DIFFERENTIABLE ARCHITECTURE SEARCH FOR KEYWORD SPOTTING

by

Tong Mo

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

With the popularity of smart mobile devices, the need to control mobile devices using voice is increasing. Also, there is greater expectation for the accuracy of keyword spotting. Many existing researches have applied neural networks to keyword spotting, and have great performances. However, at the same time, for keyword spotting on mobile devices, there is still a need to further reduce the parameter size and improve the recognition accuracy simultaneously.

In this thesis, I apply the neural network architecture search approach to keyword spotting, and proposed a Differentiable Architecture Search Approach for keyword spotting. This approach can design multiple neural network models for keyword spotting through search. In this thesis, I proposed eight specific neural network models designed by this approach. All models beat the state-of-the-art model based on the evaluation on Google Commands Dataset, with similar or much smaller parameter sizes.

Acknowledgments

I would like to thank all the people who contributed in some way to the work described in this thesis. First and foremost, I would like to extend my most sincere thanks to my supervisor, Dr. Di Niu. Dr. Di Niu provided very comprehensive guidance during my research and his supervision help me to learn how to be a good researcher. I would also like to thank Yakun for cooperating with me in my research, she gave me very valuable suggestions in many details. I would also like to thank Mohammad and Keith, both of whom had several helpful discussions with me during my research.

I would like to acknowledge friends who supported me during my time here. I thank all my friends in the lab, Bang, Shan, Shengyao, Yakun, Keith, Qikai, Mingjun, Jiuding, Fred, Yaochen and Jerry. I cherish the opportunity to get along with you.

Finally, I want to express my sincere gratitude to my family. First I would like to thank Mom, Dad and other family members for their constant love and support. Additionally, I would like to thank my lover Boping for being my strength throughout the times.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Acronyms	Definition
CNN	Convolutional Neural Network
DARTS	Differentiable Architecture Search
DAS_KWS	Differentiable Architecture Search for Keyword Spotting
DCT	Discrete Cosine Transform
DNN	Deep Neural Network
ENAS	Efficient Neural Architecture Search
FFT	Fast Fourier Transform
HMM	Hidden Markov Model
MFCC	Mel Frequency Cepstral Coefficient
NAS	Neural Architecture Search
PCEN	Per-Channel Energy Normalization
ResNet	Deep Residual Network
RNN	Recurrent Neural Network

Chapter 1

Introduction

1.1 Keyword Spotting

With the rapid development of smartphones and smart tablets, voice-recognition-related technology are becoming progressively popular. Several famous technology companies have launched related products, such as Apple's Siri, Microsoft's Cortana and Amazon's Alexa. Keyword spotting is a typical problem in speech processing, it deals with the identification of keywords in utterances. Basically, keyword spotting process is performed in the cloud after audio recordings from users' mobile devices are transferred to the cloud [1] [2]. The advantage of this is very obvious. In the cloud, one can use very large scales neural network frameworks to improve keyword spotting accuracy. However, in such a frequent data transmission process, privacy issues will be tricky, and some important information of users is at risk of leakage [2]. Therefore, many researchers have studied on small footprint keyword spotting models in an effort to obtain small size models that can be deployed on low power and performance-limited devices [2]. Small footprint keyword spotting usually only focuses on recognition of common commands, such as 'yes', 'no', 'on' and 'off', and enables these keywords to be quickly identified directly on the device and controls the device accordingly.

With the continuous innovation of computer science technology, keyword spotting models have been continuously updated in the past twenty years. Two decades

ago, the commonly used technique for keyword spotting is the Keyword Filler Hidden Markov Model (HMM) [5][6]. Generally, this approach uses every keyword to train an HMM model and uses the non-keyword segments to train a filter HMM model [5][7]. In the model operation, because the most likely state sequence needs to be found, the Viterbi algorithm needs to be used for decoding, which is very computationally expensive [8][9]. In addition, some work built keyword spotting models based on large margin formulations, which surpass HMM models in terms of accuracy, but have high detection latency [10].

