Polyphonic Sound Event Detection with Weak Labeling

Yun Wang

CMU-LTI-18-017

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis committee:

Prof. Florian Metze, Chair (Carnegie Mellon University)
Prof. Alex Waibel (Carnegie Mellon University)
Prof. Alex Hauptmann (Carnegie Mellon University)
Dr. Aren Jansen (Google Inc.)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Language and Information Technologies

© 2018 Yun Wang

Abstract

Sound event detection (SED) is the task of detecting the type as well as the onset and offset times of sound events in audio streams. It is useful for multimedia retrieval, surveillance, *etc*. SED is difficult because sound events exhibit diverse temporal and spectral characteristics, and because they can overlap with each other.

Ideally, SED systems should be trained with *strong labeling*, which provides the type, onset time and offset time of each sound event occurrence. However, such labeling is formidably tedious to produce by hand. Current research on SED often uses *weak labeling*. This thesis deals with two types of weak labeling: *presence/absence labeling*, which only states which types of events are present in each recording without any temporal information, and *sequential labeling*, which only provides the order of sound events, but without timestamps. Even if the training data is weakly labeled, we still want our SED systems to localize the sound events in time.

SED with presence/absence labeling is usually treated as a multiple instance learning (MIL) problem, which requires a pooling function. In this thesis, we compare six pooling functions both theoretically and empirically, and establish the linear softmax pooling function as the optimal. Using this function, we build a state-of-the-art network that not only recognizes the types of sound events, but also localizes them temporally.

SED with sequential labeling has not received much attention. In this thesis, we propose a novel connectionist temporal localization (CTL) framework, which successfully makes use of the extra temporal information in sequential labeling compared with presence/absence labeling.

Transfer learning is a popular technique to deal with insufficient training data. In this thesis we extract features from two neural networks trained for out-of-domain tasks, and show that these features can improve the SED performance when the training corpus is small.

Keywords: Sound event detection (SED), weak labeling, multiple instance learning (MIL), pooling function, connectionist temporal classification (CTC), connectionist temporal localization (CTL), transfer learning

Contents

List of Figures

List of Tables

List of Abbreviations

AP	average precision
AUC	area under the curve
BPTT	back-propagation through time
BS	bag-space
CASA	computational auditory scene analysis
CHIL	Computers in the Human Interaction Loop
CLEAR	Classification of Events, Activities and Relationships
CMO	confusion matrix ordering
CMU	Carnegie Mellon University
CNN	convolutional neural network
CRNN	convolutional and recurrent neural network
CTC	connectionist temporal classification
CTL	connectionist temporal localization
DCASE	Detection and Classification of Acoustic Scenes and Events
DNN	deep neural network
ES	embedding-space
ESC	Environmental Sound Classification
FLAC	Free Lossless Audio Codec
FN	false negative
FP	false positive
GMM	Gaussian mixture model
GRU	gated recurrent unit
HME	hierarchical mixture of experts
HMM	hidden Markov model
HTML	hypertext markup language
iff	if and only if
IS	instance-space
KL	Kullback-Leibler (divergence)
LIUM	Laboratoire d'Informatique de l'Université du Maine
LSTM	long short-term memory
MAP	mean average precision

MAUC	mean area under the curve
MED	multimedia event detection
MFCC	Mel-frequency cepstral coefficients
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
ROC	receiver operator characteristic
SED	sound event detection
SGD	stochastic gradient descent
SLAT	strong labeling assumption training
SMI	standard multiple instance (assumption)
SN-C	SoundNet, convolutional
SN-CR	SoundNet, convolutional and recurrent
SN-F	SoundNet, fully connected
SN-R	SoundNet, recurrent
SVM	support vector machine
TDNN	time-delay neural network
TED	Technology, Entertainment, Design
TEDLIUM	(see TED and LIUM)
TALNet	tagging and localization network
TP	true positive
TREC	Text Retrieval Conference
TUT	Tampere University of Technology
VGG	Visual Geometry Group
WER	word error rate
YFCC	Yahoo Flickr Creative Commons

Chapter 1

Introduction

The environment of our daily life is filled with a complex mixture of sound events, such as cars passing by in the streets, or doors opening and closing in the office. Sound events exhibit great variability in a number of aspects. They may originate from various sources (*e.g.* human, animals, machinery, nature; see Fig. 1.1), and these sources may be either fixed or moving. In terms of spectral characteristics, sound events may be either 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.

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

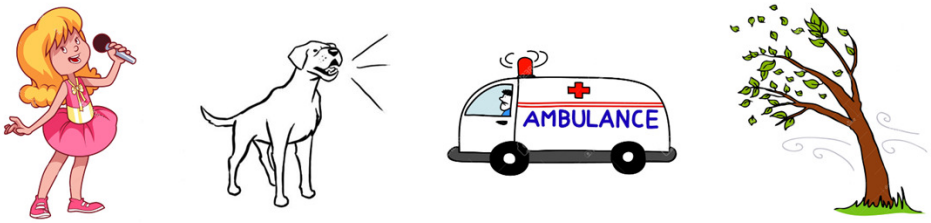


Figure 1.1: Example sound events: singing, dog barking, ambulance siren, wind.

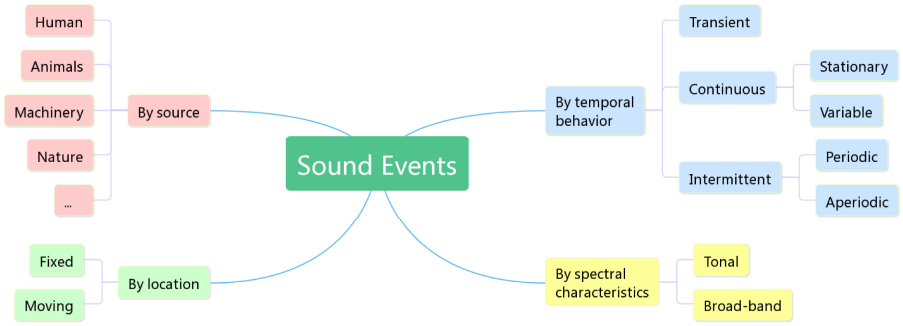


Figure 1.2: A categorization of sound events.

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. While there are works that approach the task of sound event detection jointly with acoustic scene recognition [2, 3] or with source separation [4, 5], we focus on the task of *sound event detection* alone in this thesis.

The term "sound event detection" (SED) actually entails two subtasks: *classification* and *localization*. Classification refers to determining the type(s) of the sound event(s) occurring in an audio recording, without pinpointing the onset and offset times of the events. It is also known as "audio tagging" from a perspective of information retrieval. Localization refers to determining the precise onset and offset times of each sound event occurrence. While the scope of many works are limited to audio tagging, we keep the task of localization in mind throughout this thesis.

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 [6], screams and shouts in subway trains [7, 8], and gunshots [9]. In the fast developing area of autonomous driving, it is also necessary for smart cars to be able to detect certain sounds in the environment (*e.g.* honks of other cars, sirens of ambulances) and respond to them properly. 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 [10, 11] or defined by humans [12, 13, 14] 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 less than 0 dB. Some SED systems have made a simplifying assumption that there can only be one sound event active at a time; such systems are called *monophonic* SED systems. On the other hand, in this thesis we do not make this assumption, and tackle the more realistic scenario of *polyphonic* sound event detection directly. The systems we build are able to detect occurrences of sound events that overlap with each other.

Another difficulty in developing SED systems is the lack of data. Until 2016, most SED systems were trained with no more than 20 hours of audio. The amount of audio data also restricted the number of sound event types that could be handled. Some corpora provided annotations of dozens of sound event types, but many of these types were so rare that they suffered from nearly zero recall. This difficulty has been alleviated by the release of the Google Audio Set [15] in March 2017, which contains about 8 months of audio data and involves 527 sound event types. However, SED is still far from being a solved task.

The current eminent difficulty of SED is the weakness of the labeling. Ideally, the training data for SED systems should be labeled with the type, onset time and offset time of each sound event occurrence. This type of labeling is called *strong labeling*. Strong labeling makes SED a fully supervised machine learning problem, for which there are plenty of mature models and techniques. However, such labeling can be formidably tedious to create by hand. As a result, the research on how to learn to detect sound events with *weak labeling* has become popular. In this thesis we deal with two types of weak labeling. The first type, which we call *presence/absence*

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

labeling[2], only tells us whether each sound event type is active or not in each recording, and does not contain any temporal information whatsoever. The second type, which we call *sequential labeling*, is slightly stronger: it describes the order in which sound events occur, but still does not provide any timestamps.

In the remainder of this chapter, we will first give a review of the history and state of the art of sound event detection. We will review the development of models, features, and corpora for SED. After that, we will highlight the contributions of this thesis and how they push the frontier of the research.

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

1.1.1 Evolution of Models

A multitude of models have been used to model sound events. An early example is hidden Markov models (HMMs), *e.g.* [16, 17, 18, 19]. 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 (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 were popular because they enjoy the use of "language models" of sound events to rule out unlikely sound event sequences. In the CHIL[3] project [20], it was shown [21] that an HMM-based system [17] localized sound events better than a system that classified short segments of audio with support vector machines (SVMs) [22], even though the latter was more accurate at classifying individual segments.

A drawback of HMMs is the inability to deal with polyphony. To compensate for this, a multi-pass decoding procedure was proposed in [2] 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) [23]. 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

[2] When the term "weak labeling" is used in the literature, it often only refers to presence/absence labeling.

[3] "CHIL" stands for "computers in the human interaction loop".

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 [4, 24], 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, [25] 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 quickly became 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 [26, 27] 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.* [28]) or an HMM (*e.g.* [29]). The simple feed-forward neural network in [28] yielded a significantly better polyphonic SED performance than the HMM system with NMF pre-processing in [24].

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 [30, 31, 32, 33, 34, 35] as well as to detect sound events in mixtures [36, 37]. The CNN in [37] again yielded superior performance compared with the feed-forward network in [29].

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 sound events usually do not exhibit long-range dependence, the unlimited context may 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 [13, 38, 39, 40]. A more recent study [41] 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. Such convolutional and recurrent neural networks (CRNNs) have now become a common model for SED.

The studies above relied upon strong labeling, *i.e.* knowing the exact onset and offset times of the sound event occurrences. In the past two years, SED with presence/absence labeling has become a hot topic of research, because this is the form of labeling provided by the large-scale Google Audio Set [15]. SED with presence/absence labeling is usually treated as a multiple instance learning (MIL) [42] problem: frames in a recording are regarded as instances in a bag, and the presence or absence of events is only known on the bag level. A common approach to solve this MIL problem is to predict the presence or absence of events on the frame level using any type of the neural networks introduced above, and then use a "pooling function" to aggregate the frame-level predictions into a recording-level prediction which can be compared against the ground truth. Conventional pooling functions for MIL include the max pooling function [43, 44] and the noisy-or pooling function [45, 46, 47]. More recent studies have seen the use of many other types of pooling functions, including average pooling [48], "max-of-means" pooling [49], various flavors of softmax pooling [50, 51, 52], and attention-based pooling functions [53, 54, 55, 56, 57]. Many state-of-the-art systems have emerged using these pooling functions, and this thesis will compare some of the pooling functions both theoretically and empirically. Besides aggregating frame-level predictions into a recording-level prediction, more complicated MIL methods have been applied to SED as well, such as multiple instance support vector machines (mi-SVM) [44] and manifold regularization on graphs [3].

Besides presence/absence labeling, weak labeling may also come in the form of sequential labeling. Sequential labeling is currently the predominant form of supervision for training speech recognition systems: we only know the phoneme sequence for each utterance, but not the temporal alignment. Standard technique for training speech recognizers with sequential labeling

include connectionist temporal classification (CTC) [58], RNN transducers [59], and attention [60, 61]. Despite its pivotal importance in speech recognition, sequential labeling has not received much attention in other applications, with an important reason being the lack of data. One of the few examples is [62], where the authors applied CTC to the detection of actions in video recordings given only the order of actions in each recording. We have done some preliminary work of applying CTC to sound event detection [63, 64], but the results were limited. In this thesis, we modify the CTC framework and present a novel connectionist temporal localization (CTL) framework that successfully brings out the advantage of sequential labeling over presence/absence labeling.

1.1.2 Feature Extraction

Before the times of deep learning, the choice of acoustic features was important for sound event detection. Many types of low-level acoustic features were explored, such as spectral, cepstral and perceptual features [65, 22]. In [13] and [63], we extracted as many as 6,669 dimensions of low-level features originally designed for emotion recognition [66] using the OpenSMILE [67] toolkit. Such high-dimensional features often require dimensionality reduction as a post-processing step. Common dimensionality reduction techniques include principal component analysis (PCA) [68] and independent component analysis (ICA) [69].

The advent of deep learning eliminated the need for complex feature engineering. SED systems in the form of convolutional or convolutional and recurrent neural networks usually take the spectrogram or filterbank outputs as input features; such input features can be treated as images, and the convolutional layers in the networks are good at recognizing patterns in them. Some studies (*e.g.* [70]) even feed raw waveforms directly into a neural network, with the confidence that the network can learn the necessary filters by itself.

The study of SED is constantly troubled by the lack of data, and researchers have resorted to transfer learning to alleviate the problem. A common technique of transfer learning is to train a model for a source task with plenty of training data, and then use this model as a feature extractor for the target task. When trained properly, such a feature extractor can highlight the useful information in the input, which can be hard to discover without enough training data. Many models have been trained as feature extractors for SED. Two widely used examples are the VGGish network [71] and SoundNet [72]: the former was trained for audio classification; the latter was trained to predict visual object and scenes from the audio channel. The VGGish network and SoundNet were both trained in a supervised fashion, but models for feature extraction can be trained on unsupervised tasks, too. In [73], a denoising auto-encoder network was trained to reconstruct

filterbank outputs corrupted with noise, and features were extracted from a bottleneck layer in the network. The "Look, Listen and Learn" network [74] was trained to predict whether a video frame and an audio segment matched with each other, and it was found that the network learned a representation of audio that could be used for other audio tasks such as SED. In [75], a neural network was trained to learn relationships such as "X is more like Y than like Z" on triples of audio samples, and it also proved able to produce features that could be transferred to other tasks. In this thesis, we investigate how much the supervised feature extractors (VGGish and SoundNet) can help the learning of SED with weak labeling.

1.1.3 Corpora for Sound Event Detection

Data is of paramount importance to machine learning tasks such as SED. Over the years, the community has compiled many corpora for the training and evaluation of SED systems. The corpora have become larger and larger in size, and their content has also got more and more diverse and realistic. This section gives a review of some notable corpora for SED.

An early collection of sound events is ESC-50[4] [76]. This corpus comprises 2,000 short clips of 5 seconds long, each clip containing an instance of one of 40 sound event types recorded in a clean environment. This corpus is of limited use by itself because the controlled clean recording environments are drastically different from the complex acoustic scenes encountered in real life, but the sound event samples can be used to generate synthetic data as a means of data augmentation.

A corpus of sound events recorded in real-life environments is UrbanSound [77]. 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 organized. The CHIL project [20] conducted a CLEAR[5] evaluation in 2006 and 2007 [78, 79]; more recently, the DCASE[6] challenge has taken place in 2013[7], 2016[8], 2017[9], and 2018[10]. 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

[4] "ESC" stands for "environmental sound classification".
[5] "CLEAR" stands for "classification of events, activities and relationships".
[6] "DCASE" stands for "detection and classification of acoustic scenes and events".
[7] http://c4dm.eecs.qmul.ac.uk/sceneseventschallenge/
[8] http://www.cs.tut.fi/sgn/arg/dcase2016/
[9] http://www.cs.tut.fi/sgn/arg/dcase2017/
[10] http://dcase.community/challenge2018

rooms. The corpus contained 3 hours of development data and 2 hours of evaluation data[11]. 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 [82] 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. The DCASE 2017 SED challenge [83] included a weakly labeled SED task, which provided a much larger corpus. This corpus includes a training set of 51,172 recordings, a public test set of 488 recordings, and a private evaluation set of 1,103 recordings. Each recordings lasts 10 seconds, so the total duration of the data is about 147 hours. The corpus involves 17 types of vehicle and warning sounds that are of interest for smart cars. A special feature of this corpus is that the training set comes with only presence/absence labeling; the test and evaluation sets are still strongly labeled in order to evaluate the localization of sound events.

Researchers have also collected private corpora of sound events at various sites as a complement to the insufficient data publicly available. 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[12] [38]. This corpus was first described in [84] and named "TUT-SED 2009" in [41], and it has been used in a number of studies at TUT [19, 2, 24, 26, 28, 38, 41].

At Carnegie Mellon University (CMU), we have also collected and annotated 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. The corpus has had three versions since its first documentation in [85]; our previous work used Version 2 [13, 14, 63, 64]. 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. 48 types of sound events are annotated with exact onset and offset times; because some event types are rare, we merged them

[11] These numbers follow the description of the CLEAR 2007 evaluation in [18] and [80]. Accounts differ in other references: [81] says the CLEAR 2006 evaluation used 5 seminars each lasting 10 $\sim$ 20 minutes; [79] says the CLEAR 2007 evaluation used 100 minutes of seminar data for development and 200 minutes for evaluation.

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

Name		No. of Recordings	Avg. Rec. Duration	Total Duration	No. SE Types	No. SE Instances	Average Polyphony
ESC-50 [76]		2,000	5 s	2.8 h	50	2,000	
UrbanSound [77]		1,302	75 s	27.0 h	10	3,075	
CLEAR 2007 [18, 80]				5.0 h	12	1,454	
DCASE 2016 [82]		32	214 s	1.9 h	18	1,465	
DCASE 2017 [83]		52,763	10 s	146.6 h	17	N/A	
TUT-SED 2009 [84]		103	660 s	18.9 h	61	10,278	2.53
Noiseme [85]	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 [15]		2.1 million	10 s	8 months	527	N/A	

Table 1.1: A summary of corpora available for SED.

down to 17 types in our previous work. The average polyphony level ranges between 1.40 and 1.43 in the non-silence regions.

In March 2017, Google released Audio Set [15]. This corpus is hundreds of times larger than most of the aforementioned corpora, and is a superset of the DCASE 2017 data. 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 most corpora, Audio Set only comes with *presence/absence labeling*: the annotation of Audio Set does not specify the exact onset and offset times of the sound events, but only indicates whether each type of sound event is present or absent in each 10-second excerpt. Because of this, Audio Set has made the study of SED with presence/absence labeling a hot topic in the last two years.

A summary of the corpora available for SED can be found in Table 1.1. The DCASE 2017 corpus and the Google Audio Set are the primary corpora used in this thesis.

1.2 Contributions of This Thesis

In this thesis we make the following contributions:

- The choice of the pooling function is important in multiple instance learning (MIL) systems for SED with presence/absence labeling. We make a theoretical and empirical comparison of six common pooling functions: max pooling, noisy-or pooling, average pooling, linear softmax pooling, exponential softmax pooling, and attention pooling, and establish linear softmax pooling as the optimal of the six.

- Using the linear softmax pooling function, we build a network called TALNet[13] that can perform audio tagging and localization simultaneously. This system closely matches the state-of-the-art performance

[13] "TAL" stands for "tagging and localization".

on the Google Audio Set [15], while reaching a strong performance on the DCASE 2017 challenge data [83] without any adaptation. As far as we know, this is the first system to exhibit such good performance on both corpora.

- We develop on our previous work and propose the first successful SED system trained with sequential labeling. The system uses a connectionist temporal localization (CTL) framework, which is adapted from the connectionist temporal classification (CTC) framework often used for speech recognition. On a subset of Audio Set with synthetic sequential labels, the system closes one third of the performance gap between presence/absence labeling and strong labeling.
- We investigate the use of the VGGish network [71] and SoundNet [72] as transfer learning feature extractors for SED. We train variants of SoundNet that reach a lower validation loss than the original, and conduct extensive experiments to provide empirical knowledge about when transfer learning is helpful.

The remainder of this thesis is organized as follows. Chapter 2 gives a review of machine learning models and techniques relevant to this thesis, including various architectures of neural networks, connectionist temporal classification (CTC), multiple instance learning (MIL), and transfer learning. In Chapter 3, we study sound event detection with presence/absence labeling. We make a theoretical and empirical comparison of six pooling functions for multiple instance learning (MIL), and present our state-of-the-art TALNet which performs audio tagging and localization well at the same time. In Chapter 4, we approach SED with sequential labeling using a connectionist temporal localization (CTL) framework, which is adapted from the connectionist temporal classification (CTC) framework. We demonstrate how the CTL framework overcomes the difficulties encountered in previous studies and brings out the advantage of sequential labeling over presence/absence labeling. In Chapter 5, we present how we trained variants of SoundNet better than the original, and examine the results of extensive experiments to understand when transfer learning is helpful. Finally, Chapter 6 summarizes the contributions of this thesis, and discusses potential applications and future work.

Our previous work that lays the foundation for this thesis has been published in various conferences [13, 14, 63, 64]. A part of the content of Chapter 3 has been published in Interspeech 2018 [86]. Applications of the MIL systems and techniques in this thesis have been published in [87] and [88]. We have also made to submissions to ICASSP 2019 [89, 90], based on the content of Chapters 3 and 4, respectively.

The code for some of the experiments in Chapters 3 and 4 are available on GitHub at `https://github.com/MaigoAkisame/cmu-thesis`.

Chapter 2

Review of Machine Learning Techniques

2.1 Deep Learning and Neural Networks

2.1.1 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, *etc*. All the models used in this thesis are neural networks, too.

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

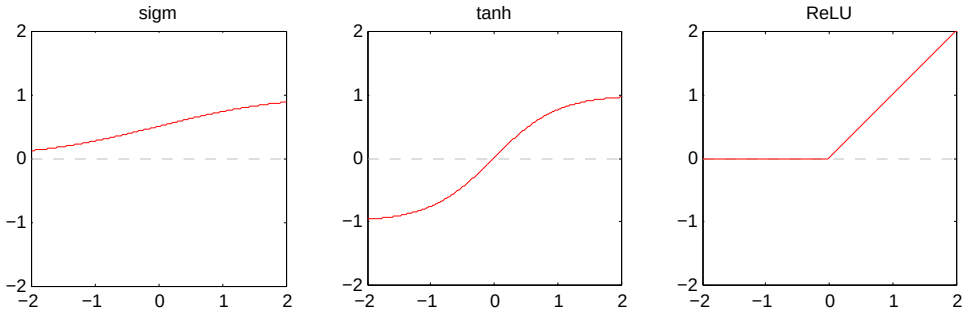


Figure 2.1: Common activation functions used in neural networks.

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 are

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 $t_i = 0$.

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* [91], which in essence is the procedure of repeatedly applying the chain rule of differentiation. Modern deep learning toolkits, such as Theano [92], TensorFlow [93], and Torch [94] (succeeded by PyTorch [95]), 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

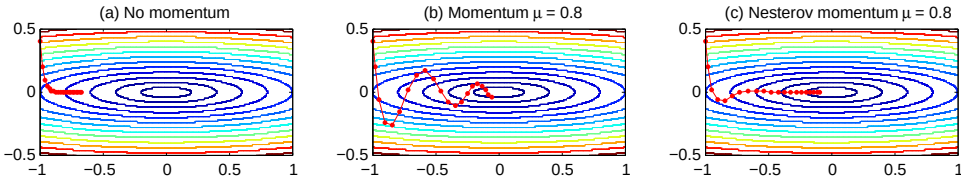


Figure 2.2: The effect of momentum in gradient descent. The loss function being minimized is $f(x, y) = x^2 + 25y^2$. The starting point is $(-1.0, 0.4)$; the learning rate is 0.01 in all the three subfigures. The red dotted curve shows the trajectory of the network parameters in the first 20 iterations.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

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* [96]. 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 [97], Adagrad [98], Adadelta [99], and Adam [100], 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 [101], and batch normalization [102]. These algorithms also apply to the more complex network structures to be introduced in the sections below.

2.1.2 Recurrent Neural Networks (RNN)

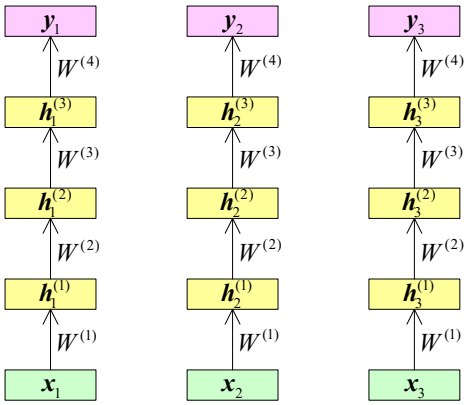
When applied to a sequence input, a feed-forward neural network processes each frame independently, as shown in Fig. 2.3 (a). This means the prediction $\boldsymbol{y}_t$ at time t is only based on the input $\boldsymbol{x}_t$ at the same moment, without using any context information, which can be important 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:

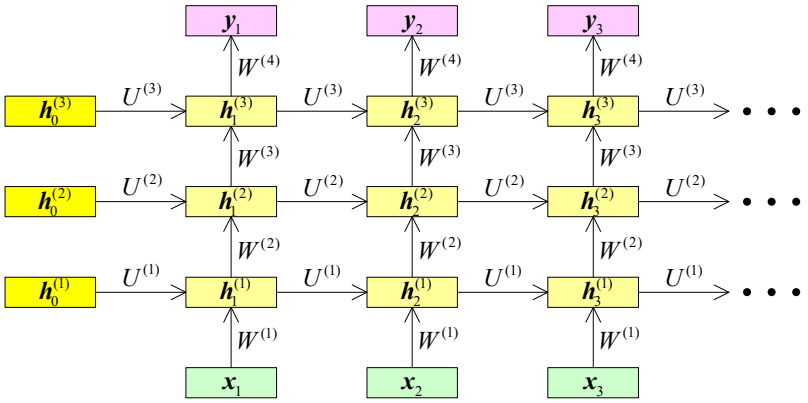
$$\boldsymbol{h}_t^{(l)} = \sigma(U^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}^{(l)}) \tag{2.8}$$

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.

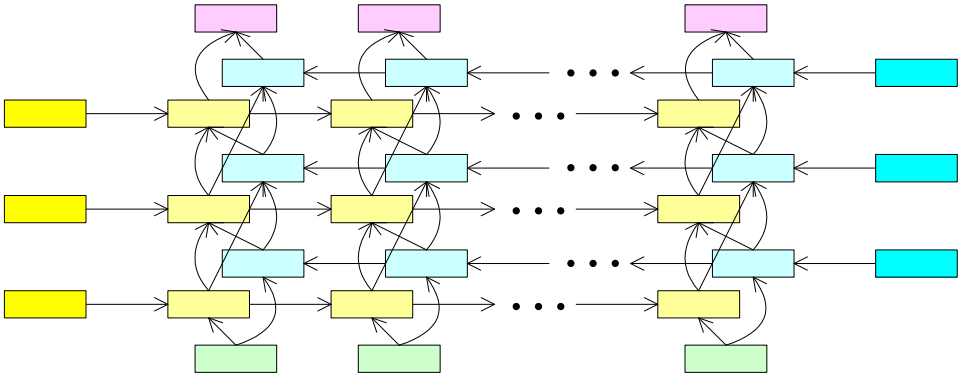
In the RNN structure described above, information flows in only one direction along the time axis. This means the prediction at any time step can only make use of information at and before this time step. But the



(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 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.

future context is often as important as the past context, so it is desirable to use a *bidirectional RNN* structure [103], 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:

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

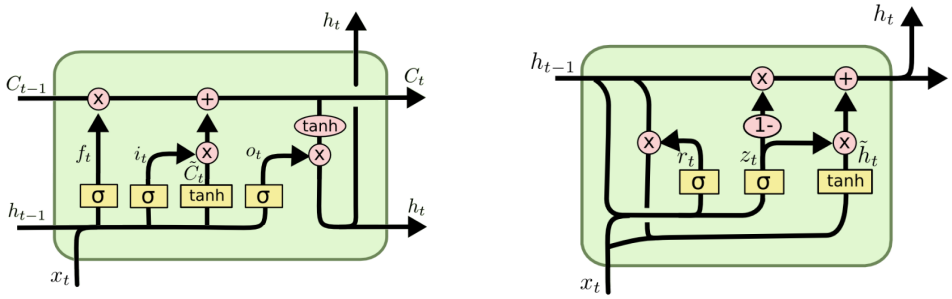
$$\overleftarrow{\boldsymbol{h}}_t^{(l)} = \sigma(\overleftarrow{U}^{(l)}\overleftarrow{\boldsymbol{h}}_{t+1}^{(l)} + W^{(l)}\boldsymbol{h}_t^{(l-1)} + \overleftarrow{\boldsymbol{b}}^{(l)}) \tag{2.9b}$$

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 the simple non-linear functions (Eqs. 2.2a, 2.2b and 2.2c) often encounter difficulty in training, due to a phenomenon called *gradient vanishing* or *gradient explosion* [104]. 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) [105]. 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 [106] and *gated recurrent units* (GRUs) [107].

