, \boldsymbol{t}) = -\log \sum_{i=1}^n t_i y_i$

Table 2.1: Output layer activation functions and loss functions suitable for different types of machine learning tasks. The two forms of categorical cross-entropy loss function are equivalent when only one $t_i = 1$ and all other $t_i = 0$. The first form is the standard form; the second form is used in Section 3.1.

are many algorithms to minimize the loss function; most of them depend on the *gradient* $\nabla L(\boldsymbol{\theta})$ of the loss function with respect to the network parameters. The gradient can be computed using a procedure called *error back-propagation* [56], which in essence is the procedure of repeatedly applying the chain rule of differentiation. Modern deep learning toolkits, such as Theano [57], TensorFlow [58], and Torch [59], can perform error back-propagation automatically, so there is no need to derive formulas of the gradient by hand.

The easiest algorithm to minimize the loss function is *gradient descent*. It is an iterative algorithm; in each step, we compute the gradient $\nabla L(\boldsymbol{\theta})$, and update the network parameters by subtracting the gradient times a learning rate λ:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \lambda \nabla L(\boldsymbol{\theta}_i) \tag{2.4}$$

where the subscript stands for the number of iterations. Every once in a while, the network is evaluated on a validation corpus (called a *checkpoint*); if the performance on the validation corpus stops improving, the learning rate λ is reduced.

The gradient descent algorithm needs to compute the gradient on the entire training corpus before each update to the network parameters. To accelerate training, *stochastic gradient descent* (SGD) is often employed in practice. In SGD, the training corpus is divided into many *minibatches*. The network parameters are updated after scanning and accumulating the gradient on each minibatch. In this way, the parameters get updated more often, and because each minibatch offers a slightly different gradient, the parameters are less likely to get stuck in a bad local minimum. The time it takes to go over the entire training data is called an *epoch*. It is customary to shuffle the minibatches in order to avoid the network learning false knowledge from the order of the minibatches. The learning rate is also often adjusted after each complete pass over the training data, *i.e.* one checkpoint is applied per epoch. For very large training corpora, validation

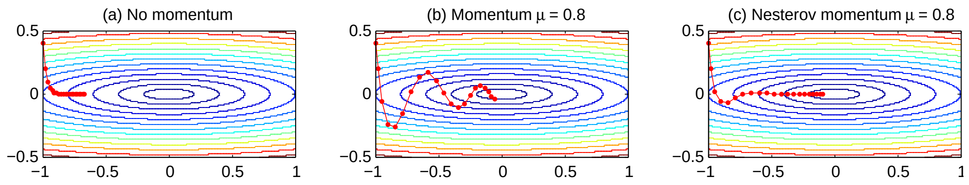


Figure 2.2: The effect of momentum in gradient descent. The loss function being minimized is $f(x, y) = x^2 + 25y^2$. The starting point is $(-1.0, 0.4)$; the learning rate is 0.01 in all the three subfigures. The red dotted curve shows the trajectory of the network parameters in the first 20 iterations.

can be performed more often, *i.e.* one epoch contains multiple checkpoints.

Another commonly used technique to accelerate training is *momentum*. With momentum, the update to the network parameters not only includes the gradient on the current minibatch, but also includes the total update in the past discounted by a momentum coefficient $\mu \in (0, 1)$. Let $\boldsymbol{\delta}_{i+1}$ be the difference between the parameters before and after the $(i+1)$-th minibatch, then the procedure of momentum-acclerated SGD can be summarized as:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \boldsymbol{\delta}_{i+1} \tag{2.5a}$$

$$\boldsymbol{\delta}_{i+1} = \mu\boldsymbol{\delta}_i - \lambda\nabla L(\boldsymbol{\theta}_i) \tag{2.5b}$$

The initial total update $\boldsymbol{\delta}_0$ is set to zero. Momentum is helpful when the loss function has a narrow ravine. When momentum is not used, in order to avoid oscillation across the ravine, the learning rate must be set to a small value, leading to slow progress along the bottom of the ravine. With momentum, the component of the gradient along the ravine adds up from iteration to iteration, resulting in faster convergence (see Fig. 2.2 (a) and (b)).

A direct improvement to the momentum method is the *Nesterov momentum* [60]. Since we know a part of the update to the parameters is adding $\mu\boldsymbol{\delta}_i$, we can use this look-ahead to evaluate the gradient at $\boldsymbol{\theta}_i + \mu\boldsymbol{\delta}_i$, instead of $\boldsymbol{\theta}_i$. The formulas of Nesterov momentum can be written as:

$$\boldsymbol{\delta}_{i+1} = \mu\boldsymbol{\delta}_i - \lambda\nabla L(\boldsymbol{\theta}_i + \mu\boldsymbol{\delta}_i) \tag{2.6a}$$

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \boldsymbol{\delta}_{i+1} \tag{2.6b}$$

The effect of Nesterov momentum is shown in Fig. 2.2 (c); the look-ahead reduces the oscillation across the ravine.

When implementing neural networks using deep learning toolkits, evaluating the gradient at a point other than $\boldsymbol{\theta}_i$ may incur a formidable amount of computation. A common practice is to redefine $\boldsymbol{\theta}'_i = \boldsymbol{\theta}_i + \mu\boldsymbol{\delta}_i$ as the network parameters. With some simple variable substitutions, Nesterov momentum can be reformulated as:

$$\boldsymbol{\delta}_{i+1} = \mu\boldsymbol{\delta}_i - \lambda\nabla L(\boldsymbol{\theta}'_i) \tag{2.7a}$$

$$\boldsymbol{\theta}'_{i+1} = \boldsymbol{\theta}'_i + \mu^2\boldsymbol{\delta}_i - (1+\mu)\lambda\nabla L(\boldsymbol{\theta}'_i) \tag{2.7b}$$

Momentum and Nesterov momentum can be regarded as indirect ways of using different learning rates for different directions in the parameter space: along the ravine, the updates in different iterations add up, effectively increasing the learning rate; in the direction perpendicular to the ravine, the updates cancel out, effectively decreasing the learning rate. A number of optimization algorithms, such as RMSprop [61], Adagrad [62], Adadelta [63], and Adam [64], maintain a different learning rate for each individual parameter directly, and often converge faster or to a better set of final parameters than simple SGD.

Because neural networks often have a huge number of parameters, overfitting can occur easily. There are various algorithms to regularize the network parameters, such as simple L_2 regularization, dropout [65], and batch normalization [66]. These algorithms also apply to the more complex network structures to be introduced in the sections below.

2.2 Recurrent Neural Networks (RNN)

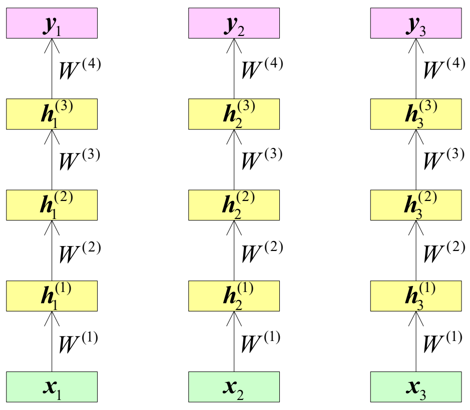
When applied to a sequence input, a feed-forward neural network processes each frame independently, as shown in Fig. 2.3 (a). This means the prediction $\boldsymbol{y}_t$ at time t is only based on the input $\boldsymbol{x}_t$ at the same moment, without using any context information, which can be important for many machine learning tasks. One way to make use of the context is to splice the input features of several consecutive frames, but this only provides limited context. A more principled way is to use a *recurrent neural network* (RNN).

The structure of an RNN is shown in Fig. 2.3 (b). The value of each hidden layer not only depends on the layer below it at the same time step, but also depends on the value of the same layer at the previous time step. Denote by $\boldsymbol{h}_t^{(l)}$ the value of the l-th hidden layer at time t, then a RNN can be described by the following formula:

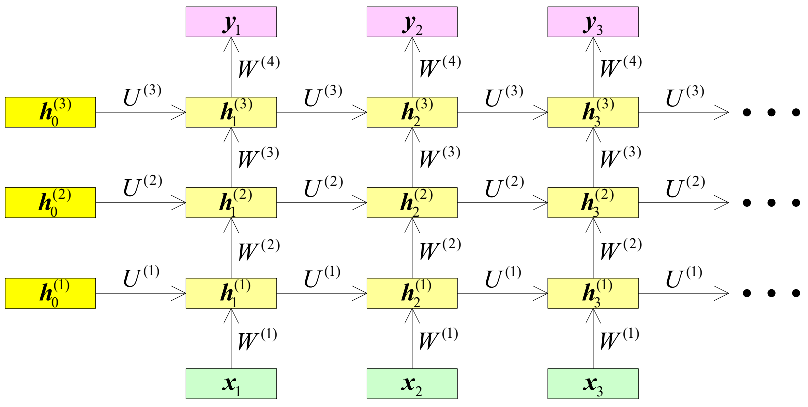
$$\boldsymbol{h}_t^{(l)} = \sigma(U^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}^{(l)}) \tag{2.8}$$

The matrix $U^{(l)}$ is the *recurrent weight matrix* of the l-th layer. The initial states $\boldsymbol{h}_0^l$ of the hidden layers may be set to zero, or they may be treated as parameters of the network and optimized during training. Equations for the first hidden layer and the output layer need to be modified slightly, but we omit them for conciseness.

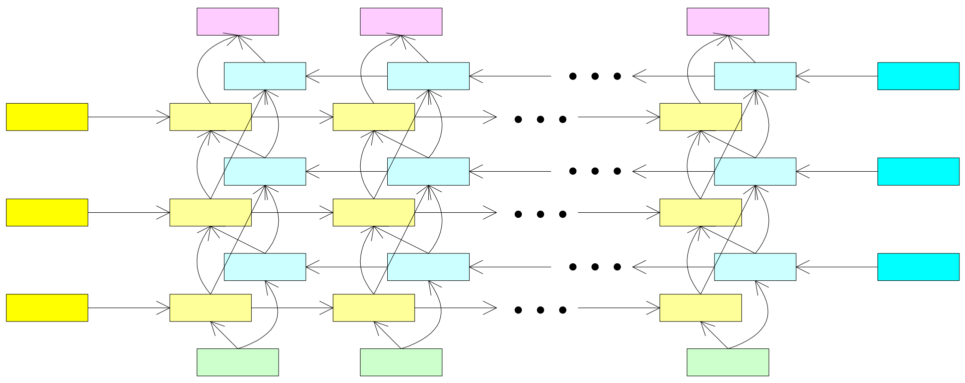
In the RNN structure described above, information flows in only one direction along the time axis. This means the prediction at any time step



(a) A feed-forward neural network applied to a time sequence. Subscripts denote time steps, and superscripts denote layers.



(b) A recurrent neural network (RNN) applied to a time sequence. Subscripts denote time steps, and superscripts denote layers.



(c) A bidirectional recurrent neural network (RNN) applied to a time sequence. To avoid clutter, the names of variables are omitted.

Figure 2.3: The structures of a feed-forward neural network, a recurrent neural network (RNN), and a bidirectional RNN.

can only make use of information at and before this time step. But the future context is often as important as the past context, so it is desirable to use a *bidirectional RNN* structure [67], as shown in Fig. 2.3 (c). Now each hidden layer consists of a forward chain and a backward chain; both chains of each layer are connected to both chains of the next layer. Besides the forward recurrent weights $\overrightarrow{U}^{(l)}$ and biases $\overrightarrow{\boldsymbol{b}}^{(l)}$, the network contains another set of parameters, the backward recurrent weights $\overleftarrow{U}^{(l)}$ and biases $\overleftarrow{\boldsymbol{b}}^{(l)}$. Let $\overrightarrow{\boldsymbol{h}}_t^{(l)}$ be the value of the forward chain in the l-th hidden layer at time t, $\overleftarrow{\boldsymbol{h}}_t^{(l)}$ the value of the backward chain, and $\boldsymbol{h}_t^{(l)}$ the concatenation of the two. The dynamics of a bidirectional RNN is described by the following formulas:

$$\overrightarrow{\boldsymbol{h}}_t^{(l)} = \sigma(\overrightarrow{U}^{(l)}\overrightarrow{\boldsymbol{h}}_{t-1}^{(l)} + W^{(l)}\boldsymbol{h}_t^{(l-1)} + \overrightarrow{\boldsymbol{b}}^{(l)}) \tag{2.9a}$$

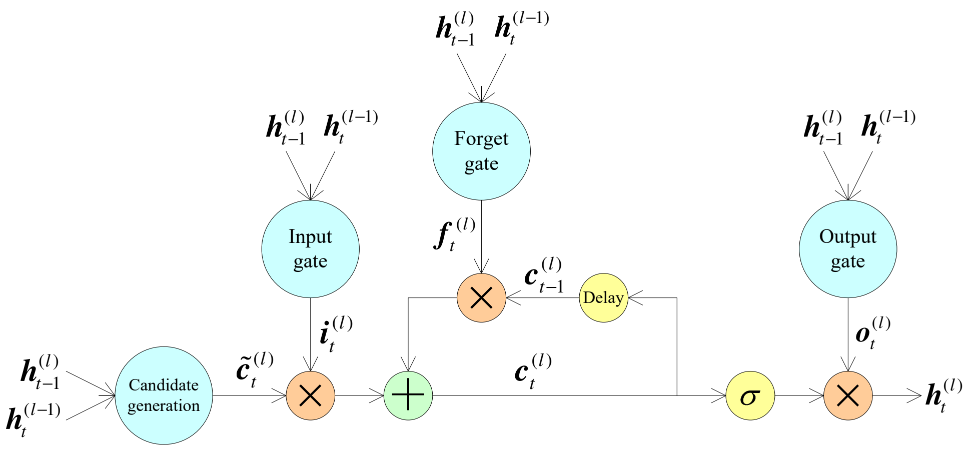
$$\overleftarrow{\boldsymbol{h}}_t^{(l)} = \sigma(\overleftarrow{U}^{(l)}\overleftarrow{\boldsymbol{h}}_{t+1}^{(l)} + W^{(l)}\boldsymbol{h}_t^{(l-1)} + \overleftarrow{\boldsymbol{b}}^{(l)}) \tag{2.9b}$$

A bidirectional RNN enjoys unlimited context, *i.e.* the prediction at any time step has access to the entire input sequence.

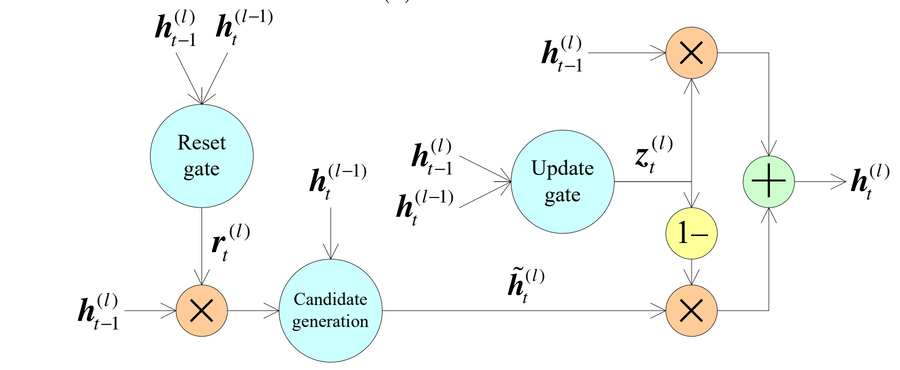
Recurrent neural networks that use any of the simple non-linear functions (Eqs. 2.2a, 2.2b and 2.2c) are called *vanilla RNNs.* Vanilla RNNs often encounter difficulty in training, due to a phenomenon called *gradient vanishing* or *gradient explosion* [68]. In RNNs, the gradient of the loss function with respect to the network parameters are computed using an algorithm called *back-propagation through time* (BPTT) [69]. As the error is propagated through time in a hidden layer, it is repeatedly multiplied by the recurrent weight matrix. If the spectral radius (*i.e.* the maximum absolute value of its eigenvalues) of the recurrent weight matrix is smaller than 1, the error will vanish, which means that distant context has little effect on the prediction. If the spectral radius is larger than 1, the error will explode, causing the training to diverge.

The gradient explosion problem can be solved by *gradient clipping*: if the absolute value of any element of the gradient exceeds a threshold Θ, then set the element to either Θ or $-\Theta$ depending on its sign. However, to solve the gradient vanishing problem, it is necessary to use more complicated non-linear functions than the ones introduced earlier. Such non-linear functions often make use of the gating mechanism, and may contain a "memory cell" to preserve information for a long time. Two widely used non-linear functions are *long short-term memory* (LSTM) cells [70] and *gated recurrent units* (GRUs) [71].

The structure of an LSTM cell is shown in Fig. 2.4 (a). It maintains two state variables: the cell state $\boldsymbol{c}_t^{(l)}$, and the output $\boldsymbol{h}_t^{(l)}$. Inputs to the LSTM cell includes the output $\boldsymbol{h}_t^{(l-1)}$ from the layer below, and the output $\boldsymbol{h}_{t-1}^{(l)}$ from the previous time step (for simplicity, we only discuss the case of unidirectional RNNs). These inputs are used to generate a candidate value



(a) An LSTM cell



(b) A gated recurrent unit (GRU)

Figure 2.4: The structures of an LSTM cell and a gated recurrent unit (GRU).