With the increasing popularity of neural networks, lots of solutions for keyword spotting based on neural networks appeared in recent years. Guoguo Chen applied deep neural networks (DNN) to achieve a great improvement over the previous HMM solutions [7]. Tara N. Sainath and Carolina Parada first applied convolutional neural networks (CNN) to the problem of keyword spotting, and proposed that keyword spotting should aim at fewer parameters (smaller footprint) [1]. Raphael Tang and Jimmy Lin improved on the previous convolutional neural network framework and applied Deep residual networks (ResNets) to the model, which outperformed Tara's model. Based on tests on the classic public dataset Google Commands Dataset, Raphael and Jimmy's model (named as Honk by themselves) has the state-of-the-art performance so far [2].

So far, the state-of-the-art in the field of keyword spotting is the Honk model which is based on convolutional neural networks. However, due to the accumulation of a large number of Resnet Blocks in the Honk model, the model size is still large. In addition, the positions of the various types of layers inside the model are completely based on manual design, and the optimal performance under this model size has not been achieved. Thus, this paper will explore the best convolutional neural network architecture to solve the problem of keyword spotting. I will not be limited to the structure Raphael and Jimmy proposed, but will redesign the neural network structure from the lowest level, considering where different types of layers should be placed. The design of the convolutional neural network will be based on Neural Architecture Search (NAS) method, and related information will

be explained in the next section.

1.2 Neural Architecture Search

In the previous section, it was introduced that the state-of-the-art of keyword spotting is based on the convolutional neural network architecture. In order to explore the method of optimizing this CNN architecture, it is necessary to deeply study the area where CNN is most applied, image recognition. The most classic approach for image recognition is to design various convolutional neural network architectures. In 2017, Barret Zoph first proposed that a neural network architecture could be searched, rather than manually designed [11]. His Neural Architecture Search (NAS) approach outperformed the state-of-the-art on CIFAR-10 dataset.

Barret's NAS method is based on an RNN controller. The controller first samples a child architecture and then trains it to convergence to see its performance. Then the controller explores other better architectures based on the performance. This process needs to be repeated for hundreds of iterations and has an extremely large running time complexity and space complexity. Usually, the running time of NAS exceeds 30,000 GPU hours for CIFAR-10 dataset, which means that for many ordinary research institutions or laboratories, they do not have enough computing resources to reproduce NAS methods.

To reduce GPU Hours, several approaches for speeding up NAS have been proposed. Hieu Pham proposed the Efficient Neural Architecture Search (ENAS) approach based on NAS [12]. The main contribution of ENAS is to increase NAS performance by requiring all child models to share weights from scratch to convergence in order to escape training of each child model. By sharing parameters, ENAS reduced GPU hours by 1000x compared to NAS. Hanxiao Liu proposed hierarchical representations for efficient architecture search by imposing a particular structure of the search space [13]. Han Cai proposed a new NAS framework, in which the meta-controller explores the space of the architecture by means of network transformation operations [14]. The above mentioned Neural architecture

search methods are optimized in runtime, but they still treat the architecture search as a black-box optimization problem within a discrete search space, which results in a great number of necessary architecture evaluations [4].

According to this, Haoxiao Liu and Karen Simonyan approached the neural architecture search problem from a different angle. Instead of searching for a discrete set of candidate architectures, they make the search space continuous so that the architecture can be optimized by gradient descent according to its validation set performance [4]. Since it is a gradient-based optimization, they called their method Differentiable ARchiTecture Search (DARTS). Comparing to inefficient black-box search, DARTS can achieve competitive performance with the state-of-the-art on CIFAR-10 and ImageNet dataset, with much less computation time (30 to 90 GPU hours) [4].The convolutional cell designed by DARTS can achieve 2.76% test error on CIFAR-10 dataset using 3.3M parameters, which is comparable to the state-of-the-art result by regularized evolution, and can achieve 26.7% top-1 error on ImageNet dataset which is competitive with the state-of-the-art result by reinforcement learning [4].

Considering the existing neural architecture search approaches and the specific requirements of the keyword spotting problem, I summarize the current situation of keyword spotting as follows:

- As of now, the best-performing model for keyword spotting converts keyword audio utterances into an image-like structure, and then designs a convolutional neural network for recognition.