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 $\tilde{\boldsymbol{c}}_t^{(l)}$ 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



(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). Figures are edited from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

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:

$$\tilde{\boldsymbol{c}}_t^{(l)} = \sigma_c(U_c^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_c^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_c^{(l)}) \tag{2.10a}$$

$$\boldsymbol{i}_t^{(l)} = \text{sigm}(U_i^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_i^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_i^{(l)}) \tag{2.10b}$$

$$\boldsymbol{f}_t^{(l)} = \text{sigm}(U_f^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_f^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_f^{(l)}) \tag{2.10c}$$

$$\boldsymbol{o}_t^{(l)} = \text{sigm}(U_o^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_o^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_o^{(l)}) \tag{2.10d}$$

$$\boldsymbol{c}_t^{(l)} = \boldsymbol{f}_t^{(l)} \odot \boldsymbol{c}_{t-1}^{(l)} + \boldsymbol{i}_t^{(l)} \odot \tilde{\boldsymbol{c}}_t^{(l)} \tag{2.10e}$$

$$\boldsymbol{h}_t^{(l)} = \boldsymbol{o}_t^{(l)} \odot \sigma_h(\boldsymbol{c}_t^{(l)}) \tag{2.10f}$$

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:

$$\boldsymbol{z}_t^{(l)} = \text{sigm}(U_z^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_z^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_z^{(l)}) \tag{2.11a}$$

$$\boldsymbol{r}_t^{(l)} = \text{sigm}(U_r^{(l)}\boldsymbol{h}_{t-1}^{(l)} + W_r^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_r^{(l)}) \tag{2.11b}$$

$$\tilde{\boldsymbol{h}}_t^{(l)} = \sigma_h(U_h^{(l)}(\boldsymbol{r}_t^{(l)} \odot \boldsymbol{h}_{t-1}^{(l)}) + W_h^{(l)}\boldsymbol{h}_t^{(l-1)} + \boldsymbol{b}_h^{(l)}) \tag{2.11c}$$

$$\boldsymbol{h}_t^{(l)} = (\boldsymbol{1} - \boldsymbol{z}_t^{(l)}) \odot \boldsymbol{h}_{t-1}^{(l)} + \boldsymbol{z}_t^{(l)} \odot \tilde{\boldsymbol{h}}_t^{(l)} \tag{2.11d}$$

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 one minus the update gate $(1 - \boldsymbol{z}_t^{(l)})$. When the value $\boldsymbol{f}_t^{(l)}$ or $(1 - \boldsymbol{z}_t^{(l)})$ stays 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 RNNs using simpler non-linear functions.

2.1.3 Convolutional / Time-Delay Neural Networks (CNN / TDNN)

Another network structure suitable for exploiting context information is *convolutional neural networks* (CNNs). Such networks were initially called *time-delay neural networks* (TDNNs) and applied to speech recognition [108, 109, 110, 111, 112]; later they also saw extensive use in image recognition [113, 114, 115].

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 ($m \times n$ is called the *stride* of the pooling layer), 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) input 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 audio 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 audio-related 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 applying pooling across time when the time resolution of the feature maps has been reduced to the desired value.

The benefits of CNNs include *shift invariance* and *locality*. For image recognition, shift invariance means that the prediction for an image should not change when the object of interest moves within the image. Such a property is also partially desirable for audio signals: a phoneme or sound event should be recognized no matter where it occurs in an audio signal, and a limited shift along the frequency axis should not affect the prediction. CNNs ensure shift variance by applying the same convolution kernel to all parts of the input. Locality means that the network has a sense of which parts of the input are next to each other and which parts are far apart. This is ensured by using neurons that receives information only from neurons representing a neighboring region in the layer below. A consequence of locality is that, when applied to audio, CNNs do not have unlimited context as RNNs do. On the other hand, tasks that work with audio may not need an unlimited context, because audio usually does not exhibit long-range dependence the way natural language does.

2.2 Connectionist Temporal Classification (CTC)

When training recurrent and convolutional neural networks, the ideal scenario is to have frame-wise supervision available. For example, in speech recognition, this means we know the exact onset and offset times of each phoneme or sub-phonemic state; in sound event detection, this means we 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 real-world tasks, however, the supervision often comes in the form of a label sequence without alignment. For example, in the typical scenario of training a speech recognition system, the only supervision available is the phoneme sequences of the training utterances, without an alignment between phonemes and frames. While sometimes it is possible to create alignments either manually or automatically, it can be desirable to avoid this extra effort. *Connectionist temporal classification* (CTC) [116] 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 labels 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 labels, such as `ABBA`; (2) it requires that each frame must output a label, 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* label, denoted by "`-`", to the frame-wise output vocabulary. The mapping function works in two steps: first, it reduces consecutive repeating labels into a single one; second, it removes the blank labels. In this way, the alignments `-AB--BAA` and `ABBB-BBA` will both map to the label sequence `ABBA`.

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 [117]. 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 labels inserted at the beginning, at the end, and between every pair of labels. 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 label by l_i. Also let $y_t(c)$ be the probability of the label c in the output distribution at step t. Define $\alpha_t(i)$ as the total probability of the

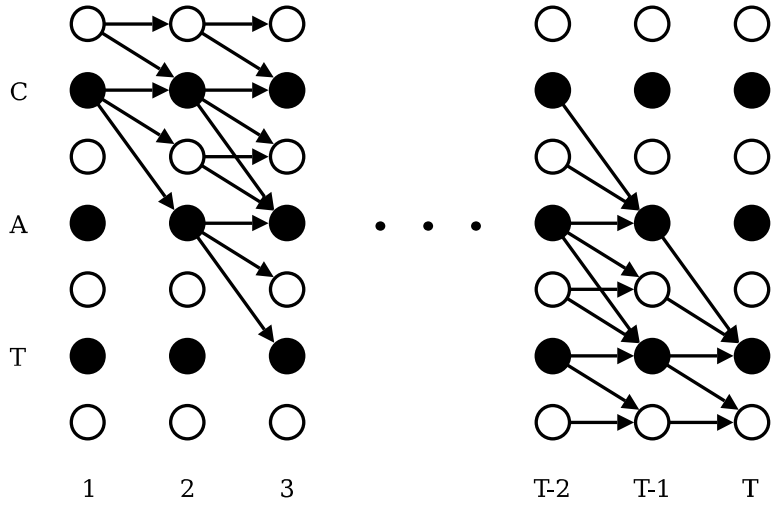


Figure 2.5: The trellis for computing the CTC loss function, taken from [116]. The target label sequence is `CAT`. Black circles represent non-blank labels, and white circles represent blanks.

path landing on l_i at time t and emitting all the labels along the way. At each time step, the path is allowed to stay at the same label, transition to the next label, or skip over the next label, but the skipping can only happen when the label skipped over is a blank, and the two labels 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 label or the blank 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 label 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.

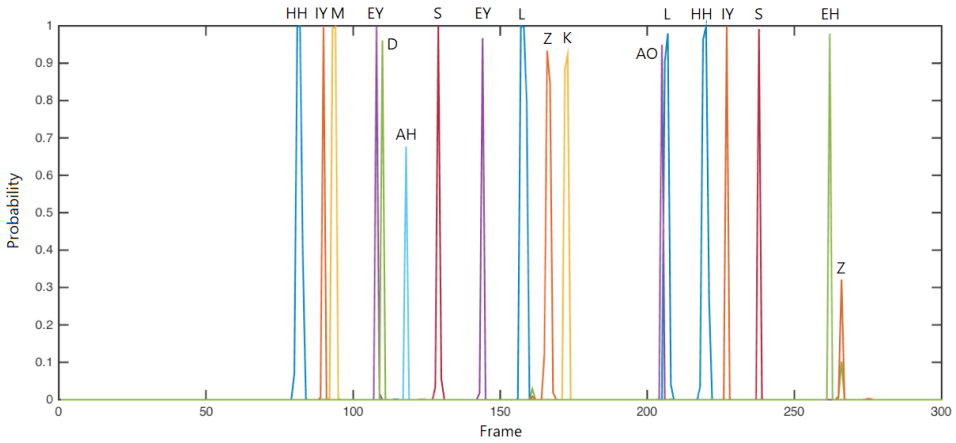


Figure 2.6: The "peaky" output of a CTC speech recognition network on the utterance "*He made a sales call, he says*". Each colored line stands for the probability of a phoneme. Best viewed in color.

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 label 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 [118]. 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 label. Afterwards, "peaks" will occur in the probabilities of non-blank labels, and the sequence formed by reading off the labels 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.

In Chapter 4 of this thesis, we modify the CTC framework to obtain a connectionist temporal localization (CTL) framework. We then apply the latter to the problem of sound event detection with sequential labeling, *i.e.* when the supervision comes in the form of sequences of sound event boundaries without frame-wise alignment.

2.3 Multiple Instance Learning

Multiple instance learning (MIL) is a special case of machine learning. In standard supervised machine learning, we have a set of training instances, each having its own ground truth label. In multiple instance learning, however, we do not know the labels of each individual training instance. Instead, the instances are grouped into bags, and labels are known for the bags only.

The article [119] gives an example of a MIL problem: drug discovery. The task is to predict whether a given molecule is an effective drug or not. A molecule may adopt multiple 3-D shapes (called "conformations"), and it is possible that some conformations of a molecule are effective drugs, while others are not. However, the training data only tells us whether each molecule is effective or not, without providing any information about the conformations. In this example of MIL, the molecules are bags, and their conformations are instances.

The relationship between the bag label and the instance labels may vary. In the situation when all the labels are binary, a common case is the *standard multiple instance (SMI) assumption*: a bag is positive if it contains at least one positive instance, and negative if it only contains negative instances. In the drug discovery problem, for example, a molecule is an effective drug iff it has at least one effective conformation. The task of MIL may be to learn a classifier either for bags or individual instances. Again, in the same example, we may only want to classify molecules as effective or not, but we may also be interested in the question of which conformation(s) make a molecule effective.

The article [42] gives a comprehensive review of methods for MIL, and classifies them into a taxonomy of the following three paradigms:

- **Instance-space (IS):** A classifier is learned on the instance level, whose decisions on the instances in a bag are aggregated to make the bag-level prediction.
- **Bag-space (BS):** The classifier makes decisions directly on the bag level, without making predictions for the instances. However, the model does not extract an explicit feature representation of bags.
- **Embedding-space (ES):** Similar to the bag-space paradigm, but bags are mapped into vectorial representations, which are then classified with a general-purpose classifier.

Generally speaking, bag-space and embedding-space methods are better at classifying bags, but only instance-level methods can make predictions for individual instances.

In the instance-space paradigm, the instance-level predictions are aggregated into the bag-level prediction using a *pooling function*. The choice of

the pooling function depends on the relationship between the instance labels and the bag label. Under the SMI assumption, a natural choice is the max pooling function (*i.e.* taking the "most positive" instance-level prediction as the bag-level prediction), but other pooling functions have also been used for other benefits.

In Chapter 3 of this thesis, we will formulate the task of sound event detection with presence/absence labeling as an instance-space MIL problem, and compare the effectiveness of six different pooling functions.

2.4 Transfer Learning

In supervised machine learning, we often run into situations when we do not have enough labeled training data for the target task. However, there may be plenty of labeled training data available for a different but related task, or for the same task on a different domain. Since the tasks or domains are related, it is natural to hope that we can borrow the knowledge embedded in the training data for the related task or domain to help the learning of the target task. Such borrowing of knowledge is called *transfer learning*.

For example, suppose we are running an online shopping website, and we receive a huge volume of product reviews. Some of these reviews may be spam, and we would like to build a classifier to filter the spam out. We do not have enough labeled training data for the spam filter, but we do have an abundant amount of labeled data for classifying the sentiment of product reviews. Because both tasks involve some extent of language understanding, we can reasonably hope that such knowledge of language can be shared across the two tasks. We may first build a sentiment classifier, then adapt the model to the task of spam detection; or we may extract features for product reviews using the sentiment classifier and build a spam filter that takes these features as input. Both approaches count as typical examples of transfer learning.

The article [120] gives a good review of the types of transfer learning scenarios, and the techniques suitable for each scenario. It defines transfer learning as the action of using knowledge learned from a source task on a source domain to help the learning of a source task on a target domain. Rigorously speaking, the "domain" refers to the feature space the input data lies in and how the data is distributed, while the "task" refers to the output label space and the input-output correspondence to be learned. In general, both the source/target domains and the source/target tasks may be different. But in this thesis, we are only concerned with the case of *inductive transfer learning*, which refers to the case of different tasks on the same domain. This is because all the SED tasks we study are conducted on the domain of natural audio recordings with rich sound events, but the sets of sound event types to be recognized differ from corpus to corpus.

Methods for inductive transfer learning are classified into the following categories in [120] (we use the more concise terminology in Sec. 3.2 of [121]):

- **Instance transfer:** Training instances for the source task are selected or weighted to form the training data for the target task;
- **Model transfer:** A model is learned for the source task, then adapted for the target task;
- **Feature transfer:** A model is learned for the source task, then used as a feature extractor for the target task;
- **Relation transfer:** These methods are used for learning relations between entities, and is irrelevant to this thesis.

In this thesis, we primarily explore the potentials of *feature transfer*. In Chapter 5, we use two neural networks built for different tasks as feature extractors, and study whether they improve the performance of sound event detectors. We would also like to mention that the "TALNet" we build in Sec. 3.3 qualifies as a degenerate case of model transfer.

Chapter 3

Sound Event Detection with Presence/Absence Labeling

This chapter studies the learning of sound events from *presence/absence labeling*: it is only known whether each type of sound event is present or absent in each audio recording; no temporal information is provided whatsoever. Presence/absence labeling has become the mainstream scenario of sound event detection since the release of the Google Audio Set [15] in March 2017; a task of SED with presence/absence labeling has been included in the DCASE challenges of both 2017 and 2018 [83, 122].

SED with presence/absence labeling can be regarded as a multiple instance learning (MIL) problem [42]. We regard each audio recording as a bag, and the frames in the recording as instances. For each sound event type, the presence/absence label serve as the bag label: if an event type is present in a recording, then the recording is a positive bag. The bag labels and the instance labels conform to the standard multiple instance (SMI) assumption: a bag is positive iff at least one of its instances is positive; in other words, an event type is present in a recording iff it is present in any frame of the recording.

Following the terminology of the DCASE 2017 challenge, the task of recognizing which types of sound events are present in a recording without localizing them temporally is called *audio tagging*. Because we are interested in both audio tagging and localization, we approach the problem of polyphonic SED with presence/absence labeling using the instance-space paradigm (see Fig. 3.1). We train an instance-level classifier that predicts the probability of each sound event type at each frame. This classifier may be a convolutional and/or recurrent neural network that communicates information across frames. To accommodate for polyphony, the instance-level probabilities are produced by neurons with the sigmoid activation function, and do not have to sum to one across sound event types. The instance-level predictions are then aggregated (or "pooled") into a bag-level

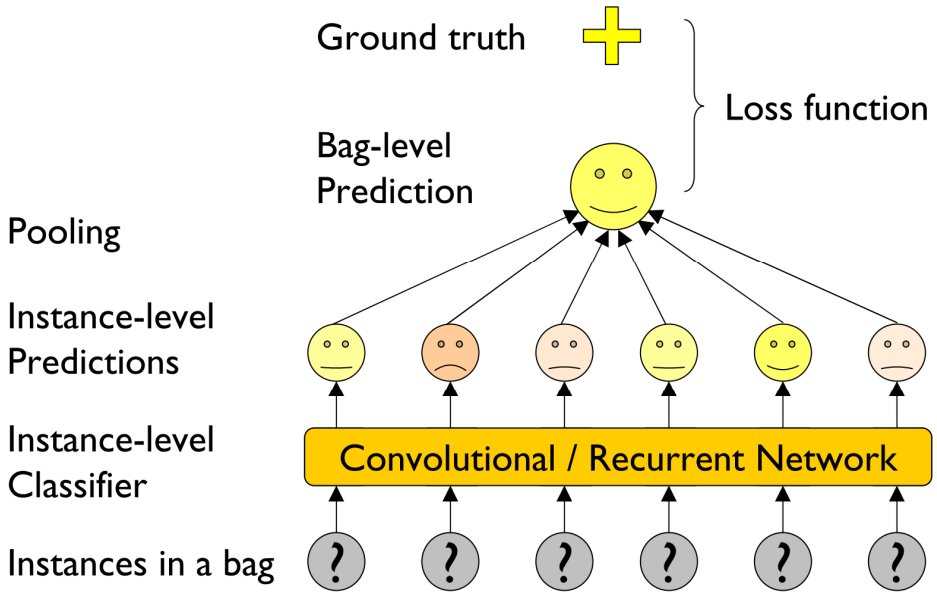


Figure 3.1: Block diagram of an instance-space multiple instance learning (MIL) system for SED.

probability for each sound event type, which can be compared with the bag labels to form a loss function. After the network has been trained, the instance-level predictions can be used for localizing the sound events in time.

A key decision when designing a MIL system is the choice of the pooling function. Traditionally, the max pooling function [43, 44] and the noisy-or pooling function [45, 46, 47] are commonly used. Since the DCASE challenge of 2017, we have also seen the usage of the average pooling function [48], two softmax pooling functions based on linear weighting [50] and exponential weighting [51] respectively, as well as an attention-based pooling function [53, 54]. In this chapter, we compare these pooling functions both theoretically and empirically.

While we do so, we end up with a deep convolutional and recurrent neural network that can perform audio tagging and localization simultaneously. We call this network TALNet, where "TAL" stands for "tagging and localization". This network closely matches the state-of-the-art performance on the Google Audio Set, while exhibiting strong performance on the DCASE 2017 challenge without any adaptation. As far as we know, this is the first system that performs so well on both corpora at the same time.

The content of this chapter is organized as follows. In Sec. 3.1, we compare the max and noisy-or pooling functions. We demonstrate that max pooling succeeds while noisy-or pooling fails on two tasks: a phoneme recognition task on TED talks, and the SED task of the DCASE 2017 challenge, and offer a theoretical explanation. In Sec. 3.2, we study the

behavior of the average pooling function, the two softmax pooling functions and the attention-based pooling function on the DCASE challenge. We show that while they alleviate the high false negative rate observed with the max pooling function, some of them in turn make too many false positives; it is only the linear softmax pooling function that achieves a good balance between false negatives and false positives. In Sec. 3.3, we present TALNet and compare it with a number of state-of-the-art systems found in the literature.

3.1 The Max and Noisy-Or Pooling Functions

3.1.1 Motivation

Consider the instances $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in a MIL problem. Let f be an instance-level classifier, and it makes a prediction $y_i = f(\mathbf{x}_i)$ for each instance. These instance-level predictions are aggregated into a bag-level prediction y using a pooling function.

The *max* pooling function is

$$y = \max_i y_i \tag{3.1}$$

It can be motivated from two perspectives. The first looks at how instance-level and bag-level decisions are made. Suppose the instance-level decisions are made by a thresholding rule: an instance $\mathbf{x}_i$ is classified as positive iff y_i is above or equal to a certain threshold θ. According to the SMI assumption, the bag should be classified as positive iff at least one $y_i \geq \theta$, *i.e.* $\max_i y_i \geq \theta$. This is equivalent to aggregating the instance-level predictions y_i into the bag-level prediction y using the max pooling function, and applying the same threshold θ to the bag-level prediction. This reasoning is valid no matter what the instance-level classifier is.

The second perspective looks at the training procedure of the classifier, assuming it is a support vector machine (SVM) $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \in \mathbb{R}$. Let $t \in \{1, -1\}$ be the label of the bag. If the bag is positive ($t = 1$), the training needs to make sure that at least one instance satisfies $y_i \geq 1$; 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:

$$t \cdot \max_i y_i \geq 1 \tag{3.2}$$

If we aggregate the instance-level predictions y_i into the bag-level prediction y using the max pooling function, then the training procedure can be simplistically regarded as training a bag-level SVM to satisfy $t \cdot y \geq 1$.

SVMs optimize the margin between positive and negative instances. However, many other classifiers (including neural networks) optimize a

probabilistic objective function (such as *binary cross-entropy* listed in Table 2.1), and they treat both the instance-level prediction y_i and bag-level prediction y as probabilities. In this case, the SMI assumption (a bag is negative iff all instances are negative) dictates a different relationship between y_i and y:

$$y = 1 - \prod_i (1 - y_i) \tag{3.3}$$

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 appears to be a more natural choice for probabilistic classifiers, but it assumes that all the instances in a bag are mutually independent. This assumption is questionable in the case of SED, because the frames in a recording are clearly correlated.

Now we have two pooling functions motivated in different ways. The noisy-or pooling function appears to be more suitable for probabilistic classifiers, but relies upon a questionable assumption of independence. Before we move on to validate them using experiments, we would like to further analyze them theoretically in terms of the gradient flow.

3.1.2 The Gradient Flow

A key step in the training of neural networks 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 y is aggregated from the instance-level predictions $y_1, \ldots, y_n \in [0, 1]$, using either the max (Eq. 3.1) or noisy-or (Eq. 3.3) 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 binary cross-entropy loss function

$$L = -t \log y - (1 - t) \log(1 - y) \tag{3.4}$$

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

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:

$$L = -t \log y_k - (1 - t) \log(1 - y_k) \tag{3.5}$$

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

$$\frac{\partial L}{\partial z_k} = -\frac{t}{y_k} \cdot y_k(1-y_k) + \frac{1-t}{1-y_k} \cdot y_k(1-y_k) \tag{3.6}$$

$$= y_k - t \tag{3.7}$$

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

The derivative in Eq. 3.7 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 convolutional and/or recurrent layers can alleviate this problem to some extent: convolutional layers can pass on the error signal as far as their perceptive fields reach; recurrent layers can pass on the error signal until the forget gate is closed (which usually spans the duration of one sound event occurrence). However, if a sound event occurs multiple times far from each other 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:

$$L = \begin{cases} -\log\left[1 - \prod_i (1-y_i)\right], & \text{if } t = 1 \\ -\log \prod_i (1-y_i), & \text{if } t = 0 \end{cases} \tag{3.8}$$

This depends on all the instance-level predictions. Its derivative w.r.t. any z_i can be calculated to be:

$$\frac{\partial L}{\partial z_i} = \begin{cases} -y_i(1-y)/y, & \text{if } t = 1 \\ y_i, & \text{if } t = 0 \end{cases} \tag{3.9}$$

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. In this light, 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.

3.1.3 Experiment 1: Phoneme Recognition on TEDLIUM

This was a proof-of-concept experiment that we did before applying MIL to SED, and it turned out to reveal a lot of relevant properties of the max and noisy-or pooling functions.

Experiment Setup

We conducted a phoneme recognition task on the TEDLIUM corpus. Phonemes can be regarded as substitutes for sound events, although there are two notable differences: phonemes are normally short, and they do not overlap. We did not perform full speech recognition to avoid the extra complexity introduced by the lexicon and the language model.

The TEDLIUM corpus[1] 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 lexicon; we only retained the 39 "real" phonemes and discarded all noise markers such as "breath" and "cough".

The baseline system we compared against was a Theano [92] re-implementation of the example CTC system[2] in the EESEN toolkit [123], which outperformed the original. This system took phoneme sequences as training labels. 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 label.

We adapted the baseline system to build two MIL systems with presence/absence labeling, using the max and noisy-or pooling functions, respectively. The input and hidden layers were identical to the baseline system. The output layer consisted of 39 neurons, without the one neuron dedicated to

[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.

System	Hyperparameters		Phoneme Error Rate			
	Grad. clip	Init. LR	Train	Valid.	Dev.	Test
Baseline (CTC)	10^{-4}	3	4.8	15.4	13.9	14.9
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 3.1: The optimal hyperparameters and phoneme error rates of the various systems on the TEDLIUM corpus.

the blank label. These neurons used the sigmoid activation function. Even though phonemes cannot overlap in time, we did not want to enforce this restriction in the network. The frame-level predictions are then aggregated across time into an utterance-level prediction (a 39-dimensional vector).

The baseline CTC system was trained to minimize the negative log-likelihood averaged over frames. The MIL systems were trained to minimize the binary cross-entropy loss; the loss was averaged over utterances and phonemes for the max pooling system, but averaged over frames and phonemes for the noisy-or pooling system, since the utterance-level probabilities could be decomposed into a product of frame-level probabilities. The optimization algorithm was stochastic gradient descent (SGD) with a Nesterov momentum of 0.9 [96]. Each minibatch contained 20,000 frames; an epoch consisted of about 2,000 minibatches. All the systems were trained for 24 epochs, with the learning rate staying constant in the first 12 epochs, and then halved in each of the next 12 epochs. We found it essential to apply proper gradient clipping and a large initial learning rate, in order to ensure that the network could get through the initial stage of instability safely and progress with large enough steps after that. The gradient clipping limit and initial learning rate of each system are also listed in Table 3.1.

We decoded all the systems using simple *best path decoding*: choosing the most probable label (phoneme or blank) at each frame, collapsing consecutive duplicate labels, and removing blanks. Since the output layer of the two MIL systems did not have a neuron for the blank label, we set the prediction to blank if the probability of the most probable phoneme was smaller than 0.5. We evaluated all the systems using phoneme error rate (PER).

Experiment Results

The phoneme error rates of the three systems on all the partitions of the data are listed in Table 3.1. The evolution of the validation PER of the systems is plotted in Fig. 3.2; the PERs on the development and test sets followed similar trends. The baseline CTC system learnt fast and accurately, reaching a PER of 30% after the first epoch and converging to 15%. The max pooling system reached a PER of 43%. Even though there was a gap between

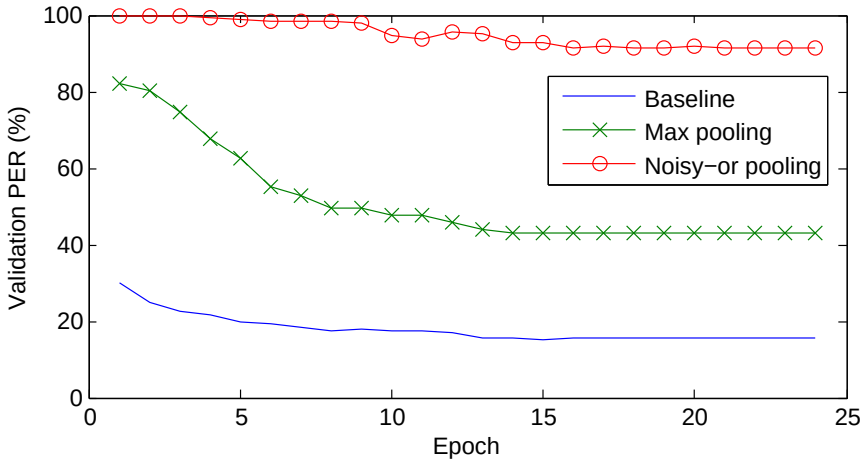


Figure 3.2: Evolution of the validation phone error rate (PER) of the various systems on the TEDLIUM corpus.

System	Decoded Phoneme Sequence	PER
Ground truth	V EH R IY IH K S T R IY M T ER EY N	
Baseline (CTC)	ER IY EH K S T R IY M TH R IY	8/15
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 3.2: The predicted phoneme sequences of the various systems on an example utterance. The ground truth transcription is "very extreme terrain".

the max pooling system and the baseline, the learning can be regarded as successful considering that the max pooling system only saw presence/absence labeling during training. The noisy-or pooling system, however, exhibited a PER above 90% even after 24 epochs, and its predictions were mostly blank.

Table 3.2 shows the predicted phoneme sequences of the various systems on an example utterance in the validation set, and Fig. 3.3 shows the underlying frame-wise predictions. The CTC system produces narrow peaks that align well with the actual position of the phonemes, but the peaks do not indicate the timespan of each phoneme. The max pooling system produces wide peaks that clearly indicate the onset and offset time of each phoneme, which is exactly the desired behavior for localization. The noisy-or pooling system, contrary to our expectations, fails to predict anything meaningful, and only three phonemes receive non-negligible probabilities.

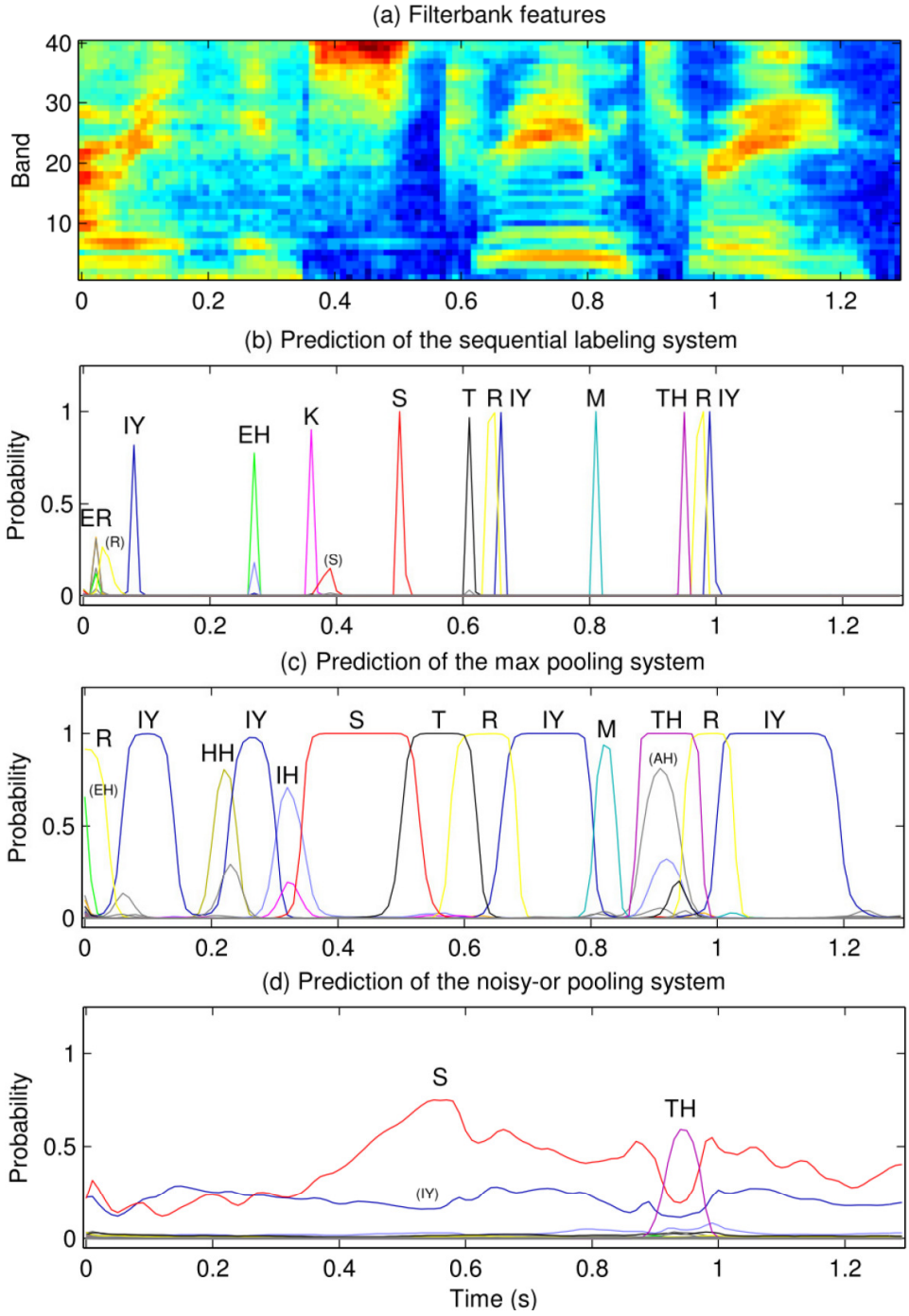


Figure 3.3: The frame-wise predictions of the various systems on an example utterance. The ground truth transcription is "very extreme terrain" (both the baseline and the max pooling systems mis-recognize "terrain" as "three"). Phonemes are differentiated by color. Peaks are annotated with their corresponding phonemes; phonemes in parentheses were not selected in the best path decoding.