$\tilde{\boldsymbol{c}}_t^{(l)}$ of the cell state, and to control the *input gate*, *forget gate*, and *output gate*. The new cell state $\boldsymbol{c}_t^{(l)}$ may accept contributions from the candidate input $\tilde{\boldsymbol{c}}_t^{(l)}$ and the previous cell state $\boldsymbol{c}_{t-1}^{(l)}$; the input and forget gates controls whether they are turned on or off. Finally, the output $\boldsymbol{h}_t^{(l)}$ of a cell is its cell state passed through a simple non-linear function and modulated by the output gate. The behavior of an LSTM cell can be described by the following equations:

$$\tilde{\boldsymbol{c}}_t^{(l)} = \sigma_c(U_c^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_c^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_c^{(l)}) \tag{2.10a}$$
$$\boldsymbol{i}_t^{(l)} = \mathrm{sigm}(U_i^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_i^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_i^{(l)}) \tag{2.10b}$$
$$\boldsymbol{f}_t^{(l)} = \mathrm{sigm}(U_f^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_f^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_f^{(l)}) \tag{2.10c}$$
$$\boldsymbol{o}_t^{(l)} = \mathrm{sigm}(U_o^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_o^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_o^{(l)}) \tag{2.10d}$$
$$\boldsymbol{c}_t^{(l)} = \boldsymbol{i}_t^{(l)} \odot \tilde{\boldsymbol{c}}_t^{(l)} + \boldsymbol{f}_t^{(l)} \odot \boldsymbol{c}_{t-1}^{(l)} \tag{2.10e}$$
$$\boldsymbol{h}_t^{(l)} = \boldsymbol{o}_t^{(l)} \odot \sigma_h(\boldsymbol{c}_t^{(l)}) \tag{2.10f}$$

The $\odot$ sign stands for element-wise multiplication. The input, forget and output gate must use the logistic sigmoid non-linearity in order to produce values between 0 and 1. The non-linear functions σ_c for the candidate cell state and σ_h for the output are configurable; the tanh function is a popular choice.

The structure of a gated recurrent unit (GRU), shown in Fig. 2.4 (b), is simpler. It maintains only one state variable $\boldsymbol{h}_t^{(l)}$. It has a *update gate* $\boldsymbol{z}_t^{(l)}$, which has a similar role to the forget gate in an LSTM cell. There is no independent input gate; in other words, the input gate is coupled with the update gate, so that their values must sum to one. There is also no output gate; instead, a *reset gate* $\boldsymbol{r}_t^{(l)}$ is inserted between the previous and current time steps when generating a candidate state variable $\tilde{\boldsymbol{h}}_t^{(l)}$. The behavior of a GRU is described by the following equations:

$$\boldsymbol{z}_t^{(l)} = \mathrm{sigm}(U_z^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_z^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_z^{(l)}) \tag{2.11a}$$
$$\boldsymbol{r}_t^{(l)} = \mathrm{sigm}(U_r^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_r^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_r^{(l)}) \tag{2.11b}$$
$$\tilde{\boldsymbol{h}}_t^{(l)} = \sigma_h(U_h^{(l)}(\boldsymbol{r}_t^{(l)} \odot \boldsymbol{h}_{t-1}^{(l)}) + W_h^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_h^{(l)}) \tag{2.11c}$$
$$\boldsymbol{h}_t^{(l)} = (\boldsymbol{1} - \boldsymbol{z}_t^{(l)}) \odot \tilde{\boldsymbol{h}}_t^{(l)} + \boldsymbol{z}_t^{(l)} \odot \boldsymbol{h}_{t-1}^{(l)} \tag{2.11d}$$

As with LSTM cells, the update and forget gates must use the logistic sigmoid non-linearity, while the candidate state variable often adopts the tanh non-linearity.

In the LSTM structure, there is a path from the previous cell state $\boldsymbol{c}_{t-1}^{(l)}$ to the current cell state $\boldsymbol{c}_t^{(l)}$ that only goes through the multiplication with the forget gate $\boldsymbol{f}_t^{(l)}$. Likewise, in the GRU structure, there is a path from $\boldsymbol{h}_{t-1}^{(l)}$ to $\boldsymbol{h}_t^{(l)}$ that only goes through the multiplication with the update gate $\boldsymbol{z}_t^{(l)}$. When these gates stay open (*i.e.* have values close to 1) for many time steps, the error can flow back through the gates without much attenuation. This solves the gradient vanishing problem, and gives LSTM and GRU networks much longer memory than vanilla RNNs.

2.3 Convolutional Neural Networks (CNN)

Convolutional neural networks (CNNs) are another way to make use of context information in prediction. They are most widely used in image recognition (*e.g.* [55, 72]).

A convolutional neural network usually consists of *convolutional layers*, interweaved with *pooling layers*. The data passed between the layers are in the form of 3-dimensional tensors, each slice of which is called a *feature map*. We denote the p-th feature map at the output of the l-th layer by the matrix by $H_p^{(l)}$. The parameters of a convolutional layer include a 4-dimensional kernel tensor $W^{(l)}$ and a 3-dimensional bias tensor $B^{(l)}$. Let $W_{pq}^{(l)}$ and $B_p^{(l)}$ be 2-dimensional slices of the kernel and bias tensors, then the behavior of a convolutional layer is described by:

$$H_p^{(l)} = \sigma\left(\sum_q [W_{pq}^{(l)} * H_q^{(l-1)}] + B_p^{(l)}\right) \tag{2.12}$$

where the asterisk stands for 2-dimensional convolution, and σ is a non-linear function.

The behavior of pooling layers is simpler. A $m \times n$ pooling layer divides each input feature map into regions of $m \times n$ pixels, and computes a statistics for each region as the output. The most common statistics include the maximum and the average. An optional non-linearity may be applied to the pooling result.

When applied to image recognition, the neural network only needs to make one prediction for an entire image, which is represented as 1 (for gray-scale images) or 3 (for color images) huge feature maps. The layers are usually arranged in a way such that convolutional layers increase the number of feature maps, and pooling layers reduce the size of the feature maps. When the feature maps are sufficiently small, they are often flattened into one single vector, followed by one or more fully connected layers to make the prediction.

CNNs may be applied to speech signals in two ways. The first way is to take the spectrogram or filterbank outputs as input. In this case, the input is a 2-dimensional feature map whose axes are time and frequency, and can be treated the same way as an image. The second way is to take the raw waveform as input. In this case, the input is a 1-dimensional feature map, which means the convolutional layers should perform 1-dimensional convolutions instead of 2-dimensional ones. In speech tasks, we often want a sequence output at a certain frame rate (*e.g.* 10 ms for speech recognition, 100 ms for sound event detection). To achieve this, we can stop using pooling layers when the step size along the time axis of the feature maps have been reduced to the desired value.

The benefits of using CNNs for image recognition are *shift invariance* and *locality*. Shift invariance means that the prediction for an image does not change when the object of interest moves within the image. This is irrelevant for speech applications: shifting a sound in the input audio along the time axis should also shift the output; shifting a sound along the frequency changes the quality of the sound and may also affect the output. Locality means each neuron in a CNN only receives information from neurons representing a neighboring region in the layer below. This implies that CNNs, when applied to speech, do not have unlimited context as RNNs do. The range in the input audio that may provide information for a given neuron is called the *receptive field* of the neuron; it is crucial to design CNNs whose neurons have receptive fields of appropriate sizes.

2.4 Connectionist Temporal Classification (CTC)

In any of the neural network architectures introduced so far, the loss function is the sum or average of the loss at each frame. To define the loss at each frame, it is necessary to have frame-wise supervision. For example, in speech recognition, we need to know the exact onset and offset times of each phoneme or sub-phonemic state; in sound event detection, we need to know the onset and offset times of each sound event occurrence, or, in other words, which events are active at each frame. The sequence formed by concatenating frame-level labels is called an *alignment*. In sequence learning tasks, the supervision often comes in the form of a label sequence without alignment; while sometimes it is possible to create alignments, it can take extra effort. *Connectionist temporal classification* (CTC) [73] is one way of defining a sequence-level loss function that depends only on the label sequence, making it possible to train neural networks without alignment.

The CTC loss function on a single training sequence is the negative log-likelihood of the probability of the label sequence. It is the sum of the probabilities of all alignments that can be mapped to the label sequence. The simplest way to map an alignment to a label sequence is to reduce all consecutive repeating tokens to a single one. For example, if the outputs at the six frames of a sequence are `ABBBAA`, then it will map to the label sequence `ABA`. This mapping function has two drawbacks: (1) it cannot produce label sequences with consecutive repeating tokens, such as `ABBA`; (2) it requires that each frame must output a token, while it can make more sense to allow some frames (*e.g.* silent frames) to output nothing. To overcome these drawbacks, CTC adds a *blank* token, denoted by "`-`", to the frame-wise output vocabulary. The mapping function works in two steps: first, it reduces consecutive repeating tokens into a single one; second, it removes the blank tokens. In this way, the alignments `-AB--BAA` and `ABBB-BBA` will both map to the label sequence `ABBA`.

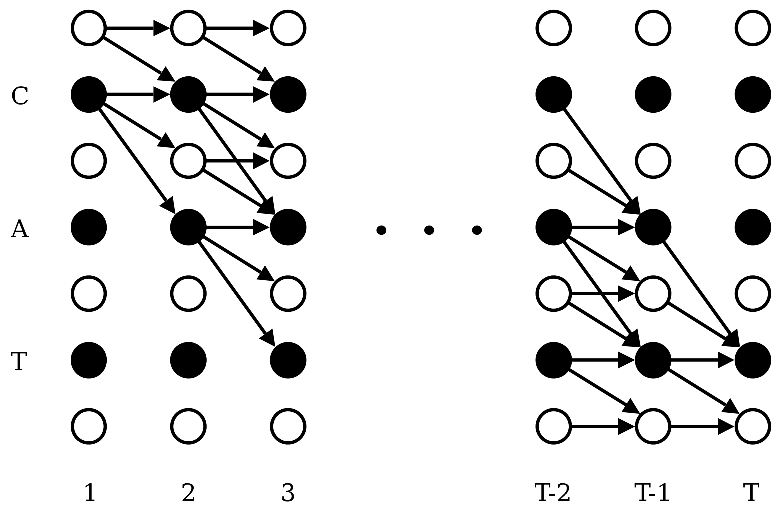


Figure 2.5: The trellis for computing the CTC loss function, taken from [73]. The target label sequence is `CAT`. Black circles represent non-blank tokens, and white circles represent blanks.

Given the frame-level output distributions, the total probability of a label sequence can be computed using a dynamic programming algorithm, similar to the forward algorithm in HMMs [74]. The computation is conducted on the trellis shown in Fig 2.5. The horizontal axis, which goes from 1 to T, stands for time steps; the vertical axis represents the target label sequence with blank tokens inserted at the beginning, at the end, and between every pair of tokens. Each alignment that can map to the target sequence corresponds to a path in the trellis.

We denote by L the target label sequence augmented with blanks, and its i-th token by L_i. Also let $y_t(c)$ be the probability of the token c in the output distribution at step t. Define $\alpha_t(i)$ as the total probability of the paths landing on L_i at time t and emitting all the tokens along the way. At each time step, the path is allowed to stay at the same token, transition to the next token, or skip over the next token, but the skipping can only happen when the token skipped over is a blank, and the two tokens around it are different. CTC assumes that the outputs at all the time steps are mutually independent given the input sequence, because context dependency can be taken care of by the recurrent layers of the network. Therefore it does not model transition probabilities, and the α's are computed with the following recurrence formula:

$$\alpha_t(i) = \begin{cases} y_t(L_i) \sum_{j=1}^{i} \alpha_{t-1}(j), & \text{if } i \leq 2 \\ y_t(L_i) \sum_{j=i-1}^{i} \alpha_{t-1}(j), & \text{if } i > 2, \text{ and } L_i = L_{i-2} \\ y_t(L_i) \sum_{j=i-2}^{i} \alpha_{t-1}(j), & \text{if } i > 2, \text{ and } L_i \neq L_{i-2} \end{cases} \tag{2.13}$$

The path is allowed to start at either the first non-blank token or the blank

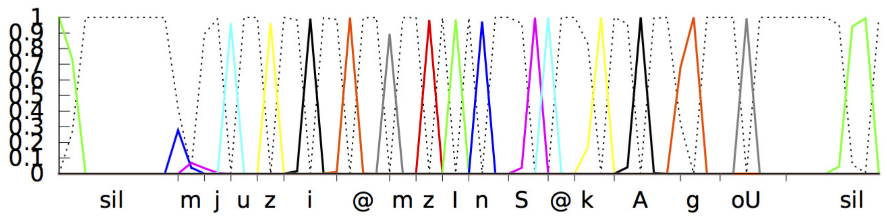


Figure 2.6: The "peaky" output of a CTC speech recognition network on the utterance "*museums in Chicago*". Each colored line stands for the probability of a phoneme; the dotted line stands for the probability of the blank token. Taken from `https://research.googleblog.com/2015/09/google-voice-search-faster-and-more.html`.

before it, so the α's are initialized as:

$$\alpha_1(i) = \begin{cases} y_1(L_i), & \text{if } i \leq 2 \\ 0, & \text{if } i > 2 \end{cases} \tag{2.14}$$

The path can finish at either the last non-blank token or the blank after it, so the total probability of the target label sequence is

$$P(L) = \alpha_T(|L| - 1) + \alpha_T(|L|) \tag{2.15}$$

where $|L|$ is the length of the sequence L.

Decoding on a network with a CTC output layer can be performed in two ways. The theoretically correct way is to find the label sequence that has the largest total probability. Doing so would require *prefix search*, and it is relatively hard to implement. A simpler way of decoding is *best path decoding*. It takes the token that has the maximum probability at each time step to form an alignment, and then maps the alignment to a label sequence. Because CTC does not model transition probabilities, this is equivalent to finding the *best path* through the trellis, and mapping the path to a label sequence. Best path decoding is a reasonable approximation of prefix search decoding.

The evolution of the output of a CTC network during training exhibits an interesting pattern [75]. In the first few epochs, the network may go through a "warm-up" stage, in which the output distributions at all time steps are dominated by the blank symbol. Afterwards, "peaks" will occur in the probabilities of non-blank tokens, and the sequence formed by reading off the tokens corresponding to the peaks will approximate the target label sequence (see Fig 2.6). The peaks normally do not span the entire duration of a phoneme, but only last one or two frames. In speech recognition, the positions of the peaks have been found to match the positions where the phonemes actually occur.

Chapter 3

Sound Event Detection with Sequential Labeling

This chapter describes the first step of our effort toward sound event detection using weak labeling. Instead of having the exact onset and offset times of each sound event occurrence, we try to learn sound events from annotations that only specify the *order* of sound events. In the case of polyphonic SED, sound events may overlap, making it hard to define sequences of sound events. To overcome this difficulty, we use annotations that specify sequences of *event boundaries* (*i.e.* onsets and offsets), instead of the events themselves. We call such annotations *sequential labeling*. Sequential labeling is easier to produce manually than strong labeling; moreover, they may be mined from textual descriptions of audio recordings. For example, the textual description "a dog barks while a car passes by" may be mapped to the following sequence of event boundaries: `engine_start, animal_start, animal_end, engine_end`. Recurrent neural networks with a *connectionist temporal classification* output layer (CTC-RNN) are well suited for learning from this form of supervision.

The content of this chapter is organized as follows. Section 3.1 describes a recurrent neural network (RNN) we trained with strong labeling for monophonic SED. This network is used for pre-training the CTC-RNN for polyphonic SED. Section 3.2 describes our experiments of polyphonic SED with a CTC-RNN on the Noiseme corpus [51]. To enforce that the CTC-RNN generate peaks around the actual locations of sound event boundaries, we strengthen the supervision a little bit by *alignment hinting*. The CTC-RNN suffers badly from overfitting. To reduce the overfitting, in Section 3.3, we replace the original low-level acoustic features with features extracted from SoundNet [53]. SoundNet is a convolutional network that tries to predict video information from audio input, and is an example of *transfer learning*. We show that using SoundNet as a feature extractor slightly improves the generalization power of the CTC-RNN, and greatly accelerates its convergence. Section 3.4 analyzes the errors made by the CTC-RNN, and points out that data scarcity is the bottleneck that limits the generalization power. Finally, in Section 3.5, we propose ideas to address this bottleneck as well as other problems to improve the SED performance, including semi-supervised learning from large weakly-labeled data, better exploiting long events, improving the temporal localization, *etc.*

The work in this chapter has been published in [8, 34, 35].

3.1 Monophonic SED with Strong Labeling

We first built a bidirectional RNN to perform monophonic SED with strong labeling. The annotation contained the exact onset and offset times of each sound event occurrence, or, in other words, which sound events were active at each frame. The network predicted active sound events on a frame-by-frame basis, and at most one sound event was predict at each frame. The network went through a series of improvements between the publications of [8] and [34]; here we describe the final version.

The network was trained on Version 2 of the Noiseme corpus [51], which contained 7.9 hours of audio and 17 sound event types. The corpus was partitioned into training, validation and test sets with a duration ratio of 3:1:1, and 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.

We extracted acoustic features using the OpenSMILE toolkit [76, 77]. We first extracted low-level features such as MFCCs and 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 feature sequences with a frame rate of 10 Hz. The dimensionality of the feature vectors was 6,669, 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 globally normalized to span the range $[-0.9, 0.9]$.

We implemented the bidirectional RNN using the Theano toolkit [57]. The network had an input layer with 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. The output layer contained 18 nodes in a softmax group, corresponding to the 17 sound event types plus a "background" type. The network was trained using the categorical cross-entropy loss function. Because multiple sound events might be active at the same time, we used the second form in Table 2.1 ($L(\boldsymbol{y}, \boldsymbol{t}) = -\log \sum_i t_i y_i$), *i.e.* we wanted to maximize the total probability of all active events, without caring about how the probability mass was distributed among them. The evaluation metric was *frame accuracy*. Also considering overlapping sound events, we regarded a frame as correctly classified if any of the ground-truth sound event types got the highest predicted probability. One minus the frame accuracy is called the *frame error rate*.