- The structure of the design is to stack Resnet blocks, which results in a large number of model parameters. In the field of image recognition, this has been proven to be not very optimal.

- For the on-device keyword spotting purpose, it is necessary to further reduce the number of model parameters and obtain comparable or even better performance than the existing state-of-the-art model.

- The solution this paper proposed is based on an existing neural architecture search method, searching for a convolutional neural network suitable for keyword spotting.

And the reasons I choose DARTS as the baseline model are summarized as follows:

- Keyword spotting and image recognition have many similarities, so when designing a convolutional neural network, the performance of each NAS model in image recognition should be an important reference.
- DARTS can achieve comparable results to the state-of-the-art results on classical image datasets, like CIFAR-10, CIFAR-100 and ImageNet.
- Due to the data efficiency of gradient-based optimization, DARTS can achieve competitive performance with less computation resources.

Based on the above summarization, in this thesis, I will propose a differentiable architecture search approach for keyword spotting, in order to achieve better keyword spotting accuracy with a smaller model size.

1.3 Neural Architecture Search for Keyword Spotting

So far, very little work has been done on neural architecture search for keyword spotting. As far as we know, only Tom Veniat proposed a stochastic adaptive neural architecture search approach for keyword spotting. His approach is to adapt the architecture to the difficulty of the recognition of audios, i.e. choosing a simple architecture when the keyword spotting is easy and a larger architecture when the keyword spotting is difficult [15]. His main objective is to reduce computation and energy consumption, but the spotting accuracy is pretty lower than the state-of-the-art result. In summary, existing works on neural architecture search for keyword spotting choose from existing architectures rather than searching and designing an

architecture, and also sacrifice a large amount of accuracy for the purpose of energy consumption reduction.

In this thesis, I approach the keyword spotting problem from a different angle, propose a differentiable architecture search approach for keyword spotting, and design the whole architecture from the ground up. The key steps in building this model are as follows:

- Preprocess the audio utterances, make sure the processed data can be used as the input of the convolutional neural network without losing the information carried by the audio.
- Design the CNN architecture based on neural architecture search approaches, specifically search for what type of layer should be used between every two nodes in the architecture, instead of choosing from specific architectures.
- Figure out specific operations (layers) suitable for keyword spotting. On the premise of ensuring the keyword spotting accuracy of the model, reduce the number of parameters as much as possible.

I proposed eight specific models obtained by the differentiable architecture search approach. All eight models outperform the state-of-the-art model proposed by Tang [2], with much less or comparable parameter sizes, based on tests on Google Commands Dataset [16].

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 mainly introduces how to pre-process speech samples, how to design a differential architecture search approach for keyword spotting, and how to select suitable candidate operations through previous research work. Chapter 3 proposes eight specific models based on the differential architecture search approach, and details the design ideas, model architecture, and model-related parameters. Chapter 4 introduces the experimental

test of our model on Google Commands Dataset, and gives comparison with the evaluation results of baseline models. Finally, we summarize our work and discuss future works in chapter 5.

Chapter 2

Model Implementation

2.1 Introduction

This chapter introduces the details of our model implementation. The remainder of this chapter is organized as follows. Section 2.2 introduces the method for preprocessing the speech samples. Section 2.3 analyzes in detail and determines the candidate operations that our model should apply, then introduces the Differentiable Architecture Search algorithm for Keyword Spotting.

2.2 Input Preprocessing

The inputs of our models are 65,000 one-second long utterances (.wav files) of 30 short words from Google's Speech Commands Dataset [16]. Detailed experimental setup will be described in next chapter. Extracting the best parametric representation of acoustic signals is an important task to deliver a better result in recognition. Classical acoustic signals extraction approaches include Per-Channel Energy Normalization (PCEN) and Mel Frequency Cepstral Coefficient (MFCC) [17][18]. In our models, we choose MFCC.

MFCC has been the popular features used for speech recognition and keyword spotting since it was created [19]. Its wide applications are due to its ability to represent the speech amplitude spectrum in a compact form. Moreover, MFCC

is based on human hearing perceptions and human ear's critical bandwidth with frequency that cannot detect frequencies above 1KHz [20]. Thus, MFCC has a clean form which can be easily processed in models and also can capture important characteristic and information of phonetic in speech.