Analysis: Why Does Noisy-Or Pooling Fail?

We have observed that the noisy-or pooling system exhibited some unexpected symptoms: it trained slowly, and preferred to predict most phonemes as negative. We suspected that this indicated the system had difficulty in starting to learn, but when we tried initializing the network parameters to the values after one epoch of max pooling training (with a validation PER of 82.1%), we observed that noisy-or training immediately brought the PER back to above 90%, generating almost all blank outputs. Looking at the frame-predictions in Fig. 3.3 (c) and (d), we find that the noisy-or pooling function is inherently too *harsh on false positives* and *lenient on false negatives*.

Noisy-or pooling is harsh on false positives precisely because it erroneously assumes that frames in a recording are independent. Because consecutive frames in a sequence are often correlated, when the system produces a false positive, it normally generates a peak that spans several frames. This peak should be penalized only once, not for every frame it spans. The noisy-or pooling function, however, multiplies the $1 - y_i$ of each frame, which makes the bag-level prediction y extremely close to 1, and results in a large loss $-\log(1 - y)$. For example, in Fig. 3.3 (c), the max pooling system produces a false positive for the phoneme `TH`. The frame-level prediction y_i exceeds 0.999 for 7 frames; the maximum value is $1 - 2 \times 10^{-7}$. With the max pooling function, this incurs a loss of $-\log(2 \times 10^{-7}) \approx 15.5$. If we used noisy-or pooling instead, this false positive would incur a loss of at least $-\log(1 - 0.999) \times 7 \approx 48$. As a result, the noisy-or pooling system only dared to generate a small peak for the phoneme `TH`, as shown in Fig. 3.3 (d). 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.

Noisy-or pooling is lenient on false negatives because the system may believe it has made the correct bag-level prediction for a positive bag, while all the instances-level predictions are negative. This also stems from the multiplication in the noisy-or pooling function. 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 frame-level predictions of the noisy-or pooling system, however, are hardly visible in Fig. 3.3 (d); they actually stay around 0.02 throughout the 130-frame utterance. The noisy-or pooling function then calculates a bag-level prediction of $y \approx 1 - (1 - 0.02)^{130} \approx 0.93$. Recall that the error signal at the i-th frame is $\partial L / \partial z_i = -y_i(1 - y)/y$ (Eq. 3.9). When the bag-level prediction is close to one, the system will no longer make much effort to boost the frame-level predictions. The phoneme `IY` illustrates a more extreme case: its predicted probability fluctuates around 0.2, resulting in a bag-level prediction of $y \approx 1 - (1 - 0.2)^{130} \approx 1 - 2.5 \times 10^{-13}$, The error

signal computed by Eq. 3.9 is virtually zero, causing the network to stop learning. Now we see that, even though noisy-or pooling provides an error signal for all frames, the term $(1-y)/y$ can cause it to vanish. Such "gradient vanishing" would limit the use of the noisy-or pooling function to very small bags (*e.g.* less than 10 instances); unfortunately, phoneme recognition and SED usually feature sequences with hundreds of frames.

3.1.4 Experiment 2: The DCASE 2017 Challenge

Experiment Setup

We also tested out the max and noisy-or pooling functions on Task 4 of the DCASE 2017 challenge[4] [83]. This task consists of two subtasks: Task A is audio tagging, *i.e.* determining which events are present in each recording, but without localizing the events; Task B is sound event detection, *i.e.* Task A plus localization. Both subtasks consider 17 events related to vehicles and warning sounds.

The data consists of a training set (51,172 recordings), a public test set (488 recordings), and a private evaluation set (1,103 recordings). All the recordings come from Google Audio Set [15], and are 10-second excerpts from YouTube videos. The test and evaluation sets are strongly labeled so they can be evaluated for both subtasks, but the training set only comes with presence/absence labeling. Also, the test and evaluation sets have balanced numbers of the events, but the training set is unbalanced. We set aside 1,142 recordings from the training set to make a balanced validation set, and used the remaining 50,030 recordings for training. For this experiment, we did not do anything about the class imbalance in the training data.

We compared max pooling and noisy-or pooling against a simple baseline that does not use MIL. In the baseline system, the instance-level predictions are not pooled into a bag-level prediction; instead, the ground truth label of each bag is assigned to all its instances, and the loss function is computed on an instance-by-instance basis. In plainer words, if a sound event is labeled as present for a recording, it is assumed to be present in every frame of the recording. This scheme of training has been applied in [71] and [124], and named *strong labeling assumption training* (SLAT) in the latter. At first glance this scheme wouldn't seem to work, because it pollutes the labels of all the negative instances in positive bags, and such polluted instances may even outnumber the instances that are truly positive. However, if we think about the discriminative nature of neural networks, we will realize that this is not a problem. A neural network only models the conditional distribution $P(y|\mathbf{x})$, where $\mathbf{x}$ is the input feature and y is the label. Even though SLAT alters the distribution $P(\mathbf{x}|y)$ a lot for the label $y = 1$, it does

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

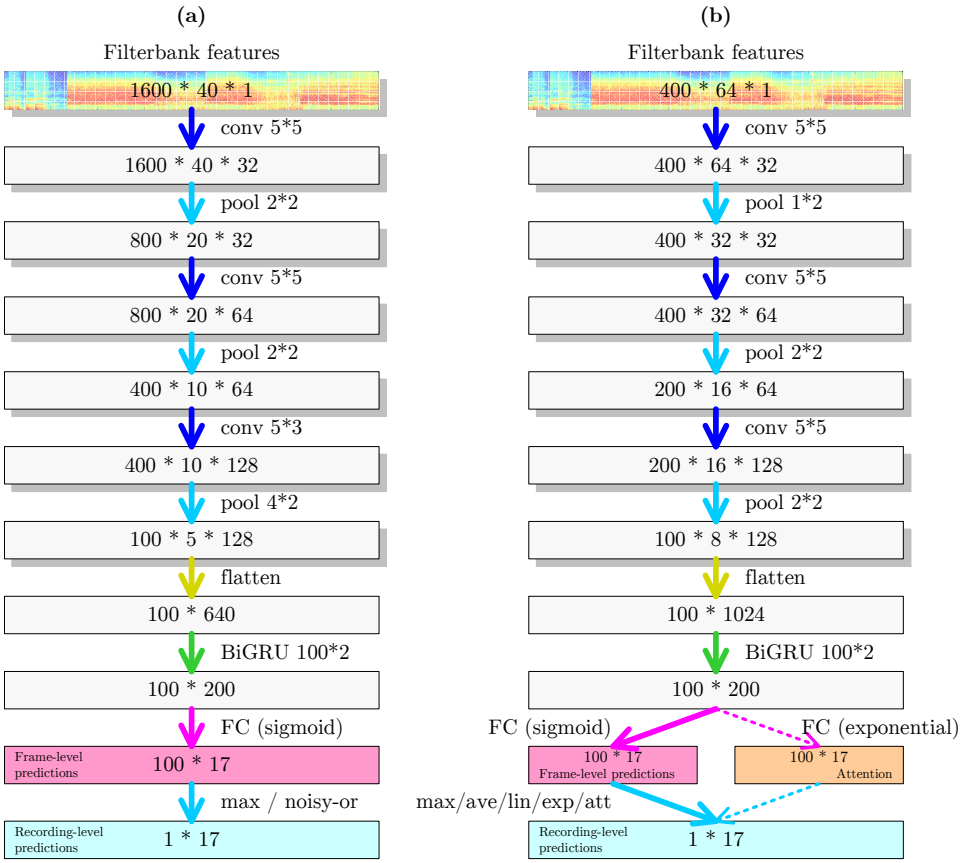


Figure 3.4: The structures of the networks for audio tagging and SED used in Secs. 3.1.4 and 3.2.3. Shadowed boxes stand for 3-D tensors; their sizes are specified in the order of time frames, frequency bins, and feature maps. Plain boxes stand for 2-D tensors; the first dimension is time. All the convolutional layers use the ReLU activation; the numbers after "conv" (*e.g.* 5*5) specify the size of the kernel. All the pooling layers following convolutional layers perform max pooling; the numbers after "pool" (*e.g.* 2*2) specify the stride. "FC" is short for "fully connected". At the output end, the "attention" block is only used with the attention pooling function.

not affect $P(y|\mathbf{x})$ much. The distribution $P(y|\mathbf{x})$ can be regarded as being learned separately for positive instances and negative instances. SLAT does not pollute the labels of the positive instances; it also will not affect the label distribution $P(y|\mathbf{x})$ over the negative instances much if the polluted instances are few compared to unpolluted negative instances. For SED, most sound events are absent in most recordings, so it is hopeful that SLAT can work and serve as a baseline.

We trained a convolutional and recurrent neural network (CRNN) using the Keras toolkit [125] with the Theano [92] backend. The structure of

Figure	3.4 (a)	3.4 (b)	3.10
Used in Sec.	3.1.4	3.2.3	3.3.2
Toolkit	Keras + Theano	PyTorch	PyTorch
Trained on	DCASE 2017	DCASE 2017	Audio Set
# conv. layers	3	3	10
# pool. layers	3	3	5
Output pooling	SLAT/max/noisy-or	max / ave / lin / exp / att	
Batch norm.	No	No	Yes
Dropout prob.	SLAT, noisy-or: 0.0 max: 0.1	0.0	0.0
Optimizer	SGD + Nesterov 0.9	Adam	Adam
Data balancing	No	Yes	Yes
Batch size	100	100	250
Gradient clipping	SLAT: 0.1 max: N/A noisy-or: 10^{-4}	N/A	N/A
Initial LR	SLAT, max: 0.1 noisy-or: 0.3	3×10^{-4}	10^{-3}
Checkpoint	1 epoch (500 batches)	1 epoch (500 batches)	1,000 batches
Validation metric	Val. loss	Val. loss	Val. MAP
LR decay factor	max: 0.8 others: N/A	lin: 0.5 exp: 0.3 others: 0.1	0.8

Table 3.3: Detailed information and hyperparameters of all the networks for audio tagging and SED used in Chapter 3. Batch normalization is applied to the convolutional layers only, before the ReLU activation. Dropout is applied before each convolutional layer, and on both sides of the GRU layer. Batch size is in the number of 10-second recordings. "LR" is short for "learning rate".

the network is shown in Fig. 3.4 (a)[5]. The input is filterbank features extracted with the LibROSA toolkit[6] [126]; it has 40 frequency bins and 1,600 time steps spanning 10 seconds (*i.e.* with a frame rate of 160 Hz). The convolutional and pooling layers reduce the frame rate to 10 Hz, whose

[5]All the CRNNs in this thesis have one and only one GRU layer. We optimized some hyperparameters such as the number of convolutional layers and their sizes (see Table 3.3), but we did not explore alternative structures such as zero or more than one recurrent layer, or replacing the GRU layer with an LSTM layer.

[6]Configuration for filterbank feature extraction: The waveform is downsampled to 16 kHz; frames of 480 samples (30 ms) are taken with a hop of 100 samples (6.25 ms); each frame is Hanning windowed and padded to 2,048 samples before taking the Fourier transform; the filterbank of 40 triangle filters spans a frequency range from 0 Hz to 5.8 kHz. The upper frequency limit of the filterbank was a mistake; it should have been 8 kHz.

output is fed into a bidirectional GRU layer with 100 neurons in each direction. A fully connected layer with 17 neurons and the sigmoid activation function then predicts the probability of each sound event at each frame, which can be used for SED (Task B); these frame-level predictions are also aggregated with either the max or the noisy-or pooling function to produce recording-level predictions for audio tagging (Task A)[7]. When training the baseline SLAT system, the loss function is computed by comparing the frame-level predictions directly to the recording-level labels of the training data; when training the max pooling and noisy-or pooling systems, the loss function is computed by comparing the recording-level predictions to the recording-level labels.

We minimized the cross-entropy averaged over both recordings and events using the SGD algorithm with a Nesterov momentum of 0.9 a batch size of 100 recordings. We tuned the initial learning rate for each network and applied gradient clipping where necessary; see the first column of Table 3.3 for details. For the max pooling network, we applied dropout with a rate of 0.1, and decayed the learning rate with a factor of 0.8 when the lowest validation loss did not see any updates for 3 consecutive epochs, which contributed marginally to the performance.

The frame-level and recording-level probabilities predicted by the network must be thresholded to generate output for evaluation. The optimal thresholds, however, depends on the evaluation metrics. The DCASE 2017 challenge used standard metrics for SED defined in [127]. Task A (audio tagging) was evaluated by the micro-average F_1 on the recording level. Let TP (true positive) be the number of correctly predicted sound events, FN (false negative) be the number of missed sound events, and FP (false positive) be the number of spuriously predicted sound events, accumulated over all recordings and sound event types. The F_1 is defined as the harmonic average of the precision and recall:

$$F_1 = \frac{2}{\left(\frac{TP}{TP+FP}\right)^{-1} + \left(\frac{TP}{TP+FN}\right)^{-1}} = \frac{2TP}{2TP + FP + FN} \tag{3.10}$$

Task B (SED with localization) was evaluated by both the micro-average error rate (ER) and the micro-average F_1 on 1-second segments. The F_1 is defined in the same way as Eq. 3.10, but with TP, FN, and FP counted at the segment level and accumulated over all segments and sound event types. The ER is defined in terms of substitution (S), deletion (D) and insertion (I) errors. For each segment, the number of substitutions is defined as the minimum of false negatives and false positives; any extra false negative counts as a deletion, and any extra false positive counts as an insertion. Let

[7] The baseline SLAT system also uses the max pooling function to make recording-level predictions.

System	Val. Set	Test Set			Evaluation Set		
	Task A	Task A	Task B		Task A	Task B	
	F1	F1	ER	F1	F1	ER	F1
Baseline (SLAT)	53.7	49.2	82.4	41.4			
Max pooling	53.3	50.1	79.7	39.4	50.7	75	46.9
Noisy-or pooling	53.4	49.6	97.9	4.8			

Table 3.4: The performance of the SLAT, max pooling and noisy-or pooling systems on both subtasks of the DCASE 2017 challenge. All numbers are in percentages.

$N = TP + FN$ be the total number of true occurrences of sound events at the segment level, then the ER can be computed by

$$ER = \frac{S + D + I}{N} = \frac{FN + FP - S}{TP + FN} \tag{3.11}$$

We found it critical to tune the threshold for each sound event class individually. We devised an iterative procedure to tune class-specific thresholds to optimize the micro-average F_1 for Task A: first, we tuned the threshold for each sound event class to maximize the F_1 of that class alone; then, we repeatedly picked a random class and re-tuned its threshold to optimize the micro-average F_1, until no improvements could be made. After each epoch of training, we tuned the thresholds on the validation data to optimize the audio tagging F_1; the model with the highest F_1 was picked as the final model. The thresholds obtained on the validation data were applied to the test and evaluation data for both Task A and Task B.

Experiment Results and Analysis

The performance of the three systems are listed in Table 3.4. We got in touch with the organizers of the challenge and had our max pooling system evaluated on the private evaluation set; our system would have ranked 3rd on Task A and 4th on Task B among the nine participants of the challenge[8]. In the text below, however, we will compare systems based on their performance on the public test set. From the table we can see that all the three systems achieved similar performance on Task A. For Task B, the max pooling system performed similarly to the baseline SLAT system, but the noisy-or pooling system failed miserably, just like it did for the phoneme recognition task.

One may doubt the value of the max pooling system if it only performs on par with the simple SLAT system. However, we would like to argue that the success of the SLAT system depends on the limited extent of label

[8]The higher ranking teams used combinations of either models or features, while we trained a single system using a single type of feature.

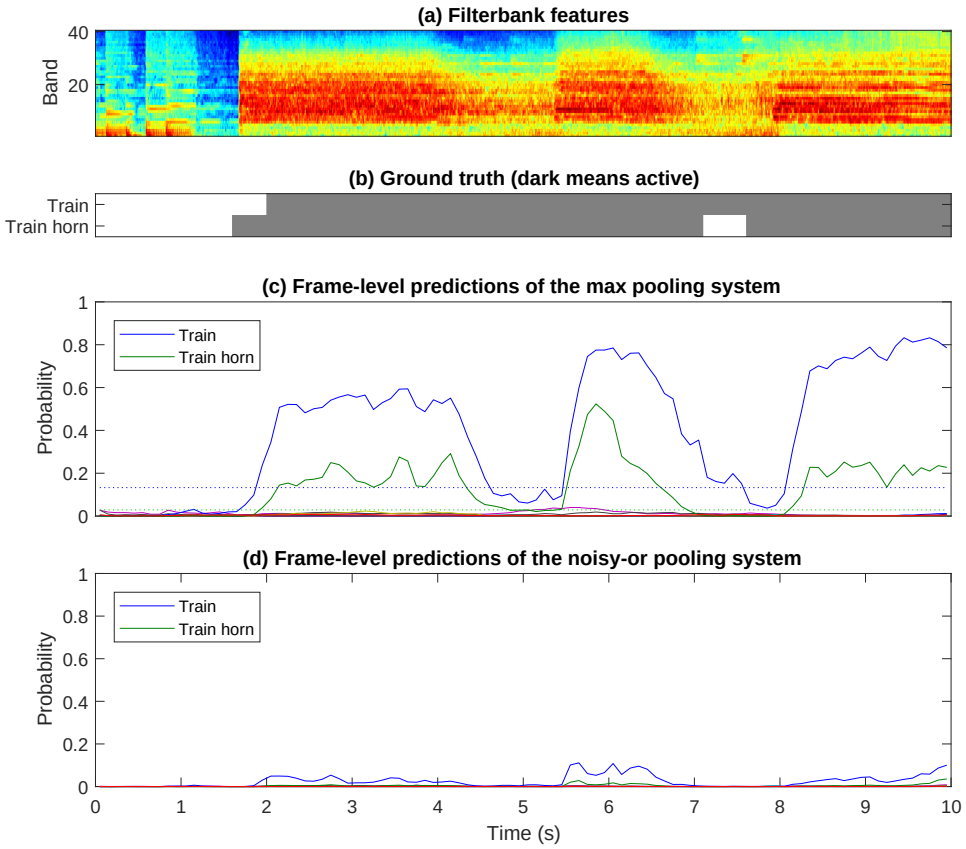


Figure 3.5: The frame-level predictions of the max and noisy-or pooling systems on the example test recording "`10i60V1RZkQ`". In (c), the dotted lines indicate the thresholds for the two event classes. Best viewed in color.

pollution. Because the training set of the DCASE 2017 data is only weakly labeled, we could not measure how many of the labels are polluted in this set. Nevertheless, we were able to measure it on the strongly labeled test set. It turned out that, averaged over the 17 sound event types, 5.0% of the instances were positive and 95.0% were negative; among the negative instances, only 2.3% had polluted labels. The small amount of polluted data was the reason for SLAT's success. In [124] and [128], it has also been reported that SLAT does not perform as well as MIL. Therefore, rather than resting satisfied with the simple SLAT method, we would like to explore more principled MIL approaches that do not depend on the distribution of the labels.

The noisy-or system failed for Task B for the same reasons why it failed for the phoneme recognition task: the noisy-or pooling function is harsh on false positives and lenient on false negatives, resulting in infinitesimal frame-level predictions. Fig. 3.5 illustrates the frame-level predictions of the

True positive (TP)	1,311	(30.1%)
False negative (FN)	3,049	(78.2%)
False positive (FP)	976	(22.4%)
Precision		57.3%
Recall		30.1%
F_1		**39.4%**
Substitution (S)	548	(12.6%)
Deletion (D)	2,501	(57.4%)
Insertion (I)	428	(9.8%)
Error rate (ER)		**79.7%**

Table 3.5: Breakdown of the errors (in number of 1-second segments) made by the max pooling system on Task B of the DCASE 2017 challenge.

max pooling and the noisy-or pooling systems on an example test recording. The recording contains the sound of a train sounding its horn intermittently. Fig. 3.5 (c) indicates that the max pooling system is able to locate the intervals during which the horn is sounding and produce reasonable frame-level probabilities; it is even able to detect the pause of the horn between 4 and 5 seconds which is not labeled in the ground truth. Fig. 3.5 (d), however, indicates that the noisy-or pooling system suffers from the same problem observed in the phoneme recognition experiment: its frame-level predictions are too small. Even though these small frame-level predictions can make up reasonable recording-level probabilities through the multiplication in the noisy-or pooling function (Eq. 3.3), they are not intuitive as frame-level probabilities for SED. Because we applied the same thresholds to both subtasks, most sound events were missed for Task B. This is why we saw an error rate close to 100% and an F_1 close to zero.

From both the phoneme recognition experiment and the DCASE 2017 challenge, we can draw the conclusion that noisy-or pooling is not suitable for localization. There are two reasons behind this: (1) the noisy-or pooling function makes the false assumption that the frames in a recording are independent; (2) noisy-or pooling does not produce intuitively compatible predictions on the recording level and the frame/segment level. On the other hand, max pooling performs reasonably on both levels. Because max pooling is more principled than the simple SLAT method, we will use max pooling as the baseline for the study in the next section.

3.2 The Average, Softmax and Attention Pooling Functions

3.2.1 Motivation

The experiments in the previous section show that the max pooling function provides a reasonable solution to SED. But if we break down the errors made by the max pooling system on Task B (see Table 3.5), we see that the system produces a lot more false negatives than false positives, resulting in a low recall. We conjecture this is due to the inefficient gradient flow of the max pooling function: because only the maximum instance receives an error signal, if a sound event occurs in more than one region of a recording, it is possible that only one of the occurrences can be detected.

Recent literature has seen the use of the average pooling function, two types of softmax pooling functions and an attention-based pooling function, which claim to alleviate this problem. The attention-based pooling function is receiving especially much attention from researchers. All these four pooling functions compute the bag-level prediction y as a weighted average of the instance-level predictions y_i:

$$y = \frac{\sum_i y_i w_i}{\sum_i w_i} \tag{3.12}$$

Actually, max pooling can also be regarded as a weighted average, where the maximum instance gets a weight $w_i = 1$, and all other instances get a weight $w_i = 0$. The average, softmax and attention pooling functions assign weights in ways that the non-maximum instances receive some error signal, too, so the gradient flow can be made easier.

The average pooling function [48] assigns an equal weight to each instance. The bag-level prediction y is calculated by

$$y = \frac{1}{n} \sum_i y_i \tag{3.13}$$

where n is the number of instances in the bag.

The first softmax pooling function [50], which we call *linear softmax*, assigns to each instance a weight that is proportional to the instance-level prediction, y_i. The bag-level prediction y is calculated by

$$y = \frac{\sum_i y_i^2}{\sum_i y_i} \tag{3.14}$$

The second softmax pooling function [51], which we call *exponential softmax*, assigns a weight of $\exp(y_i)$ to an instance whose instance-level prediction is y_i. The bag-level prediction y is calculated by

$$y = \frac{\sum_i y_i \exp(y_i)}{\sum_i \exp(y_i)} \tag{3.15}$$

The attention pooling function [53, 54] employs a separate output layer in the neural network to learn the weights w_i used for the weighted average. In other words, separate parts of the network are used to learn the *probabilities* of the sound events and the *confidences* about these probabilities. The bag-level prediction y is then calculated by Eq. 3.12.

All the new pooling functions introduced above assign a positive weight to instances that do not reach the maximum during training, so that if a sound event occurs multiple times in a recording, the instance-level probability y_i at all the occurrences have a chance to be boosted. While this facilitates the gradient flow, it makes the pooling functions deviate from the standard multiple instance (SMI) assumption. The two softmax pooling functions are relatively more loyal to the SMI assumption, because the weight w_i is monotonic in the instance-level prediction y_i; the linear softmax pooling function is even more so because w_i approaches 0 when y_i approaches 0. The average and attention pooling functions, on the other hand, appear to defy the SMI assumption; the attention pooling function may learn large w_i's where the y_i's are small. This behavior, however, may be justified by the motivation of decoupling the probability and the confidence: the attention (*i.e.* weight) should be placed on instances where the system has a high confidence about its judgment, regardless of whether this judgment says a sound event is likely to occur there or not.

3.2.2 The Gradient Flow

Now we would like to compute the error signal that the output layer passes down to lower layers.

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 y is aggregated from the instance-level predictions $y_1, \ldots, y_n \in [0, 1]$ using the average (Eq. 3.13), linear softmax (Eq. 3.14), exponential softmax (Eq. 3.15), or attention (Eq. 3.12) pooling function; in the case of the attention, y also depends on the learned weights $w_1, \ldots, w_n$. The instance-level predictions must lie within $[0, 1]$, so they are produced with sigmoid units, *i.e.* $y_i = \mathrm{sigm}(z_i)$. The weights only need to be non-negative; in our experiments we generate them with exponential units, *i.e.* $w_i = \exp(v_i)$.

To compute the error signal, we need to find the derivative of the loss function w.r.t. all the z_i's (and v_i's in the case of attention). The loss function is

$$L = -t \log y - (1 - t) \log(1 - y) \tag{3.16}$$

Its derivative w.r.t. the z_i's and v_i's can be decomposed as:

$$\frac{\partial L}{\partial z_i} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial y_i} \frac{\mathrm{d} y_i}{\mathrm{d} z_i} \tag{3.17}$$

$$\frac{\partial L}{\partial v_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_i}\frac{\mathrm{d}w_i}{\mathrm{d}v_i} \tag{3.18}$$

Some terms do not depend on the concrete expression of the pooling function, which we compute here:

$$\frac{\partial L}{\partial y} = -\frac{t}{y} + \frac{1-t}{1-y} \tag{3.19}$$

$$\frac{\mathrm{d}y_i}{\mathrm{d}z_i} = y_i(1-y_i) \tag{3.20}$$

$$\frac{\mathrm{d}w_i}{\mathrm{d}v_i} = w_i \tag{3.21}$$

We notice that $\partial L/\partial y$ is negative when $t = 1$ and positive when $t = 0$, and that both $\mathrm{d}y_i/\mathrm{d}z_i$ and $\mathrm{d}w_i/\mathrm{d}v_i$ are always positive. The different behavior of the pooling functions will depend mainly on the middle terms $\partial y/\partial y_i$ and $\partial y/\partial w_i$.

With the average pooling function (Eq. 3.13), we have

$$\frac{\partial y}{\partial y_i} = \frac{1}{n} \tag{3.22}$$

This term is positive and constant for all instances. The consequence is that all frames get boosted when the bag label is positive (t = 1), and all frames get suppressed when the bag label is negative (t = 0). While it is generally a good thing to bring y_i closer to t, the gradient exhibits no selectivity across the frames, so the average pooling function can be expected to have poor localization performance.

With the linear softmax function (Eq. 3.14), we have

$$\frac{\partial y}{\partial y_i} = \frac{2y_i - y}{\sum_j y_j} \tag{3.23}$$

This term is positive iff $y_i > y/2$. Considering the terms $\partial L/\partial y$ and $\mathrm{d}y_i/\mathrm{d}z_i$ as well, we know that when $t = 1$, $\partial L/\partial z_i$ is negative when $y_i > y/2$ and positive when $y_i < y/2$. This indicates an interesting behavior: when the ground truth bag label is positive, gradient descent will boost the z_i's (and the y_i's) of the frames where the frame-level prediction y_i is sufficiently large, but suppress the frames where y_i is small. This can lead to clean and well-localized predictions of sound events. When the ground truth bag label is negative ($t = 0$), we see the opposite behavior: frames with large y_i's are suppressed, and frames with small y_i's are boosted. However, because the threshold between "large" and "small" is $y/2$, in the long run all the y_i's will be brought down to zero. This analysis shows that the linear softmax function has desirable gradient flow properties.

With the exponential softmax function (Eq. 3.15), we have

$$\frac{\partial y}{\partial y_i} = (1 - y + y_i) \cdot \frac{\exp(y_i)}{\sum_j \exp(y_j)} \tag{3.24}$$

Unlike the linear softmax function, this partial derivative is always positive. As a result, all the frames get boosted when the bag label is positive ($t = 1$), and all the frames get suppressed when the bag label is negative ($t = 0$). It is not directly clear whether this will lead to poor localization performance like the case of average pooling, because to some extent Eq. 3.24 still emphasizes frames with larger probabilities.

With the attention pooling function (Eq. 3.12), we have

$$\frac{\partial y}{\partial y_i} = \frac{w_i}{\sum_j w_j} \tag{3.25}$$

$$\frac{\partial y}{\partial w_i} = \frac{y_i - y}{\sum_j w_j} \tag{3.26}$$

The first partial derivative, $\partial y / \partial y_i$, is always positive. As a result, the attention pooling function will also boost all frames when $t = 1$ and suppress all frames when $t = 0$. The strength of the boosting or suppression is higher where the learned weight is higher, which is desirable. The second partial derivative, $\partial y / \partial w_i$, is positive iff $y_i > y$. When the bag label is positive ($t = 1$), this will boost the attention of the frames where $y_i > y$ and suppress the attention elsewhere, causing attention to concentrate on the frames with large predictions. This is also reasonable behavior. But when the bag label is negative ($t = 0$), the attention will concentrate on the frames with small predictions. This agrees with the motivation of decoupling probability and confidence, but it can cause incompatible bag-level and instance-level predictions. Consider a case when the bag label is negative, but the instance-level predictions vary between positive and negative. The attention will focus on the negative instances and make a correct bag-level prediction, but then the instances that are less attended to will end up as false positives on the instance level.

3.2.3 Experiment: The DCASE 2017 Challenge

Experiment Setup

We compared the average pooling function, the two softmax pooling functions and the attention pooling function on the DCASE 2017 challenge, using the max pooling function as a baseline. We trained a convolutional and recurrent neural network (CRNN) similar to the one used in Sec. 3.1.4, whose structure is shown in Fig. 3.4 (b). To speed up the training, we implemented the network using the PyTorch toolkit [95] instead of Keras + Theano.