The network was trained with the stochastic gradient descent (SGD) algorithm. Each minibatch consisted of 5 sequences of 500 frames; longer training sequences were split into sequences shorter than 500 frames, cutting in the middle of silence segments whenever possible. The initial learning rate was 0.005, and we applied a Nesterov momentum coefficient of 0.9. We adopted an adaptive learning rate schedule: after every epoch, if the validation frame error rate decreased by more than 1% relative, the learning rate was increased by 5%; when the validation frame error decreased by less than 0.5% relative, the learning rate was decreased by 20%; training was terminated when the validation frame error rate decreased by less than 0.1% relative for 5 consecutive epochs.

We found two tricks helpful for improving the frame accuracy. The first trick was setting the initial bias of the forget gates to one, in order to encourage remembering in the early stages of training. This is an effective practice first proposed in [78], and emphasized in [79]. Biasing the forget gates increased the frame accuracy by 1.8% absolute [34]. The second trick was 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). This multiplied the amount of training data by four. During training, we adjusted the learning rate after every quarter pass through the augmented training data; during testing, we averaged the probabilities predicted on the four copies of features for each audio file before selecting the maximum. Data augmentation contributed another 0.7% to the frame accuracy.

We trained four such networks using different random initializations. The average frame accuracy of the four networks was 54.0%. The single best network reached an accuracy of 55.5%, and was used to initialize the CTC model.

3.2 Polyphonic SED with Sequential Labeling

Building upon the bidirectional RNN for frame-wise monophonic SED, we continued to train a CTC-RNN that learnt to perform polyphonic SED from weak labeling. Even though we had the exact onset and offset times of the sound event occurrences in the annotation, we discarded the timing information to produce weak labeling in the form of sequences of event boundaries. For example, if the content of an audio recording could be described as "a dog barks while a car passes by", then the sequence of event boundaries would be `engine_start, animal_start, animal_end, engine_end`.

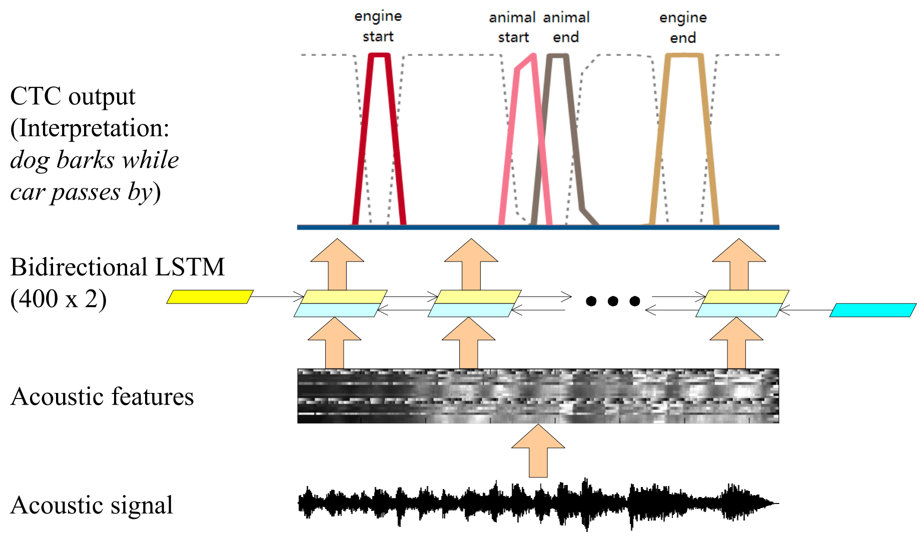

engine start
animal start
animal end
engine end
CTC output (Interpretation: *dog barks while car passes by*)
Bidirectional LSTM (400 x 2)
Acoustic features
Acoustic signal

Figure 3.1: Structure of the CTC-RNN for polyphonic SED with sequential labeling.

3.2.1 Training the CTC-RNN

The CTC-RNN was also implemented using the Theano toolkit. The structure of the CTC-RNN is shown in Fig. 3.1. It accepted the same form of input as the frame-wise RNN, and the structure was also identical up to the hidden layer. The output layer now contained 35 neurons in a softmax group – two for the onset and offset of each of the 17 sound event types, and one for the blank token. The network was trained to minimize the per-frame CTC loss. That is, let P_i be the total probability of all alignments that can be mapped to the ground-truth sequence of event boundaries for the i-th training sequence, and T_i be its duration, then the objective function to be minimized was $L = -\sum_i T_i \log P_i / \sum_i T_i$.

The network was also trained on Version 2 of the Noiseme corpus. Data augmentation was not applied this time; we only used the left channel and OpenSMILE 1.0.1. The training algorithm was SGD with a Nesterov momentum of 0.9. The learning rate was initialized to 0.3; it was kept constant until 200 epochs, and decayed with a factor of 0.99 every epoch until reaching 500 epochs.

We found it necessary to apply gradient clipping in order to avoid the gradient explosion problem. Without gradient clipping, the magnitudes of the gradients in the first few epochs would be larger than in subsequent epochs; this would force us to use a smaller learning rate which would slow down the convergence. Moreover, without gradient clipping, a single large value in the gradients could result in an abrupt surge in the training loss, which could take up to 100 epochs to compensate for, or even cause the

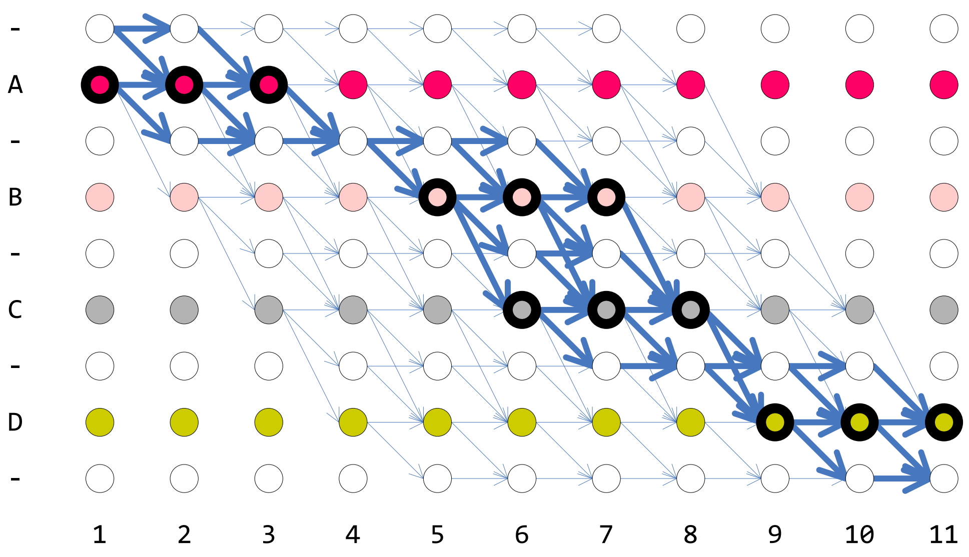


Figure 3.2: An example of alignment hinting with tolerance $k = 1$. The horizontal axis stands for time, and the vertical axis stands for the augmented label sequence. The ground-truth timestamps of the four non-blank tokens are Frames 2, 6, 7, 10 respectively. If no alignment hinting was applied, the total probability of the label sequence `ABCD` would be the sum of the probabilities of all paths that go through the arrows. With alignment hinting, the path must go through the bold states for non-blank tokens, and only paths that go through the bold arrows are included in the sum.

training to crash. We found 0.001 to be a good clipping limit.

By inspecting the output of the CTC-RNN on the 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 positions where the boundaries of the sound events occurred (see Fig. 3.4 (a)). This means the model picked up spurious patterns from random positions in the feature sequences. In order to lead the CTC-RNN to find the correct alignment, we strengthened the annotations a little bit by approximately using the timing information in the annotations. 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 L_i in the augmented token sequence is not a blank, and occurs at frame t_i according to the annotation, then all $\alpha_t(i)$ with $|t - t_i| > k$ would be set to zero. This constraint still leaves 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. We call this technique *alignment hinting*; Fig. 3.2 shows an example when the tolerance $k = 1$.

3.2.2 Quantitative and Qualitative Evaluation

Evaluation of the CTC-RNN was conducted from two aspects. Quantitatively, we computed the token error rate (TER) of the network output. We decoded the CTC output with best path decoding to obtain sequences of sound event boundaries, compared them with the ground-truth sequences, and computed the TER in the same way as word error rate (WER) in speech recognition. Qualitatively, we inspected the probability distribution predicted by the CTC-RNN at each frame to see whether it produced peaks for sound event boundaries at the right positions.

Fig. 3.3 shows the effect of gradient clipping and alignment hinting on the evolution of the training loss and TER. Because we didn't use any cross-validation, we used both the validation and test sets (40% of the entire corpus) as testing data. The first three curves indicate that gradient clipping not only avoided the surges in the training loss and TER curves, but also allowed us to use a larger learning rate (0.3 vs 0.1). The last three curves indicate that alignment hinting helped the model to converge faster and to a better solution. With a hinting tolerance of $k = 5$ (*i.e.* windows of 1 second), the final training TER was 13%, and the final test TER was 81%. Even though these numbers were lower than the unhinted case, overfitting remained a problem.

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. 3.4, we plot the output of the networks with and without alignment hinting on some training and test recordings.

Graph (a) shows the output of the network without alignment hinting on a training recording. At the middle of this recording were three cannon shots, signified by the `pulse, white, nature` sequence repeated three times. The unhinted network was able to recover this sequence correctly, but most tokens were placed far away from the positions where they really occurred, and the tokens tended to cluster together. Graph (b) shows the output of a hinted network (with tolerance $k = 5$) on the same recording. Now we see that the alignment hinting forced the peaks to be generated near the actual starts and ends of the sound events.

Graphs (c) and (d) show the output of the same hinted network on two test recordings. The CTC-RNN was able to detect the span of some sound event instances (*e.g.* the speech segments in graph (d)), notably transient ones (*e.g.* the pulses in graph (c)). However, many sound events were still missed (*e.g.* the cheering, music and engine noise).

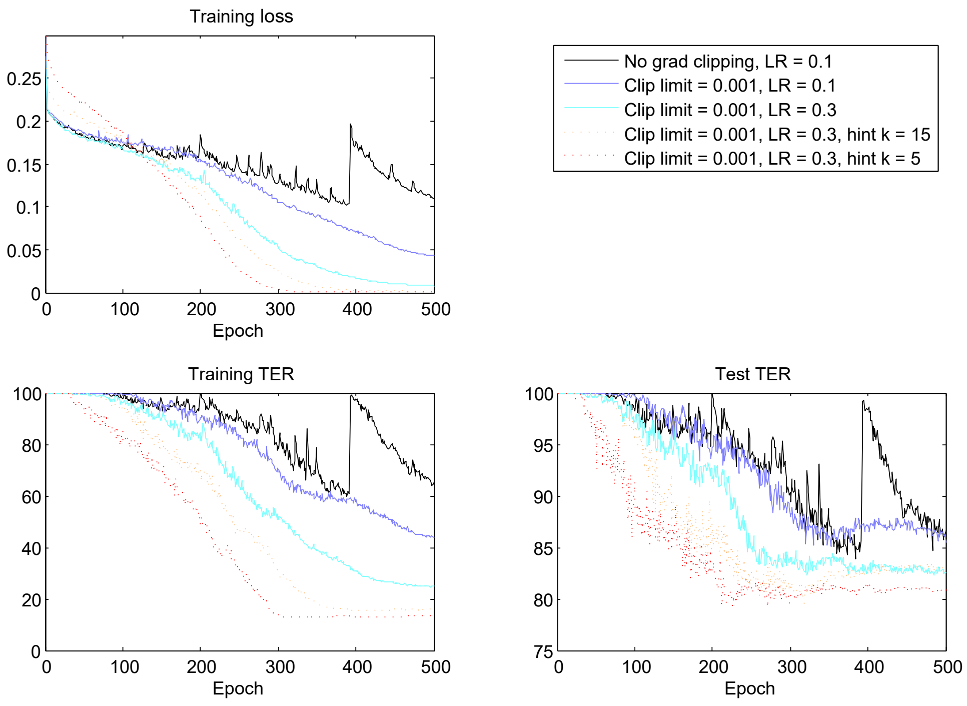


Figure 3.3: Training curves of the CTC-RNN with different gradient clipping limits, initial learning rates, and alignment hinting tolerances. The tolerances $k = 15$ and $k = 5$ correspond to windows of 3 seconds and 1 second. Note the different scales of the training and test TERs, which indicates severe overfitting. Best viewed in color.

3.3 Improving the Acoustic Features with Transfer Learning

Even though the CTC-RNN was able to detect some sound events (*e.g.* speech and pulses), it still generalized poorly to new data. We surmise that the limited amount of training data was a main reason. Not only did this limit the number of instances that the network could learn from, but it also restricted the complexity of the network so it couldn't enjoy all the benefits of deep learning. Since Google Audio Set [52] was not available yet at this time, we resorted to *transfer learning* to solve these problems. Transfer learning is the technique of taking knowledge learned from a task with larger data ("source task") and applying it to a related task with smaller data ("target task"). It has been successfully applied to acoustic scene recognition in [80], where the knowledge was transferred from six other acoustic scene recognition tasks with different label sets than the target task. In our work, we transferred knowledge from a visual object and scene recognition task to help sound event detection. The knowledge was distilled

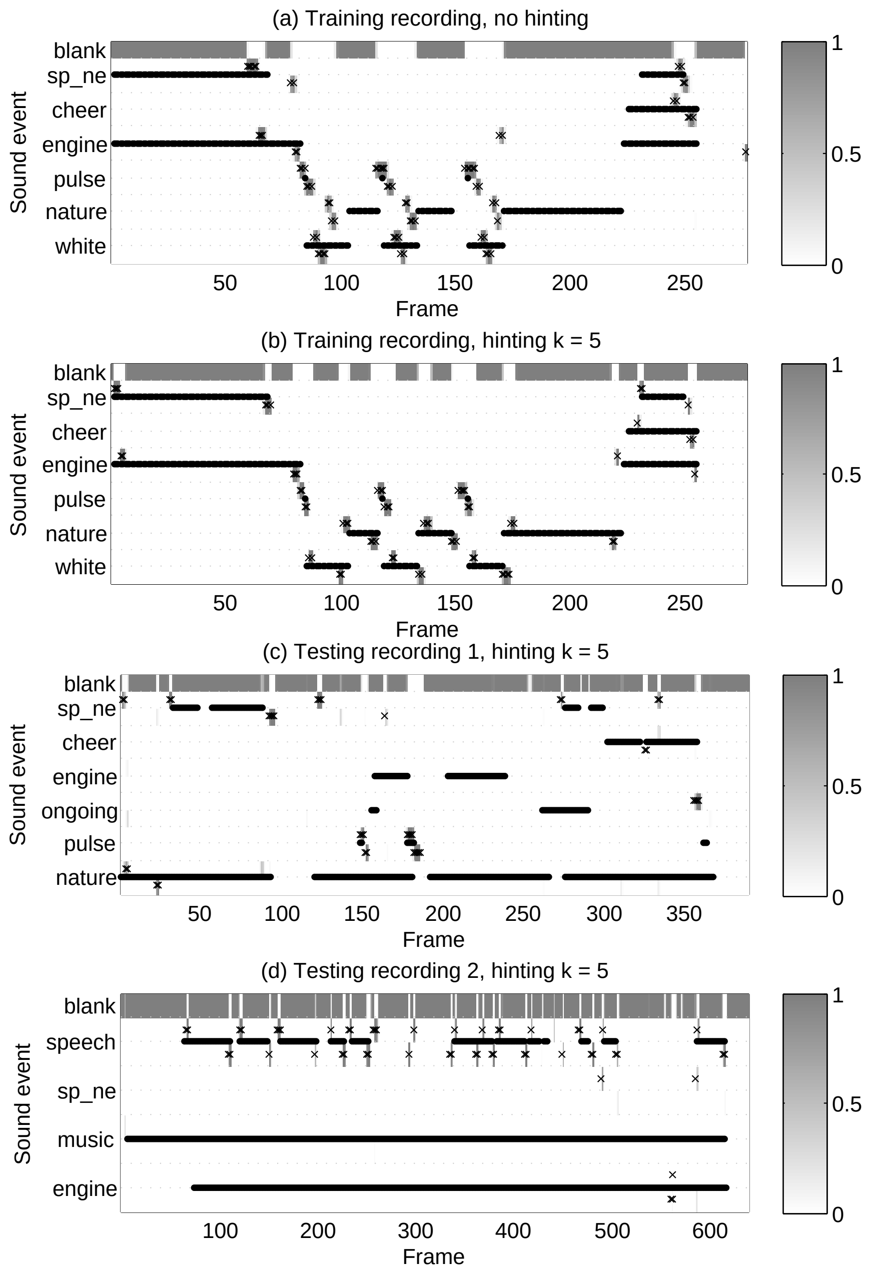


Figure 3.4: Example predictions of the CTC-RNN on some training and test recordings. Each row of the graphs stands for an output token; each sound event type is associated with two rows – its start and end tokens. Shades of gray signify the output probability. Crosses mark the most probable token at 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. "Sp_ne" stands for "non-English speech". Unimportant sound events for these examples are omitted.