In our models, we follow the traditional process of the generation of MFCC, including six main steps [20][21].

- **Pre–emphasis**: This step processes acoustic signal through a filter that accentuates higher frequencies. This process aims to increase the energy of acoustic signal at higher frequency and is not a necessary process.

- **Analog to Digital Conversion**: To convert the original acoustic signals to digital signals, this step segments the speech utterances into small frames with the length of 10 msec. An acoustic signal is divided into frames of N samples. E.g. for a 1 second long signal, N should be 100.

- **Windowing**: In this step, the signal is multiplied with a Hamming window or a Hanning window. The windowing process equation is given as follows [20]:

$$Y(n) = X(n) * H(n) \tag{2.1}$$

  where n = number of samples in each frame, Y(n) = Output Signal, X(n) = Input Signal, H(n) = Hamming or Hanning window function. For Hamming window:

$$H(n) = 0.54 - 0.46 * cos(2 * \pi * n/(N-1)),\ 0 \leq n \leq N-1 \tag{2.2}$$

  For Hanning window:

$$H(n) = 0.5 - 0.5 * cos(2 * \pi * n/(N-1)),\ 0 \leq n \leq N-1 \tag{2.3}$$

- **Fast Fourier Transform (FFT)**: This step converts each frame of N samples from time domain into frequency domain. This conversion is based on

traditional fast Fourier Transform [20].

- **Mel Processing**: Use a bunch of triangle filters to compute a weighted sum of filter spectral components, if the frequencies range in FFT spectrum is too wide. Then compute the Mel for the frequency f in Hz as follows [20][21].

$$F(Mel) = 2595 \log_{10}(1 + \frac{f}{700}) \tag{2.4}$$

- **Discrete Cosine Transform (DCT)**: In this stage, we convert the log Mel spectrum into time domain based on Discrete Cosine Transform (DCT), and obtain the Mel Frequency Cepstrum Coefficient. Now, each input audio utterance is transformed into a sequence of vector [20][21].

In this thesis, I segment each speech utterances into 101 frames and use 40 different frequencies to compute MFCC. In this way, 1*101*40 dimensional Mel-Frequency Cepstrum Coefficient (MFCC) frames are constructed, which have a similar structure with a one-channel image . To test the robustness of our models and increase the difficulty of spotting, we also add background noise consisting of whitenoise, pink noise, and human-made noise, then perform random shift to each sample. Details will be shown in the experiment section. The whole feature extraction process is shown in Fig. 2.1.

2.3 Differentiable Architecture Search for Keyword Spotting (DAS_KWS)

In this section, we introduce our differentiable architecture search approach for keyword spotting in details, based on DARTS by Liu, NAS by Zoph and ENAS by Cai [11][12][4]. We describe our general search space and general model structure of the differentiable architecture search approach in 2.3.1.

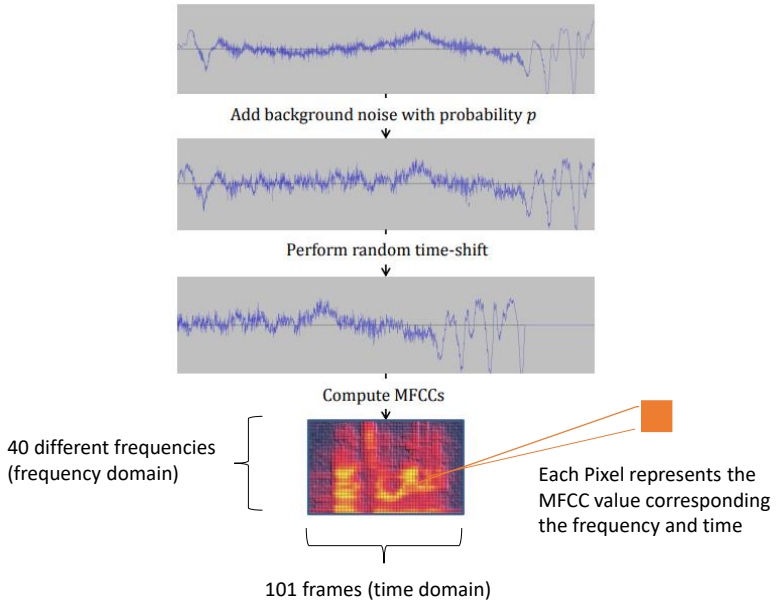

Fig. 2.1. MFCC generation [2]