The new network used a different size for the input filterbank features[9] (400×64 instead of 1600×40); the sizes of the convolutional and pooling layers were adjusted accordingly. At the output end, a fully connected layer with the sigmoid activation function produces frame-level predictions, which are then aggregated across time into recording-level predictions using any of the five pooling functions (max, average, linear softmax, exponential softmax, attention). If the attention pooling function is used, a separate fully connected layer is used to generate the attention weights, which are consumed by the pooling function. The authors of [53] used a softmax activation function for the second fully connected layer, in which the softmax acts across sound event types. We found the normalization across sound event types unnecessary, and used the exponential activation function instead. Because the attention pooling function is a weighted average across time, this is equivalent to using a softwax acting across time.

When training the network, we balanced the amount of training data of different sound event classes. This is done by creating a data generator for each sound event class that cycles through all the training recordings labeled with this class, and then sampling uniformly from all the data generators to form the training batches. This data balancing eliminates the concept of an "epoch", but we still called it an "epoch" when the training had gone through as many recordings as the training set contained. Note that in such an epoch, the data of the rare sound event classes may have been seen multiple times, while some of the data of the frequent sound event classes may not have been seen at all. The data balancing was found to accelerate the convergence, even though it did not significantly affect the evaluation metrics.

We trained the network using the Adam optimizer [100] with an initial learning rate of 3×10^{-4}. The batch size was 100 recordings, so an epoch consisted of 500 batches. We reduced the learning rate if the validation loss did not reduce for 3 consecutive epochs; the reduction factor was tuned for each pooling function (see the middle column of Table 3.3 for details). We did not apply gradient clipping or dropout. All models converged by Epoch 25, and we evaluated the models at this epoch.

A word must be said about how the predictions on 1-second segments were made. If we predicted a segment as positive as long as one frame-level prediction within the segment exceeded the threshold, we found we would end up with an overwhelming number of false positives (yielding error rates near 200%) when using the average, softmax or attention pooling functions. Instead, we decided to treat each segment as a bag containing 10 frames,

[9] Configuration for filterbank feature extraction: The waveform is downsampled to 16 kHz; frames of 1,024 samples (64 ms) are taken with a hop of 400 samples (25 ms); each frame is Hanning windowed and padded to 4,096 samples before taking the Fourier transform; the filterbank of 64 triangle filters spans a frequency range from 0 Hz to 8 kHz. This is also the configuration used in all subsequent experiments.

	Max Pool.	Ave. Pool.	Lin. Soft.	Exp. Soft.	Attention
Task A					
TP	284	297	317	298	301
FN	322	309	289	308	305
FP	364	285	359	324	317
Precision	43.8	51.0	46.9	47.9	48.7
Recall	46.9	49.0	52.3	49.2	49.7
F_1	**45.3**	**50.0**	**49.5**	**48.5**	**49.2**
Task B					
TP	1,206	2,114	1,832	2,121	1,926
FN	3,154	2,246	2,528	2,239	2,434
FP	1,253	3,758	2,187	3,437	3,309
Precision	49.0	36.0	45.6	38.2	36.8
Recall	27.7	48.5	42.0	48.6	44.2
F_1	**35.4**	**41.3**	**43.7**	**42.8**	**40.1**
Task B					
Sub.	712	1,385	1,040	1,292	1,275
Del.	2,442	861	1,488	947	1,159
Ins.	541	2,373	1,147	2,145	2,034
Error Rate	**84.7**	**105.9**	**84.3**	**100.6**	**102.5**

Table 3.6: Detailed performance of the max pooling, average pooling, linear softmax, exponential softmax, and attention systems trained and evaluated on the DCASE 2017 challenge. Errors of Task A are counted by the number of recordings; errors on Task B are counted by the number of 1-second segments.

and calculated the segment-level predictions by aggregating the frame-level predictions using whichever pooling function was used in training. This constrained the number of false positives within a reasonable range.

Experiment Results

Table 3.6 lists the performance of the max pooling, average pooling, linear softmax, exponential softmax, and attention systems on both subtasks of the DCASE 2017 challenge[10]. All the four new pooling functions significantly[11] outperform the max pooling function in terms of F_1 for both subtasks. In terms of error rate for Task B, however, only the linear softmax system slightly outperforms the max pooling system; the average pooling,

[10]The max pooling system here does not perform as well as the one in Sec. 3.1.4. However, we found the latter hard to reproduce even with the original network structure.

[11]We repeated all the experiments five times with different random seeds. For each evaluation metric, we tested every pair of systems to see if one system had a significantly lower mean than the other system. The test used was the one-tailed Welch's t-test [129], which assumes that the samples obey the normal distribution but do not necessarily have an equal variance. This test is available in Matlab as the `ttest2` function. Both claims of significance passed the test at a significance level of 0.05.

exponential softmax and attention systems yield error rates higher than 100% and are significantly[11] worse than the max pooling and linear softmax systems.

Table 3.6 also includes a breakdown of the error types. All the five systems achieve a reasonable balance between false negatives and false positives for Task A. For Task B, however, only the linear softmax system achieves a good balance. The max pooling system makes too many false negatives. The reason has been analyzed in Sec. 3.1.2: when a sound event occurs multiple times in a recording, the gradient flow of the max pooling function only boosts the prediction at one occurrence. The average pooling, exponential softmax and attention systems make too many false positives. The reason has been analyzed in Sec. 3.2.2: these pooling functions put too much weight on the frames with small-valued predictions, making it necessary to have many large-valued frame-level predictions in order to make a positive recording-level prediction. It turns out that at the operation points of these systems, the error rate is very sensitive to false positives. This is why the average pooling, exponential softmax and attention systems perform badly on Task B.

Fig. 3.6 illustrates the false positives made by the average pooling, exponential softmax and attention systems. The 10-second recording contains the conversation between a man and a child, with the sound of a bicycle chain turning from 1.5 s to 3.7 s. The DCASE challenge is only interested in the bicycle sound. On this recording, all the five systems correctly detect the bicycle sound and deny the existence of bus sounds on the recording level. However, the average pooling, exponential softmax and attention systems make false positives for the `bus` event on the frame or segment levels. In Fig. 3.6 (c) $\sim$ (g), the solid line shows the frame-level predictions for the `bus` event, the dashed line shows the recording-level prediction for the `bus` event (which is calculated from the frame-level predictions using the pooling function), and the dotted line shows the class-specific threshold for the `bus` event. In the max pooling and linear softmax systems, both the frame-level predictions and recording-level prediction stay safely below the threshold. In the exponential softmax system, the frame-level predictions exhibit undesired high values toward the end of the recording. Even though these high values do not affect the recording-level prediction much because other frames also have significant weight in the pooling function, a frame-level false positive is made at 8.7 s. Similarly, the average pooling system makes false positives around 2.6 s, 4.0 s and 9.0 s. In the attention system, we see what we have anticipated in Sec. 3.2.2: the attention (light blue line) mostly focuses on regions where the frame-level predictions are low. This correctly produces a negative recording-level prediction, but lets many frame-level false positives get away unconstrained. The frame-level false positives also produce segment-level false positives on the segments 4$\sim$5 s and 5$\sim$6 s, contradicting the recording-level prediction.

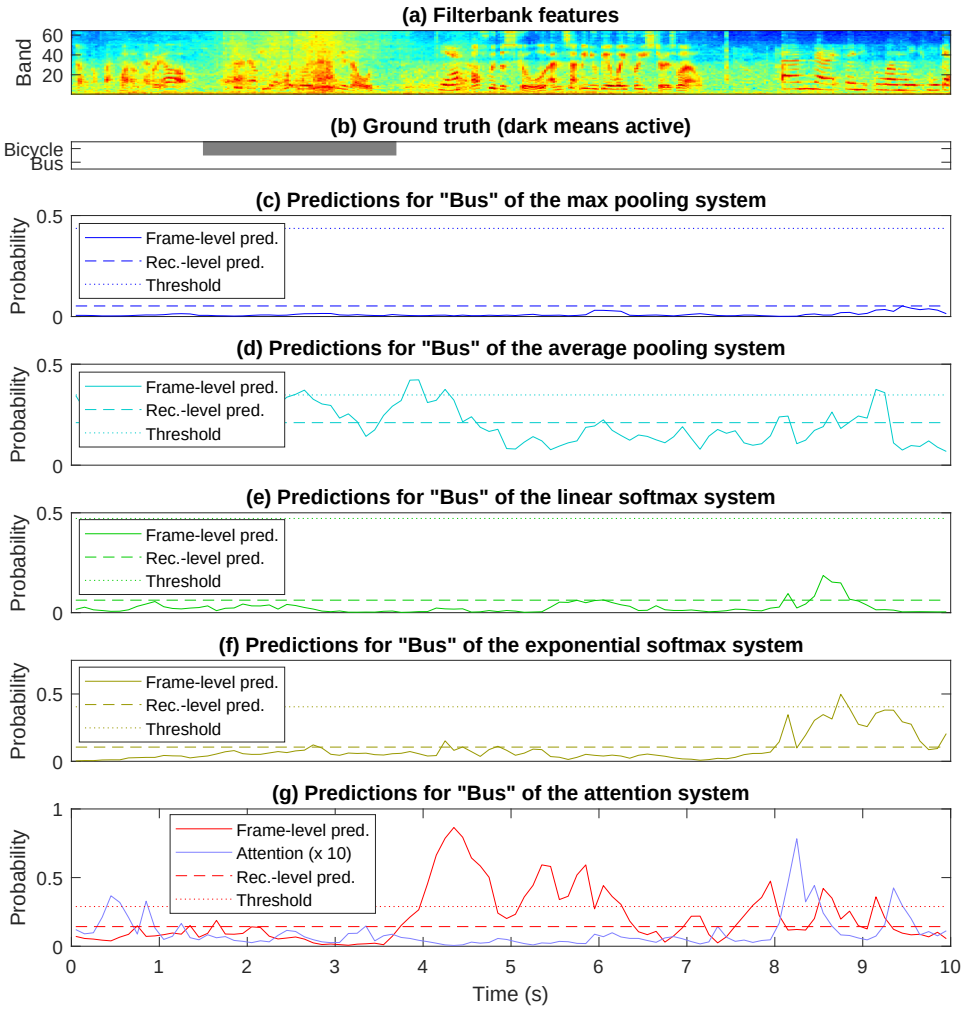


Figure 3.6: The frame-level predictions of the max pooling, average pooling, linear softmax, exponential softmax, and attention systems for the `bus` event on the example test recording "`-nqm_RJ2xj8`" (unfortunately, this recording is no longer available on YouTube). Best viewed in color.

The false positives illustrated in Fig. 3.6 are common throughout the data.

Considering both the error rate and the F_1 metric, as well as the agreement between recording-level and frame-level predictions, we recommend the linear softmax pooling function among all the pooling functions we have studied.

Class-Wise Analysis

We visualized the confusion between the 17 sound event types via confusion matrices, both on the recording level and the 1-second segment level. Due to

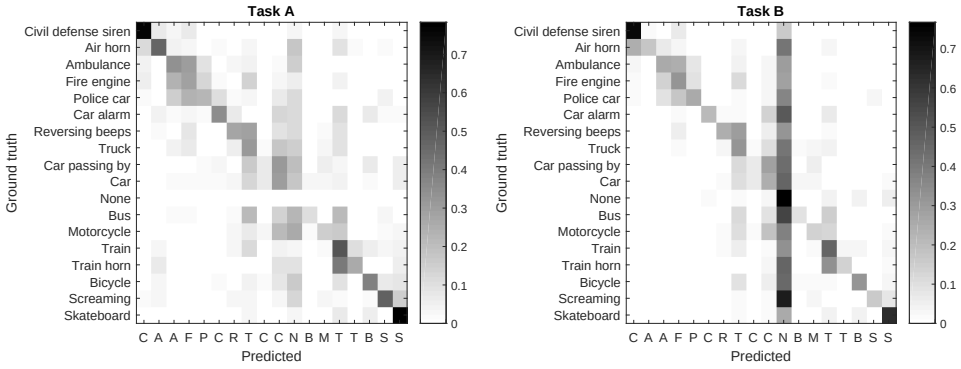


Figure 3.7: The confusion matrices of the linear softmax system on the test data of the DCASE 2017 challenge, for both Task A and Task B.

polyphony, a recording or a segment may have multiple ground truth labels, as well as multiple predicted labels. In such cases, we divide the recording or segment into fractional recordings or segments with equal weights. For example, if a recordings has the ground truth label `car` but is predicted to contain both `bus` and `train`, then we break it down into two recordings both having a weight of 1/2 and the ground truth label `car`, but one having the prediction `bus` and the other having the prediction `train`.

The confusion matrices of the linear softmax system on the test data for both Task A and Task B are shown in Fig. 3.7. For better visualization, we arranged the rows and columns of the matrices so that large entries fall near the diagonal using the confusion matrix ordering (CMO) algorithm[12] introduced in [130]. In the confusion matrix for Task A, the row for the event `None` is empty, because all test recordings contain at least one event type.

The following results can be read from the confusion matrices:

- `Civil defense siren` and `skateboard`, arranged in the corners of the confusion matrices, are the two best-learned sound event types. They are seldom recognized as other events, nor are other events often recognized as them.

- There is a lot of confusion between the sirens of the three types of emergency vehicles: `ambulance`, `fire engine`, and `police car`. This agrees with intuition.

- The event types `car`, `truck` and `train` receive a lot of false positives. This is understandable because the first two of the three event types have remarkably more training recordings than other events, which may cause the system to bias toward them even though data balancing is applied.

[12] Available in the `clana` toolkit: https://github.com/MartinThoma/clana.

- Events are missed more often on the segment level than on the recording level.

The confusion matrices of systems using other pooling functions are more or less similar, except that the max pooling system has a lot more misses on the segment level, and that the attention system has slightly more false positives for the events `bicycle` and `bus` for Task B.

We also tried to find out what properties of individual sound event types affect how well they are learned. Because the error rate metric inherently involves multiple sound event types, we studied the F_1 metric for both Task A and Task B. The properties we considered include the *frequency* and *coverage* of each sound event type. The frequency is measured by how many recordings in the training data contain a given sound event type. The coverage reflects whether a given sound event type tends to cover the whole recording or last for only a short time. It is measured by the percentage of duration covered by a given sound event type in all recordings in which the sound event type is labeled as present by the ground truth.

Calculating the coverage requires strong labeling. For the DCASE 2017 data, we could calculate the coverage of each event type on the strongly labeled test data (488 recordings). We could also estimate the coverage from the larger training data (50,030 recordings). To do so, we predicted frame-level probabilities of the events using the linear softmax system (which was found to produce balanced numbers of false negatives and false positives), and thresholded these probabilities using class-specific thresholds tuned to optimize the micro-average Task A F_1 on the validation data. We then calculated the coverage of each event type on the training recordings whose ground truth was positive for this event type.

To understand the interplay between the evaluation metrics and the properties of the sound event types, we calculated the Pearson's correlation coefficient between each pair of them. We observed the same trends for all the systems using different pooling functions (see Fig. 3.8). First, we see that the F_1's of both Task A and Task B are negatively correlated with the frequency (excluding the two extremely frequent events: `car` and `truck`). This means less common events are learned better; a partial explanation may be that frequent events tend to receive more false positives. Second, the F_1's of both Task A and Task B are strongly positively correlated with the coverage estimated from the training data. The two best-learned event types, `civil defense siren` and `skateboard`, happen to have the highest coverage ($> 70\%$). This means events that tend to last longer are learned better. There are some caveats when interpreting this result, though: first, we found that the coverage estimated from the training data did not correlate well with the coverage directly calculated from the test data; second, it is not clear whether higher coverage results in better learning, or better learning results in a higher estimated value of coverage.

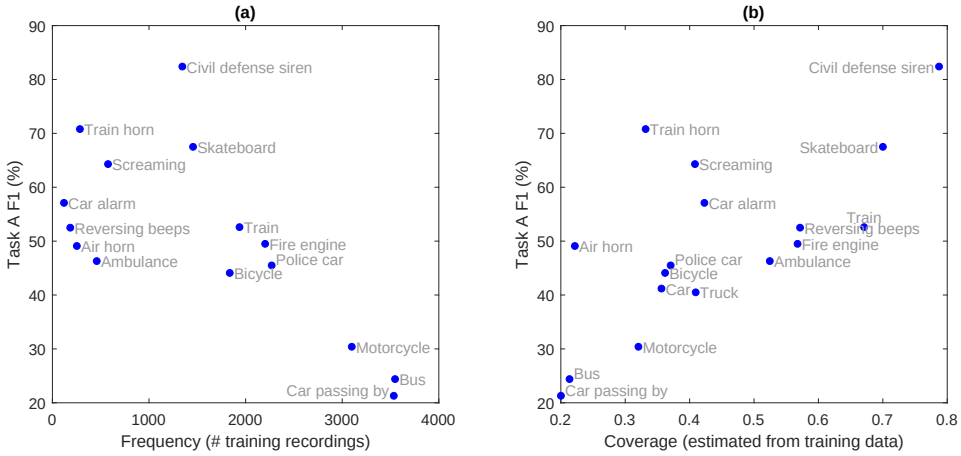


Figure 3.8: The correlation between the linear softmax system's Task A F_1 and the frequency and coverage of the event types. Each dot stands for an event type; the dots for `car` and `truck` are omitted from the subplot (a) because their frequency values are too large. The Task B F_1 is not plotted because it is highly correlated with the Task A F_1. Systems with other pooling functions exhibit similar patterns.

Discussion: Alternative Thresholding for Task B

As we tuned the hyperparameters for the experiment in this section, we found it hard to get the error rate of Task B below 80%. The authors of [53], however, achieved an error rate of 72% using the attention pooling function, ranking second among the seven participants of the challenge who submitted results for Task B. In our private correspondence with the authors, we learned that their low error rate was a result of the following techniques:

- Using a deeper network (nine convolutional layers instead of three);
- Fusing the predictions of the models at different epochs;
- Using different thresholds for Task B than Task A;
- Smoothing the detected sound events.

All of these may be regarded as engineering efforts except the third point. Recall that we tuned class-specific thresholds to optimize the Task A F_1 on the validation data, and then applied these thresholds to both subtasks on the test data. While this can be expected to yield a good performance for Task A, there is no particular reason why these thresholds should work well for Task B. In the case of the max pooling function, using the same thresholds for both subtasks guarantees compatible recording-level and frame-level predictions, but this no longer holds for other types of pooling functions. Therefore, we experimented with different thresholds for Task B.

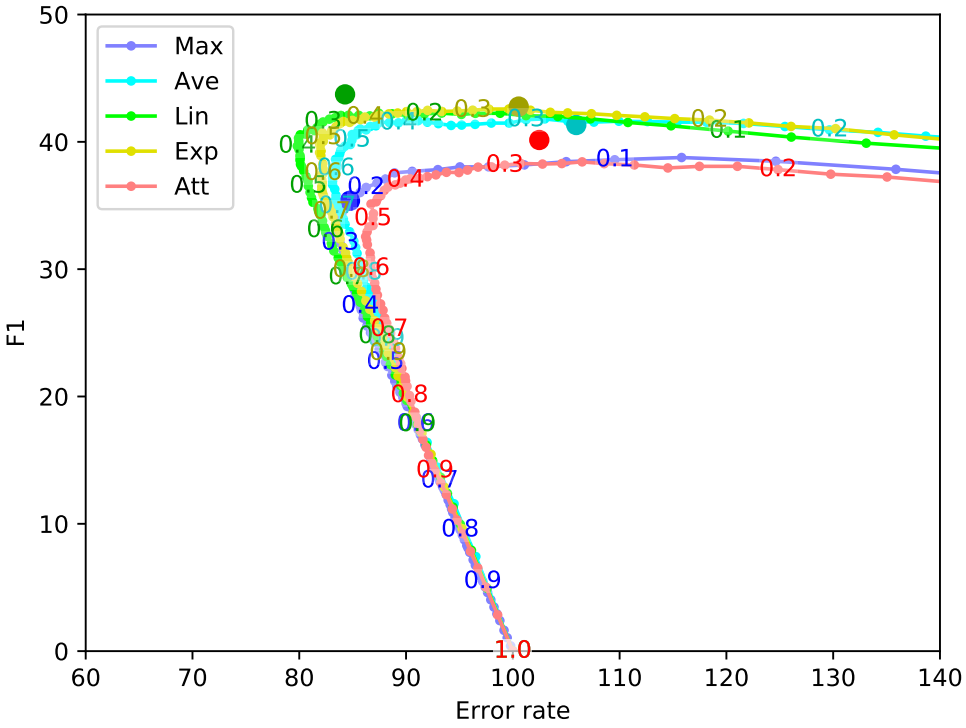


Figure 3.9: The change of the Task B error rate and F_1 metric of the various systems as the threshold varies from 0 to 1, measured on the test data of the DCASE 2017 challenge. The numbers on the curves indicate the thresholds at those points; the big dots indicate the error rate and F_1 obtained with class-specific thresholds tuned to optimize the Task A F_1 on the validation data. Best viewed in color.

A problem we ran into was the lack of validation data. The validation data of the DCASE 2017 challenge was not strongly labeled, so we could not tune the thresholds for Task B on the validation data. The authors of [53] tuned the thresholds on the public test data, and applied them to the private evaluation data. Since we did not have access to the ground truth of the evaluation data, all we could do was to tune the thresholds on the test data, and then measure the evaluation metrics on the test data as well. This inevitably introduced overfitting. To mitigate this overfitting, we no longer tuned class-specific thresholds, but tuned a global threshold for all the sound event types instead.

Fig. 3.9 shows the change of the Task B error rate and F_1 as the global threshold varies from 0 to 1. A good threshold should yield a low error rate and a high F_1, and this is represented by points lying in the upper left corner of the curves. If the threshold is lowered, false positives will increase, which quickly increases the error rate. The F_1, however, is less sensitive to false positives, because the recall tends to 100% and the precision

has a lower bound. If the threshold is raised, false negatives will increase, which adversely affects both the error rate and the F_1 metric. The optimal threshold appears to be near 0.25, 0.5, 0.35, 0.5, and 0.45 for the max pooling, average pooling, linear softmax, exponential softmax and attention systems, respectively. The authors of [53] used a threshold of 0.5 for their attention system, which is close to the optimal value.

However, if we compare all the five curves in Fig. 3.9, we see that even with a threshold tuned separately for Task B, it is still the linear softmax system that achieves the best error rate and F_1, because its curve extends more than the others to the left and above. In addition, we can look at the big dots in the figure, which indicate the actual performance when the class-specific thresholds tuned to optimize the F_1 of Task A on the validation data are directly applied to Task B (they do not have to fall on the curves). The linear softmax system has a big dot that lies close to the optimal operating point. This means the linear softmax system, unlike the average pooling, exponential softmax and attention systems, does not require tuning the thresholds separately for Task A and Task B.

The discussion in this section still points to the superiority of the linear softmax pooling function. We conjecture that the performance of the attention system in [53] can be further improved if the attention pooling function is replaced by the simpler linear softmax pooling function.

3.2.4 Additional Remarks

So far we have shown linear softmax to be the best pooling function among all the pooling functions we have studied. It has the following advantages:

- It facilitates the gradient flow;
- It achieves a good balance between false negatives and false positives;
- It makes consistent recording-level and frame-level predictions.

As a result, the linear softmax system achieves a low error rate on the segment level, as well as a high F_1 on both the recording level and the segment level.

However, we do not believe that linear softmax is the ultimate optimal pooling function for sound event detection. We notice that both softmax pooling functions are actually special cases of weighted averaging with the following weighting function:

$$w_i = y_i^{\beta} \exp(\alpha y_i) \tag{3.27}$$

The linear softmax function is obtained when $\alpha = 0, \beta = 1$, while the exponential softmax function arises when $\alpha = 1, \beta = 0$. As long as $\alpha, \beta \geq 0$, the weight w_i will be monotonically increasing in the prediction y_i. It is

unlikely that $\alpha = 0, \beta = 1$ are the optimal hyperparameters in this vast hyperparameter space. Indeed, a study concurrent with ours [52] proposed an adaptive pooling function with a weighting scheme of $w_i = \exp(\alpha y_i)$, where the coefficient α can take a different value for each sound event type, and is trained jointly with all the network parameters.

And there is no reason why we must restrict ourselves to weighting functions in the form of Eq. 3.27. The freedom to learn the weights, as provided by the attention pooling function, is still attractive. However, we have seen that if we do not impose any constraints, the attention can tend to fall on frames with low predicted probabilities, which can generate incompatible recording-level and frame-level predictions. To avoid this, we may consider learning a weighting function $w_i = w(y_i)$ that must be monotonically increasing. Such a weighting function can enjoy as much freedom as possible without suffering from too many frame-level false positives, and is a promising direction for further study.

3.3 TALNet: A Network for Large-Scale Joint Audio Tagging and Localization

In the previous two sections, we have compared six types of pooling functions on the DCASE 2017 challenge, and established the linear softmax pooling function as the optimal among the six. However, the corpus of the DCASE challenge is a small dataset after all: it contains only about 140 hours of audio for training, and involves only 17 types of vehicle and warning sounds. In this section, we would like to confirm our findings on the large-scale Google Audio Set, which contains more than 2 million 10-second audio excerpts with a total duration of 8 months, and is concerned with 527 diverse types of sound events.

Because Audio Set only comes with presence/absence labeling, the default metrics only evaluate the performance of audio tagging. Nevertheless, our goal has always been to perform audio tagging and localization simultaneously. To evaluate for both tasks, we train a network on Audio Set, and evaluate the network on both Audio Set and the DCASE challenge. Because the 527 sound event types of Audio Set is a superset of the 17 sound event types of the DCASE challenge, we do not perform any adaptation when we evaluate the network on the latter. We show that our network, which we name TALNet[13], not only closely matches the current state of the art on Audio Set, but also exhibits strong performance on the DCASE challenge. As far as we know, this is the first network that achieves such good performance on both corpora at the same time.

[13] "TAL" stands for "tagging and localization".

3.3.1 Google Audio Set: Corpus and Metrics

The Google Audio Set consists of 10-second YouTube video excerpts annotated with the presence or absence of 527 types of sound events. It contains over 2 million training recordings, in which the sound event types are not uniformly distributed. About 22 thousand recordings are selected from these to make a balanced training set, but we used the entire unbalanced set for training because the balanced training set is too small. Audio Set also provides an evaluation set of 20,371 recordings. We downloaded all the audio data from YouTube around May 2017; about 1% of the data was already unavailable at the time of the download. The audio was downsampled to 16 kHz and stored in the FLAC[14] format; it takes more than 400 GB of storage.

Since we would also evaluate the model on the DCASE 2017 challenge, we excluded the recordings belonging to the validation, test or evaluation set of the DCASE 2017 challenge from the Audio Set training data to ensure a fair evaluation. We also built a validation set for Audio Set to monitor the progress of model training. The data for this came from the DCASE validation data and a part of the Audio Set training data; its size and the distribution of sound events are comparable with the Audio Set evaluation data.

Because Audio Set only provides presence/absence labeling, it is only possible to evaluate the performance of audio tagging on this corpus. Three (or actually two) metrics are widely used in the literature that uses Audio Set: *mean average precision* (MAP), *mean area under the curve* (MAUC), and *d-prime* (d'). These metrics measure how well a system separates positive and negative recordings, and they do not require setting thresholds. The metrics are calculated as below. For each sound event type, the system generates a ranked list of the evaluation recordings, sorted by the probability of the event in descending order. For every positive recording in the list, we can set the threshold just below its probability and calculate a precision score; the average of all these precision scores is defined as the *average precision* (AP) of the event type. We can also plot the *receiver operator characteristic* (ROC) curve for this event type, whose y-axis and x-axis track the change of the true positive (TP) rate and false positive (FP) rate as the threshold varies from 1 to 0; the *area under the curve* (AUC) metric is defined as the area under the ROC curve. The AP and AUC values are averaged over all sound event types to produce the MAP and MAUC metrics. Because the MAUC metric is often very close to 1, it is warped using the following formula to get the *d-prime* metric, in order to amplify small changes:

$$d' = \sqrt{2}\Phi^{-1}(\text{AUC}) \tag{3.28}$$

[14] "FLAC" stands for "Free Lossless Audio Codec".

where Φ is the accumulative density function of the standard normal distribution. All of MAP, MAUC and d-prime are the larger the better.

3.3.2 TALNet: Training and Evaluation

We trained a convolutional and recurrent neural network (CRNN) whose structure is shown in Fig. 3.10. The structure and the hyperparameters were optimized so that all the five pooling functions (max pooling, average pooling, linear softmax, exponential softmax, and attention) achieved a good performance. The network consists of 10 convolutional layers interleaved with 5 max pooling layers, followed by one bidirectional GRU layer and one fully connected layer. The input to the network is 64-dimensional filterbank features sampled at 40 frames per second, *i.e.* a feature map of size 400 $\times$ 64. The convolutional layers have a kernel size of 3 $\times$ 3; the number of features maps were chosen so that the embedding size at each frame (*i.e.* the number of feature maps times the number of frequency bins) was 2,048. Each max pooling layer reduces the number of frequency bins by half, bringing the embedding size down to 1,024; the first two max pooling layers also reduce the frame rate by half. The output of the last max pooling layer is flattened into 100 frames of 1,024-dimensional vectors; they are fed into a bidirectional GRU layer with 512 hidden units in each direction. The final fully connected layer predicts the probabilities of the 527 sound event types for each frame, which are then pooled using any of the five pooling functions into recording-level predictions.

When training the network, we applied data balancing in the same fashion as in Sec. 3.2.3. We found it essential to apply batch normalization before the ReLU activation of each convolutional layer. We used the Adam optimizer with an initial learning rate of 10^{-3}. Each minibatch contained 250 recordings. We measured the performance of the model every 1,000 batches (called a *checkpoint*), and decayed the learning rate by a factor of 0.8 if the best MAP on the validation set saw no updates in 3 consecutive checkpoints. We did not apply dropout as we found it to hurt the performance. These hyperparameters are summarized in the last column of Table 3.3.