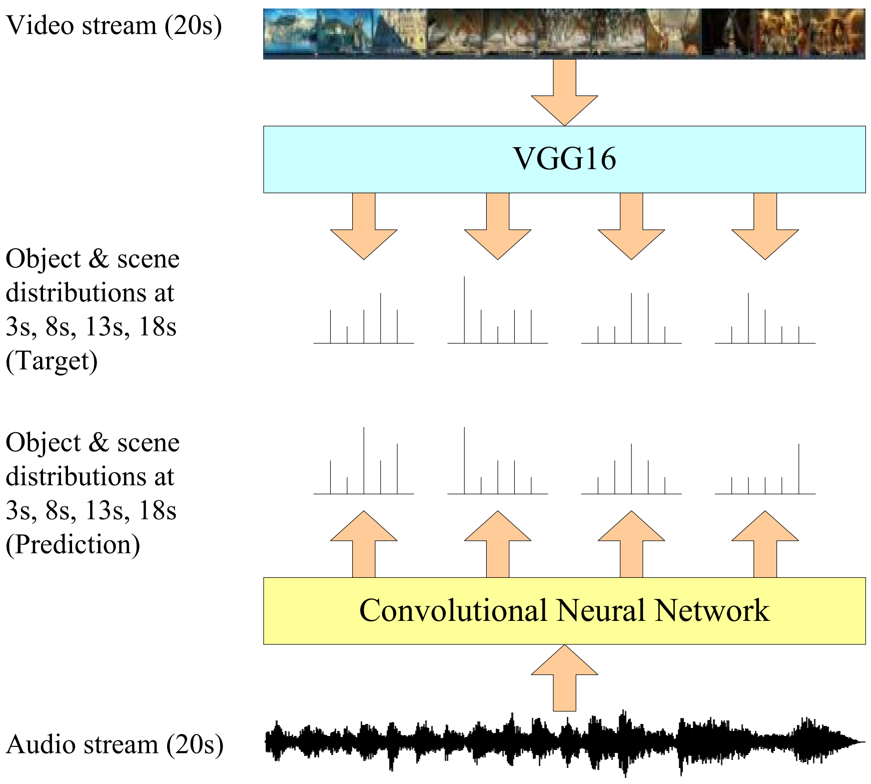

Video stream (20s)
VGG16
Object & scene distributions at 3s, 8s, 13s, 18s (Target)
Object & scene distributions at 3s, 8s, 13s, 18s (Prediction)
Convolutional Neural Network
Audio stream (20s)

Figure 3.5: The architecture of SoundNet [53]. It is trained to minimize the KL divergence between the predicted distributions and target distributions.

in a deep convolutional network called SoundNet [53]; we used it as a feature extractor for the CTC-RNN, so the latter could enjoy the deep structure and vast training data of SoundNet.

3.3.1 The Structure of SoundNet and Its Variants

The overall architecture of SoundNet is shown in Fig. 3.5. It is a deep convolutional network that takes raw waveforms as input, and tries to predict the objects and scenes in video streams at certain points. The ground truths of the objects and scenes are produced by the image recognition network VGG16 [55]. Even though what can be seen in the video may not always be heard in the audio and *vice versa*, with sufficient training data, the network can still be expected to discover the correlation between the audio and the video. After the network is trained, the activations of an intermediate layer can be considered a representation of the audio suitable for both visual object and scene recognition and sound event detection. Actually, SoundNet has outperformed other features and models by a significant margin in the acoustic scene classification task of the DCASE

Layer	input	conv1	pool1	conv2	pool2	conv3	conv4	conv5	pool5	conv6	conv7	conv8 (output)
# feature maps	1	16	16	32	32	64	128	256	256	512	1024	1000 + 401
Filter size		64		32		16	8	4		4	4	4
Activation		relu		relu		relu	relu	relu		relu	relu	softmax
Batch norm.		yes		yes		yes	yes	yes		yes	yes	
Subsampling		2	8	2	8	2	2	2	4	2	2	2
Frame rate (Hz)	22,050	11,025	1,378	689	86	43	21.5	10.8	2.69	1.35	0.67	0.34
Reception Field		2.9 ms	3.5 ms	26 ms	36 ms	0.21 s	0.37 s	0.51 s	0.79 s	1.91 s	4.13 s	8.59 s

(a) The layers of the original SoundNet.

Layer	input	conv1	pool1	conv2	pool2	conv3	conv4	conv5	pool5	fc1	fc2	fc3	output
# feature maps	1	16	16	32	32	64	128	256	256	100	100	100	1000 + 401
Filter size		64		32		16	8	4					
Activation		relu		relu		relu	relu	relu		tanh	tanh	tanh	softmax
Batch norm.		yes		yes		yes	yes	yes					
Subsampling		2	5	2	5	2	2	2	2				
Frame rate (Hz)	16,000	8,000	1,600	800	160	80	40	20	10	10	10	10	10
Reception Field		4.0 ms	4.5 ms	24 ms	29 ms	0.12 s	0.21 s	0.29 s	0.34 s	0.34 s	0.34 s	0.34 s	0.34 s

(b) The structure of SN-F, with layers above "pool5" replaced by fully connected layers.

Layer	gru1	gru2	gru3	output
# feature maps	100 × 2	100 × 2	100 × 2	1000 + 401
Activation	relu	relu	relu	softmax
Batch normalization	yes	yes	yes	
Frame rate (Hz)	10	10	10	10

(c) The higher, recurrent layers of SN-R. Layers up to "pool5" are identical to SN-F.

Table 3.1: Detailed information about the layers of SoundNet and its two variants, SN-F and SN-R.

2013 challenge and a sound event classification task on the ESC-50 corpus [53].

Information about the layers of SoundNet is listed in Table 3.1 (a). The input is a 20-second, monaural waveform with a sample rate of 22,050 Hz. The network has seven hidden convolutional layers, interspersed with max-pooling layers. Each convolutional layer doubles the number of feature maps and halves the frame rate; each max-pooling layer halves the frame rate as well. The output layer is also convolutional. It has 1,401 output units, split into two softmax groups of sizes 1,000 and 401, standing for the distributions of objects and scenes, respectively. The output layer of SoundNet has a frame rate of about 1/3 Hz. This corresponds to about 6.7 frames, but considering boundary effects, the actual output only contains the distributions of objects and scenes at 4 time steps.

To localize the onsets and offsets of sound events with reasonable precision, the CTC-RNN for sound event detection must run at a sufficient frame rate. In our previous experiments, this frame rate was set to 10 Hz. In SoundNet, only one layer ("conv5") has a frame rate close to this value. Therefore, we used the lower part of SoundNet (up to layer "conv5") as a feature extractor for the CTC-RNN. These features replaced the 50 dimensional PCA features used in previous experiments.

It might be expected that higher layers of SoundNet would compute representations of the input audio more closely related to objects and scenes, or closer to sound events. However, these layers in SoundNet were subsampled too much to be used for SED. In order to make use of the information in the higher layers, we trained two variants of SoundNet, SN-F and SN-R. Instead of using convolutional layers all the way up, we switched to *fully connected* (SN-F) or *recurrent* (SN-R) layers after the frame rate had been reduced to the desired value of 10 Hz. After three fully connected or recurrent layers, a fully connected output layer would perform the object and scene classification. Also, we changed the input sampling rate to 16,000 Hz to match the Noiseme corpus [51]. The structures of SN-F and SN-R are summarized in Tables 3.1 (b) and 3.1 (c). The values at the "pool5" layer or any higher layer could be used as input features for the CTC-RNN.

3.3.2 Training SoundNet and Its Variants

The original SoundNet was trained using the Torch [59] toolkit. As introduced in Section 1.2, the training data contained 2 million 20-second video excerpts, while the validation data was about 1/15 the size of the training data. Target object and scene distributions were extracted from keyframes at 3 s, 8 s, 13 s and 18 s of each video. The network predicted object and scene distributions at four moments in each excerpt which were 3 seconds apart; the misalignment between the timestamps of the target and predicted distributions was ignored. The network was trained to minimize the sum of the average KL divergence of the object distributions and that of the scene distributions. The optimizer was Adam [64] with a fixed learning rate of 0.001 and a momentum of 0.9. Each minibatch contained 64 videos, and the network was trained for 100,000 minibatches (about 3 epochs). Batch normalization was applied after each convolutional layer.

We trained SN-F and SN-R using the same data as SoundNet, but randomly selected 1,000 videos from the validation data to speed up the training. The two networks were implemented using the Keras [81] toolkit. Even though they predicted object and scene distributions every 0.1 s, only the predictions at 3 s, 8 s, 13 s and 18 s were used to compute the loss function (total KL divergence of objects and scenes), measured in nats per keyframe. We also used the Adam optimizer and a batch size of 64 videos, but used a decaying learning rate and no momentum. Because the training corpus was huge, we checked the loss on the 1,000-video validation set after every 160 minibatches (about 0.5% of an epoch), and decayed the learning rate by a factor of 0.9 when the minimum validation loss did not see any update for 5 checkpoints. We found this decay helpful for the network to reach a lower loss.

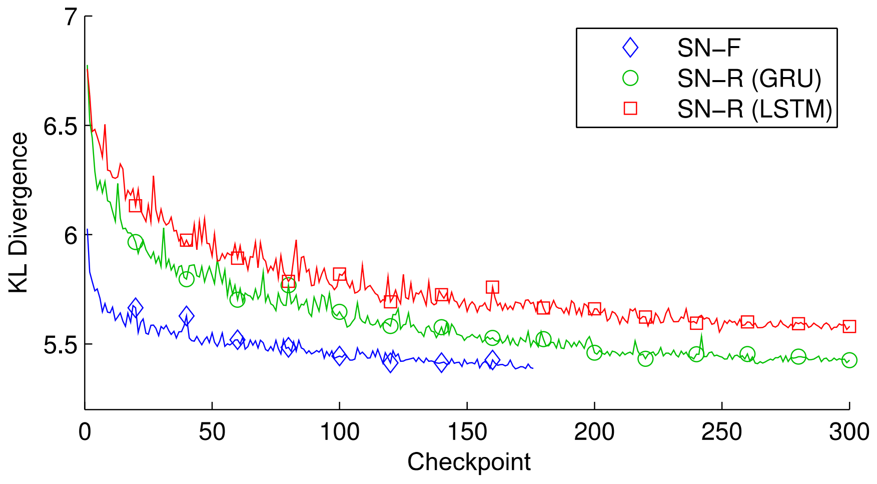


Figure 3.6: Training the variants of SoundNet: The evolution of the validation KL divergence of SN-F and SN-R, the latter using either GRU or LSTM cells.

We studied the effect of the recurrent cell type for SN-R, as well as the effect of the activation function. It turned out that GRU cells [71] reached a lower KL divergence than LSTM cells [70], but the activation function did not make a difference for either SN-F or SN-R. In Fig. 3.6, we plot the evolution of the validation loss of SN-F and SN-R, all using the "tanh" activation function. SN-F converged faster thanks to its simpler structure; by Checkpoint 175 (about 90% of an epoch), it reached a validation loss of 5.39. We trained SN-R until Checkpoint 300. With LSTM cells, the final validation loss was 5.58; with GRU cells, 5.43. For comparison, the loss of the original SoundNet on the 1,000-video validation set (3,418 frames) was 5.15, but this number was measured after excluding about 2% of the frames, because on these frames SoundNet predicted zero probabilities for some object or scene classes.

We also studied the effect of batch normalization [66]. The original SoundNet used batch normalization for all the convolutional layers, and we found it essential to do the same. In the fully connected or recurrent layers, batch normalization made no difference on the KL divergence, but we found it to slightly improve the SED performance of SN-R. Consequently, for the experiments in the next subsection, we used a SN-R with GRU cells, the ReLU non-linearity and batch normalization in the recurrent layers, as described in Table 3.1 (c). Because the non-linearity is not the final step of computation in GRU cells, batch normalization was applied *after* all the GRU computation. This is different from the convolutional layers, where batch normalization was performed *before* the non-linearity.

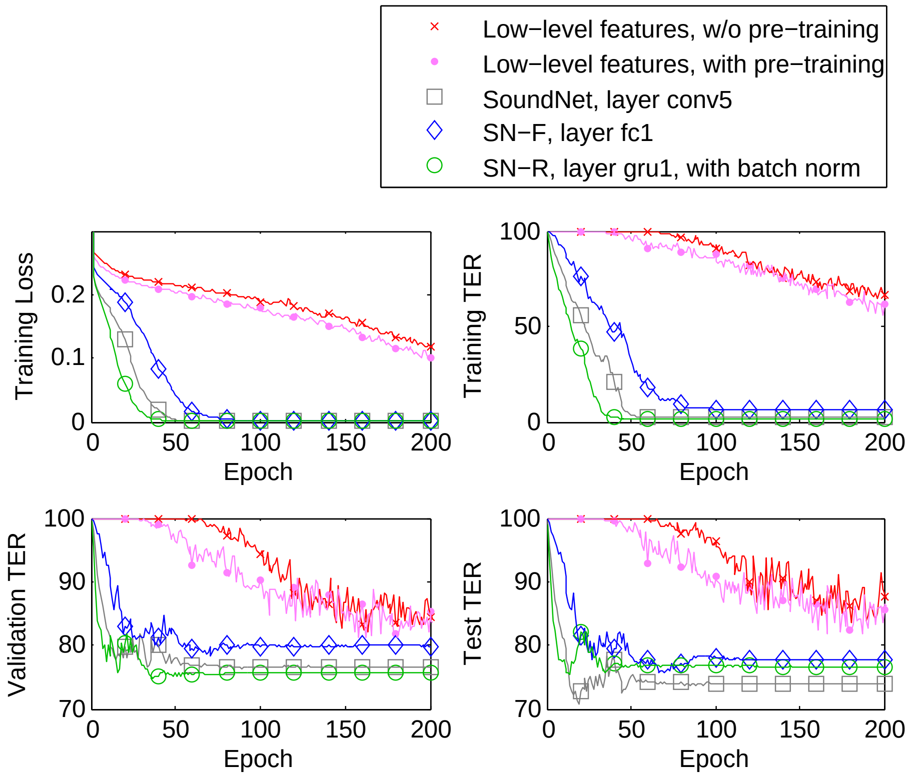


Figure 3.7: Training the CTC-RNN for sound event detection: The evolution of the training loss and the token error rate (TER) on the training, validation and test sets, using either low-level acoustic features or transfer learning features extracted from SoundNet or its variants. Note that low-level features do not yet achieve convergence at 200 epochs. Also note the different scales of the training TER vs the validation and test TERs, which indicates severe overfitting. Best viewed in color.

3.3.3 SED Using Transfer Learning Features

We repeated the SED experiments in Section 3.2, replacing the 50-dimensional low-level features with those extracted from SoundNet, SN-F or SN-R. All the setups were identical except: (1) Pre-training was found to be unnecessary, so we initialized the weight matrices of the CTC-RNN using Glorot uniform initialization [82], and set the initial bias of the forget gates to one [78, 79]; (2) The learning rate was initialized to 3.0, and was decayed by a factor of 0.8 when the token error rate on the validation set saw no update in 5 epochs; (3) The alignment hinting tolerance was set to 10 frames (*i.e.* each peak was allowed to occur within a 2-second window around the ground truth).

Fig. 3.7 shows the evolution of the loss function and the TER on the training, validation and test sets, using the "conv5" layer of SoundNet,

Feature	Layer	#Dims	Train TER	Val. TER	Test TER
Low-level	N/A	50	15.2	82.8	81.0
SoundNet	conv5	256	2.3	**76.6**	**74.0**
SN-F	pool5	256	3.5	80.5	77.4
	fc1	100	6.1	79.9	77.8
	fc2	100	6.5	82.2	79.2
	fc3	100	4.6	80.8	77.6
SN-R	pool5	256	3.0	78.9	**74.9**
	gru1	200	3.0	77.8	78.4
	gru1-BN	200	1.2	**75.8**	76.7
	gru2	200	3.0	84.5	80.6
	gru2-BN	200	2.3	82.3	79.0
	gru3	200	90.6	96.4	96.4
	gru3-BN	200	60.9	90.5	91.2

Table 3.2: SED performance when using transfer learning features: Token error rate (TER) at convergence (Epoch 200) using features extracted from different layers of SoundNet, SN-F and SN-R. "BN" means after batch normalization. The TER values of low-level features are measured at Epoch 500.

"fc1" layer of SN-F, and "gru1" layer of SN-R (after batch normalization), respectively. For comparison, the curves produced using the low-level features are also included. The transfer learning features learnt substantially accelerated the convergence. When using low-level features, the CTC network exhibited a "warm-up" stage in which it did not output anything; the pre-training shortened this stage from 60 epochs to 40 epochs. But with transfer learning features, the warm-up stage was almost non-existent. The final test-set TER was also lower than using low-level features (see Table 3.2). Actually, before we switched to SoundNet features, we had tried several techniques on the CTC-RNN (including dropout [65] and data augmentation) in order to improve the generalization, but none of these techniques brought the test TER below 80%. The transfer learning features broke this barrier easily; however, the gap between the training and test sets remained huge.