2.3.1 Overall Structure of DAS_KWS

The overall structure of DAS_KWS follows DARTS, as shown in Fig. 2.2. We search for two types of cells as the building block of the final neural architecture, normal cells and reduction cells. Then we stack the learnt cells to form a convolutional neural network [4]. For cell, we follow the definition in DARTS and ENAS. A cell is a directed acyclic graph consisting of an ordered N node chain, while each node $x^{(i)}$ is a feature map in convolutional networks. Between each two nodes, there is a directed edge (i, j) corresponding to an operation $o^{(i,j)}$ (e.g., convolution, max pooling)[12][4]. Each cell has two input nodes and one output node. The inputs are the outputs of the previous two cells and the output is obtained through concatenation on all intermediate nodes [4]. Each intermediate node is determined by all its previous nodes [4]:

$$x^{(i)} = \sum_{i<j} o^{(i,j)}(x^{(i)}) \tag{2.5}$$

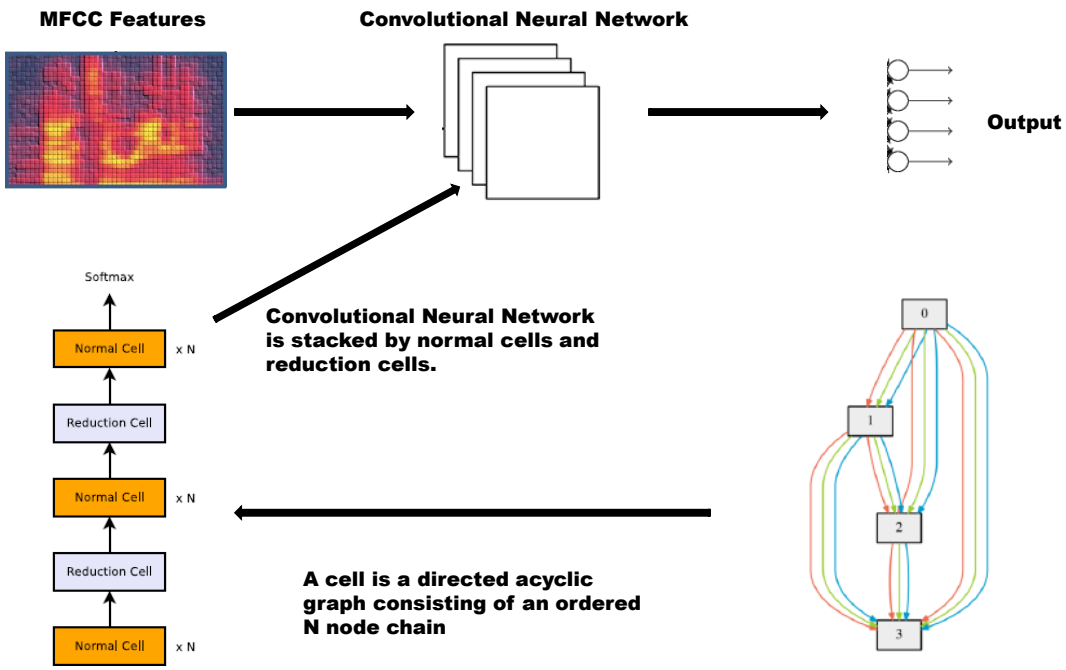

Fig. 2.2. Overall Structure of DAS_KWS

In sum, the overall structure and process of DAS_KWS are summarized as follows.

- The topmost structure is to use the convolutional neural network to process the MFCC features and get the classification output. But the structure of convolutional neural networks is unknown.
- The convolutional neural network is formed by stacking learnt normal cells and reduction cells. Generally, a reduction cell is stacked after every N normal cells.
- A cell is a directed acyclic graph consisting of an ordered sequence of N nodes. Learning a cell is equivalent to learning which operations should be applied to the edge between each two nodes [4].