We evaluated the network using the metrics of both Audio Set and the DCASE 2017 challenge. When evaluating on the DCASE challenge, we selected the 17 columns corresponding to the 17 sound event types out of the 527 columns in the network output. The Audio Set metrics do not require setting thresholds, but the DCASE metrics do. Therefore, at each checkpoint, we tuned class-specific thresholds on the DCASE validation data to optimize the audio tagging F_1, and then applied these thresholds to the DCASE test data. For each system, we picked a checkpoint at which all the metrics were close to the best value ever reached. This usually happened between the 10th and 20th checkpoints.

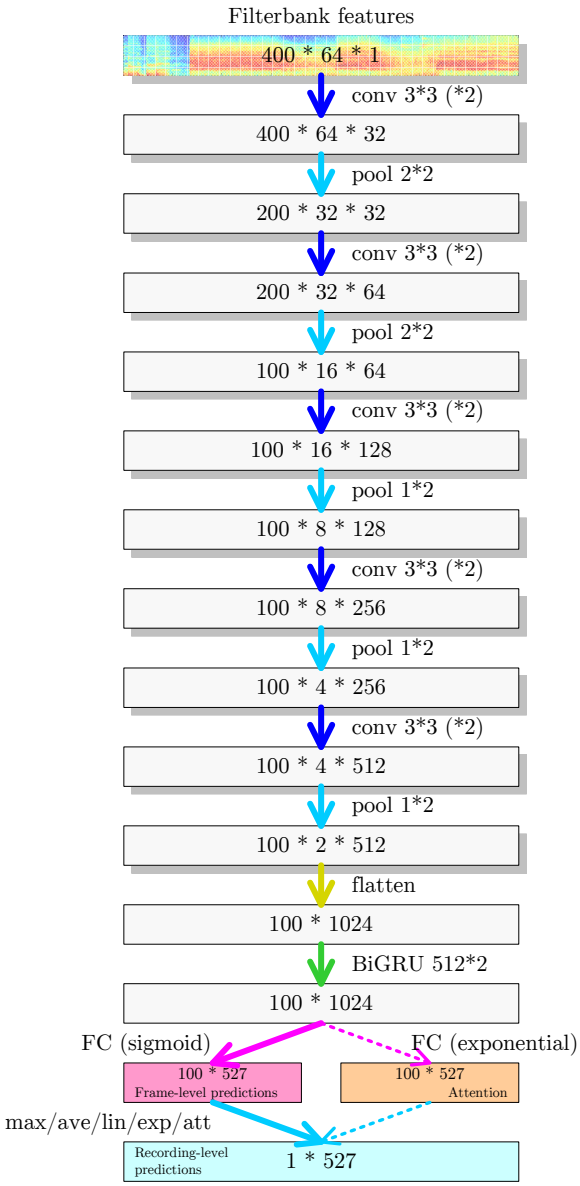


Figure 3.10: The structure of TALNet for joint audio tagging and localization. Shadowed boxes stand for 3-D tensors; their sizes are specified in the order of time frames, frequency bins, and feature maps. Plain boxes stand for 2-D tensors; the first dimension is time. All the convolutional layers use the ReLU activation; the numbers after "conv" (*e.g.* 3*3) specify the size of the kernel, and an additional "(*2)" means the layer is repeated twice. All the pooling layers following convolutional layers perform max pooling; the numbers after "pool" (*e.g.* 2*2) specify the stride. "FC" is short for "fully connected". At the output end, the "attention" block is only used with the attention pooling function.

Group	System	No. of Training Recs.	Audio Set			DCASE 2017		
			MAP	MAUC	d'	Task A	Task B	
						F1	ER	F1
TALNet (Sec. 3.3)	Max pooling	2M	0.351	0.961	2.497	52.6	81.5	42.2
	Average pooling		0.361	**0.966**	2.574	**53.8**	101.8	**46.8**
	Linear softmax		0.359	**0.966**	**2.575**	52.3	**78.9**	45.4
	Exp. softmax		**0.362**	0.965	2.554	52.3	89.2	46.2
	Attention		0.354	0.963	2.531	51.4	92.0	45.5
Literature	Hershey [71, 15]	1M	0.314	0.959	2.452			
	Kumar [128]	22k	0.213	0.927				
	Shah [48]	22k	0.229	0.927				
	Wu [131]	22k		0.927				
	Kong [54]	2M	0.327	0.965	2.558			
	Yu [55]	2M	**0.360**	**0.970**	**2.660**			
	Chen [56]	600k	0.316					
	Chou [57]	1M	0.327	0.951				
DCASE only (Sec. 3.2.3)	Max pooling	50k				45.3	84.7	35.4
	Average pooling					**50.0**	105.9	41.3
	Linear softmax					49.5	**84.3**	**43.7**
	Exp. softmax					48.5	100.6	42.8
	Attention					49.2	102.5	40.1

Table 3.7: Audio tagging and localization performance of TALNet, measured on the evaluation set of Audio Set and the public test set of the DCASE 2017 challenge. The performance of various systems in the literature and systems trained with DCASE data only is also included for comparison. Bold numbers indicate the best performance in each group.

The performance of the network using different pooling functions is listed in the top row of Table 3.7. In terms of audio tagging performance, the linear softmax pooling function is not the optimal; it is better than the max and attention pooling functions, but worse than the average and exponential softmax pooling functions[15]. However, the linear softmax pooling function is still the only pooling function that maintains a balance between false negatives and false positives for the localization task. It has a significantly lower Task B error rate than the average, exponential softmax and attention pooling functions, which make too many false positives, and a significantly higher Task B F_1 than the max pooling function, which makes too many false negatives[16].

In the middle row of Table 3.7, we list the results on Audio Set reported in all the literature that we can find. Not all these systems used the entire unbalanced training set of 2 million recordings; some only used the balanced

[15] Just like in Sec. 3.2.3, we repeated the experiments five times and conducted the one-tailed Welch's t-test with a significance level of 0.05. In terms of MAP, the linear softmax pooling function is significantly better than the max and attention pooling functions, but significantly worse than the average and exponential softmax pooling functions. In terms of d' (regarded as a more precise version of MAUC), the linear softmax pooling function is only significantly better than the attention pooling function. In terms of Task A F_1, no difference is significant.

[16] Significance again verified by the one-tailed Welch's t-test at a level of 0.05.

training set of 22,000 recordings. The only competitive systems are those of Kong *et al.* [54], Chou *et al.* [57], and Yu *et al.* [55]; our linear softmax system outperforms the first two and closely matches the last[17]. We would like to remark that the systems of Kong *et al.*, Chou *et al.* and Yu *et al.* either do not perform localization well, or do not perform localization at all. The system of Kong *et al.* [54] uses the attention pooling function. As we have demonstrated, this pooling function can cause many false positives on the frame level, and suffer from a high error rate. The system of Chou *et al.* [57] applies attention to the frame-level losses instead of the frame-level predictions, but it is likely to suffer from the same problem. The system of Yu *et al.* [55] uses multi-level attention: attention layers are built upon multiple hidden layers, whose outputs are concatenated and further processed by a fully connected layer to yield a recording-level prediction. This deviates from the instance-space (IS) paradigm of MIL, and falls into the embedding-space (ES) paradigm instead: the outputs of the attention layers can be regarded as an explicit vectorial representation of the audio recording. It is normal for ES methods to outperform IS methods on the bag level, but ES systems cannot make instance-level predictions. Our TALNet is the first system we know that achieves good performance for both audio tagging and localization at the same time.

In the bottom row of Table 3.7, we also list the performance of the systems in Sec. 3.2.3, which were trained only on the DCASE data. We see that all the evaluation metrics got better when we moved from the DCASE data to Audio Set. This is not simply a result of adding more training data, because most of the Audio Set data is not related to the 17 DCASE event types. Rather, we regard this as a successful case of *model transfer*[18]. Here the source task is the 527-class SED on Audio Set, and the target task is the 17-class SED on the DCASE challenge. TALNet, which is trained for the source task on a large corpus, turned out to have learned more relevant knowledge for the target task than a network trained directly for the target task on a small corpus.

We also analyzed the correlation between the class-wise performance of the 527 sound event types and their frequency and coverage. This time we estimated the coverage on the evaluation data (about 20,000 recordings), because the training data (about 2 million recordings) was too big. Just like in Sec. 3.2.3, we found all the class-wise metrics (AP, AUC, d') positively correlated with the coverage. However, we found almost no correlation

[17]Because we could not repeat the experiments of Yu *et al.*, we used the one-tailed one-sample t-test (function `ttest` in Matlab) to test whether the mean performance of TALNet with the linear softmax pooling function is significantly worse than the reported performance of Yu *et al.*'s system. The gap between TALNet and Yu *et al.*'s system turned out to be indeed significant at a level of 0.05.

[18]This is actually a degenerate case of model transfer, because the model is not adapted at all when applied to the target task.

between the metrics and the frequency this time. These trends apply uniformly to all the five systems using different pooling functions.

3.4 Summary

In this chapter, we have shown that multiple instance learning (MIL) is a viable choice for sound event detection with presence/absence labeling. Even though the training labels do not contain any information about the temporal location of sound events, MIL is able to localize them with reasonable accuracy.

The choice of the pooling function is an important question in MIL. In this chapter, we have extensively studied six pooling functions: max pooling, noisy-or pooling, linear softmax, exponential softmax, and attention. Based on both theoretical analysis and experimental validation, we have established linear softmax as the optimal of the six. The linear softmax function facilitates the gradient flow, achieves a good balance between false negatives and false positives, and makes consistent bag-level and instance-level predictions. Nevertheless, we also point out potential directions for discovering even better pooling functions, such as interpolating between the linear and exponential softmax functions, or regularizing attention with monotonicity constraints.

Based upon these findings, we built a network called TALNet that achieves state-of-the-art performance on the Google Audio Set and strong performance on the DCASE 2017 challenge at the same time. As far as we know, this is the first system that exhibits such good performance simultaneously on the two tasks of audio tagging and localization.

Chapter 4

Sound Event Detection with Sequential Labeling

The previous chapter has studied sound event detection with presence/absence labeling. While presence/absence labeling is the prevalent form of annotation currently available, we are still interested in find out whether a slightly stronger form of annotation can improve the performance of SED.

In this chapter, we explore *sequential labeling*, which is stronger than presence/absence labeling but still weaker than frame-wise labeling. Roughly speaking, sequential labeling tells us the order of sound events occurring in each recording, but not the exact onset and offset times of each occurrence. It is similar to the form of supervision used to train contemporary speech recognition systems: each utterance is labeled with an ordered sequence of phonemes, but the exact onset and offset times of the phonemes are unknown. Just as in the case of presence/absence labeling, we are not only interested in detecting which types of sound events are present or recovering the order of these events, but also interested in the precise localization of the onsets and offsets of each occurrence.

There is one difference between the tasks of polyphonic SED and speech recognition: sound events can overlap, while phonemes cannot. It can be hard to define the order between overlapping sound events. To get around this problem, we define sequential labeling as an ordered sequence of event boundaries (*i.e.* onsets and offsets), instead of the event themselves. For example, if a recording contains a dog bark *followed by* the sound of a car, we label the recordings as `<dog> </dog> <car> </car>`[1]. On the other hand, if a recording contains a bark *while* a car passes by, then we label it as `<car> <dog> </dog> </car>`. When two different sound events start or end at exactly the same time, we choose an arbitrary order for the onset or offset labels involved.

[1] We borrow the notation from HTML: `<event>` signifies the onset of an event, and `</event>` signifies the offset of an event.

In modern speech recognition systems, *connectionist temporal classification* (CTC) is a popular technique to deal with sequential labeling. CTC establishes a multiple-to-one mapping between the frame-level predictions of a neural network and the label sequence of an recording, and, during training, attempts to maximize the total probability of all frame-level predictions corresponding to the ground truth label sequence. This idea may be directly borrowed and applied to SED with sequential labeling. While the standard CTC is good at detecting short events, it does not localize long events well due to a "peak clustering" phenomenon. We modify the standard CTC framework and propose a connectionist temporal localization (CTL) framework. This framework solves the "peak clustering" problem and closes about one third of the gap between presence/absence labeling and strong labeling, measured in terms of frame-level F_1 macro-averaged across sound event types.

Sequential labeling is easier to produce manually than strong labeling. Moreover, they may be mined from textual descriptions of audio recordings, such as "a dog barks while a car passes by". Nevertheless, building a sequentially labeled corpus of a size similar to Google Audio Set is still an enormous task beyond the abilities of the author's lab. In order to test out algorithms designed for sequential labeling and compare them to those designed for strong labeling and presence/absence labeling, we generated strong labeling and sequential labeling for the Audio Set recordings automatically with TALNet, a state-of-the-art multiple instance learning (MIL) system trained with presence/absence labeling in Chapter 3. The automatic labels are no doubt noisy, but they suffice to demonstrate the extra advantage that sequential labeling brings about compared to presence/absence labeling.

The content of this chapter is organized as follows. Sec. 4.1 describes the procedure we used to automatically generate strong labeling and sequential labeling for Audio Set recordings, and the resulting corpus used in the experiments in this chapter. We also establish an upper baseline and a lower baseline with strong labeling and presence/absence labeling, respectively. Sec. 4.2 demonstrates that a direct application of CTC to SED with sequential labeling is able to detect short events, but it fails to localize long events due to a "peak clustering" phenomenon. We analyze the cause and, in Sec. 4.3, propose a connectionist temporal localization (CTL) framework, which successfully surpasses the performance of presence/absence labeling. In Sec. 4.4, we combine a CTL system with a MIL system for further improvement. In Sec. 4.5, we briefly discuss the problem of generalizing to different data, and Sec. 4.6 summarizes the contributions of this chapter.

4.1 Data Preparation

4.1.1 Automatic Generation of Strong and Sequential Labels

We decided to select recordings from the entire Google Audio Set [15] to build the training, validation and evaluation sets for the experiments in this chapter. This decision was made based upon both the quantity and the quality of the data. One alternative choice we considered but did not adopt was to use the data of the DCASE 2017 challenge [83], which is a subset of Audio Set. The total duration of this data (140 hours) could be considered enough for training, but this data is only concerned with 17 types of vehicle and warning sounds, which are not diverse enough. In addition, a significant portion of the recordings ($> 40\%$) contain only one occurrence of a sound event spanning the entire recording, which do not bring out the difference between presence/absence labeling and sequential labeling. Another choice was the Noiseme corpus [85]. The advantage of this corpus is that it is strongly labeled by hand, but this corpus only contains about 13 hours of audio, which is too little for training. We do briefly discuss using the Noiseme corpus for testing in Sec. 4.5, though.

We first generated strong labels for the Audio Set recordings using TALNet – the best system we trained in Sec. 3.3, *i.e.* the network shown in Fig. 3.10 with the linear softmax pooling function. This system produced frame-level probabilities for each sound event type; we binarized them using class-specific thresholds to generate strong labels. The thresholds were tuned to maximize the audio tagging F_1 micro-averaged across the 527 sound event types on the evaluation data of Audio Set. This was not an impeccable action – a more prudent choice would be to tune the thresholds on the validation data, but it is acceptable since it only affects the generation of the labels and does not directly affect any of the experiments. An inspection of the strong labels revealed that there were many very short occurrences of sound events and very short gaps between occurrences of the same sound event type. These could have been eliminated by smoothing, but we chose not to apply any smoothing because (1) it was hard to pick optimal parameters for smoothing, and (2) smoothing would have eliminated the occurrences of many transient sound events (*e.g.* knocks) altogether.

Audio Set comes with a repertoire of 527 types of sound events. Using all of them would be quite a burden for the experiments; in addition, not all the sound event types have a high enough quality. Many of the event types form a hierarchy, such as "`speech` – `male speech`", and "`animal` – `dog` – `bark`". Quite often, a recording is only labeled with one event type in such chains, even though its hypernyms and/or hyponyms also apply. For example, a recording containing a man's speech may be only labeled with `speech`, not `male speech`. Because of this deficiency in the training data, TALNet also often failed to predict labels such as `male speech` when it should. We

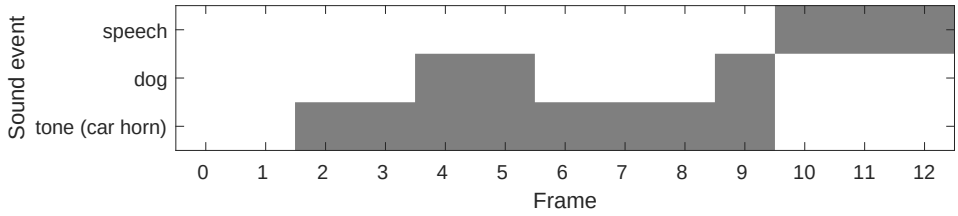


Figure 4.1: An example of the strong labels generated by TALNet. Dark means active. The corresponding sequential labels are: `<tone> <dog> </dog> <dog> </dog> </tone> <speech> </speech>`.

believed it was necessary to reduce the set of sound event types. The Noiseme corpus [85] proposed an ontology of 42 sound event types, which served as a good starting point. Starting from this ontology, we merged some confusable event types (*e.g.* `heavy engine`, `light engine`, `quiet engine`), and ended up with 35 classes (which we call "common events"). Because the Audio Set ontology was designed with the noiseme ontology in mind, it was not hard to find the Audio Set event type(s) corresponding to each common event. The 35 common events, and the corresponding noiseme and Audio Set event types are listed in Table 4.1. The 527-class strong labeled were reduced to the 35 common events using the following rule: at each frame, a common event is active iff any of its corresponding Audio Set event is active. For simplicity, we did not descend down the hierarchy in this step: if TALNet predicted the Audio Set event `male speech` as active at a frame but `speech` as inactive, then the common event `speech` was labeled as inactive.

Reducing the set of sound event types also reduced the amount of training data. We discarded Audio Set recordings that did not contain any of the 35 common events, according to the predictions of TALNet. Because the common events `speech`, `sing`, `music` and `crowd` had overwhelmingly large numbers of recordings, we further excluded recordings that did not contain any of the other 31 common events. This filtering was applied to the unbalanced training recordings of Audio Set to generate the training data for the experiments in this chapter; the resulting training data contained 359,741 recordings totaling nearly 1,000 hours, which account for about 18% of the original unbalanced training recordings. The same filtering was applied to the validation and evaluation data used in Chapter 3; the number of surviving recordings was 4,879 and 5,301, respectively.

Table 4.2 lists some statistics of the data we gathered for the experiments in this chapter. On average, each recording contains about 6 event occurrences. Less than 10% of the training recordings contain only one event occurrence, indicating that the ground truth of this data is less "boring" than the DCASE data. The average duration of the event occurrences is under 1 second. Such a short average duration can be attributed to two

Common event	**Corresponding noiseme(s)**	**Corresponding Audio Set event(s)**
00: speech	speech speech_ne mumble	0: speech
01: sing	singing music_sing	27: singing
02: music	music music_sing	137: music
03: laugh	laugh	16: laughter
04: cry	cry	22: crying, sobbing
05: scream	scream	14: screaming
06: cheer	cheer	66: cheering
07: applause	applause	67: applause
08: crowd	crowd	69: crowd
09: human	human	41: breathing 47: cough 49: sneeze 50: sniff
10: child	child	6: babbling
11: dog	anim_dog	74: dog
12: cat	anim_cat	81: cat
13: bird	anim_bird	111: bird
14: knock	knock	359: knock
15: thud	thud	460: thump, thud
16: clap	clap	63: clapping 427: gunshot, gunfire
17: click	click	491: click
18: bang	bang	466: bang
19: beep	beep	481: beep
20: clatter	clatter	489: clatter
21: rustle	rustle	487: rustle
22: scratch	scratch	474: scratch
23: hammer	hammer	419: hammer
24: rub	washboard	476: rub
25: engine	engine_heavy engine_light engine_quiet power_tool	343: engine 424: power tool
26: phone	phone	389: telephone
27: whistle	whistle	402: whistle
28: squeak	squeak	361: squeak
29: tone	tone	482: ping 483: ding 308: Vehicle horn, car horn, honking 318: air horn, truck horn
30: siren	siren	396: siren
31: water	water	288: water
32: wind	wind micro_blow other_distorted	283: wind 285: wind noise (microphone)
33: radio	radio	525: radio
34: white	white_noise	520: white noise

Table 4.1: The 35 common events, and their corresponding noisemes and Audio Set sound event types.

	Training	Validation	Evaluation
# recordings	359,741	4,879	5,301
Average # event occurrences per recording	6.3	5.4	5.4
% recordings with only one event occurrence	8.6 %	16.2 %	12.3 %
Average duration of event occurrence	0.89 s	0.80 s	0.81 s
Polyphony in non-silence regions	1.11	1.09	1.07
% adjacent onset/offset pairs	81.0 %	83.7 %	86.4 %
% overlapping boundaries	10.9 %	9.7 %	8.1 %

Table 4.2: Some statistics of the data used for the experiments in Chapter 4.

reasons: first, we included many types of events that are inherently short (*e.g.* `knock`, `clap`, `click`); second, our choice of no smoothing left us with many short occurrences of events. The polyphony level is barely above 1, which means there is not much overlap between the event occurrences.

Finally, we derived sequential labels from the strong labels generated by TALNet. An example is shown in Fig. 4.1. When multiple event boundaries occur at the same time, we had to impose an order on these boundaries: we always put the offset labels before the onset labels; the offset labels and onset labels were sorted according to the order in Table 4.1 among themselves. This specific way to determine the order should not matter much, because only about 10% of the event boundaries coincide exactly in time. Also, we found that more than 80% of the time, the onset label of an event is immediately followed by its corresponding offset label.

4.1.2 Baseline Systems

We built two baseline systems: the first system is trained with strong labeling, and serves as an upper bound; the second system is a multiple instance learning (MIL) system trained with presence/absence labeling, and serves as a lower bound. The performance of an SED system that effectively makes use of sequential labeling should fall between the two bounds, and approach the upper bound as closely as possible.

The structure of the baseline systems are shown in Fig. 4.2 (a) and (b). The network takes the same input as in Chapter 3: 64-dimensional filterbank features extracted at 40 Hz. The network contains 6 convolutional layers and 6 max pooling layers. The convolutional layers use the ReLU activation function, and batch normalization is applied before the ReLU. The first convolutional layer produces 16 feature maps, and each subsequent convolutional layer doubles the number of feature maps. On the other hand, each pooling layer reduces the number of frequency bins by half, so the embeddings produced by all pooling layers have a dimensionality of 512. The first two pooling layers also half the frame rate, so the number of frames for a 10-second recording reduces from 400 to 100. The structure up to here (and the GRU layer to follow) is shared across all the systems in this chapter;

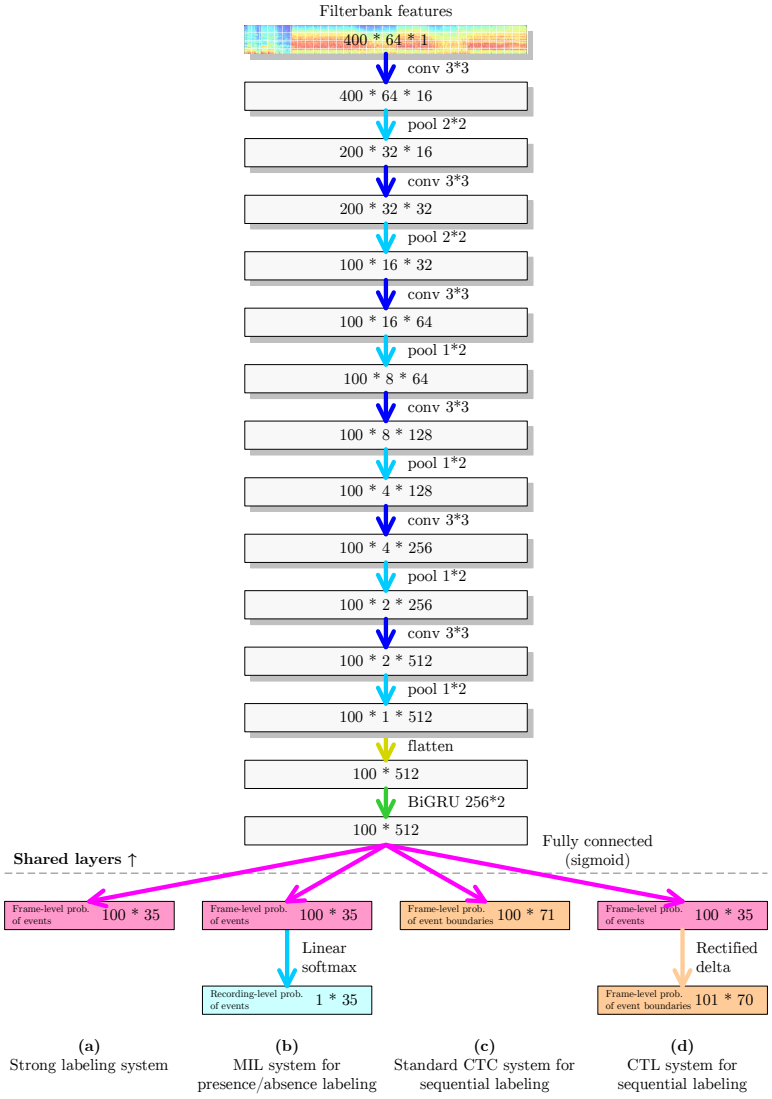


Figure 4.2: The structures of the strong labeling baseline system, the multiple instance learning (MIL) baseline system, the standard CTC system, and the connectionist temporal localization (CTL) system used in Chapter 4. The layers above the dashed line are shared across the four systems. Shadowed boxes stand for 3-D tensors; their sizes are specified in the order of time frames, frequency bins, and feature maps. Plain boxes stand for 2-D tensors; the first dimension is time. All the convolutional layers use the ReLU activation; the numbers after "conv" (*e.g.* 3*3) specify the size of the kernel. All the pooling layers following convolutional layers perform max pooling; the numbers after "pool" (*e.g.* 2*2) specify the stride. Batch normalization is applied to all the convolutional layers before the ReLU activation. Note that the output layer of the CTL system has only 70 units, because there is no unit for the blank label.

it has been found to produce optimal or near optimal performance for all the experiments.

The output of the last max pooling layer is flattened and fed into a GRU layer; finally, a fully connected layer with the sigmoid activation function predicts frame-level probabilities of the 35 common events. In the strong labeling system, these frame-level probabilities are directly compared with the labels generated by TALNet to compute the binary cross-entropy loss. In the MIL system, the frame-level probabilities of events are aggregated using the linear softmax pooling function into recording-level probabilities of events, and then compared with the ground truth. Note that the ground truth here is aggregated from the strong labels generated by TALNet using the standard multiple instance assumption. We did not use the original presence/absence labels provided by Audio Set because they would make this system less comparable with the other systems in this chapter.

The evaluation metric we use throughout this chapter is the frame-level F_1 macro-averaged over all the 35 common events. This metric differs from the micro-average segment-level F_1 used in the DCASE challenge, and we made such choices on purpose. Measuring F_1 on the frame level instead of the segment level avoids the influence of the segment length, and eliminates the need for an algorithm to aggregate the frame-level predictions within a segment. And we chose the macro-average (calculating the F_1 for each event type then taking the average) instead of the micro-average (aggregating the TP, FN and FP counts first then calculating the F_1) because the former allows the evaluation of individual sound event types. The class-specific thresholds were tuned to optimize this macro-averaged frame-level F_1 on the validation data, then applied to the evaluation data.

The baseline systems were trained using the Adam optimizer with a constant learning rate of 10^{-3}. The loss function was binary cross-entropy, averaged over frames and events for the strong labeling system and over recordings and events for the MIL system. We applied the same data balancing technique as described in Sec. 3.2.3 to make all event types account for an equal fraction of the training data. We used a batch size of 500 recordings, and checked the performance every 200 batches (called a checkpoint). At each checkpoint, we tuned class-specific thresholds to optimize the macro-average frame-level F_1 on the validation data, and then applied the thresholds to the evaluation data. We report the highest evaluation F_1 obtained within 100 checkpoints. The same settings were used to train all the networks in this chapter, except that the loss function differs from network to network.

The strong labeling system reached an F_1 of 67.38%, while the MIL system reached an F_1 of 55.83%. Fig. 4.3 demonstrates the frame-level predictions of the two baseline systems on an evaluation recording. This recording contains the sound of a dog whining intermingled with speech, and will be referred to as the "whining dog" recording hereafter. While

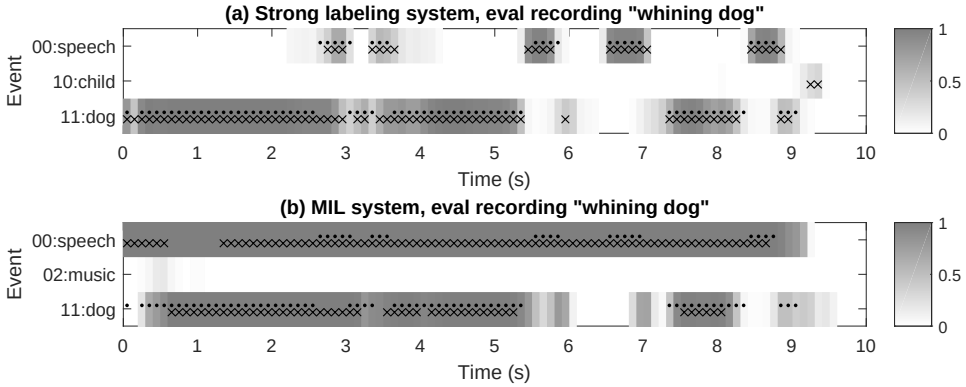


Figure 4.3: The frame-level predictions of the two baseline systems on the evaluation recording `0F04c_rY4aw` ("whining dog"). Shades of gray indicate the frame-level probabilities of the sound events; crosses indicate where these probabilities exceed the class-specific thresholds, while dots indicate the ground truth. Events that do not receive significant probability anywhere in the recording are omitted.

both systems localize the `dog` sounds well, the MIL system does not localize the `speech` sounds as well as the strong labeling system. This is probably because the MIL system was trained with weaker supervision.