Next, we looked at which layer of SoundNet or its variants yielded features that led to the best SED performance. Table 3.2 shows the TER on the training, validation and test sets after 200 epochs when using features extracted from different layers. We found that features extracted from the "conv5" layer of the original SoundNet remained competitive. With SN-F, features extracted from the higher, fully connected layers yielded better SED performance than low-level features, but still fell short of SoundNet's "conv5" layer. With SN-R, we first noticed that it was always better to

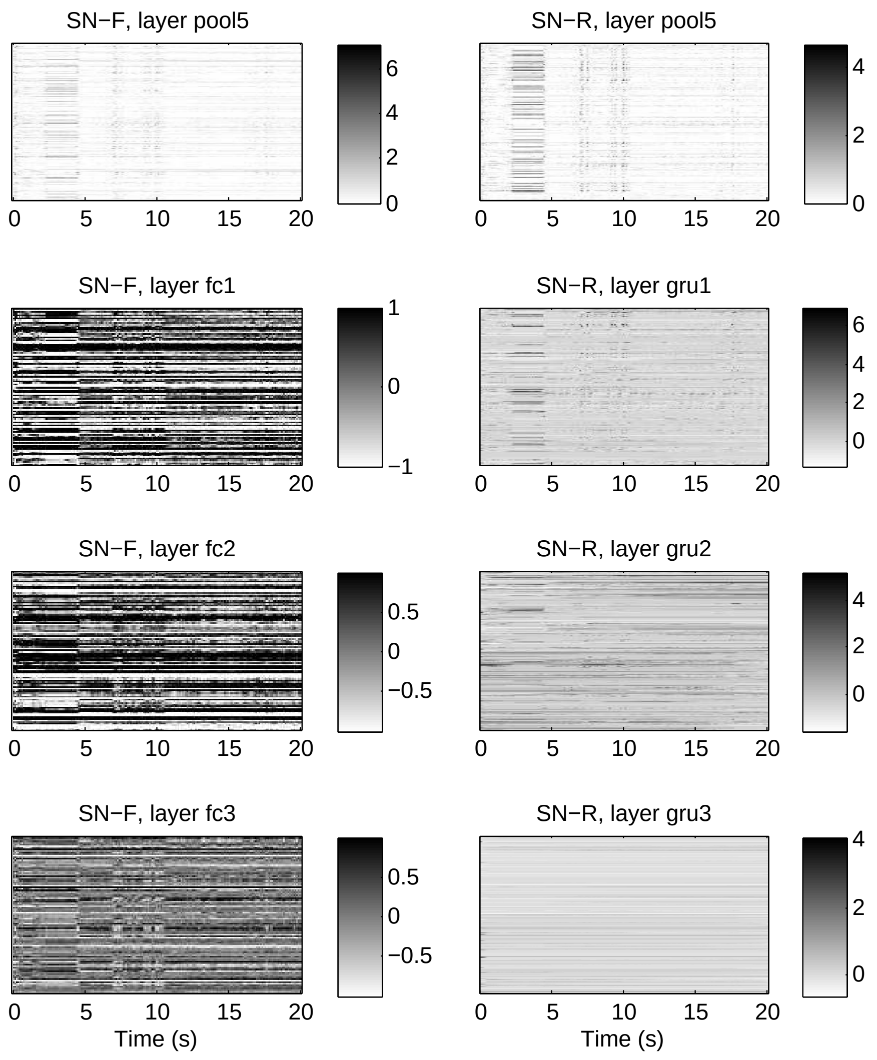


Figure 3.8: The activations of the higher layers of SN-F and SN-R on a validation recording. For the recurrent layers of SN-R, the activations have been batch normalized.

extract features after batch normalization. We also noticed the counter-intuitive phenomenon that the SED performance got worse as features were extracted from higher layers.

We give a tentative explanation of this performance deterioration by visualizing the activations of some higher layers of SN-F and SN-R in Fig. 3.8. We can see a clear transition in the activations at 4.5 s, which was preserved in all the layers of SN-F. In SN-R, however, the transition became blurred out at the "gru2" layer, and disappeared altogether at the "gru3" layer. This indicates that recurrent layers, which have access to information at distant moments, might not be good at representing local information. The fully connected layers of SN-F, on the other hand, maintained a reception field of 0.34 seconds, and therefore were able to concentrate on what happened within this time window.

3.4 Error Analysis

The limited improvement in the token error rate (81% to 74%) prompted us to conduct a thorough error analysis of the system predictions. We computed an error rate for each sound event type from the predictions of the CTC-RNN with original SoundNet features to understand what was happening with the system.

Even though the token error rate metric consists of insertion, deletion and substitution errors, it is hard to attribute the individual errors to each sound event type. This is because there can be multiple alignments that give the same token error rate. For example, if the reference sequence is `ABC` and the hypothesized sequence is `DE`, then there are three alignments that all give a TER of 100%:

```
        ABC                 ABC                 ABC
        DE-                 D-E                 -DE
```

But they differ on which token is deleted and which are substituted. To avoid this arbitrariness, we compute the *miss rate* and *false alarm rate* of each type of token, defined as the number of misses or false alarms divided by the the number of occurrences in the reference. A token is considered to be missed if it occurs in the reference but does not align with an identical token in the hypothesis; a token is considered to be a false alarm (FA) if it occurs in the hypothesis but does not align with an identical token in the reference. In the example above, the tokens `A`, `B`, and `C` are all considered to have one miss, and the tokens `D` and `E` to have one false alarm, regardless of the alignment. There is still a little arbitrariness in this definition if one considers the case of *flipping* two tokens, *e.g.* when the reference is `AB` and the hypothesis is `BA`. There are again three alignments that all give a TER of 100%:

```
        AB                  AB-                 -AB
        BA                  -BA                 BA-
   A: 1 miss, 1 FA     A: 1 miss, 1 FA
   B: 1 miss, 1 FA                          B: 1 miss, 1 FA
```

But the counts of misses and false alarms for the tokens `A` and `B` are different. Luckily we found that such flipping occurred rarely enough in our hypotheses, so the fluctuation in the miss and FA rates caused by choosing an alignment arbitrarily could be ignored.

In Fig. 3.9 we plot the miss and FA rates of each sound event type. Each

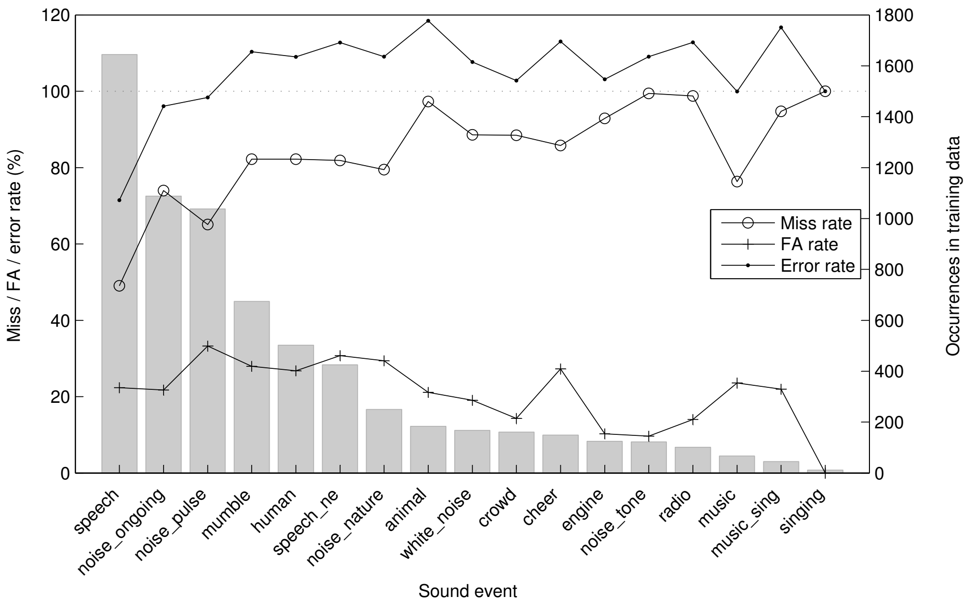


Figure 3.9: The miss rate, false alarm rate and total error rate of each sound event type (lines). The bars indicate the number of occurrences in the training data.

sound event type actually corresponds to two types of tokens (an onset token and an offset token); we plot the average miss and FA rates of the two. The total error rate of each event type, defined as the sum of the miss and FA rates, is also plotted in the figure. The event types are ordered by the number of occurrences in the training data. It can be seen that the miss rate is negatively correlated with the number of occurrences, while the FA rate is positively correlated but has a smaller range. Put together, the total error rate is lower for more frequent sound event types. It turns out, however, that only the three most frequent sound event types (speech, ongoing noise, pulse noise) obtained an error rate less than 100%. All the other sound event types, due to their insufficient number of occurrences (mostly around 200), contributed adversely to the token error rate. This result demonstrates how much the success of a CTC model can depend on the availability of sufficient training data. The size of the corpus may need to be increased by four or five times before a significant reduction of the TER can be observed.

Another problem revealed by the error analysis is that, some types of long-lasting sound events, such as "music" and "crowd", are identified as rare events in Fig. 3.9 because of their small number of *occurrences*, even though they cover a long total *duration* as shown in Fig. 1.3 (b). In the CTC models we used, each occurrence of a sound event provided two target CTC tokens at the boundaries, while the long duration of the events provided no supervision. This indicates we were not making sufficient use of the duration of long events.

Nonetheless, we should emphasize again the effectiveness of the CTC-RNN in detecting pulse noises, an example of transient noises. The CTC-RNN system in Sec. 3.3 obtained an error rate of 98% for pulse noises (65% miss + 33% FA), while the frame-wise system in Sec. 3.1 got 107% (90% miss + 17% FA). Even though these numbers are not directly comparable, "pulse noise" was the third best detected sound event type with the CTC-RNN, while it fell in the long tail of "badly detected" event types with the frame-wise system.

3.5 Proposed Work

Current research on SED with CTC suffers from two problems: there amount of training data available is limited, and there isn't a principled way to exploit long-lasting sound events. It is also crucial for the model to maintain a good temporal resolution in order to localize sound events in time, but we have observed that the peaks CTC produces tend to cluster together without alignment hinting, and that the activations of the higher layers of SN-R get blurred out across time. I propose to tackle these problems in the last year of my PhD.

3.5.1 Semi-Supervised Training with More Data

To increase the amount of training data, I plan to automatically discover reliable training instances from large but weakly-labeled data. Google Audio Set [52] is an abundant source of such data. As introduced in Sec. 1.2, Google Audio Set contains about 600 times as much data as the Noiseme corpus (v2). For example, the corpus contains more than 4,000 instances of cheering, and more than 15,000 instances of engine noise.

The Google Audio Set is only labeled with the presence or absence of each sound event type in each 10-second excerpt. This raises the question of how we can incorporate this data into a CTC system that accepts sequential labeling. I propose two methods to generate strong labeling on Google Audio Set, and then derive sequential labeling from there. The first method is to train a SED system with presence/absence labeling on Google Audio Set, which is the content of Chapter 4. Even though trained with weak labeling, this system should be able to localize sound events in the Audio Set data, thus producing labeling suitable for CTC. The second method is to run existing SED systems (including the frame-wise system in Sec. 3.1 and the CTC system in Sec. 3.3) on the Audio Set data to discover candidate sound event instances. Both approaches can be subsumed under the framework of *semi-supervised learning*. The sequential labeling generated by the two methods will no doubt be noisy, but the two methods will be able to validate each other to select reliable recordings to add into the training data.

Using Google Audio Set has another merit: the recordings in it last only 10 seconds, which is much shorter than the average recording length of the Noiseme corpus (75 s). At this length, we can hopefully eliminate the "alignment hinting" introduced in Sec. 3.2.

The incorporation of Google Audio Set in the training data will allow me to re-evaluate the performance of the CTC-RNN for SED, the benefits of transfer learning based features, and the effect of using fully connected or recurrent layers on top of convolutional layers. Such evaluation will not only be done with the TER metric, but also with more standard metrics for SED, such as segment-based and event-based error rate and F1 [83]. Besides measuring the overall performance, I will also look at the performance of individual sound events, to see whether CTC is equally effective for transient, continuous and intermittent sound events.

3.5.2 Better Exploiting Long Sound Events

While CTC models are good at detecting short events, currently there is no systematic way of exploiting long events. Even though a sound event may last for as long as one minute, a CTC network can only make use of the start and end tokens at the two boundaries, and may not spend any effort learning what a long event sounds like when it is ongoing.

To overcome this problem, I propose to add a "middle" token for each sound event type, in addition to the existing "start" and "end" tokens. The CTC network is required to generate "middle" tokens during the timespan of long events once every m seconds. For example, if a dog barks while a car passes by, and the car noise lasts for a long time, the target label sequence may be: `engine_start, engine_middle, animal_start, animal_end, engine_middle, engine_end`. The length of the interval between consecutive "middle" tokens, m, will need to be optimized by experiment.

3.5.3 Improving the Temporal Localization of CTC

It has been shown in Sec. 3.2.2 that the peaks predicted by CTC tend to cluster together without alignment hinting, instead of aligning with the actual boundaries of sound events. This is because the recurrent layers underneath the CTC layer can propagate information across time, making the prediction at each frame depend not only on the current frame but also on information from the past and the future. While it is sometimes necessary to look backward and forward in time to decide which sound event is active at the current frame, local information is also important for obtaining a correct alignment. The vanilla CTC model is not good at the latter.

In [84], a parallel structure has been proposed to improve the temporal localization power of a CTC-based speech recognizer. A stack of local, fully connected layers is placed in parallel with the usual recurrent layers. Each of the two branches has its own sigmoid output layer; the values of the two output layers are averaged before computing the CTC loss function. Because the frames in the fully connected branch have limited reception fields, the combined prediction is forced to give sufficient consideration to the local information. This structure has been shown to moderately reduce the word error rate, but significantly improve the alignment. The same idea of training a feed-forward branch that focuses on local information in parallel with a recurrent network in order to improve its temporal localization power has also been applied to SED in [85].

I would like to apply this parallel structure to my CTC network for SED as well. Hopefully this will make the predicted peaks align better with the actual event boundaries, and even make alignment hinting unnecessary.

3.5.4 Training a Transfer Learning Feature Extractor that Maintains Temporal Resolution

Although transfer learning based features extracted from convolutional and/or recurrent neural networks can exhibit superior performance in SED compared to low-level features, they must maintain a sufficient temporal resolution in order to localize sound events in time. However, current transfer learning networks do not have the necessary temporal resolution: the original SoundNet tends to make similar predictions at different keyframes in the same recording, and our SN-R network exhibits "blurred" activations at the higher recurrent layers.

My personal experience with tuning hyperparameters for RNNs indicates that this may be a result of underfitting. When the network is trained with the SGD algorithm, this can happen when the magnitude of the gradient is large in the initial stage of training but decreases fast with time: in order to avoid gradient explosion in the initial stage, the learning rate must be set to a low value, which results in slow learning in subsequent epochs. This could be remedied with gradient clipping and a larger learning rate.

I will try re-training the SN-R network using the SGD algorithm with gradient clipping, and/or look for solutions to the underfitting problem when using the Adam optimizer.

3.5.5 Comparing and Combining Different Transfer Learning Features

In Chapter 4, I will train an end-to-end deep network for SED on the Google Audio Set. This network bears many similarities with SoundNet: they are both trained with a different objective than SED with sequential labeling, and they are trained with a similar amount of training data. As such, the lower layers of both networks can be treated as a transfer learning feature extractor for SED with sequential labeling. Nevertheless, there is an important distinction between the two networks: the network trained on Audio Set is also trained for SED, while SoundNet is trained on an out-of-domain image learning task. An interesting comparison would be to see if the features learned by the former would yield better SED performance than the latter because they encodes more relevant information. The two types of features may also be combined to further improve the performance.

Chapter 4

Sound Event Detection with Presence/Absence Labeling

This chapter studies the learning of sound events from *presence/absence labeling*. This type of labeling is even weaker than sequential labeling; it is only known whether each type of sound event is present or absent in each audio recording.

SED from presence/absence labeling can be regarded as a multiple instance classification (MIC) problem [36], which falls into the broad framework of multiple instance learning (MIL). In MIC, we do not have the ground truth labels for individual instances. Instead, the instances are grouped into *bags*, and labels are known for the bags only. The relationship between the bag label and the instance labels may vary; a common case is the *standard multiple instance (SMI) assumption*: a bag is labeled as positive if it contains at least one positive instance, and negative if it only contains negative instances. The task of MIC is to learn a classifier either for bags or individual instances.

We formulate polyphonic SED from presence/absence labeling as a MIC problem as follows. First, the prediction of each sound event type is carried out independently, even though they may depend on a certain shared high-level representation of the audio. For example, the prediction may be performed with a neural network with a separate output layer for each sound event type but shared hidden layers. Now polyphonic SED becomes a set of binary classification problems, one for each sound event type. We regard each audio recording as a bag, and the frames in the recording as instances. A bag is labeled as positive if and only if a certain sound event type is active in the corresponding recording. We learn an instance-level classifier that makes a prediction for each individual frame, which can be used to localize sound events in time. The instance-level predictions are aggregated (or "pooled") into a bag-level prediction, which can be compared with the bag labels to form a loss function. Two commonly used pooling functions are the max function [37, 38] and the noisy-or function [39–41]; we compare the two functions both theoretically and empirically.

The content of this chapter is organized as follows. In Sec. 4.1, we give the motivation of the max and noisy-or pooling functions in the framework of multiple instance learning (MIL), and point out some plausible advantages of the noisy-or pooling function. To compare the two pooling functions, we conduct proof-of-concept experiments on a speech recognition task on TED talks in Sec. 4.2. This task is easier than polyphonic SED because phonemes in speech do not overlap, and they have a smaller variation in both duration and acoustic characteristics than sound events. Surprisingly, max pooling succeeds on this task while noisy-or pooling fails. We then analyse the results and discuss some defects of noisy-or pooling that render it unsuitable for sequence learning tasks. In Sec. 4.3, we propose to apply MIL with the max pooling function to polyphonic SED on large corpora, *e.g.* the Google Audio Set. We propose to localize the sound events in time even though the labels do not provide any timing information, and to automatically classify sound events as transient, continuous or intermittent.

4.1 Pooling Functions in Multiple Instance Learning

4.1.1 Motivation of the Max and Noisy-Or Pooling Functions

Let's consider a bag of instances in a multiple instance classification problem. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be the instances, and t be the label of the bag. For convenience, we may use either $\{1, -1\}$ or $\{1, 0\}$ to represent the labels of the positive and negative classes. Let f be an instance-level classifier, and it makes a prediction $y_i = f(\mathbf{x}_i)$ for each instance. Training of the classifier means ensuring that the y_i's and t satisfy certain constraints, or minimizing a loss function that depend on the two.