Thus, the entire keyword spotting problem is simplified as: which operations to choose from, and how to determine which operation to use through learning.

2.3.2 Operations Options

In this thesis, an operation is associated with a directed edge between two nodes, that transforms a node. In other words, an operation is a layer in neural network, e.g. a convolutional layer, a max pooling layer.

Reviewing previous research, Sainath first proposed the use of convolutional neural networks to solve the problem of keyword spotting[1]. Then, Tang redesigned a convolutional neural network based on Resnet, and obtained the state-of-the-art results on the Google Commands Dataset [2].

Sainath's model is an improvement on the Deep Neural Network (DNN) model. Previously, only pure DNN was used to solve the problem of keyword spotting. Sainath places one or two convolutional layers before the DNN, which greatly improves spotting accuracy. In summary, Sainath's model first passes the input through one or two convolutional layers, then passes the output of these convolutional layers to a linear low-rank layer or one to two DNN layers, and obtains outputs by softmax function. For these convolutional layers, it chooses normal bias-free convolutional layer with weights $W \in R^{(m*r)*n}$, where m is the width, r is the height and n is the number of feature maps. The values of m and r can be determined artificially, as long as the following rules are followed: $m \leq width \quad of \quad input$, and $r \leq height \quad of \quad input$, where in this thesis the width and height of the input represent the input feature dimension in time and frequency respectively. The overall structure diagram of Sainath's model is shown in Fig. 2.3 [1].

Based on the above-mentioned overall structure, Sainath proposed several specific models. Here we list two classic ones: trad-fpool3, which is their base model; and tpool2, the most accurate variant of the models they explored. Schematic diagrams of the two models are shown in Fig. 2.4 and corresponding parameters are shown in Table. 2.1 and Table. 2.2 [1].

Sainath first proposed the use of CNN for keyword spotting, and we found the following improvement points.

- Only the basic convolutional layers are used instead of advanced convolu-

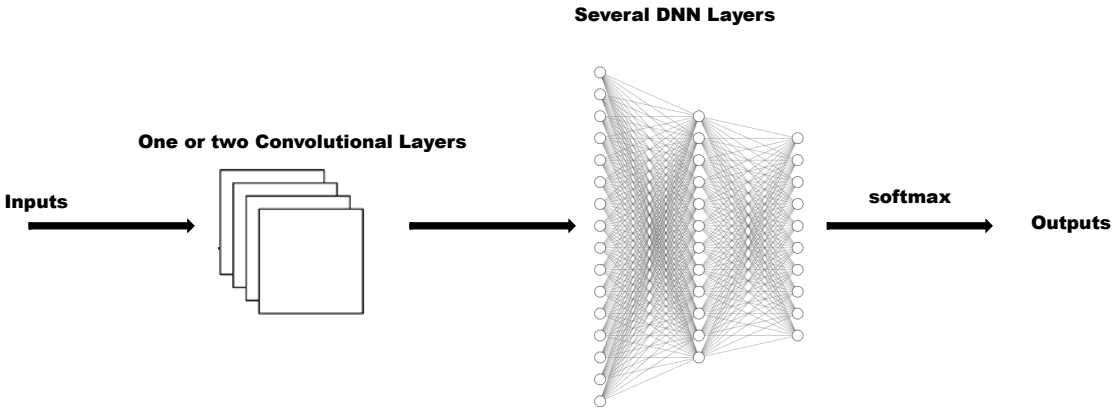

Fig. 2.3. Overall Structure of Sainath's model [1]

layer type	m	r	n
conv	20	8	64
conv	10	4	64
lin	-	-	32
dnn	-	-	128
softmax	-	-	4
Number of Parameters	244K		

TABLE 2.1
ARCHITECTURE FOR TRAD-FPOOL3 [1]

layer type	m	r	n
conv	21	8	94
conv	6	4	94
lin	-	-	32
softmax	-	-	4
-	-	-	-
Number of Parameters	7400K		