4.2 Standard CTC for SED with Sequential Labeling

We built a SED system that handles sequential labeling with the standard CTC, whose structure is shown in Fig. 4.2 (c). The layers up to the GRU layer are identical to the baseline systems. The final fully connected layer has 71 output neurons, which predict frame-level probabilities of the onset and offset labels of the 35 common events, plus a blank label. These frame-level probabilities are used to compute the probability of the ground truth label sequence using the standard CTC forward algorithm as introduced in Sec. 2.2. This network was trained using the same hyperparameters as the baseline systems; the loss function was the log probability of the ground truth label sequence, averaged across frames.

During evaluation, we decoded the predictions of the network using *best-path decoding*: for each frame, we selected the label with the largest probability, no matter whether it was an onset label, an offset label, or a blank label. We did not apply class-specific thresholding. To generate frame-level predictions of events (*i.e.* localization) for evaluation, we looked at the selected labels for each event type: for each innermost matching pair of onset/offset labels, we marked the event as active from the frame of the

onset to the frame of the offset. Non-matching or non-innermost matching onset/offset labels were ignored.

This procedure is better illustrated with an example. Suppose the frame-level probabilities of event boundaries (and the blank label) are given in the following matrix:

blank	.30	**.40**	.01	.20	.10	.05	.05	**.60**	.10
`<cat>`	**.40**	.20	**.95**	.10	.10	.05	.05	.10	.10
`</cat>`	.05	.10	.01	.10	.10	.10	**.70**	.30	**.60**
`<dog>`	.20	.20	.01	**.50**	**.60**	.20	.10	.05	.10
`</dog>`	.05	.10	.02	.10	.10	**.60**	.10	.05	.10
Frame	1	2	3	4	5	6	7	8	9

Picking the most probable label at each frame yields the following labels:

	–							–	
	`<cat>`		`<cat>`						
							`</cat>`		`</cat>`
				`<dog>`	`<dog>`				
						`</dog>`			
Frame	1	2	3	4	5	6	7	8	9

For the `cat` event, the onset label at Frame 3 and the offset label at Frame 7 form an innermost matching pair, so the `cat` event is marked as active from Frame 3 to Frame 7. The onset label at Frame 1 and the offset label at Frame 9, even though they match, are not an innermost matching pair, so they are ignored. For the `dog` event, the onset labels at Frames 4 and 5 should be treated as a single onset. In such a case, we still pick the innermost onset label (at Frame 5) to match with the offset label at Frame 6. The resulting frame-level predictions of events for evaluation is then ("+" stands for active):

`cat`	−	−	+	+	+	+	+	−	−
`dog`	−	−	−	−	+	+	−	−	−
Frame	1	2	3	4	5	6	7	8	9

The standard CTC algorithm, unfortunately, only reached a macro-average frame-level F_1 of 31.91%, which was even lower than the MIL baseline. By inspecting the behavior of the standard CTC system on some evaluation recordings, we found that it was actually good at detecting short events, but had a prevalent failure mode on long event occurrences. We plot the frame-level predictions of the network on three evaluation recordings in Fig. 4.4. Graph (a) shows the predictions on a recording

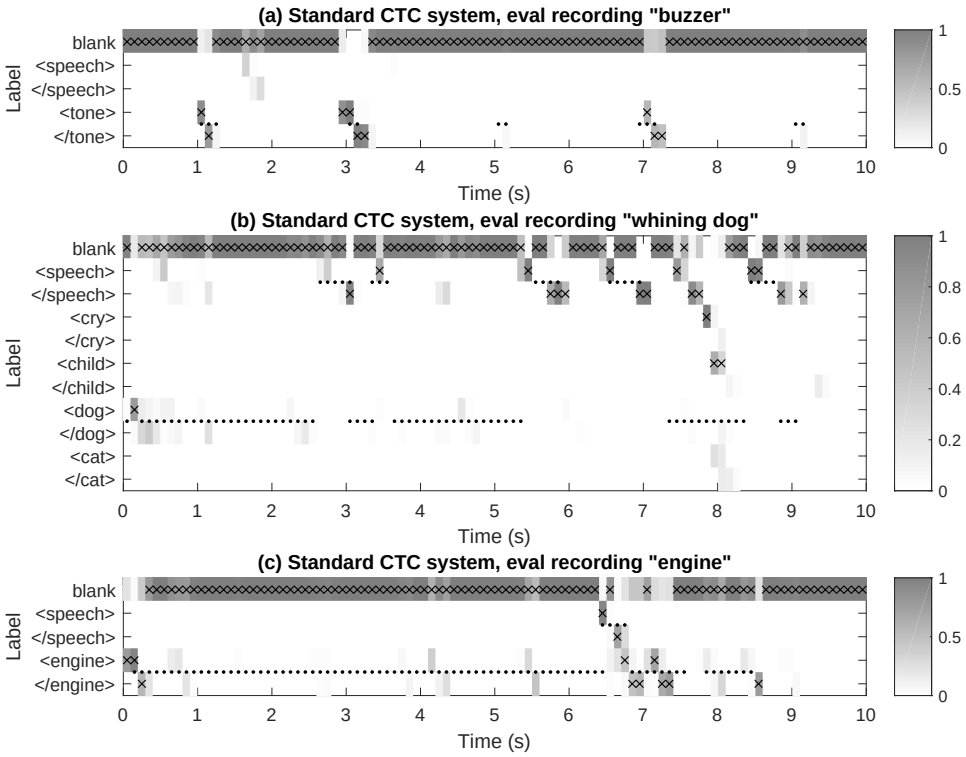


Figure 4.4: The frame-level predictions of the standard CTC system on three evaluation recordings: `-2EKWgTNEYU` ("buzzer"), `0F04c_rY4aw` ("whining dog"), and `-z3n74RK92U` ("engine"). Shades of gray indicate the frame-level probabilities of event boundaries as well as the blank label; crosses indicate the most probable label at each frame, while dots indicate the ground truth events. Events whose boundaries do not receive significant probability anywhere in the recordings are omitted.

containing intermittent buzzer tones: three of the five occurrences were actually successfully detected. Graph (b) shows the predictions on the "whining dog" recording: while many of the short segments of speech were correctly detected, the longer dog whines were localized badly. Instead of generating a clear onset peak at the beginning of each occurrence and a clear offset peak at the end, the CTC system generated many vague pairs of onset and offset labels clustered together at random places in the recording. This failure mode is more pronounced in Graph (c): the recording contains continuous engine noise, but the CTC system predicted many clustered pairs of onset and offset peaks instead. This problem, which we dub the "peak clustering" problem, has already been pointed out in our previous work [63].

Why does the CTC system tend to generate clustered onset and offset peaks? There can be several reasons behind this. First, adjacent onset and offset labels are an extremely common pattern in the training label

sequences. As the statistics indicate, more than 80% of the onset labels in the training data are immediately followed by their corresponding offset labels. As a result, CTC may misunderstand a pair of onset and offset labels as collectively indicating the existence of an event, instead of understanding them as separately indicating the event boundaries. Second, the CTC loss function only mandates the order of the predicted labels, without imposing any temporal constraints. In this case, the recurrent layer of the network will prefer to emit onset and offset labels next to each other, because this minimizes the effort of memory. Third, the output layer of the CTC network is designed to predict frame-level probabilities of event *boundaries*; it is expected to keep "silent" both when an event is inactive and when an event is continuing. When the network predicts the onset and offset labels of a long event occurrence next to each other, it is actually not violating this expectation too much, and does not have enough incentive to correct this behavior.

The third point actually points to a more serious problem with the standard CTC: the network is trained to ignore the potentially huge differences in the acoustic features when an event is continuing and when it is inactive. This is in contrast with the baseline systems, which predict frame-level probabilities of the events *themselves*: they are trained to make different predictions when an event is continuing and when an event is inactive, so they naturally learn to leverage the differences in the acoustic features. This illuminates a way to make CTC work: the network should predict frame-level probabilities of the events themselves just as the baseline systems do, and the frame-level probabilities of the event boundaries can be derived from those of the events themselves. This derivation step can be regarded as a regularizer for the boundary probabilities, which helps to prevent the network from falling into the undesirable local optima of clustered peaks.

4.3 Connectionist Temporal Localization (CTL) for SED with Sequential Labeling

4.3.1 The CTL Forward Algorithm

Inspired by the analysis in the previous section, we modified the CTC framework into a connectionist temporal localization (CTL) framework for SED with sequential labeling. The forward algorithm used during training was also adapted accordingly, which we explain in this section.

The structure of a network using the CTL framework is shown in Fig. 4.2 (d). After the GRU layer, a fully connected layer predicts the frame-level probabilities of events in the same way as the baseline systems. The following layer implements a fixed algorithm (*i.e.* without tunable

parameters) to derive frame-level probabilities of event boundaries. Finally, a forward algorithm computes the total probability of the ground truth label sequence based on these frame-level boundary probabilities.

We use a simple "rectified delta" operation to derive boundary probabilities from event probabilities. More formally, let $y_t(\texttt{e})$ be the probability of the event `e` being active at frame t. Here $1 \leq t \leq T$, where T is the number of frames in the recording in question. Let $z_t(\texttt{<e>})$ and $z_t(\texttt{</e>})$ be the probabilities of the onset and offset labels of the event `e` at frame t. We calculate them using the following equations:

$$\begin{aligned} z_t(\texttt{<e>}) &= \max[0, y_t(\texttt{e}) - y_{t-1}(\texttt{e})] \\ z_t(\texttt{</e>}) &= \max[0, y_{t-1}(\texttt{e}) - y_t(\texttt{e})] \end{aligned} \tag{4.1}$$

In these equations we allow t to range from 1 to $T + 1$, in order to accommodate events that start at the first frame or end at the last frame. When $y_0(\texttt{e})$ or $y_{T+1}(\texttt{e})$ is referenced, we assume it to be 0.

A problem that emerges immediately is that the sum of the $z_t(l)$'s for all the boundary labels l may exceed one, in which case it is hard to assign a probability $z_t(\text{blank})$ to the blank label. To avoid this problem, we treat the boundary labels at a frame as *mutually independent*, instead of *mutually exclusive*. Under this assumption, we may calculate the probability of not emitting any boundary labels as:

$$\epsilon_t = \prod_l [1 - z_t(l)] \tag{4.2}$$

where l goes over the onset and offset labels of all event types. The probability of emitting a single boundary label l is then:

$$p_t(l) = z_t(l) \prod_{l' \neq l} [1 - z_t(l')] \tag{4.3}$$

If we define

$$\delta_t(l) = \frac{z_t(l)}{1 - z_t(l)} \tag{4.4}$$

Then Eq. 4.3 reduces to

$$p_t(l) = \epsilon_t \cdot \delta_t(l) \tag{4.5}$$

The assumption that boundary labels at the same frame are mutually independent seems to eliminate the need for the blank label. Indeed, in the standard CTC, the blank label serves two purposes: (1) to allow emitting nothing at a frame, and (2) to separate consecutive repetitions of the same label. With the independence assumption, the first purpose is naturally achieved. As for the second point, we make the following simplification: when mapping the frame-level emissions to the recording-level label sequence, we no longer collapse consecutive repeating labels into

a single one. With this simplification, the blank label can be removed altogether.

The independence assumption also allows us to assess the probability of emitting multiple labels at the same frame, which is not possible with the standard CTC. The probability of emitting multiple labels $l_1, \ldots, l_k$ together at frame t can be calculated as

$$\begin{aligned} p_t(l_1, \ldots, l_k) &= \prod_{i=1}^{k} z_t(l_i) \prod_{l \notin \{l_1, \ldots, l_k\}} [1 - z_t(l)] \\ &= \epsilon_t \prod_{i=1}^{k} \delta_t(l_i) \end{aligned} \tag{4.6}$$

Note that this probability does not specify an order of the labels $l_1, \ldots, l_k$. We make the assumption that this is also the probability of emitting these labels in any specific order. One may argue that one should distribute the probability $p_t(l_1, \ldots, l_k)$ equally to all possible orders, *i.e.* define the probability of any specific order as $p_t(l_1, \ldots, l_k)/k!$. But our assumption can be more reasonable considering that when we generated the ground truth label sequences, boundaries that coincide in time were assigned an arbitrary order, while there should actually be no order at all. Eq. 4.6 requires the labels $l_1, \ldots, l_k$ to be all different, but we ignore this constraint in the algorithm below.

Now we can formulate our CTL forward algorithm. What we want to find is the total probability of emitting the ground truth label sequence $L = \{l_1, \ldots, l_{|L|}\}$, regardless of which labels are emitted at which frame. What we are given is the frame-level probabilities of events $y_t(\mathtt{e})$, from which we can derive the probability $p_t(\cdot)$ of emitting zero, one or more labels at each frame by Eq. 4.6. Let $\alpha_t(i)$ be the probability of having emitted exactly the first i labels of L after t frames. The α's can be computed with the following recurrence formula:

$$\begin{aligned} \alpha_t(i) &= \sum_{j=0}^{i} \alpha_{t-1}(i-j) \cdot p_t(l_{i-j+1}, \ldots, l_i) \\ &= \sum_{j=0}^{i} \alpha_{t-1}(i-j) \cdot \epsilon_t \prod_{k=i-j+1}^{i} \delta_t(k) \end{aligned} \tag{4.7}$$

In the summation, the index j stands for the number of labels emitted at frame t. The initial values are:

$$\alpha_0(i) = \begin{cases} 1, & \text{if } i = 0 \\ 0, & \text{if } i > 0 \end{cases} \tag{4.8}$$

The final value, $\alpha_{T+1}(|L|)$, is the total probability of emitting the label sequence L, and its negative logarithm is the contribution of the recording in question to the loss function.

System	F_1 (%)
Max concurrence = 1	59.92
Max concurrence = 2	57.49
Max concurrence = 3	53.63
Strong labeling (upper baseline)	67.38
MIL (lower baseline)	55.83

Table 4.3: Performance of the CTL network with different values of max concurrence.

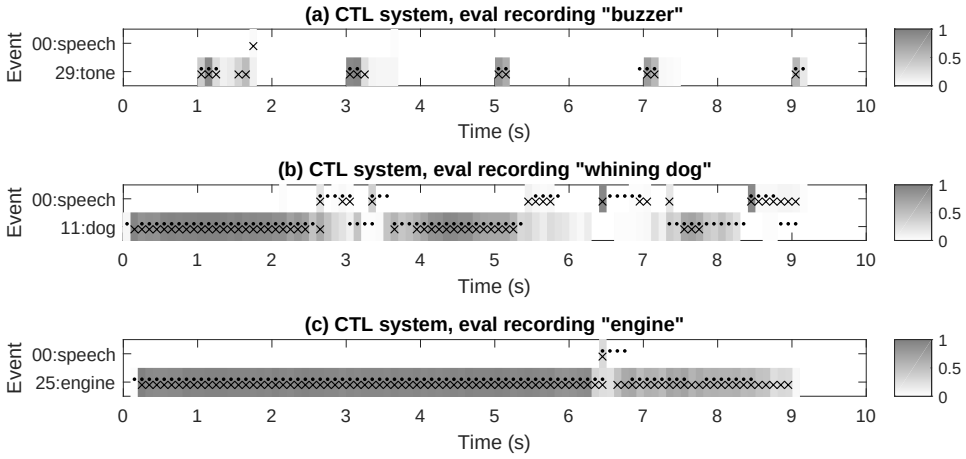


Figure 4.5: The frame-level predictions of the CTL system on three evaluation recordings: `-2EKWgTNEYU` ("buzzer"), `0F04c_rY4aw` ("whining dog"), and `-z3n74RK92U` ("engine"). Compare with Figs. 4.3 and 4.4.

Eq. 4.7 allows emitting arbitrarily many labels at the same frame. When the ground truth label sequence is long, this can pose a problem of time complexity. In practice, it is rare (only about 10% of the time) to observe multiple labels emitted at the same frame. Therefore, it can be desirable to limit the maximum number of concurrent labels, *i.e.* the maximum value of j in Eq. 4.7. We call this maximum value the *max concurrence*.

4.3.2 Experiment Results

We trained a CTL network the same way as the standard CTC network. We tried out a max concurrence from 1 to 3, and the resulting F_1's are listed in Table 4.3. The results indicate that it is sufficient to allow only one label to be emitted at each frame; allowing concurrent labels is actually harmful. Also, if we compare the best performance of the CTL system (59.92%) against the baseline systems, we find that it closes about one third of the gap between the MIL system (55.83%) and the strong labeling system

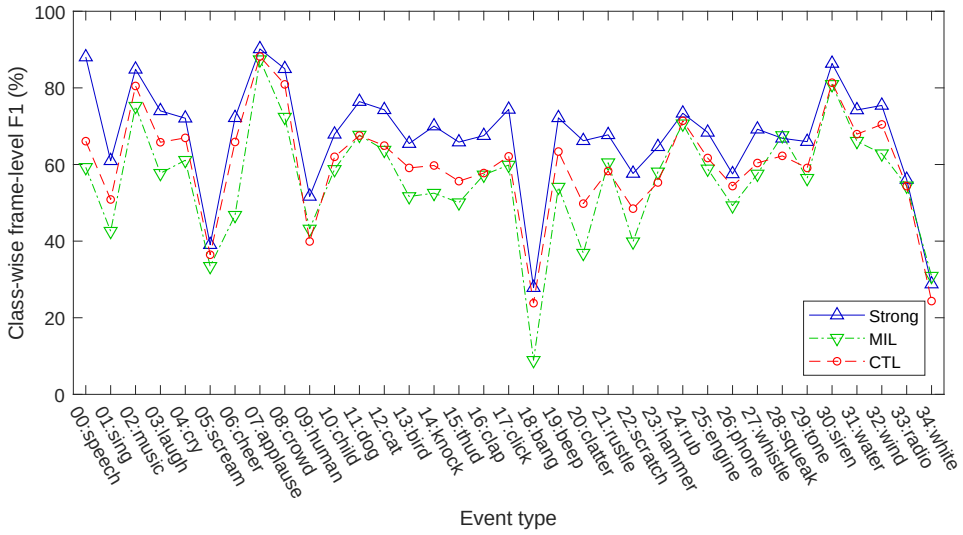


Figure 4.6: The class-wise frame-level F_1's of the CTL system (with a max concurrence of 1) compared with the two baseline systems. Best viewed in color.

(67.38%).

Fig. 4.5 illustrates the predictions of the CTL system (with a max concurrence of 1) on some evaluation recordings. Compared with the standard CTC system (Fig. 4.4), the CTL system no longer fails for long event occurrences (such as the `dog` sounds in Graph (b), and the `engine` noise in Graph (c)). The huge improvement in the F_1 can be mainly attributed to getting rid of this failure mode. Compared with the baseline systems (Fig. 4.3), the CTL system localizes the `speech` segments in the "whining dog" recording better than the MIL system, but still not as well as the strong labeling system. This exemplifies how the CTL system closes part but not all of the gap between the two baselines.

Fig. 4.6 demonstrates the class-wise frame-level F_1's of the CTL system (with a max concurrence of 1) and the two baseline systems. For 28 out of the 35 common events, the F_1 of the CTL system falls between the two baselines. This indicates that the improvement from the MIL baseline to the CTL system is not contributed by particular event types; most of the event types benefited from the use of sequential labeling.

4.3.3 Discussion: An Alternative CTL Algorithm

As we explored the details of the CTL algorithm, we tried out an alternative way of deriving the boundary probabilities $z_t(\texttt{<e>})$ and $z_t(\texttt{</e>})$ from the event probabilities $y_t(\texttt{e})$. Assuming the event probabilities as independent

across adjacent frames, we derived the boundary probabilities as:

$$\begin{aligned} z_t(\texttt{<e>}) &= [1 - y_{t-1}(\texttt{e})] \cdot y_t(\texttt{e}) \\ z_t(\texttt{</e>}) &= y_{t-1}(\texttt{e}) \cdot [1 - y_t(\texttt{e})] \end{aligned} \tag{4.9}$$

Training with this alternative CTL algorithm turned out to be much more difficult than using Eq. 4.1 in Sec. 4.3.1. During the training we observed two failure modes. The first failure mode is shown in Fig. 4.7 (a): for a random subset of event types (*e.g.* `cry`, `human`, `bird`) the frame-level probabilities stayed at one for almost all frames, except for being 0.5 at the first and last frames. This phenomenon usually manifested itself after only a few minibatches, and never disappeared. Fortunately, we were able to solve this problem by initializing the bias of the final fully connected layer to -1, in order to discourage large frame-level predictions.

After applying the initial bias, the network was able to train normally for 5 to 10 checkpoints. It reached a peak F_1 around 43%. The frame-level predictions at this checkpoint is shown in Fig. 4.7 (b); we can observe some occurrences of `dog` noises taking shape. However, after this, the F_1 started to drop, and the network got stuck in a new failure mode shown in Fig. 4.7 (c): the frame-level predictions exhibited many peaks scattered here and there.

Neither of the two failure modes was observed when deriving boundary probabilities from event probabilities using Eq. 4.1. Here we attempt to give an explanation why Eq. 4.1 works better than its alternative, Eq. 4.9. First, we notice that in both failure modes, in any two consecutive frames, there is always at least one frame whose probability is either 0 or 1. In this case, the Eqs. 4.1 and 4.9 are equivalent. In other words, these failure modes incur the same loss when using either equation. These failure modes appear to be attractive local optima for Eq. 4.9, but this is not the case for Eq. 4.1. A probable cause is that the "success mode" incurs a significantly smaller loss when using Eq. 4.1 than when using Eq. 4.9, so the network can avoid getting stuck in the failure modes when using the former equation. The typical "success mode" for long event occurrences is like Fig. 4.5 (c): the frame-level probability stays at a high value close to 1 (*e.g.* 0.9) for a long time. In such stable regions, Eq. 4.1 derives almost zero probabilities for the boundary labels. On the other hand, Eq. 4.9 derives a probability of about $0.9 \times 0.1 = 0.09$ for all boundary labels. This can lead the system to believe that something is likely to be emitted in the long stable regions, which lowers the probability of the ground truth label sequence (which is empty in these regions) and increases the loss.

The lesson to be learned from the failure of the alternative CTL algorithm is: it may not be always good to attach probabilistic interpretations to numbers, because it often requires making unrealistic assumptions of independence. Actually, we have already learned this lesson once with the noisy-or pooling function in Sec. 3.1.

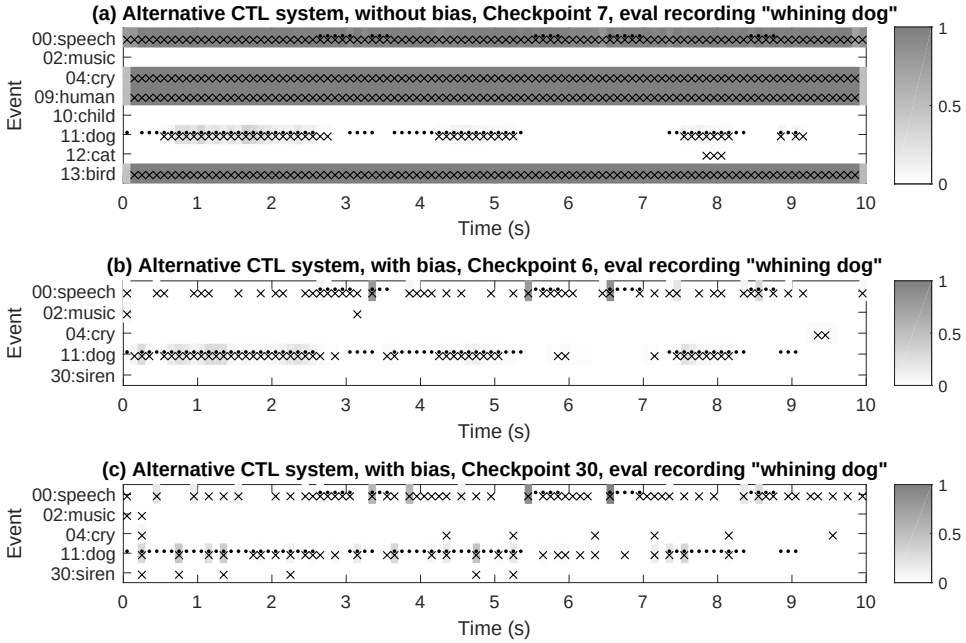


Figure 4.7: The frame-level predictions of the system using the alternative CTL algorithm on the evaluation recording `0F04c_rY4aw` ("whining dog"), captured at various checkpoints during training. Graphs (a) and (c) illustrate two failure modes.

4.4 Combining CTL with MIL

Combining different systems for the same task often brings improvements at no cost. When sequential labeling is available for training a SED system, presence/absence labeling is automatically available. Therefore, it is a natural idea to combine a CTL system with a MIL system to see if it can improve the performance of the former.

Studying the combination of CTL and MIL systems has another potential use. Because sequential labeling is harder to collect than presence/absence labeling, we may have more data with presence/absence labeling available than data with sequential labeling. In this scenario, a combined system can make use of both types of data at the same time.

Since the CTL system and the MIL system have an identical structure up to the fully connect layer that predicts frame-level probabilities of events, the system combination boils down to assigning proper mixing weights to the CTL and MIL loss functions. In our previous experiments in this chapter, we found that the loss of MIL systems usually stabilized around 0.02, while the loss of CTL systems usually stabilized around 0.2. For the experiments in this section, we fixed the weight of the CTL loss to 1, and tried out the following weights for the MIL loss: 30 (emphasizing the MIL loss more), 10

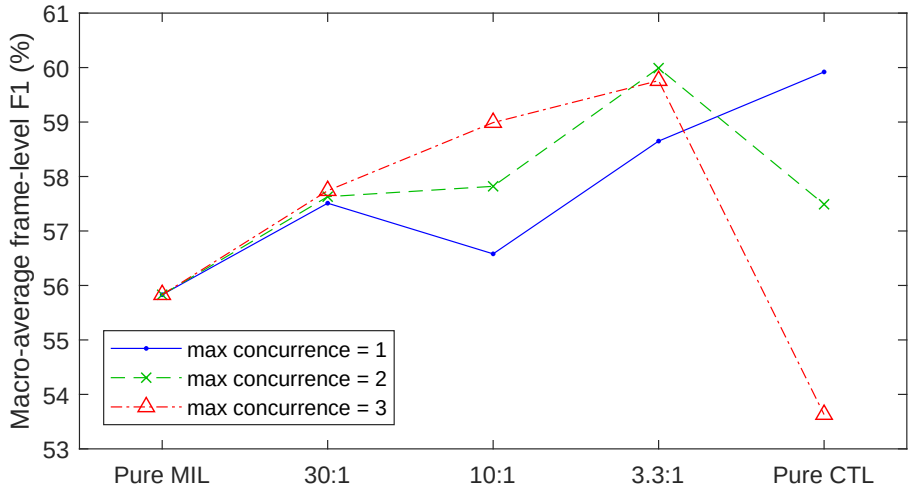


Figure 4.8: The effect of combining a CTL system with a MIL system using different mixing weights.

(weighting both losses equally), and 3.3 (emphasizing the CTL loss more). We also tried out different values of the max concurrence: 1, 2 and 3.

Fig. 4.8 shows the macro-average frame-level F_1 obtained with different mixing weights of the loss functions and different values of max concurrence. Due to the variance inherent in the experiments, the trend of how the F_1 changes with the mixing weights is not totally consistent for different values of max concurrence. Nevertheless, a mixing weight of 3.3:1 is generally a good choice, and gives a marginal improvement on top of pure CTL.

4.5 Discussion: Generalization to New Data

When we selected the training data for this chapter in Sec. 4.1, we excluded the Noiseme corpus because it only contained 13 hours of audio. Nevertheless, the Noiseme corpus has the advantage that it is strongly labeled by hand. In this section, we discuss its use as a test corpus.

We generated the ground truth for the Noiseme corpus by mapping the original strong labels to the 35 common events according to Table 4.1. To test out a network's performance on the Noiseme corpus, we applied the class-specific thresholds tuned on the validation data of Audio Set, and measured the macro-average frame-level F_1.

Unfortunately, the above procedure yielded miserable F_1 numbers, usually around 10%. We found an important reason to be that the class-specific thresholds tuned on Audio Set were inappropriate for the Noiseme corpus. To solve this problem, we performed 2-fold cross validation for the class-specific thresholds on the Noiseme corpus: we divided the corpus into two equal parts, tuned the thresholds on the two parts separately, and

System	Audio Set F_1 (%)	Noiseme F_1 (%)
Baselines		
Strong labeling	67.38	21.75
MIL	55.83	20.44
CTL		
Max concurrence = 1	59.92	20.66
Max concurrence = 2	57.49	21.30
Max concurrence = 3	53.63	20.12
CTL + MIL (weight 1:3.3)		
Max concurrence = 1	58.65	20.02
Max concurrence = 2	59.99	21.00
Max concurrence = 3	59.76	20.90

Table 4.4: The macro-average frame-level F_1 of some systems Chapter 4, measured on both the Audio Set evaluation data and the Noiseme corpus.

applied the thresholds tuned on one part to the other. This brought the F_1 numbers up to around 20%; this was also close to the oracle F_1 which would be obtained if the class-specific thresholds were tuned on the entire Noiseme corpus directly.

In Table 4.4, we list the F_1's of many systems trained in this chapter, measured on the evaluation set of Audio Set as well as on the Noiseme corpus. We observe that the F_1's measured on the Noiseme corpus are much lower than those measured on Audio Set. In addition, the F_1's measured on the Noiseme corpus also exhibit a much smaller variance, and they do not follow the same trend as the F_1's measured on Audio Set. In this case, it is hard to say whether the variation is due to the different performance of the systems or due to random fluctuation. It is because we could not make any conclusive judgments from the F_1's measured on the Noiseme corpus that we chose to use Audio Set for evaluation.

The huge gap between the F_1's measured on the two corpora indicates that the systems trained in this chapter are probably overfitting to Audio Set. Indeed, when we inspect the predictions on the noiseme data, we find that the frame-level probabilities of most events (except `speech` and `music`) are near zero. This means the systems fail to recognize the events in the Noiseme corpus.