Suppose the classifier is a support vector machine (SVM) $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \in \mathbb{R}$. If the bag is positive ($t = 1$), then at least one of the y_i's must satisfy $y_i \geq 1$. If the bag is negative ($t = -1$), then all the instances must satisfy $y_i \leq -1$. The two cases may be subsumed by a single equation:

$$t \cdot \max_i y_i \geq 1 \tag{4.1}$$

If we aggregate the instance-level predictions y_i into a bag-level prediction using the following equation:

$$y = \max_i y_i \tag{4.2}$$

then the multiple instance SVM reduces to an ordinary SVM on bags. This is the motivation of the "max pooling" function. The max pooling function is the natural choice for max-margin classifiers such as SVMs. Both SVMs and the max pooling function exemplify the "winner-takes-all" idea: in SVMs, the decision boundary depends solely on the extreme instances (*i.e.* support vectors); with the max pooling function, the bag-level prediction depends solely on the maximum instance, and other instances do not matter.

With classifiers other than SVMs, however, "max pooling" may not be the pooling function of first choice. Let's consider a probabilistic classifier g (*e.g.* a neural network), whose predictions $y_i = g(\mathbf{x}_i) \in [0, 1]$ indicates the probability of $\mathbf{x}_i$ being a positive instance. As listed in Table 2.1, for a binary classification problem, the most common loss function is *cross-entropy*. The loss function on the bag in question is

$$L = -t \log y - (1 - t) \log(1 - y) \tag{4.3}$$

where $t \in \{0, 1\}$ is the bag label, and y is the predicted probability of the bag being positive. The SMI assumption states that a bag is negative only if all of its instances are negative. If we assume all the instances in the bag are independent, the bag-level prediction y can be decomposed as

$$y = 1 - \prod_i (1 - y_i) \tag{4.4}$$

This is called the "noisy-or" pooling function, because the equation effectively implements the logical "or" gate if all the y_i's were binary. The noisy-or pooling function is the natural choice with probabilistic classifiers, if we can assume independence between instances in a bag.

To summarize, the max and noisy-or pooling functions are motivated by different types of classifiers. The max pooling function is suited for SVMs, and is desirable if we prefer a "winner-takes-all" behavior; the noisy-or pooling function is suited for probabilistic classifiers, and every frame plays a role in the loss function. Since neural networks produce probabilistic predictions, the noisy-or pooling function seems preferable. However, in the case of sound event detection, the assumption of independence between instances in a bag is questionable because the instances are frames in a recording, and they are clearly correlated with each other.

4.1.2 Relationship Between the Noisy-Or Pooling Function and CTC

In Chapter 3, we used CTC to deal with the incompleteness of sequential labeling. Naturally, we wonder if CTC can also be used to deal with the incompleteness of presence/absence labeling. In this subsection, we show that a connection can be established between CTC and the noisy-or pooling function by extending the CTC model.

First we take a step back: we no longer model the boundaries of sound events, but return to modeling the events themselves. Suppose an audio

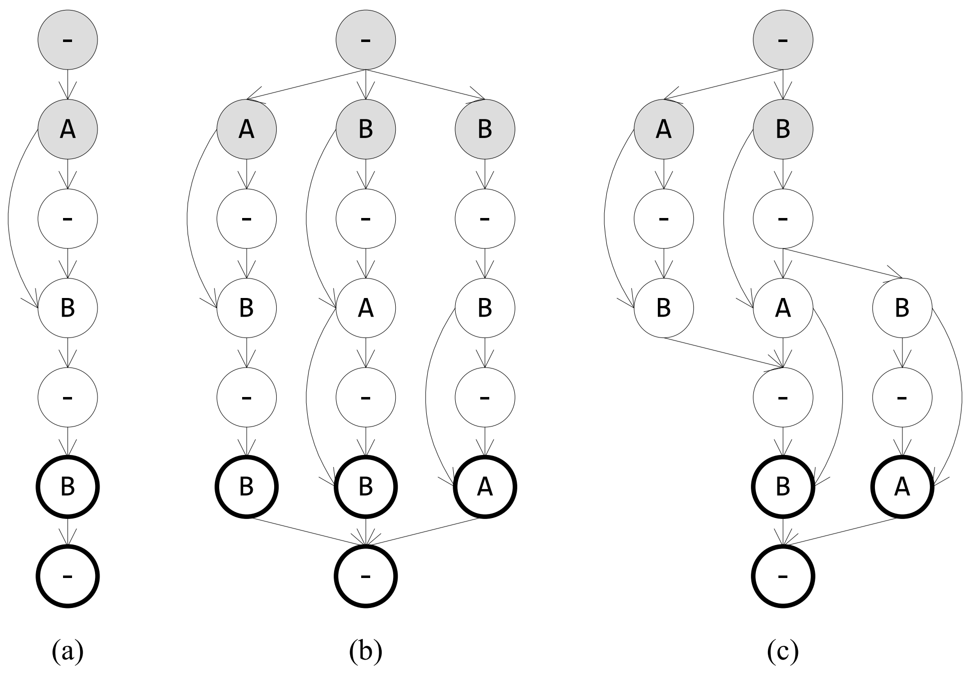


Figure 4.1: (a) CTC state graph accepting a single label sequence `ABB`. (b) CTC state graph accepting three label sequences `ABB`, `BAB`, and `BBA`. (c) Determinized and minimized version of (b).

recording contains the siren of an ambulance (`A`) and two dog barks (`B`). If the order of the three sound events are unspecified, or if they overlap in time, then the label sequence of the recording becomes ambiguous. Any of the output sequences `ABB`, `BAB`, and `BBA` is acceptable, and we may want to maximize the total probability of the three output sequences.

This maximization problem is solvable by extending the state graph of CTC. Fig. 4.1 (a) shows the CTC state graph when the only acceptable label sequence is `ABB`. An arc is drawn between two states if it is allowed to transition from one state to the other in one time step. As indicated by Eq. 2.13, transitions are allowed between adjacent states, or states that are two steps apart and do not carry the same token. When computing the alpha trellis, the shaded states are initialized with non-zero probabilities at the first time step, and the total probability of the label sequence is obtained by summing the alpha values of the bold states at the last time step. When multiple label sequences are acceptable, these sequences may be juxtaposed in the state graph as in Fig. 4.1 (b). Treating the state graph as a finite state automaton, it may be determinized and minimized to the form in Fig. 4.1 (c).

It can be easily imagined that as the number of sound events in a recording grows, the number of paths in the CTC state graph will grow factorially. To avoid this problem, we can create a separate CTC output layer for each sound event type above the shared recurrent layers; each CTC output layer has a single unit that predicts the probability of the event token (one minus which is the probability of the blank token). In the example in the paragraph above, the CTC output layer for ambulance should accept the label sequence `A`, the CTC output layer for dog barks should accept the label sequence `BB`, while all the other CTC output layers should accept empty label sequences (denoted by ϵ). The loss function on the recording in question will be the total negative log-likelihood of the all CTC output layers:

$$L = -\log P(\texttt{A}) - \log P(\texttt{BB}) - \log P(\epsilon) \ldots \tag{4.5}$$

Such a model is suitable for a form of weak labeling that lies between sequential labeling and presence/absence labeling: what we know is the number of occurrences of each sound event type in each recording. We call this *count labeling*, and apply CTC to a speech recognition task with count labeling in Sec. 4.2.

In the case of presence/absence labeling, even the number of occurrences of each sound event type is unknown. Therefore, the CTC output layer for ambulance should accept all label sequences that contain one or more tokens `A`, and compute their total probability $P(\texttt{A}^+) = P(\texttt{A}) + P(\texttt{AA}) + P(\texttt{AAA}) + \ldots$. This is equal to one minus the probability of the empty label sequence. In CTC, it is assumed that the frames in a utterance are conditionally independent given the states of the hidden layers, and we make the same assumption here. Let $y_i(\texttt{A})$ be the probability of the token `A` at the i-th frame, then the desired total probability is given by

$$P(\texttt{A}^+) = 1 - \prod_i [1 - y_i(\texttt{A})] \tag{4.6}$$

Likewise, the output layer for an event `C` apart from ambulance and dog barks should compute the probability of the empty label sequence ϵ:

$$P(\epsilon) = \prod_i [1 - y_i(\texttt{C})] \tag{4.7}$$

We can see that Eqs. 4.6 and 4.7 have the same form as the noisy-or pooling function (Eq. 4.4).

To summarize, we have shown that the noisy-or pooling function is related to CTC, if we make the following modifications and generalizations to CTC:

1. Break up a CTC layer into many parallel CTC layers, one for each token;

2. Allow the token in each CTC layer to occur any positive number of times.

4.1.3 The Gradient Flow

A key step in the training of RNNs is the computation of the gradient, *i.e.* error back-propagation. In this subsection, we compare the gradient flow of the max and noisy-or pooling functions to see which one makes training easier.

Let $t \in \{0, 1\}$ be the ground truth label for a bag, and $y \in [0, 1]$ be the bag-level prediction. The bag-level prediction is aggregated from the instance-level predictions $y_1, \ldots, y_n \in [0, 1]$, using either the max (Eq. 4.2) or noisy-or (Eq. 4.4) pooling function. The instance-level predictions are the output of sigmoid units; let the input to the sigmoid units be $z_1, \ldots, z_n$, then we have $y_i = \mathrm{sigm}(z_i)$. Now we want to compute the error signals in back-propagation, *i.e.* the derivative of the loss function

$$L = -t \log y - (1 - t) \log(1 - y) \tag{4.8}$$

with respect to the z_i's. For convenience, we point out that the derivative of the sigmoid function is $\partial y_i / \partial z_i = y_i(1 - y_i)$.

When using the max pooling function, the bag-level prediction y is only dependent on the single maximum instance-level prediction. Let the subscript of this instance be k, then the loss function is:

$$L = -t \log y_k - (1 - t) \log(1 - y_k) \tag{4.9}$$

The derivative of L w.r.t. z_k is:

$$\begin{align} \frac{\partial L}{\partial z_k} &= -\frac{t}{y_k} \cdot y_k(1 - y_k) + \frac{1 - t}{1 - y_k} \cdot y_k(1 - y_k) \tag{4.10} \\ &= y_k - t \tag{4.11} \end{align}$$

while the derivative w.r.t. other z_i's are all zero.

The derivative in Eq. 4.11 makes sense. When the bag is positive ($t = 1$), the k-th instance receives a negative gradient, and the gradient descent algorithm will pull z_k up, so y_k gets closer to 1. When the bag is negative ($t = 0$), the gradient is positive and z_k will be pushed down. The amount of boost or suppression that z_k receives is proportional to the difference between the prediction y_k and the ground truth t.

A short-coming of the max pooling function, however, is that the error signal is only given to the single instance reaching the maximum. In the case of SED, the underlying recurrent layers can alleviate this problem to some extent, because they can pass on the error signal across time until the forget gate is closed. This usually spans the duration of one sound event occurrence. However, if a sound event occurs multiple times in a recording, then only the one that yields the maximum frame-level prediction will receive an error signal.

When using the noisy-or pooling function, the loss function takes the following form:

$$L = \begin{cases} -\log\left[1 - \prod_i (1 - y_i)\right], & \text{if } t = 1 \\ -\log \prod_i (1 - y_i), & \text{if } t = 0 \end{cases} \tag{4.12}$$

This depends on all the instance-level predictions. Its derivative w.r.t. any z_i is:

$$\frac{\partial L}{\partial z_i} = \begin{cases} -y_i(1 - y)/y, & \text{if } t = 1 \\ y_i, & \text{if } t = 0 \end{cases} \tag{4.13}$$

Let's analyze what effects this gradient has on the learning process. When the bag is negative ($t = 0$), the gradient is positive, so all z_i's will be pushed down, in proportion to the instance-level predictions y_i. When the bag is positive ($t = 1$), the gradient is negative, and all z_i's will be boosted. The strength of the boost depends on two factors. One is $(1 - y)/y$, which involves the bag-level prediction. The farther from 1 the bag-level prediction y is, the more eager the model is to boost up the instance-level predictions. The other factor is y_i itself. At first glance this may seem counter-intuitive: the instances whose y_i are already closer to 1 get boosted more. However, we should note that this is a multiple instance learning scenario, and we do not need to make all the instances positive in a positive bag. Instead, we only encourage the "hopeful" instances, and leave alone the instances that would like to stay negative.

Compared with the max pooling function, the noisy-or pooling function sends an error signal to every instance in a bag, instead of the single one with the largest instance-level prediction. In addition, it adjusts the magnitude of the error signals according to both the instance-level predictions and the bag-level prediction. It may be hoped that the noisy-or pooling function allows the gradient to flow more easily through the network, which may accelerate the training.

To sum up this section, we have seen that the noisy-or pooling function fits more naturally with probabilistic classifiers, is an extension of the CTC model, and may facilitate the gradient flow. Nevertheless, it requires independence between instances within a bag, which may not hold for sequential learning tasks such as SED. We conducted experiments to test out the relative advantages and disadvantages of the max and noisy-or pooling functions.

4.2 Proof-of-Concept Experiments with Speech Recognition

4.2.1 Experiment Setup

To empirically compare the "max" and "noisy-or" pooling functions, we conducted a set of proof-of-concept experiments with a speech recognition task. To be precise, this was only a phoneme recognition task, because both the reference and the network output were sequences of phonemes instead of words. We used phonemes as substitutes for sound events, although there were two notable differences: phonemes are normally short, and they do not overlap.

The experiments were conducted on the TEDLIUM v1 corpus[1]. The corpus consists of 206 hours of training data, 1.7 hours of development data, and 3.1 hours of testing data. We used 95% of the training data for training, and reserved the remaining 5% for validation. Ground truth phoneme sequences were generated for all the utterances from the transcriptions and the dictionary; we only retained the 39 "real" phonemes and discarded all noise markers such as "breath" and "cough".

We first tried to replicate a baseline system[2] in the EESEN toolkit [86]. Using our own Theano implementation, we trained a network with the same structure as the baseline system. The network consisted of five bidirectional LSTM layers, with 320 memory cells in each direction of each layer. The input layer had 40 neurons, which accepted 40-dimensional filterbank features[3]. The CTC output layer consisted of 40 neurons arranged in a softmax group, corresponding to the 39 phonemes plus a blank token. We used the per-frame negative log-likelihood as the loss function, and trained the network using the stochastic gradient descent (SGD) algorithm with a Nesterov momentum of 0.9. Each minibatch contained 20,000 frames; an epoch consisted of about 2,000 minibatches. The learning rate started at 3, stayed constant for 12 epochs, and was then halved each epoch for another 12 epochs. Gradient clipping was applied with a threshold of 10^{-4}. Simple best path decoding was used to generate hypothesized phoneme sequences, and the performance of the network was measured with phoneme error rate (PER).

The supervision used by the baseline system can be regarded as *sequential*. We then trained three other networks that used weaker forms of supervision. One of these networks used *count labeling*: instead of knowing the phoneme sequence for each training utterance, it was only told how many times each phoneme occurred in each utterance. The other two networks used *presence/absence labeling*, *i.e.* they only knew whether each phoneme was present or not in each utterance. These two networks used the max and noisy-or pooling functions, respectively. The structures of these networks were kept identical to the baseline network except for the output layer. Instead of having 40 neurons arranged in a softmax group, the output layer of the three weakly supervised networks contained 39 neurons with the sigmoid activation function. This can be regarded as 39 parallel CTC output layers,

[1]The corpus can be downloaded at `http://www.openslr.org/resources/7/`.
[2]`https://github.com/srvk/eesen/tree/master/asr_egs/tedlium/v1`
[3]Unlike the EESEN system, we did not use delta and double delta features.

each responsible for a single phoneme. With count labeling, the target label sequence of each CTC layer was the corresponding phoneme repeated the specified number of times; with presence/absence labeling, each output layer must generate the corresponding phoneme any positive number of times when the label is "present", and zero times when the label is "absent".

The change in the output layer also caused a change in the loss function. For the count labeling system, the CTC output layers were not degenerate, so we still used the per-frame negative log-likelihood, and also averaged it across all the phonemes. With presence/absence labeling, however, the output layer essentially degenerated to sequence-level binary classification. For the max pooling network, we used cross-entropy averaged across both sequences and phonemes; for the noisy-or pooling network, since the sequence-level probabilities could be decomposed into a product of frame-level probabilities, we used cross-entropy averaged across frames and phonemes.

The different loss functions of the networks might range in different orders of magnitude, resulting in a drifting of the optimal hyperparameters for training. To limit the effort of hyperparameter tuning, we only tuned the gradient clipping limit and the initial learning rate, and kept everything else (*e.g.* batch size, momentum, learning rate schedule) identical to the baseline system.

We still used the best path decoding algorithm for the systems trained with count and presence/absence labeling, *i.e.* picking the most probable phoneme at each frame, then reducing this alignment to a phoneme sequence by removing repeating phonemes and blank tokens. A difference from the baseline system is that there was no blank token in the CTC output layer. We dealt with this difference with the following strategy: if the most probable phoneme at a frame had a probability smaller than 0.5, we regarded this frame as emitting a blank token.

4.2.2 Experiment Results

For each of the systems trained, the optimal hyperparameters, as well as the PERs on different parts of the data, are listed in Table 4.1. The evolution of the cross-validation PER of each system is plotted in Fig. 4.2. When we compare systems in this section, we will mainly look at the cross-validation PER; the development and test PERs follow a similar trend.

First, we compare our own implementation of the baseline system using sequential labeling with the EESEN implementation. After a single epoch, the system we implemented reached 28.8% PER on the training set and 30.2% on the cross-validation set; the final PER was 4.8% on the training set and 15.4% on the cross-validation set, outperforming the EESEN implementation.

Second, we compare the count labeling system with the baseline. The

System	Hyperparameters		Phoneme Error Rate			
	Grad.clip	Init.LR	Train	Valid.	Dev.	Test
Baseline (EESEN)			13.1	18.5	19.3	18.4
Baseline (Theano)	10^{-4}	3	4.8	15.4	13.9	14.9
Count labeling	10^{-6}	100	30.1	34.5	31.7	32.5
Max pooling	0.01	0.3	40.5	43.0	39.7	40.7
Noisy-or pooling	10^{-8}	3000	91.0	91.6	91.6	91.5

Table 4.1: The optimal hyperparameters and phoneme error rates of the various systems on the TEDLIUM corpus.

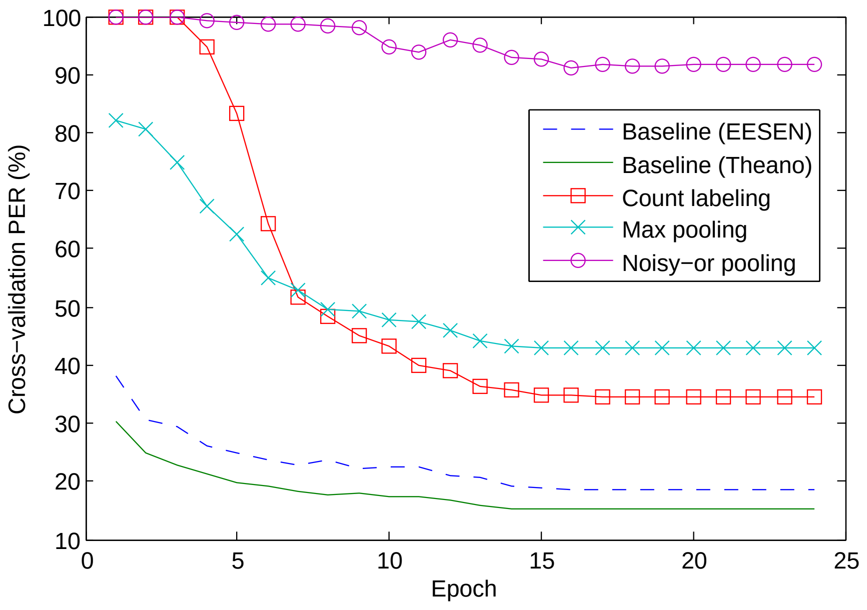


Figure 4.2: Evolution of the cross-validation PER of the various systems on the TEDLIUM corpus.

final PER (34.5%) was higher than the baseline system, but still reasonable. This means the modified CTC-RNN was able to learn from a weaker form of labeling, even though not as well as the baseline. In the first few epochs, the count labeling system exhibited high PERs above 90%. This is the "warm-up" stage of CTC systems; it lasted longer because count labeling is weaker than sequential labeling.

Next, we look at the max pooling system trained with presence/absence labeling. Because presence/absence labeling is again weaker than count labeling, the final PER (43.0%) was higher than that of the count labeling system (34.5%), but this number still indicates that the max pooling system was able to learn to detect phonemes. Also remarkable is that the max pooling system reached a PER of 82.1% after the first epoch. There was no "warm-up" stage, because max pooling is not related to CTC.

System	Decoded Phoneme Sequence																PER
Ground truth	`V`	`EH`	`R`	`IY`		`IH`	`K`	`S`	`T`	`R`	`IY`	`M`	`T`	`ER`	`EY`	`N`	
Baseline (Theano)			`ER`	`IY`		`EH`	`K`	`S`	`T`	`R`	`IY`	`M`	`TH`	`R`	`IY`		8/15
Count labeling		`EH`	`R`	`HH`		`IH`		`S`	`T`	`R`	`IY`	`M`	`TH`	`R`	`IY`		7/15
Max pooling			`R`	`IY`	`HH`	`IY`	`IH`	`S`	`T`	`R`	`IY`	`M`	`TH`	`R`	`IY`		9/15
Noisy-or pooling								`S`	`S`				`TH`	`S`			14/15

Table 4.2: The predicted phoneme sequences of the various systems on an example utterance. The ground truth transcription is "very extreme terrain".

Finally, we look at the noisy-or pooling system. Despite the "advantages" we had found, this system was not able to learn effectively with whatever configuration of hyperparameters. The PER decreased very slowly, and after 24 epochs, the best PER we observed was still above 90%. If we kept the learning rate constant and trained the model for more epochs, the PER would stabilize between 83% and 86%.

Table 4.2 shows the predicted phoneme sequences of the various systems on an example utterance in the cross-validation set, and Fig. 4.3 shows the underlying frame-wise predictions. Comparing Fig. 4.3 (b) and (c), we can see that the predictions of the CTC-based count labeling system exhibits a similar pattern to the baseline system: predictions come in sharp peaks that cluster together. The predictions of the max pooling system, shown in Fig. 4.3 (d), however, displays a very different pattern. Each phoneme is represented by a wide peak with a flat top, even overlapping with each other. This is actually a desired property, because the span of the peak can indicate the onset and offset of each phoneme. The noisy-or pooling system, whose predictions are shown in Fig. 4.3 (e), seems not to have learnt anything. All but three phonemes are predicted with a negligible probability throughout the utterance.

4.2.3 Analysis: Why Noisy-Or Pooling Fails

We have observed the symptoms of the noisy-or pooling system: it trained slowly, and preferred to predict most phonemes as negative. We suspected that this indicated the system had difficulty in starting to learn. We tried initializing the network parameters to the values after one epoch of max pooling training (with a cross-validation PER of 82.1%), but we observed that noisy-or training immediately brought the PER back to above 90%.

It turned out that the problem lay with the combination of the cross-entropy loss function and the noisy-or pooling function. When used with noisy-or pooling, the cross-entropy loss function is excessively *harsh on false alarms* but *lenient on misses*, which was the reason why the noisy-or pooling system preferred making negative predictions. This problem does not exist

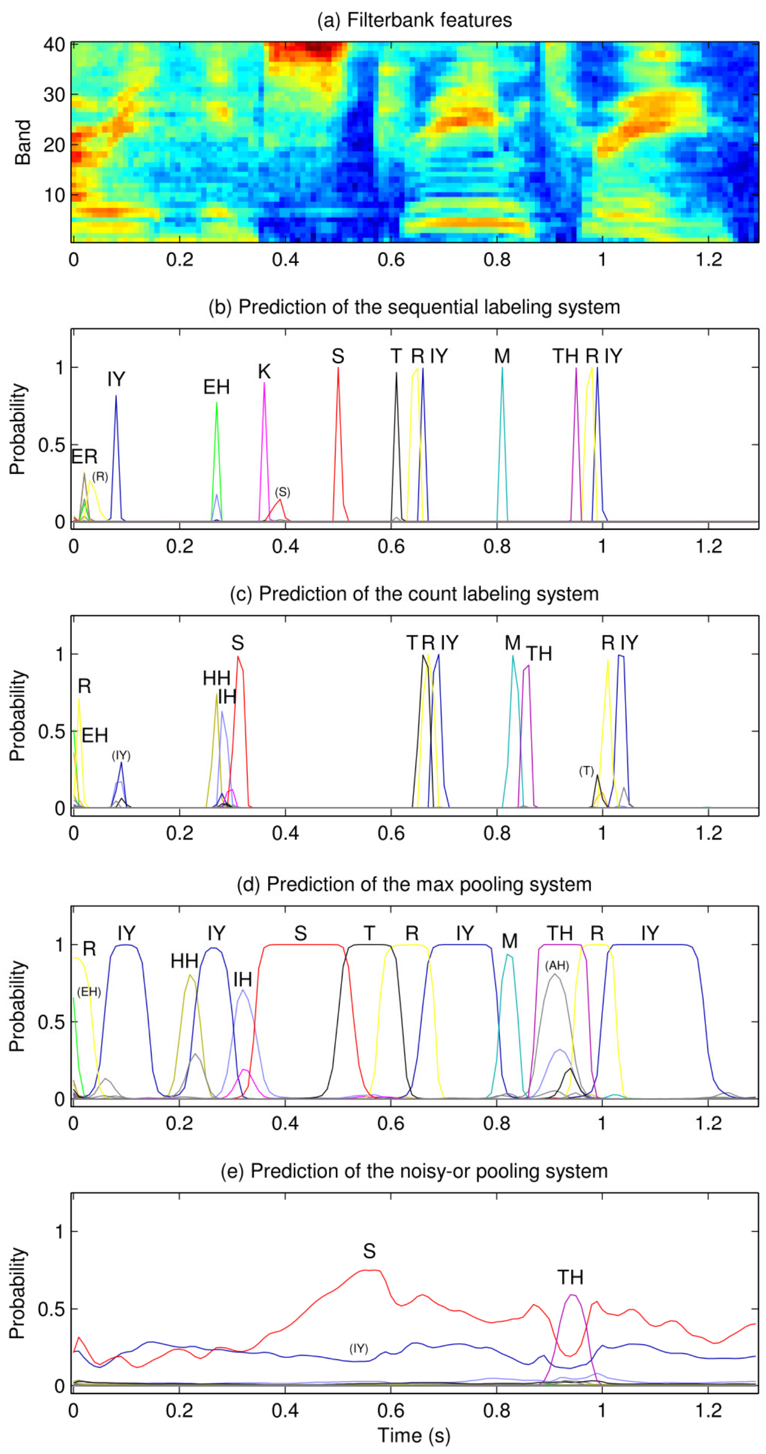


Figure 4.3: The frame-wise predictions of the various systems on an example utterance. The ground truth transcription is “very extreme terrain”. Different phonemes are plotted in different colors. Peaks are annotated with their corresponding phonemes; phonemes in parentheses did not make it into the predicted phoneme sequences.

when using the max pooling function. We will illustrate this problem by looking at some phonemes in Fig. 4.3 (d) and (e), and computing the cross-entropy loss function using both pooling functions.

Harsh on false alarms: Let's look at the phoneme TH. This phoneme does not occur in the ground truth phoneme sequence, so both the max pooling system and the noisy-or pooling system made a false alarm around 0.95 seconds. The max pooling system generated a tall and wide peak. Denote by y_i the predicted probability of the phone TH at frame i. For the seven frames $90 \leq i \leq 96$, y_i was above 0.999; the largest value was $y_{93} \approx 1 - 2 \times 10^{-7}$. When using the max pooling function, this meant a loss of $-\log(2 \times 10^{-7}) \approx 15.5$, while with the noisy-or pooling function, this meant a loss of at least $-\log(0.001) \times 7 \approx 48$. The large loss arises from the multiplication in the noisy-or pooling function ($y = 1 - \prod_i (1 - y_i)$). To avoid such large losses, the noisy-or pooling system generated a much lower and narrower peak than the max pooling system. The same situation happened with the phonemes HH and AH: the max pooling system generated a moderate peak, while the noisy-or pooling system generated no peak at all.

Note that the multiplication in the noisy-or pooling function results from the assumption of independence between instances in a bag. In the case of speech recognition or sound event detection, the instances are frames in a recording. Phonemes and sound events typically last more than one frame, causing strong correlation between the predictions on these frames. In such cases, we would want to base the loss function only on the heights of the peaks, regardless of their widths. From this perspective, the max pooling function is better suited for sequence learning problems, where the assumption of independence fails.

Lenient on misses: Let's look at the phoneme IH. This phoneme is in the ground truth phoneme sequence, and the max pooling system correctly predicts a peak (although not very tall) at 0.32 s. The probability curve predicted by the noisy-or pooling system is hardly visible in Fig. 4.3 (e); actually the y_i's for this phoneme fluctuates around 0.02. Even though this value is negligible on itself, when it is repeated throughout the utterance, it can produce a surprising effect. Suppose $y_i = 0.02$ for all the 130 frames of the utterance. Then the predicted probability of the bag being positive, *i.e.* the probability that the phoneme IH occurs anywhere in the utterance, will be $y = 1 - (1 - 0.02)^{130} \approx 0.93$ according to the noisy-or pooling function. In other words, even though the noisy-or system does not predict the phoneme IH at any frame, it believes that it has predicted it somewhere in the utterance, and will not make an effort to correct this miss.

A more extreme example is the phoneme IY. The frame-level predictions fluctuate around 0.2, which means the probability of the bag being positive is $y = 1 - (1 - 0.2)^{130} \approx 1 - 2.5 \times 10^{-13}$. This does not only mean that the noisy-or pooling system falsely believes that it has predicted the phoneme

`IY` in the utterance, but also has a bad effect on the gradient. Recall that the error signal at the i-th frame is $\partial L / \partial z_i = -y_i(1-y)/y$ (Eq. 4.13). With the bag-level prediction y close to 1, the error signal is virtually zero. Such "gradient vanishing" is the reason why the noisy-or system trained very slowly.

Now we have seen another undesirable property of the noisy-or pooling function: when the bag is large, negative instance-level predictions can make the bag-level prediction positive. In the extreme case, this can cause the gradient to vanish and the training to stall. This problem is inherent with the noisy-or pooling function, and exists even in non-sequence learning tasks. It is only with small bags (*e.g.* less than 10 instances) that this does not become a big concern.

In Sec. 4.1.2, we have shown that the noisy-or pooling function is related to CTC. One may wonder why the baseline system and the count labeling system, both being CTC systems, did not suffer from this problem. It turns out that the relationship between noisy-or pooling and CTC is rather remote. CTC can be turned into noisy-or pooling via two steps: (1) reducing the vocabulary size to one, and (2) allowing the single token to repeat any number of times. Actually the most important information in CTC is the order between different tokens; this information is totally lost in the first step. And in the second step, exponentially many paths are added into the set of acceptable paths; to be precise, $2^n - 1$ out of 2^n paths become acceptable for an utterance lasting n frames. This boosts the probability of the bag being positive to an undesirable extent. In a word, the essence of CTC is taken away when it is converted into noisy-or pooling.

The "max" and "noisy-or" pooling functions can also be regarded as examples of the general p-norm:

$$||\mathbf{x}||_p = \left(\sum_i |x_i|^p \right)^{1/p} \tag{4.14}$$

Recall the expressions of the two pooling functions:

$$\text{Max pooling:} \qquad y = \max_i y_i \tag{4.15a}$$

$$\text{Noisy-or pooling:} \qquad y = 1 - \prod_i (1 - y_i) \tag{4.15b}$$

where the y_i's are instance-level predictions, and y is the bag-level prediction. Let $u_i = -\log(1 - y_i)$, and $u = -\log(1 - y)$, then we have:

$$\text{Max pooling:} \qquad u = \max_i u_i \tag{4.16a}$$

$$\text{Noisy-or pooling:} \qquad u = \sum_i u_i \tag{4.16b}$$

Let $\mathbf{u}$ be the vector containing all the u_i's, then the max and noisy-or pooling functions correspond to the ∞-norm and 1-norm of this vector. When p gets larger, the p-norm of a vector is more and more determined by the *single largest* entry of the vector; repeated entries and small entries play a weaker and weaker role. For sequence learning tasks involving long sequences, we want to downplay repeating entries because the frames are not independent, and we want to downplay small entries in order to avoid negative frame-level predictions making the sequence-level prediction positive. The max pooling function, which corresponds to $p = \infty$, satisfies these conditions.

From the analysis in this section, we draw the following conclusions: the noisy-or pooling function may be suited for *non-sequence learning problems* with *small bags*. For such problems, it is more reasonable to assume independence between instances in a bag, negative instance-level predictions do not easily make the bag-level prediction positive, and we need the gradient to flow through every instance because there are no recurrent layers to propagate the gradient across instances. For sequence learning problems such as speech recognition and SED, max pooling is a better choice.

4.3 Proposed Work: Learning with Presence/ Absence Labeling on Large Data

I propose to train networks for SED on large data. One such corpus is Google Audio Set [52], which contains as much as 8 months of audio data. With this amount of data, I will be able to train end-to-end networks, *i.e.* ones that directly take waveforms as the input. The networks will start with convolutional layers that perform feature extraction, followed by recurrent layers that deal with context dependency. The frame-level predictions will be aggregated with the max pooling function, since we have seen that it trains effectively and predicts wide peaks desirable for the temporal localization of sound events.

Google Audio Set contains more than 500 types of sound events, which may be a burden for training. A good idea is to pick a smaller set of sound events to use as targets. Two sets of target sound events may be considered. The first is the event set used in Task 4 of the DCASE 2017 challenge[4], which includes 17 types of vehicle sound and warning sounds that are relevant for smart cars. Evaluating on this event set will enable me to compare my systems with those participating in the challenge. The second is the set of 17 types of sound events in the Noiseme corpus. Systems built for this event set will be able to generate strong labels on the Google Audio Set, which could be used to augment the training data for the experiments in Chapter 3.

[4]http://www.cs.tut.fi/sgn/arg/dcase2017/challenge/task-large-scale-sound-event-detection

The SED system will be evaluated for their ability to detect the presence of sound events in the 10-second excerpts, measured by the F_1 score of each event type. However, we are more interested in how well the systems can localize sound events in time. For this purpose, I will study how well the predictions of the networks align with the actual timespan of the sound events. Because Audio Set does not come with the exact onset and offset times of sound events, some manual inspection will be needed. If the predictions do not align with actual events to a satisfactory degree, I will try to improve them by training a deep feed-forward network alongside the recurrent neural network (as proposed in Sec. 3.5.3), in order to emphasize local information.

I am also interested in the ability of the networks to discover the internal structures of sound events. Specifically, I would like to compare the behavior of the networks on transient events (*e.g.* pulses), continuous events (*e.g.* cheering), and intermittent events (*e.g.* footsteps). Ideally, these three types of events should exhibit different patterns in the network predictions:

- Transient events should produce short predictions that last only one or a few frames, and the lengths of the intervals between occurrences should be random;
- Stationary events should produce long predictions that may last as long as an entire recording;
- Intermittent events should produce many predictions whose lengths cluster strongly around a typical value, and the lengths of the intervals between occurrences should also concentrate around a typical value.

I will compute histograms such as the number of predicted occurrences per recording, the duration of predicted occurrences, and the length of the intervals between predicted occurrences, and see if it is possible to classify an event type as transient, continuous or intermittent based on these statistics.