Such overfitting may arise from several factors. First, Audio Set and the Noiseme corpus were annotated by different groups of people, who might have had different notions about what each event sounded like. Second, the strong labels of Audio Set were automatically generated by TALNet, while those of the Noiseme corpus were produced by hand. As a result, the ground truth of the Audio Set evaluation data would share some biases and peculiarities with the Audio Set training data, while the ground truth of

the Noiseme corpus would not. For example, spurious short occurrences of events would be common throughout the Audio Set because we chose to not perform any smoothing on the labels; such occurrences should be rare in the Noiseme corpus.

4.6 Summary

In this chapter, we have studied how to perform SED with sequential labeling, *i.e.* using sequences of event boundaries as the supervision. The technique of connectionist temporal classification (CTC) is well suited for this type of supervision. However, when used to predict event boundaries directly, CTC suffers from a "peak clustering" problem. We modified the CTC framework and proposed a connectionist temporal localization (CTL) framework to overcome this difficulty. The key features of the CTL framework include: (1) Instead of predicting frame-level probabilities of event boundaries, the network explicitly predicts the frame-level probabilities of the events themselves. The boundary probabilities are then derived from the event probabilities using a "rectified delta" operation. (2) The event probabilities at the same frame are treated as mutually independent instead of mutually exclusive. This gets rid of the blank label in the forward algorithm, and allows the emission of multiple labels at the same time.

We have also proposed a method for combining a CTL system with a multiple instance learning (MIL) system. Because the CTL system explicitly predicts the frame-level probabilities of events, combining it with a MIL system becomes as simple as a weighted average of the two loss functions. Such a combination can potentially be used when some part of the training data has sequential labeling, but the other part the data only has presence/absence labeling. A three-way combination of a strong labeling system, a CTL system and a MIL system may also be considered if different parts of the training data come with strong labeling, sequential labeling and presence/absence labeling.

Because there is no corpus available with manually generated sequential labeling, we generated strong labels and sequential labels automatically for Audio Set, and used them to evaluate our systems. Using this synthetic dataset, we have shown that our CTL framework closes one third of the performance gap between a MIL system using presence/absence labeling and a strong labeling system. The combination with a MIL system gives a marginal further improvement.

Nevertheless, the performance of our systems does not generalize well to unseen data. This overfitting remains a problem to solve in the future.

Chapter 5

Transfer Learning for Sound Event Detection

Sound event detection has long been troubled by the problem of insufficient training data. In such situations, transfer learning can help by borrowing knowledge from a related task with a larger training corpus. Among the many techniques of transfer learning, *feature transfer* is a simple and popular method. It refers to the action of taking a model trained for a related task as a feature extractor for the target task. In this chapter, we study how feature transfer can can facilitate the learning of sound events. We would like to answer the following questions: Does feature transfer help? When does it help? And what types of transfer learning features help?

We consider two networks trained for different tasks as feature extractors for sound event detection. The first network, called the VGGish network [71], is trained to classify audio recordings among more than three thousand concepts not restricted to sound events. The second network, called SoundNet [72], is trained to predict the object and scene in the video stream given the audio stream. While we use the first network as is without any modification, we retrain the second network ourselves in order to study the effect of network structure and the layer from which to extract features. We also considered the "Look, Listen and Learn" network [74], which is trained to tell whether an audio segment and a video segment match with each other, but gave it up because we could not find an open-source implementation.

We apply the features extracted from both networks to all the three SED experiments we have done in Chapters 3 and 4. Two of these experiments use presence/absence labeling; they are conducted on the relatively small corpus of the DCASE 2017 challenge and the large-scale Google Audio Set, respectively. The goal is to perform audio tagging and localization well at the same time. The third experiment uses sequential labeling, and the goal is to show that the extra timing information provided by the sequential

labeling is helpful for localization. We study whether the transfer learning features achieve these goals, and whether they achieve them better than the simple filterbank features.

The content of this chapter is organized as follows. In Sec. 5.1, we describe the VGGish network and SoundNet. We especially focus on how we retrained the latter, and inspect the features extracted from the various layers of it. In Sec. 5.2, we present our experiments with transfer learning features, and answer the questions of when transfer learning helps and which types of features help. Finally, Sec. 5.3 concludes the chapter.

5.1 The Feature Extractors for Transfer Learning

5.1.1 The VGGish Network

The first network we used as a feature extractor for transfer learning is called the VGGish network[1] [71]. It is a pre-trained network that can be used off the shelf, and does not require any finetuning. Features extracted by this model have also been used in [54] and [55].

The VGGish network was trained for audio classification on a preliminary version of the YouTube-8M [132] corpus. YouTube-8M is a large corpus containing 6.1 million video recordings totaling 350,000 hours (40 years), which is 60 times as large as Audio Set. The recordings are labeled with 3,862 Knowledge Graph[2] entities, which are not restricted to sound events.

The structure of the VGGish network, shown in Fig. 5.1, is a variant of the VGG network [115] for image recognition. The network works on audio segments of 960 ms. The input to the network is filterbank features of the audio segment, with a size of 96 frames and 64 frequency bins. The filterbank features are consequently processed by 6 convolutional layers, 4 max pooling layers and 3 fully connected layers. The last of the 3 fully connected layer is a bottleneck layer, which generate a 128-dimensional embedding for each audio segment. The embedding is then fed into a final fully connected layer with the softmax activation, in order to classify the audio segment among the 3,862 Knowledge Graph entities of YouTube-8M.

We use the 128-dimensional embeddings as a type of acoustic features for transfer learning, and we call them "VGGish features". The download page of Audio Set[3] provides such embeddings extracted at 1 Hz (*i.e.* one feature vector per second). Since we want to perform SED at a time resolution of 0.1 s, we re-extracted the embeddings ourselves from 960 ms windows

[1] A TensorFlow [93] implementation of the VGGish network is available online at https://github.com/tensorflow/models/tree/master/research/audioset.

[2] Knowledge Graph is a knowledge base used by Google; see https://www.google.com/intl/es419/insidesearch/features/search/knowledge.html for more information.

[3] https://research.google.com/audioset/download.html

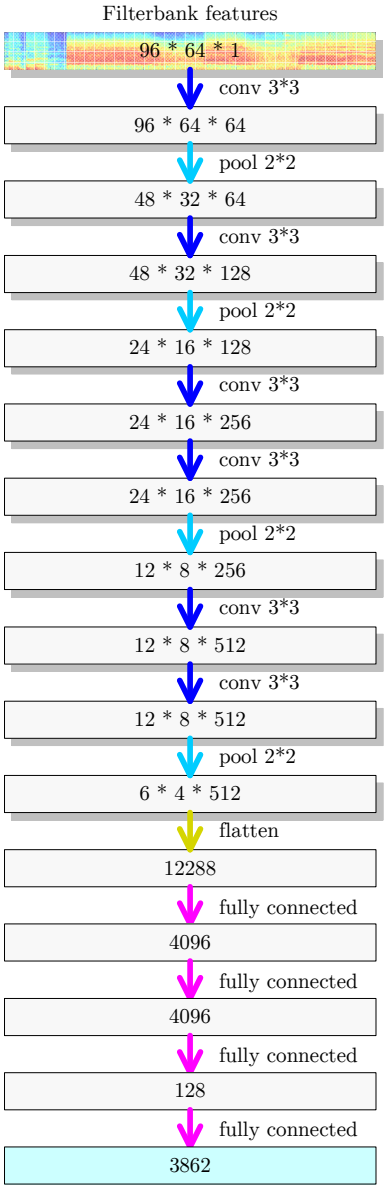


Figure 5.1: The structure of the VGGish network, used as a feature extractor for transfer learning. Shadowed boxes stand for 3-D tensors; their sizes are specified in the order of time frames, frequency bins, and feature maps. Plain boxes stand for 1-D tensors. All convolutional layers and fully connected layers use the ReLU activation, except the final fully connected layer which uses the softmax activation; all pooling layers perform max pooling. The "$m*n$" at each layer specify the size of the kernel on the time and frequency dimensions. The convolutional layers do not perform subsampling; the pooling layers have a subsampling stride equal to the kernel size.

shifting 0.1 s at a time. For each recording of 10 seconds, this produced 100 feature vectors of 128 dimensions.

5.1.2 SoundNet and Its Variants

The Structures of SoundNet and Its Variants

The second network we used as a feature extractor for transfer learning is SoundNet [72]. The overall architecture of SoundNet is shown on the left side of Fig. 5.2. 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 [115]. 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. The features extracted with SoundNet have outperformed other features and models by a significant margin in the acoustic scene classification task of the DCASE 2013 challenge and a sound event classification task on the ESC-50 corpus [72].

Detailed information about the layers of SoundNet is shown on the right side of Fig. 5.2 (b). The input is a 20-second, monaural waveform with a sampling rate of 22,050 Hz. The network has eight convolutional layers interspersed with five 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, which is also convolutional, has 1,401 output units. These units are 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 for 20-second recordings, 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 a reasonable temporal resolution, we need to extract features at a sufficient frame rate. We have done all our SED experiments at a frame rate of 10 Hz. Only one layer ("conv5") in SoundNet has a frame rate close to this value. As a result, if we were to extract features from SoundNet, we would be forced to extract the features from this layer. However, we are curious whether the higher layers would produce better features for SED, since they are closer to the final classification layer, and may encode information that is more directly related to concrete concepts such as objects, scenes, or sound events. In order to be able to extract features from higher layers, we trained

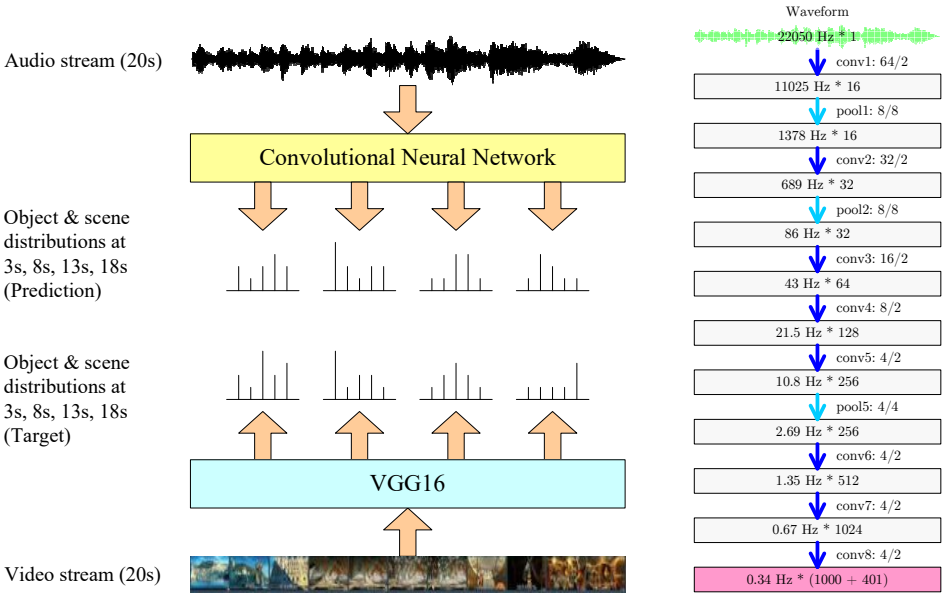


Figure 5.2: The structure of the original SoundNet. The left side shows the overall architecture; the convolutional neural network (CNN) is trained to minimize the KL divergence between the predicted and target distributions. The right side lists the layers of the CNN. All tensors are 2-D; their sizes are specified as "frame rate * number of feature maps". All convolutional layers use the ReLU activation, except the final layer which uses the softmax activation; all pooling layers perform max pooling. The "m/n" at each layer specify the size of the kernel (m) and the subsampling stride (n).

four variants of SoundNet, in which the frame rate stays constant once it has been reduced to 10 Hz.

The structures of the variants of SoundNet are shown in Fig. 5.3. The sampling rate of the input waveform is changed to 16,000 Hz to match what we have done with the Google Audio Set and the Noiseme corpus. All of the variants share the layers up to "pool5", which reduce the frame rate to exactly 10 Hz. The output of the "pool5" layer has a dimensionality of 128, which is chosen to match the dimensionality of the VGGish features. The four variants use different types of layers for the next three layers, and they are named after the types:

- SN-F: Three fully connected layers;
- SN-C: Three convolutional layers;
- SN-CR: Two convolutional layers followed by one bidirectional GRU layer;
- SN-R: Three bidirectional GRU layers.

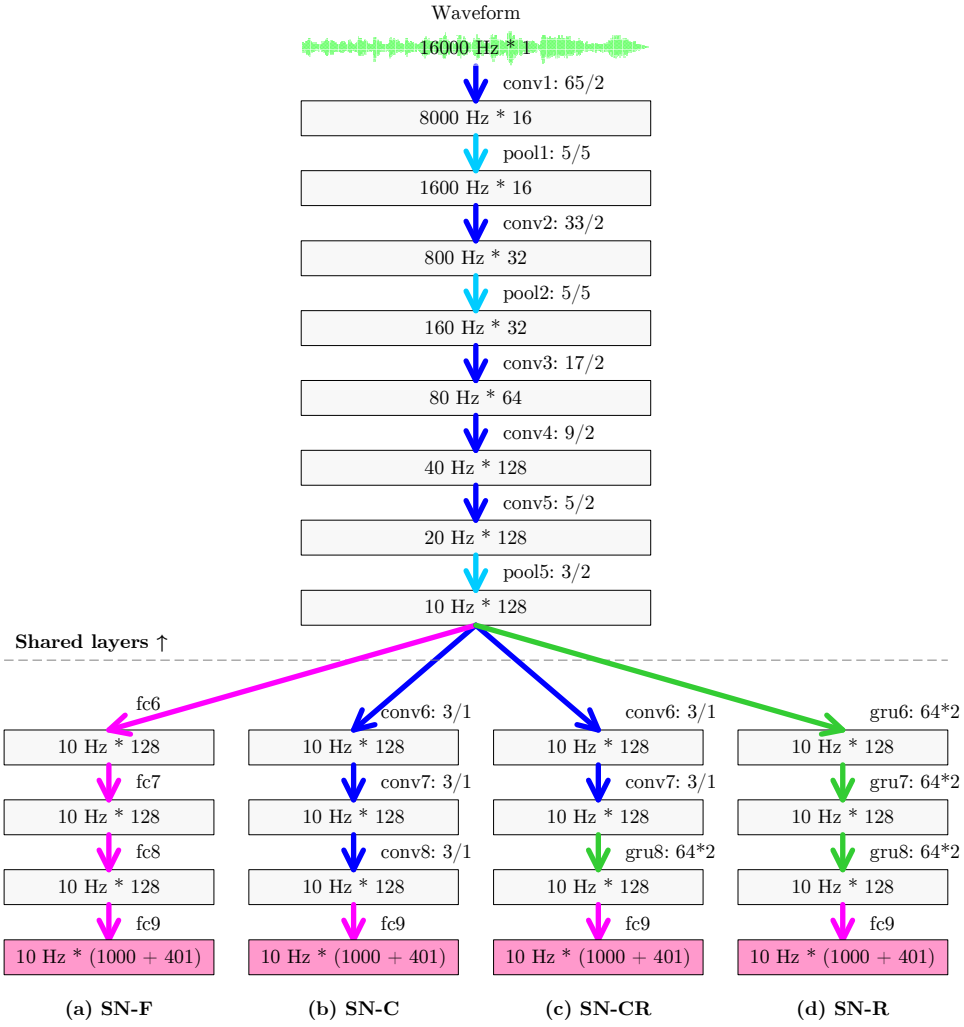


Figure 5.3: The structures of the four variants of SoundNet. The three layers after the "pool5" layer are different, and the variants are named after the types of these layers. Batch normalization is applied to all the convolutional layers before the ReLU activation.

These three layers do not change the frame rate, and they keep the dimensionality of the embedding at 128. Finally, a fully connected layer with 1,401 neurons predicts the probabilities of 1,000 objects and 401 scenes, just as in the original SoundNet. All the fully connected layers except the last one and all the convolutional layers use the ReLU activation function.

We will extract features after the "pool5" layer and the next three layers from the four variants. The features will be named with a pattern of "SN-CR-6": the prefix specifies which variant the features come from; the suffix specifies the layer after which they are extracted. All the features

are collectively referred to as "SoundNet features", in contrast to VGGish features and filterbank features.

Training SoundNet and Its Variants

The training corpus of SoundNet consists of 2 million videos, accompanied by a validation set of 147 thousand videos. This is approximately equal in size to the Google Audio Set. Since the network is intended to be used as a feature extractor instead of a classifier, there is no evaluation data. The videos come from either the YFCC100M[4] corpus [133] 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 [115], contains the distributions of 1,000 objects and 401 visual scenes at keyframes. From the filenames, we infer that the 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[5].

The original SoundNet was trained using the Torch [94] toolkit. It was trained to minimize the KL divergence between the predicted object and scene distributions and the ground truth on the validation data (the object KL divergence is added to the scene KL divergence, and averaged over the keyframes). Even though the interval between the keyframes do not agree with the frame rate of the output layer of SoundNet, the four keyframes were assigned as the ground truth of the four time steps of the output layer anyway. The optimizer was Adam with a fixed learning rate of 10^{-3} 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 to all convolutional layers before the ReLU activation, except the last layer.

We trained the four variants of SoundNet using the PyTorch [95] toolkit. We used the same training set, but to speed up the validation, we randomly selected 1,000 videos from the validation data. The ground truth distributions were compared against the predicted distributions at exactly 3 s, 8 s, 13 s and 18 s to compute the KL divergence. To start with, we also used the Adam optimizer with a fixed learning rate of 10^{-3} (but without momentum) and a batch size of 64 videos. 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). Batch normalization was applied to all convolutional layers before the ReLU activation. We tried applying batch normalization to the other types of layers or removing the

[4] "YFCC" stands for "Yahoo Flickr Creative Commons".

[5] https://projects.csail.mit.edu/soundnet/

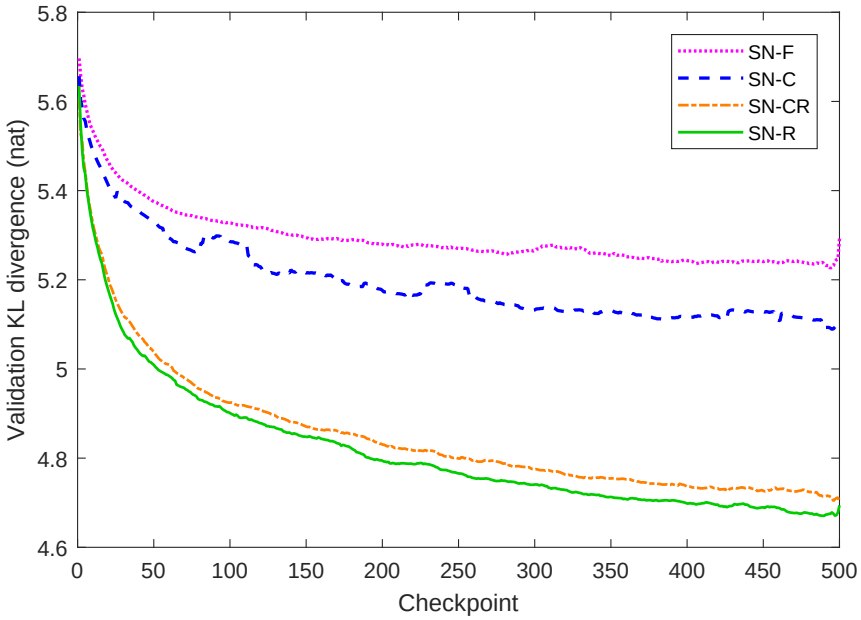


Figure 5.4: Training curves of the four variants of SoundNet, showing how the validation KL divergence decrease over time. The curves are smoothed with a moving average filter of length 31. The final rise in the curves is an artifact of this smoothing, and should be ignored.

batch normalization from convolutional layers, but found that these did not have a significant effect on the performance.

The evolution of the validation KL divergence up to 500 checkpoints (about 3 epochs) while training the four variants of SoundNet is depicted in Fig. 5.4. The variants SN-R and SN-CR exhibit a clear advantage over the other two variants. For reference, if we evaluated the original SoundNet on the 1,000-video validation set, we would get an average KL divergence of 5.13 nats[6], which is on par with our purely convolutional variant, SN-C. We may claim that we have trained two variants of SoundNet (SN-R and SN-CR) that perform object and scene classification significantly better than original, and we no longer use the other two variants (SN-C and SN-F) hereafter.

We tried tuning the hyperparameters of training for better performance. We found it marginally beneficial to increase the batch size to 128 videos. The checkpoint size was kept at 160 minibatches, which would now be about 1% of an epoch. We also found it helpful to decrease the learning rate in an exponential fashion: multiplying it by 0.998 every checkpoint. We let the two variants (SN-R and SN-CR) train for 48 hours, at the end of which the learning rate became so small that the training had converged. SN-R reached

[6]This number was measured after excluding about 2% of the keyframes, because on these frames SoundNet predicted zero probabilities for some object or scene classes.

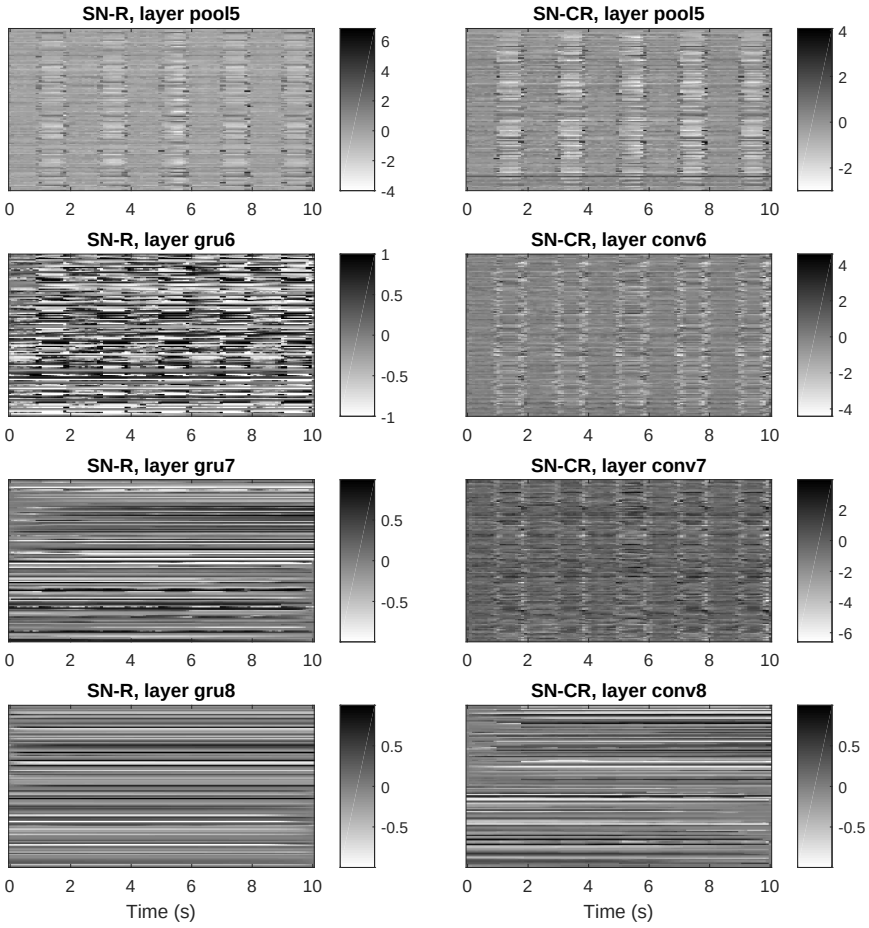


Figure 5.5: The features extracted from various layers of two variants of SoundNet, SN-R and SN-CR, for the Audio Set evaluation recording `-2EKWgTNEYU` ("buzzer").

a lowest KL divergence of 4.467 nats at Checkpoint 1,657 (16.57 epochs); SN-CR reached a lowest KL divergence of 4.523 nats at Checkpoint 2,133 (21.33 epochs).

Inspecting the SoundNet Features

Before applying the SoundNet features to experiments, let's first inspect them to see if there are any obvious problems. We ran the variants SN-R and SN-CR on the "buzzer" recording we used in Chapter 4, and plot the features in Fig. 5.5. The recording contains intermittent buzzer tones. The boundaries of the buzzer tones are clearly visible in the features extracted after the "pool5" and "gru6" layers of SN-R and the "pool5", "conv6" and "conv7" layers layers of SN-CR, but they are smeared out in the features

extracted from higher layers. The "smeared out" features may not have the necessary temporal resolution for SED.

We speculate that the smearing out effect is due to the inadequate quality of the supervision. First, the objects and scenes may not change frequently in the videos. It is not rare for a 20-second video to contain only one shot. Second, the supervision may be imposed at places a few frames off from the correct places. Influence by these two factors, the recurrent layers in the networks will tend to "remember" their predictions at one frame and reproduce the same predictions at nearby frames, because doing so maximizes the chance of "guessing" correctly. Finally, the supervision is so sparse in time that it does not emphasize the transition between different objects and scenes, so the recurrent layers do not adequately learn when to forget, either.

5.2 Transfer Learning Experiments

In this section, we repeat the experiments in Chapters 3 and 4 with the VGGish features and the various SoundNet features. The first two experiments are joint audio tagging and localization with presence/absence labeling, carried out on the DCASE challenge and the Google Audio Set, respectively; they correspond to Secs. 3.2 and 3.3. The last experiment is localization with sequential labeling, and corresponds to Chapter 4.3.

5.2.1 Transfer Learning for the DCASE Challenge

In this subsection, we repeat our experiments in Sec. 3.2 with transfer learning features. The task of the experiments is to perform audio tagging (Task A) and localization (Task B) simultaneously on the corpus of the DCASE 2017 challenge, which is a small (140 hours) subset of Audio Set and involves only 17 types of vehicle and warning sounds. The purpose of the experiments in Chapter 3 was to compare different pooling functions for multiple instance learning. Since the conclusion was clear that the linear softmax pooling function was the best, we only use this one pooling function in this subsection, and turn our focus to the comparison of different transfer learning features among themselves and against the filterbank features. The evaluation metrics include the F_1 for Task A, plus the error rate and F_1 calculated on 1-second segments for Task B. All the metrics are micro-averaged across the event types.

The structure of the network we built for this task is shown in Fig. 5.6. This network has several notable differences from the networks in Chapter 3. First, it is a rather small network, because the training corpus is small. Second, the intermediate layers are 2-D instead of 1-D. This is because the 128 dimensions of the transfer learning features do not form a frequency axis as the filterbank features do; in other words, a transfer learning feature

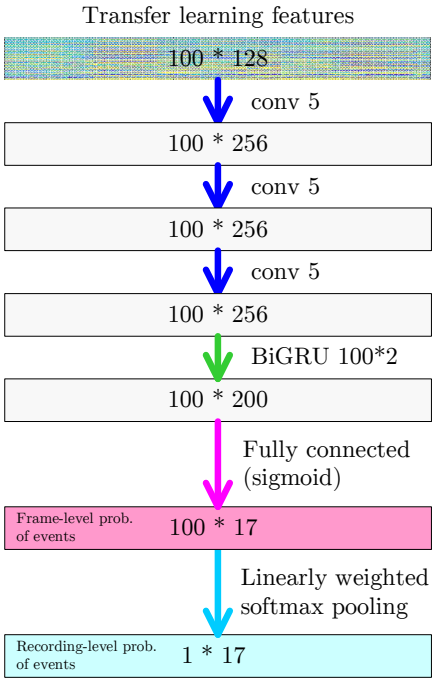


Figure 5.6: The structure of the system used in Sec. 5.2.1. "conv 5" specifies the kernel size. Batch normalization is not applied.

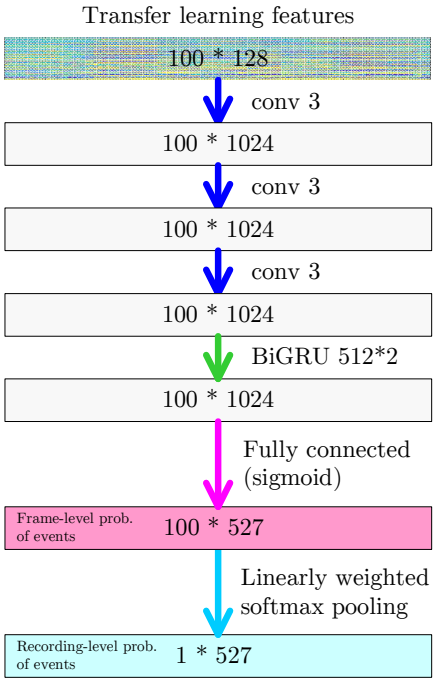


Figure 5.7: The structure of the system used in Sec. 5.2.2. "conv 3" specifies the kernel size. Batch normalization is applied to all the convolutional layers before the ReLU activation.

sequence should be treated as 128 feature maps of size 100×1, instead of 1 feature map of size 100×128. Finally, there are also no max pooling layers, because there is no frequency axis to reduce, and the features were extracted at the target time resolution from the beginning.

The network was trained in a very similar way to Sec. 3.2.3. We found it necessary to apply dropout to prevent overfitting. We applied dropout before each convolutional layer and on both sides of the GRU layer, and tuned the dropout probability. We also tuned the factor of learning rate decay: we multiplied the learning rate by this factor whenever the validation loss did not see any updates for 3 consecutive epochs. All other hyperparameters were kept the same as in the middle column of Table 3.3.

We evaluated all the systems with different transfer learning features at Epoch 35, when they have all converged. For each system, class-specific thresholds were tuned on the validation set of the DCASE challenge for Task A, then applied directly to Task B. The performance numbers, together with the optimal dropout probability and learning rate decay factor, are listed in Table 5.1.

Feature	Dropout prob.	LR decay factor	Task A	Task B	
			F1	ER	F1
Filterbank	0.0	0.5	49.5	84.3	43.7
VGGish	0.3	0.1	56.4	73.5	50.4
SN-R-5	0.3	0.1	50.2	80.5	44.6
SN-R-6	0.2	0.1	50.2	80.9	43.0
SN-R-7	0.0	0.1	44.4	88.5	38.2
SN-R-8	0.1	0.3	39.1	98.0	33.9
SN-CR-5	0.3	0.1	51.3	81.4	45.2
SN-CR-6	0.3	0.1	53.8	81.4	45.3
SN-CR-7	0.5	0.3	51.2	78.1	45.7
SN-CR-8	0.1	0.1	45.2	91.9	37.6

Table 5.1: The optimal hyperparameters and performance of the system in Sec. 5.2.1 with different types of features.