Chapter 5

Contributions and Timeline

5.1 Contributions of This Proposal

This proposal aims to demonstrate that it is possible to build neural networks that learn to perform sound event detection (SED) with weak labeling. I deal with two types of weak labeling: *sequential labeling*, where the reference takes the form of sequences of tokens (*e.g.* event boundaries) without precise time markers, and *presence/absence labeling*, where we only know whether each type of sound event is present or absent in each recording. I would like to show that it is possible to localize the onset and offset times of sound event instances, even though such information is not directly present in the labeling. I will also demonstrate the benefits of feature extraction by transfer learning, and improve upon existing transfer learning feature extractors.

Below is a summary of the points I have achieved or will achieve in the next year:

- For sequential labeling:
  - Show that it is possible to learn SED from sequential labeling with connectionist temporal classification (CTC) (Sec. 3.2);
  - Show that the SED performance can be improved by semi-supervised training on a large corpus, even though the labels on this corpus are noisy (Sec. 3.5.1);
  - Show that CTC can accurately detect both long events and short events (Sec. 3.5.2), and is better at detecting short events than frame-wise models (Sec. 3.4);
  - Show that the temporal localization power of CTC networks can be improved by training recurrent layers and feed-forward layers in parallel (Sec. 3.5.3).
- For transfer learning:

  - Show that features extracted by transfer learning can preserve the temporal resolution and produce better performance than low-level features (Sec. 3.3 and 3.5.4);
  - Compare the performance of features transferred from an image learning task and an audio learning task, and combine the two for even better performance (Sec. 3.5.5).

- For presence/absence labeling:
  - Show that it is possible to train an end-to-end SED system from presence/absence labeling on a large corpus such as Google Audio Set, using the multiple instance learning (MIL) framework (Sec. 4.3);
  - Make a comparison between the "max" and "noisy-or" pooling functions (Sec. 4.1 and 4.2);
  - Show that it is possible to localize sound events in time even though the labeling does not contain temporal information (Sec. 4.3);
  - Show that the SED system can tell the difference between transient, continuous and intermittent sound events (Sec. 4.3);
  - Compare our SED performance with the participants of the DCASE 2017 challenge (Sec. 4.3).

5.2 Potential Applications

This proposal does not only contribute to the field of sound event detection. A direct downstream application of SED is multimedia event detection (MED), which is the task of detecting what activity is happening in video recordings (*e.g.* parade, birthday party). MED is usually performed in two stages. The first stage generates a high-level representation of each recording, which can be either a single vector or a sequence of frame-wise vectors. The confidence of sound events being active at each frame is a popular representation. The second stage performs binary or multi-class classification on the representations of recordings to decide which activities are active. Common classifiers include support vector machines (SVMs), recurrent neural networks (RNNs), and a model called "recurrent SVMs" we have devised in [9] which combines the advantages of the two. Having improved sound event detectors will no doubt also improve the performance of multimedia event detection.

The techniques studied in this work, including connectionist temporal classification (CTC) and multiple instance learning (MIL), can also apply to other sequence learning tasks with weak supervision. For example, they may be used to detect scenes and actions in videos (*e.g.* fighting), if the data comes with textual descriptions that specify sequences or the presence/absence of scenes and actions but do not specify their exact starting and ending times [33]. These techniques may also used to diagnose pathological speech: if we collect speech from a sufficient number of patients labeled with the types of speech anomalies they are diagnosed with (*e.g.* stuttering), we can train a system that learns what these types of anomalies sound like and helps in diagnosing future patients. They may also be applied to motion tracking data to recognize the moving states of people carrying mobile devices (*e.g.* walking, running, driving), because the users may only provide information about what they did but not exactly when they did it. Another application is the detection and localization of malicious code or vulnerabilities in programs [87], because often we only know that a program has a bug but do not know which part of the code causes the bug.

5.3 Timeline

A tentative timeline for the proposed work is given below.

Time	**Work**	**Corresponding Section**
10/2017 – 11/2017	Train SED systems with presence/absence labeling on Google Audio Set	4.3
11/2017 – 12/2017	Generate strong and sequential labelings on Google Audio Set, and perform semi-supervised training with sequential labeling	3.5.1
01/2018	Train sequential labeling systems that better exploit long events	3.5.2
02/2018 – 03/2018	Improve the temporal localization of CTC by training a recurrent network and a feed-forward network in parallel	3.5.3
03/2018 – 04/2018	Train transfer learning feature extractors that preserve temporal resolution	3.5.4
05/2018	Compare and combine different transfer learning features	3.5.5
06/2018 – 07/2018	Thesis writing and defense	

Table 5.1: Timeline for the proposed work.

Bibliography

[1] D. Wang and G. J. Brown, *Computational auditory scene analysis: Principles, algorithms, and applications*. Wiley-IEEE Press, 2006.

[2] Y.-T. Peng, C.-Y. Lin, M.-T. Sun, and K.-C. Tsai, “Healthcare audio event classification using hidden Markov models and hierarchical hidden Markov models,” in *International Conference on Multimedia and Expo (ICME)*, IEEE, 2009, pp. 1218–1221.

[3] P. Laffitte, D. Sodoyer, C. Tatkeu, and L. Girin, “Deep neural networks for automatic detection of screams and shouted speech in subway trains,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2016, pp. 6460–6464.

[4] C. Clavel, T. Ehrette, and G. Richard, “Events detection for an audio-based surveillance system,” in *International Conference on Multimedia and Expo (ICME)*, IEEE, 2005, pp. 1306–1309.

[5] S. Chaudhuri, M. Harvilla, and B. Raj, “Unsupervised learning of acoustic unit descriptors for audio content representation and classification,” in *Proceedings of Interspeech*, ISCA, 2011, pp. 2265–2268.

[6] B. Byun, I. Kim, S. M. Siniscalchi, and C.-H. Lee, “Consumer-level multimedia event detection through unsupervised audio signal modeling,” in *Proceedings of Interspeech*, ISCA, 2012, pp. 2081–2084.

[7] Y. Wang, S. Rawat, and F. Metze, “Exploring audio semantic concepts for event-based video retrieval,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2014, pp. 1360–1364.

[8] Y. Wang, L. Neves, and F. Metze, “Audio-based multimedia event detection using deep recurrent neural networks,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2016, pp. 2742–2746.

[9] Y. Wang and F. Metze, “Recurrent support vector machines for audio-based multimedia event detection,” in *International Conference on Multimedia Retrieval (ICMR)*, ACM, 2016, pp. 265–269.

[10] X. Zhou, X. Zhuang, M. Liu, H. Tang, M. Hasegawa-Johnson, and T. Huang, “HMM-based acoustic event detection with AdaBoost feature selection,” in *Multimodal Technologies for Perception of Humans*, Springer, 2008, pp. 345–353.

[11] A. Mesaros, T. Heittola, A. Eronen, and T. Virtanen, “Acoustic event detection in real life recordings,” in *European Signal Processing Conference (EUSIPCO)*, IEEE, 2010, pp. 1267–1271.

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

[13] D. D. Lee and H. S. Seung, “Algorithms for non-negative matrix factorization,” in *Advances in Neural Information Processing Systems (NIPS)*, 2001, pp. 556–562.

[14] T. Heittola, A. Mesaros, T. Virtanen, and A. Eronen, “Sound event detection in multisource environments using source separation,” in *Workshop on Machine Listening in Multisource Environments (CHiME)*, 2011, pp. 36–40.

[15] T. Heittola, A. Mesaros, T. Virtanen, and M. Gabbouj, “Supervised model training for overlapping sound events based on unsupervised source separation,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2013, pp. 8677–8681.

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

[17] O. Gencoglu, T. Virtanen, and H. Huttunen, “Recognition of acoustic events using deep neural networks,” in *European Signal Processing Conference (EUSIPCO)*, IEEE, 2014, pp. 506–510.

[18] M. Ravanelli, B. Elizalde, K. Ni, and G. Friedland, “Audio concept classification with hierarchical deep neural networks,” in *European Signal Processing Conference (EUSIPCO)*, IEEE, 2014, pp. 606–610.

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

[20] M. Espi, M. Fujimoto, Y. Kubo, and T. Nakatani, “Spectrogram patch based acoustic event detection and classification in speech overlapping conditions,” in *Joint Workshop on Hands-free Speech Communication and Microphone Arrays (HSCMA)*, IEEE, 2014, pp. 117–121.

[21] H. Zhang, I. McLoughlin, and Y. Song, "Robust sound event recognition using convolutional neural networks," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2015, pp. 559–563.

[22] H. Phan, L. Hertel, M. Maass, and A. Mertins, "Robust audio event recognition with 1-max pooling convolutional neural networks," *arXiv e-prints*, Apr. 2016. [Online]. Available: `http://arxiv.org/abs/1604.06338`.

[23] K. J. Piczak, "Environmental sound classification with convolutional neural networks," in *International Workshop on Machine Learning for Signal Processing (MLSP)*, IEEE, 2015.

[24] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017.

[25] N. Takahashi, M. Gygli, B. Pfister, and L. Van Gool, "Deep convolutional neural networks and data augmentation for acoustic event detection," *arXiv e-prints*, Apr. 2016. [Online]. Available: `http://arxiv.org/abs/1604.07160`.

[26] Y. Tokozume and T. Harada, "Learning environmental sounds with end-to-end convolutional neural network," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2017, pp. 2721–2725.

[27] A. Gorin, N. Makhazhanov, and N. Shmyrev, "DCASE 2016 sound event detection system based on convolutional neural network," DCASE2016 Challenge, Tech. Rep., 2016.

[28] M. Espi, M. Fujimoto, K. Kinoshita, and T. Nakatani, "Exploiting spectro-temporal locality in deep learning based acoustic event detection," *EURASIP Journal on Audio, Speech, and Music Processing*, 2015.

[29] G. Parascandolo, H. Huttunen, and T. Virtanen, "Recurrent neural networks for polyphonic sound event detection in real life recordings," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2016, pp. 6440–6444.

[30] S. Adavanne, G. Parascandolo, P. Pertilä, T. Heittola, and T. Virtanen, "Sound event detection in multichannel audio using spatial and harmonic features," in *Workshop on Detection and Classification of Acoustic Scenes and Events (DCASE)*, IEEE, 2016, pp. 6–10.

[31] T. Hayashi, S. Watanabe, T. Toda, T. Hori, J. Le Roux, and K. Takeda, "Bidirectional LSTM-HMM hybrid system for polyphonic sound event detection," in *Workshop on Detection and Classification of Acoustic Scenes and Events (DCASE)*, IEEE, 2016, pp. 35–39.

[32] E. Çakır, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen, "Convolutional recurrent neural networks for polyphonic sound event detection," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 6, pp. 1291–1303, 2017.

[33] D.-A. Huang, F.-F. Li, and J. C. Niebles, "Connectionist temporal modeling for weakly supervised action labeling," in *European Conference on Computer Vision*, 2016, pp. 137–153.

[34] Y. Wang and F. Metze, "A first attempt at polyphonic sound event detection using connectionist temporal classification," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2017, pp. 2986–2990.

[35] Y. Wang and F. Metze, "A transfer learning based feature extractor for polyphonic sound event detection using connectionist temporal classification," in *Proceedings of Interspeech*, ISCA, 2017, pp. 3097–3101.

[36] J. Amores, "Multiple instance classification: Review, taxonomy and comparative study," *Artificial Intelligence*, vol. 201, pp. 81–105, 2013.

[37] T.-W. Su, J.-Y. Liu, and Y.-H. Yang, "Weakly-supervised audio event detection using event-specific gaussian filters and fully convolutional networks," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2017, pp. 791–795.

[38] A. Kumar and B. Raj, "Audio event detection using weakly labeled data," in *Multimedia Conference*, ACM, 2016, pp. 1038–1047.

[39] O. Maron and T. Lozano-Pérez, "A framework for multiple-instance learning," in *Advances in Neural Information Processing Systems (NIPS)*, 1998, pp. 570–576.

[40] C. Zhang, J. C. Platt, and P. A. Viola, "Multiple instance boosting for object detection," in *Advances in Neural Information Processing Systems (NIPS)*, 2006, pp. 1417–1424.

[41] B. Babenko, P. Dollár, Z. Tu, and S. Belongie, "Simultaneous learning and alignment: Multi-instance and multi-pose learning," in *Workshop on Faces in Real-Life Images: Detection, Alignment, and Recognition*, 2008.

[42] B. Raj and A. Kumar, "Audio event and scene recognition: A unified approach using strongly and weakly labeled data," in *International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2017, pp. 3475–3482.

[43] K. J. Piczak, "ESC: Dataset for environmental sound classification," in *Multimedia Conference*, ACM, 2015, pp. 1015–1018.

[44] J. Salamon, C. Jacoby, and J. P. Bello, “A dataset and taxonomy for urban sound research,” in *Multimedia Conference*, ACM, 2014, pp. 1041–1044.

[45] R. Stiefelhagen, K. Bernardin, R. Bowers, J. Garofolo, D. Mostefa, and P. Soundararajan, “The CLEAR 2006 evaluation,” in *International Evaluation Workshop on Classification of Events, Activities and Relationships*, 2006, pp. 1–44.

[46] R. Stiefelhagen, K. Bernardin, R. Bowers, R. Rose, M. Michel, and J. Garofolo, “The CLEAR 2007 evaluation,” in *International Evaluation Workshop on Classification of Events, Activities and Relationships*, 2007, pp. 3–34.

[47] X. Zhuang, X. Zhou, M. A. Hasegawa-Johnson, and T. S. Huang, “Real-world acoustic event detection,” *Pattern Recognition Letters*, vol. 31, no. 12, pp. 1543–1551, 2010.

[48] A. Temko, R. Malkin, C. Zieger, D. Macho, C. Nadeu, and M. Omologo, “CLEAR evaluation of acoustic event detection and classification systems,” in *International Evaluation Workshop on Classification of Events, Activities and Relationships*, 2006, pp. 311–322.

[49] A. Mesaros, T. Heittola, and T. Virtanen, “TUT database for acoustic scene classification and sound event detection,” in *European Signal Processing Conference (EUSIPCO)*, IEEE, 2016, pp. 1128–1132.

[50] T. Heittola, A. Mesaros, A. Eronen, and T. Virtanen, “Audio context recognition using audio event histograms,” in *European Signal Processing Conference (EUSIPCO)*, IEEE, 2010, pp. 1272–1276.

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

[52] J. F. Gemmeke, D. P. W. 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 *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2017, pp. 776–780.

[53] Y. Aytar, C. Vondrick, and A. Torralba, “SoundNet: Learning sound representations from unlabeled video,” in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 892–900.

[54] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li, “YFCC100M: The new data in multimedia research,” *Communications of the ACM*, vol. 59, no. 2, pp. 64–73, 2016.

[55] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, ACM, 2015.

[56] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 6088, pp. 533–536, 1988.

[57] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, May 2016. [Online]. Available: `http://arxiv.org/abs/1605.02688`.

[58] M. Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. [Online]. Available: `http://tensorflow.org/`.

[59] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A Matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[60] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/\mathrm{sqr}(k))$," *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[61] T. Tieleman and G. Hinton, *RMSprop: Divide the gradient by a running average of its recent magnitude*, Coursera: Neural Networks for Machine Learning, Lecture 6.5, 2012.

[62] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 7, pp. 2121–2159, 2011.

[63] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," *arXiv e-prints*, Dec. 2012. [Online]. Available: `http://arxiv.org/abs/1212.5701`.

[64] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv e-prints*, Dec. 2014. [Online]. Available: `http://arxiv.org/abs/1412.6980`.

[65] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[66] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, ACM, 2015, pp. 448–456.

[67] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[68] 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.

[69] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural networks*, vol. 1, no. 4, pp. 339–356, 1988.

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

[71] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NIPS Workshop on Deep Learning*, 2014.

[72] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[73] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *International Conference on Machine Learning (ICML)*, ACM, 2006, pp. 369–376.

[74] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[75] T. Bluche, H. Ney, J. Louradour, and C. Kermorvant, "Framewise and CTC training of neural networks for handwriting recognition," in *International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2015, pp. 81–85.

[76] F. Eyben, M. Wöllmer, and B. Schuller, "openSMILE – the Munich versatile and fast open-source audio feature extractor," in *Multimedia Conference*, ACM, 2010, pp. 1459–1462.

[77] F. Eyben, F. Weninger, F. Gross, and B. Schuller, "Recent developments in openSMILE, the Munich open-source multimedia feature extractor," in *Multimedia Conference*, ACM, 2013, pp. 835–838.

[78] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[79] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International Conference on Machine Learning (ICML)*, ACM, 2015, pp. 2342–2350.

[80] S. Mun, S. Shon, W. Kim, D. K. Han, and H. Ko, "Deep neural network based learning and transferring mid-level audio features for acoustic scene classification," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2017, pp. 796–800.

[81] F. Chollet *et al.*, *Keras*, 2015. [Online]. Available: `https://github.com/fchollet/keras`.

[82] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010, pp. 249–256.

[83] A. Mesaros, T. Heittola, and T. Virtanen, "Metrics for polyphonic sound event detection," *Applied Sciences*, vol. 6, no. 6, pp. 162–178, 2016.

[84] A. Zeyer, E. Beck, R. Schlüter, and H. Ney, "CTC in the context of generalized full-sum HMM training," 2017.

[85] Y. Xu, Q. Kong, Q. Huang, W. Wang, and M. D. Plumbley, "Attention and localization based on a deep convolutional recurrent model for weakly supervised audio tagging," in *Proceedings of Interspeech*, ISCA, 2017, pp. 3083–3087.

[86] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *Workshop on Automatic Speech Recognition and Understanding (ASRU)*, IEEE, 2015, pp. 167–174.

[87] B. Athiwaratkun and J. W. Stokes, "Malware classification with LSTM and GRU language models and a character-level CNN," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2017, pp. 2482–2486.