This experiment is a typical successful example of transfer learning. Except for the features that smear out in time (SN-R-7, SN-R-8, and SN-CR-8), all transfer learning features outperform the baseline filterbank features, in terms of both audio tagging and localization. Considering that SoundNet is trained with 1 year worth of audio while the DCASE corpus has only about 140 hours of audio for training, it is no wonder that the SoundNet features can provide useful knowledge that cannot be mined from the DCASE training data alone. The best performance is achieved by the VGGish features; this is reasonable since the VGGish network is trained with 40 years of audio. These results indicate that transfer learning is helpful when the amount of data used to train the source model is significantly larger than the amount of data available for training the target model, and the more training data, the better.

Among the SoundNet features, the SN-CR features perform better than the SN-R features, even though the SN-R network reaches a slightly lower KL divergence. This indicates that better performance on the source task does not directly translate to better performance on the target task. The difference between features extracted from different layers is rather small. The SN-CR-6 features achieve a slightly higher F_1 for audio tagging; the SN-CR-7 features achieve a slightly lower error rate for localization.

5.2.2 Transfer Learning for Large-Scale Joint Audio Tagging and Localization

In this subsection, we repeat our experiments in Sec. 3.3 with transfer learning features. The main task is to perform audio tagging on the entire Audio Set, but at the same time the system must also perform audio

Feature	Hyperparameters		Audio Set Metrics			DCASE Metrics		
	Dropout prob.	LR decay factor	MAP	MAUC	d'	Task A	Task B	
						F1	ER	F1
Filterbank	0.0	0.8	0.359	0.966	2.575	52.3	78.9	45.4
VGGish	0.3	0.8	0.340	0.962	2.510	54.0	81.1	48.6
SN-R-5	0.3	0.9	0.289	0.950	2.323	47.1	92.2	39.3
SN-CR-5	0.3	0.9	0.300	0.952	2.349	50.5	90.1	42.7
SN-CR-6	0.2	1.0	0.289	0.952	2.351	47.6	93.5	41.7
SN-CR-7	0.3	0.8	0.276	0.949	2.311	47.6	93.4	39.4

Table 5.2: The optimal hyperparameters and performance of the system in Sec. 5.2.2 with different types of features. The first row is the performance of the TALNet in Sec. 3.3.

tagging and localization on the DCASE 2017 challenge data reasonably well. Besides the metrics of the DCASE challenge, the evaluation metrics of this experiment also include the mean average precision (MAP), mean area under the curve (MAUC), and d-prime on Audio Set. The metrics evaluate how well a system ranks the test recordings for each event, and therefore do not require thresholding. Please refer to Sec. 3.3.1 for their definitions. Just like the previous subsection, we only use the linear softmax pooling function. Because Audio Set is large and the features can take up a huge amount of storage, we only try out the following types of SoundNet features in this subsection: SN-R-5, SN-CR-5, SN-CR-6, SN-CR-7.

The structure of the network we built for this task is shown in Fig. 5.7. This network is significantly shallower than TALNet, because the transfer learning features are already a result of several layers of neural processing. The network was trained in a very similar way to Sec. 3.3. We increased the batch size from 250 recordings to 500 recordings, and tuned the dropout probability and learning rate decay factor. All other hyperparameters were kept the same as in the last column of Table 3.3.

Table 5.2 lists the evaluation results. While the Audio Set evaluation metrics do not require thresholding, the DCASE metrics still do, so we tuned class-specific thresholds on the DCASE validation set for Task A, and then applied them directly to Task B. For each system evaluated, we picked a checkpoint at which all the metrics were close to the best value ever reached.

This time we no longer see the VGGish features leaving the baseline filterbank features far behind. While the VGGish features achieve a higher audio tagging F_1 and localization F_1 on the DCASE data, it fails to beat the TALNet on the other metrics. A possible reason is that the training data for TALNet (8 months) is already large enough, so it no longer needs information transferred from another task. Another possible reason is that while the DCASE challenge only involves 17 types of sound events, Audio Set is concerned with 527 types, which may not be well covered by the

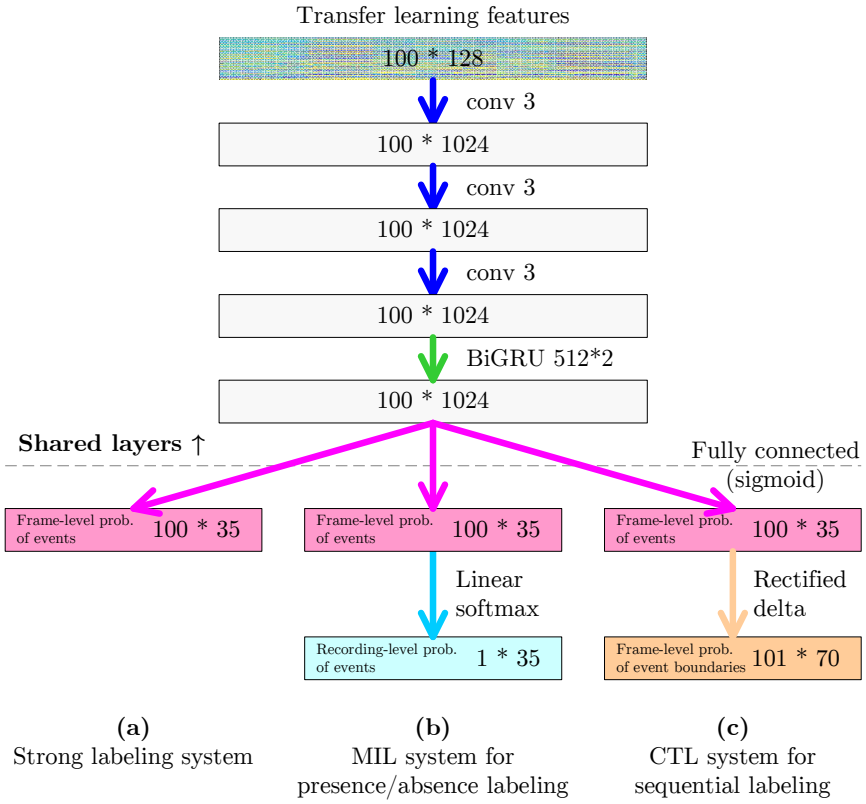


Figure 5.8: The structures of the strong labeling baseline system, the multiple instance learning (MIL) baseline system, and the CTL system used in Sec. 5.2.3. The layers above the dashed line are shared across the three systems. All tensors are 2-D; their sized are specified by the number of frames and the number of feature maps. All the convolutional layers use the ReLU activation; the number after "conv" (*e.g.* 3) specifies the size of the kernel. Batch normalization is applied to all the convolutional layers before the ReLU activation.

Knowledge Graph entities the VGGish network is trained to recognize.

All SoundNet features fall behind the baseline. This may be due to the fact that SoundNet is only trained with 1 year of audio, which is not much larger than the size of Audio Set (8 months). Among the SoundNet features, SN-CR-5 yields the best performance. This is in contradiction to our hypothesis that higher layers should produce better features.

5.2.3 Transfer Learning for Sequential Labeling

In this subsection, we repeat our main experiments in Chapter 4 with transfer learning features. The task of the experiments is to perform SED with sequential labeling on a subset of Audio Set, which accounts for about

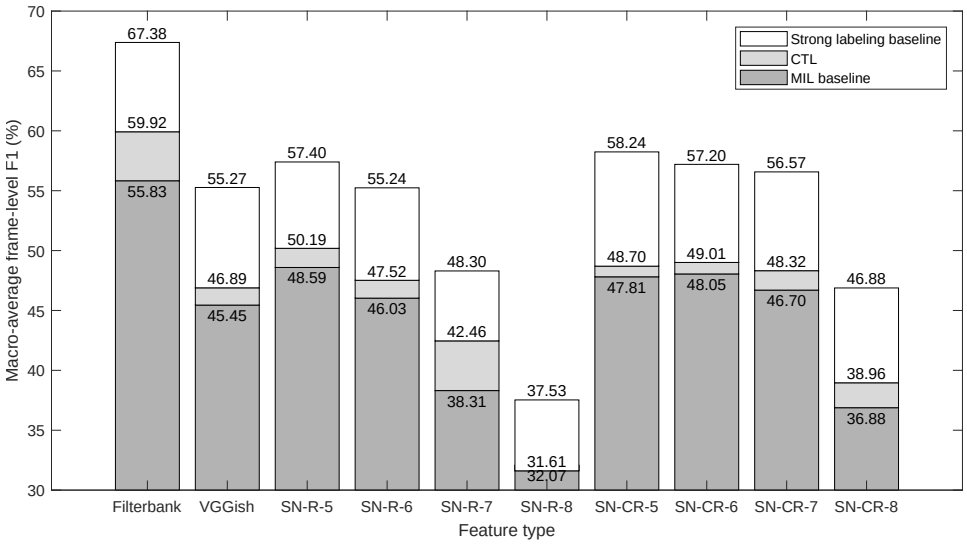


Figure 5.9: The macro-average frame-level F_1 obtained by the CTL system, compared with the strong labeling and MIL baselines, using various types of transfer learning features as well as the filterbank features.

18% the size of the entire corpus and involves 35 "common events". We build neural networks that take various transfer learning features as input, using the CTL framework which has proven successful in Chapter 4. To evaluate the performance in context, we also build a strong labeling baseline system and a multiple instance learning (MIL) baseline system for each feature type. The performance is measured by the frame-level F_1 macro-averaged across all the event types.

The structures of the systems we built are shown in Fig. 5.8. Again, the networks are shallow compared to those used in Chapter 4, because the transfer learning features already encode several layers of neural processing. The networks were trained in almost the same way as in Chapter 4, except that we found it sometimes beneficial to apply dropout. For the strong labeling baseline, we found 0.2 to be a good value for the dropout probability. For the MIL baseline and the CTL system, we found it was best to apply dropout with a probability of 0.1 or not to apply dropout at all; we chose the better of the two for each feature type. For the CTL system, the max concurrence was set to 1. The macro-average frame-level F_1's obtained by the three systems using different features are illustrated in Fig. 5.9.

The corpus for sequential learning falls between the DCASE data and the entire Audio Set, in terms of both the total duration of the audio and the number of sound event types. The outcome of the current experiment is critical, because it would provide a lot of information to answer this question: How much training data do we need so that transfer learning

no longer helps? From Fig. 5.9, we see that none of the transfer learning features surpass the performance of filterbank features. However, we are not confident to conclude that the subset of Audio Set used in this experiment is already large enough so transfer learning is no longer necessary. Remember that the labels of this data are generated by TALNet, which is trained using filterbank features, so the labels will no doubt contain idiosyncracies that are easier to approximate with filterbank features. We suspect that the good performance of filterbank features may be due to overfitting.

Among the transfer learning features, all feature types do a reasonable job except those that smear out in time (SN-R-7, SN-R-8, and SN-CR-8). A surprising fact is that SoundNet features now perform better than VGGish features, even though the latter were trained with way more data. Among the SoundNet features, SN-R-5 achieves the highest performance, while the SN-CR features extracted from layers 5, 6 and 7 are on par with each other. These results are not in agreement with what we have observed in the previous experiments. They may imply that the effect of transfer learning needs to be investigated on a case-by-case basis.

With all the feature types but SN-R-8, we see the performance of the CTL network surpass that of the MIL baseline, even though it does not close as much as one third of the gap between the two baselines. This confirms that the CTL framework is able to leverage the extra temporal information in the sequential labeling.

5.3 Summary

In this chapter, we have explored how to improve the learning of sound events by feature transfer. We used two networks as feature extractors: VGGish and SoundNet. We used the VGGish network off the shelf, while we trained several variants of SoundNet ourselves. The variants SN-R and SN-CR outperformed the original SoundNet significantly in terms of KL divergence.

The SED experiments using transfer learning features showed mixed results. A most brief summarization of the comparison between different types of features is given below (only the single best type of SoundNet features is listed):

- Experiment 1 (DCASE):
  VGGish > SN-CR-6/7 > filterbank;
- Experiment 2 (Audio Set):
  Filterbank > VGGish > SN-CR-5;
- Experiment 3 (sequential labeling):
  Filterbank > SN-R-5 > VGGish.

The results indicate that there are certain conditions for transfer learning to succeed. The most import condition is the amount of training data for the source task and the target task. When the target task has a small training corpus (*e.g.* Exp. 1), transfer learning can offer a big improvement. On the other hand, when the target task already has plenty of training data (*e.g.* Exp. 2), it is best to let the network find out on itself the most useful information in the raw input. Generally speaking, the source task has to have about 50 times as much training data as the target task to make a significant difference.

For SoundNet, we hypothesized that features extracted from higher layers would perform better because they would encode concepts more closely related to the classification targets. However, we did not observe this to be true. An important reason is the "smearing out" effect that recurrent layers had on the features, and we suspect this is due to the inadequate form of supervision used during training.

Chapter 6

Conclusion

6.1 Contributions of This Thesis

This thesis has studied the task of sound event detection when there is only weak labeling available. Weak labeling may come in the form of either presence/absence labeling or sequential labeling, and we have pushed the frontier of the research in both cases. We have also investigated transfer learning as a means to improve the performance when training data is insufficient.

Below is a summary of the contributions made by this thesis:

- We have made a theoretical and empirical comparison of six common pooling functions: max pooling, noisy-or pooling, average pooling, linear softmax pooling, exponential softmax pooling, and attention pooling. We establish the linear softmax pooling function as the best of the six, because it has the following advantages: it facilitates the gradient flow, achieves a good balance between false negatives and false positives, and makes consistent recording-level and frame-level predictions. (Secs. 3.1 and 3.2)

- Using the linear softmax pooling function, we have built a network called "TALNet" that can perform audio tagging and localization simultaneously. This system closely matches the state-of-the-art performance on the Google Audio Set [15], while reaching a strong performance on the DCASE 2017 challenge data [83] without any adaptation. As far as we know, this is the first system to exhibit such good performance on both corpora. (Sec. 3.3)

- We have modified the standard connectionist temporal classification (CTC) framework and proposed a novel connectionist temporal localization (CTL) framework for SED with sequential labeling. The CTL framework overcomes the "peak clustering" problem encountered in

previous work, and closes one third of the performance gap between presence/absence labeling and strong labeling. (Chapter 4)

- We have investigated the use of the VGGish network [71] and SoundNet [72] as transfer learning feature extractors for SED. We have trained variants of SoundNet that reach a lower validation loss than the original. Through extensive experiments, we have found that a successful application of transfer learning may require up to 50 times as much data for the source task as for the target task. (Chapter 5)

6.2 Potential Applications

This thesis 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 frame-level probabilities of sound events constitute 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) and recurrent neural networks (RNNs); we have also devised a model called "recurrent SVMs" in [14] 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 multiple instance learning (MIL) and connectionist temporal classification / localization (CTC / CTL), 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 [62]. 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 [134], because often we only know that a program has a bug but do not know which part of the code causes the bug.

6.3 Future Work

6.3.1 Continuation of Work in This Thesis

Even though we have obtained exciting results in this thesis, there is still work that could make the conclusions stronger. These include:

- Investigate the attention-based pooling function for multiple instance learning with the constraint that the attention must be monotonic in the frame-level probabilities;
- Improve the generalization of the CTL framework to new data;
- Find out the true cause of the "smearing out" effect in the features extracted from the recurrent variants of SoundNet, and, after solving the problem, study how the transfer learning performance is affected by the layer from which features are extracted.

6.3.2 Utilizing the Hierarchy of Sound Events

The sound event types in the Google Audio Set comes in an ontology. There are broader classes (*e.g.* `animal`) that subsume finer classes (*e.g.* `dog`, `cat`). In this thesis, we did not make use of this hierarchical information, and trained one big neural network to recognize all the sound event types at the same time. Such a big network may not spare enough attention to learn the finer classes, which occur fewer times in the training data than the broader classes.

There exist multiple ways to incorporate hierarchical information into neural networks. *Modular networks* [135, 136, 137] were proposed in the early days of speech recognition, when computational resources were not enough to support the training of one monolithic network to recognize all phonemes. A modular network consists of many smaller networks (called modules), each of which focuses on a subset of the classes (phonemes or sound events) to distinguish. A top-level network then summarizes the opinions of the modules to make a final prediction. For sound event detection, the Audio Set ontology can provide guidance on the creation of subsets of sound event types. Compared to a monolithic network, the modules in a modular network may be trained to better distinguish similar and confusable classes. It is also easier to add new classes to an already trained network: it suffices to add modules for recognizing the new classes and to fine-tune the top-level network; existing modules can be kept intact.

A more sophisticated way to exploit the ontology is a *hierarchical mixture of experts* (HME) [138]. An HME has a bottom-up tree structure: the leaf nodes are expert classifiers that specialize in classifying certain subsets of classes, just like the modules in modular networks; the internal nodes are gating networks that select or combine the predictions of their child nodes.

The recursive structure of an HME allows for a more logical organization of the classes to recognize, and the Audio Set ontology may be directly mapped to the structure of an HME. The tree structure of an HME may also be grown adaptively [139]; a comparison between the adaptively grown tree structure and the Audio Set ontology may be performed to discover imperfections in the latter.

6.3.3 Learning the Temporal Characteristics of Sounds

Sound events also have different temporal characteristics. There are continuous sound events which span a long time interval, transient sound events which only lasts a few frames, as well as intermittent sound events which display a rhythmic structure. It would be beneficial if the network could detect and exploit these temporal characteristics.

Such attempts have been made in [140] and [43]. The authors restrict the kernel in a convolutional layer to have a Gaussian shape with tunable width; the width will end up being larger for continuous sound events and shorter for transient sound events. To accommodate for intermittent sound events, we may also consider using kernels that have the shape of an oscillating function.

The adaptive pooling function in [52] also has the side benefit of distinguishing transient events from continuous ones. The pooling function has a class-dependent parameter α that can bias the pooling function toward either max pooling or average pooling. It is found that the pooling function will tend toward max pooling for transient events, while tending toward average pooling for continuous events.

Another approach to learning the temporal characteristics of sounds is the duration-controlled hidden semi-Markov models used in [141], in which the duration distributions of different sound event types are explicitly modeled. All the techniques above are promising ways toward learning the temporal characteristics of sounds.

Acknowledgments

I would like to express my gratitude for my PhD advisor, Prof. Florian Metze, and my master advisor, Prof. Qin Jin, for their constant support during my eight years at CMU. I would also like to thank all the committee members for their valuable advice on my thesis proposal and defense.

I would like to thank Susi Burger for her effort in compiling and annotating the Noiseme corpus, without which my research wouldn't have been possible. I am also grateful to Juncheng (Billy) Li for his cooperation in the DCASE 2017 challenge. Finally, I would like to thank Qiuqiang Kong and Yong Xu at the University of Surrey, UK, for sharing the code of their SED system and for their long discussions with me about the pooling functions.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE) [142], which is supported by National Science Foundation (NSF) grant number ACI-1548562. Specifically, it used the Comet system at San Diego Supercomputer Center (SDSC) and the Bridges system at the Pittsburgh Supercomputing Center (PSC).

Bibliography

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

[2] 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.

[3] 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.

[4] 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.

[5] Q. Kong, Y. Xu, W. Wang, and M. D. Plumbley, "A joint separation-classification model for sound event detection of weakly labeled data," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2018, pp. 321–325.

[6] 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.

[7] 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.

[8] P. Laffitte, Y. Wang, D. Sodoyer, and L. Girin, "Assessing the performances of different neural network architectures for the detection of screams and shouts in public transportation," *Expert Systems with Applications*, vol. 117, pp. 29–41, 2019.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] R. Malkin, D. Macho, A. Temko, and C. Nadeu, "First evaluation of acoustic event classification systems in CHIL project," in *Joint Workshop on Hands-Free Speech Communication and Microhpone Arrays (HSCMA)*, 2005.

[17] C. Zieger, "An HMM based system for acoustic event detection," in *Multimodal Technologies for Perception of Humans*, 2008, pp. 338–344.

[18] 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.

[19] 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.

[20] A. Waibel and R. Stiefelhagen, *Computers in the Human Interaction Loop*. Springer-Verlag London, 2009.

[21] A. Temko, R. Malkin, C. Zieger, D. Macho, C. Nadeu, and M. Omologo, "Acoustic event detection and classification in smart-room environments: Evaluation of CHIL project systems," in *IV Jornadas en Tecnología del Habla*, ISCA, 2006.

[22] A. Temko and C. Nadeu, "Classification of meeting-room acoustic events with support vector machines and variable-feature-set clustering," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2005, pp. V-505–V-508.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] 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.

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

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

[46] 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.

[47] 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.

[48] A. Shah, A. Kumar, A. G. Hauptmann, and B. Raj, "A closer look at weak label learning for audio events," *arXiv e-prints*, Apr. 2018. [Online]. Available: http://arxiv.org/abs/1804.09288.

[49] P. Joshi, D. Gautam, G. Ramakrishnan, and P. Jyothi, "Time aggregation operators for multi-label audio event detection," in *Proceedings of Interspeech*, ISCA, 2018, pp. 3309–3313.

[50] A. Dang, T. H. Vu, and J.-C. Wang, "Deep learning for DCASE2017 challenge," DCASE2017 Challenge, Tech. Rep., Sep. 2017.

[51] J. Salamon, B. McFee, and P. Li, "DCASE 2017 submission: Multiple instance learning for sound event detection," DCASE2017 Challenge, Tech. Rep., Sep. 2017.

[52] B. McFee, J. Salamon, and J. P. Bello, "Adaptive pooling operators for weakly labeled sound event detection," *arXiv e-prints*, Apr. 2018. [Online]. Available: http://arxiv.org/abs/1804.10070.

[53] Y. Xu, Q. Kong, W. Wang, and M. D. Plumbley, "Large-scale weakly supervised audio classification using gated convolutional neural network," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2018, pp. 121–125.

[54] Q. Kong, Y. Xu, W. Wang, and M. D. Plumbley, "Audio set classification with attention model: A probabilistic perspective," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2018, pp. 316–320.

[55] C. Yu, K. S. Barsim, Q. Kong, and B. Yang, "Multi-level attention model for weakly supervised audio classification," *arXiv e-prints*, Mar. 2018. [Online]. Available: http://arxiv.org/abs/1803.02353.

[56] S. Chen, J. Chen, Q. Jin, and A. Hauptmann, "Class-aware self-attention for audio event recognition," in *International Conference on Multimedia Retrieval (ICMR)*, ACM, 2018, pp. 28–36.

[57] S.-Y. Chou, J.-S. R. Jang, and Y.-H. Yang, "Learning to recognize transient sound events using attentional supervision," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 3336–3342.

[58] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International Conference on Machine Learning (ICML)*, ACM, 2014, pp. 1764–1772.

[59] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv e-prints*, Nov. 2012. [Online]. Available: http://arxiv.org/abs/1211.3711.

[60] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 577–585.

[61] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2016, pp. 4960–4964.

[62] 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.

[63] 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.

[64] 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.

[65] D. P. Ellis and K. Lee, "Features for segmenting and classifying long-duration recordings of personal audio," in *Tutorial and Research Workshop on Statistical and Perceptual Audio Processing*, ISCA, 2004.

[66] B. Schuller, S. Steidl, and A. Batliner, "The Interspeech 2009 emotion challenge," in *Proceedings of Interspeech*, ISCA, 2009, pp. 312–315.

[67] 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.

[68] R. G. Malkin and A. Waibel, "Classifying user environment for mobile applications using linear autoencoding of ambient audio," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2005, pp. 509–512.

[69] F. Kraft, R. Malkin, T. Schaaf, and A. Waibel, "Temporal ICA for classification of acoustic events ia kitchen environment," in *European Conference on Speech Communication and Technology*, 2005.

[70] E. Cakır and T. Virtanen, "End-to-end polyphonic sound event detection using convolutional recurrent neural networks with learned time-frequency representation input," *arXiv e-prints*, May 2018. [Online]. Available: http://arxiv.org/abs/1805.03647.

[71] S. Hershey *et al.*, "CNN architectures for large-scale audio classification," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2017, pp. 131–135.

[72] 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.

[73] Y. Xu, Q. Huang, W. Wang, P. Foster, S. Sigtia, P. J. Jackson, and M. D. Plumbley, "Unsupervised feature learning based on deep models for environmental audio tagging," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 6, pp. 1230–1241, 2017.

[74] R. Arandjelovic and A. Zisserman, "Look, listen and learn," in *International Conference on Computer Vision (ICCV)*, IEEE, 2017, pp. 609–617.

[75] A. Jansen, M. Plakal, R. Pandya, D. P. W. Ellis, S. Hershey, J. Liu, R. C. Moore, and R. A. Saurous, "Unsupervised learning of semantic audio representations," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2018, pp. 126–130.

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

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

[78] 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.

[79] 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.

[80] 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.

[81] 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.

[82] 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.

[83] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen, "DCASE 2017 challenge setup: Tasks, datasets and baseline system," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*, 2017.

[84] 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.

[85] 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.

[86] Y. Wang, J. Li, and F. Metze, "Comparing the max and noisy-or pooling functions in multiple instance learning for weakly supervised sequence learning tasks," in *Proceedings of Interspeech*, ISCA, 2018, pp. 1339–1343.

[87] J. Li, Y. Wang, J. Szurley, F. Metze, and S. Das, "A light-weight multimodel framework for improved environmental audio tagging," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2018, pp. 6832–6836.

[88] S.-Y. Tseng, J. Li, Y. Wang, F. Metze, J. Szurley, and S. Das, "Multiple instance deep learning for weakly supervised small-footprint audio event detection," in *Proceedings of Interspeech*, ISCA, 2018, pp. 3279–3283.

[89] Y. Wang, J. Li, and F. Metze, "A comparison of five multiple instance learning pooling functions for sound event detection with weak labeling," *arXiv e-prints*, Oct. 2018. [Online]. Available: http://arxiv.org/abs/1810.09050.

[90] Y. Wang and F. Metze, "Connectionist temporal localization for sound event detection with sequential labeling," *arXiv e-prints*, Oct. 2018. [Online]. Available: http://arxiv.org/abs/1810.09052.

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

[92] 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.

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

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

[95] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Workshop*, 2017.

[96] 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.

[97] 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.

[98] 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.

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

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

[101] 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.

[102] 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.

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

[104] 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.

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

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

[107] 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.

[108] A. Waibel, T. Hanazawa, G. Hinton, and K. Shikano, "Phoneme recognition using time-delay neural networks," ATR Interpreting Telephony Research Laboratories, Tech. Rep. TR-I-0006, 1987.

[109] A. Waibel, "Phoneme recognition using time-delay neural networks," in *Meeting of the Institute of Electrical, Information and Communication Engineers (IEICE)*, 1987, pp. 19–24.

[110] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328–339, 1989.

[111] J. B. Hampshire II and A. Waibel, "Connectionist architectures for multi-speaker phoneme recognition," in *Advances in Neural Information Processing Systems (NIPS)*, 1990, pp. 203–210.

[112] H. Sawai, "Frequency-time-shift-invariant time-delay neural networks for robust continuous speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 1991, pp. 45–48.

[113] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

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

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

[116] 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.

[117] 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.

[118] 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.

[119] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," *Artificial intelligence*, vol. 89, no. 1, pp. 31–71, 1997.

[120] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[121] N. Segev, "Transfer learning using decision forests," Master's thesis, Technion – Israel Institute of Technology, 2015.

[122] R. Serizel, N. Turpault, H. Eghbal-Zadeh, and A. P. Shah, "Large-scale weakly labeled semi-supervised sound event detection in domestic environments," *arXiv e-prints*, Jul. 2018. [Online]. Available: http://arxiv.org/abs/1807.10501.

[123] 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.

[124] A. Kumar and B. Raj, "Deep CNN framework for audio event recognition using weakly labeled web data," *arXiv e-prints*, Jul. 2017. [Online]. Available: http://arxiv.org/abs/1707.02530.

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

[126] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in Python," in *Proceedings of the 14th Python in Science Conference*, 2015, pp. 18–25.

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

[128] A. Kumar, M. Khadkevich, and C. Fügen, "Knowledge transfer from weakly labeled audio using convolutional neural network for sound events and scenes," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2018, pp. 326–330.

[129] B. L. Welch, "The generalization of 'Student's' problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1–2, pp. 28–35, 1947.

[130] M. Thoma, "Analysis and optimization of convolutional neural network architectures," Master's thesis, Karlsruhe Institute of Technology, 2017.

[131] Y. Wu and T. Lee, "Reducing model complexity for DNN based large-scale audio classification," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 2018, pp. 331–335.

[132] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "YouTube-8M: A large-scale video classification benchmark," *arXiv e-prints*, Sep. 2016. [Online]. Available: http://arxiv.org/abs/1609.08675.

[133] 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.

[134] 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.

[135] A. Waibel, "Modular construction of time-delay neural networks for speech recognition," *Neural Computation*, vol. 1, no. 1, pp. 39–46, 1989.

[136] A. Waibel, "Consonant recognition by modular construction of large phonemic time-delay neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 1989, pp. 215–223.

[137] A. Waibel, H. Sawai, and K. Shikano, "Modularity and scaling in large phonemic neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 12, pp. 1888–1898, 1989.

[138] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[139] J. Fritsch, M. Finke, and A. Waibel, "Adaptively growing hierarchical mixtures of experts," in *Advances in Neural Information Processing Systems (NIPS)*, 1997, pp. 459–465.

[140] U. Bodenhausen and A. Waibel, "The Tempo 2 algorithm: Adjusting time-delays by supervised learning," in *Advances in Neural Information Processing Systems (NIPS)*, 1991, pp. 155–161.

[141] T. Hayashi, S. Watanabe, T. Toda, T. Hori, J. Le Roux, and K. Takeda, "Duration-controlled LSTM for polyphonic sound event detection," *IEEE/ACM Transactions on Audio, Speech and Language Processing*, vol. 25, no. 11, pp. 2059–2070, 2017.

[142] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr, "XSEDE: Accelerating scientific discovery," *Computing in Science & Engineering*, vol. 16, no. 5, pp. 62–74, 2014.