TABLE 2.2
ARCHITECTURE FOR TPOOL2 [1]

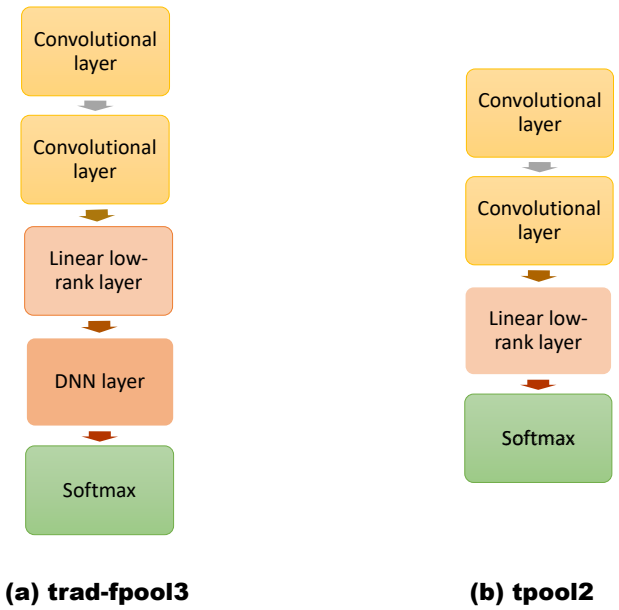

Fig. 2.4. Baseline Model; (a) tradfpool3, (b) tpool2

tional layers, such as Separate Convolution, Dilated Convolution, Grouped Convolution and other advanced convolutional layers that have proven to be excellent in image recognition.

- Each layer is sequentially stacked together, without using complex neural network architectures, such as ResNet [22], MobileNets [23]. There should be more complicated connections between nodes.

- When there's a need to reduce the size of the model, it simply deletes one or two layers, which makes the test accuracy of the model lower. Operations with many parameters in some locations can actually be replaced with simpler operations.

Tang improved on Sainath's model and got the state-of-the-art results on Google Commands Dataset. Tang mainly improved the model in two aspects, one is to use dilated convolutional layers, and the other is to apply deep residual networks.

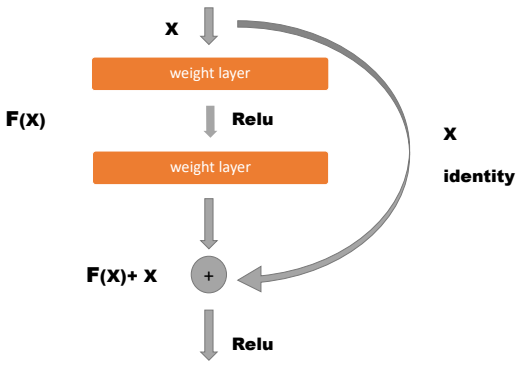

Fig. 2.5. The Residual Learning Block

Deep residual networks (ResNet) are a great breakthrough in deep learning that allowed everyone to train a much deeper neural network. They were first applied on image recognition and obtained results that beat the state-of-the-art performance. The motivation for inventing Resnet was to solve the degradation problem of extremely deep neural networks [22][24]. The degradation problem is: with the network depth rising, accuracy is saturated and then rapidly degrades. And this degradation problem is not caused by overfitting, because when the number of neural network layers is increased to make the network deeper, not only the test accuracy will decrease, but the train accuracy will also decrease significantly. They solved the degradation problem by proposing a residual learning block, to let the layers fit a residual mapping instead of expecting each stacked layers to fit a desired mapping [22]. The residual learning block is shown in Fig. 2.5.

Tang proposed a deep residual learning approach for keyword spotting based on the standard ResNet architectures. Their model is formed by stacking multiple ResNet blocks, where each ResNet block consists of a bias-free convolution layer with weights $W \in R^{(m*r)*n}$ (where m is the width, r is the height and n is the number of feature maps), ReLU activation units, and a batch normalization layer [2]. Their model architecture is shown in Fig. 2.6.

Based on the architecture, Tang proposed three baseline models, called res8-narrow, res15 and res26, according to different numbers of residual blocks [2]. Pa-

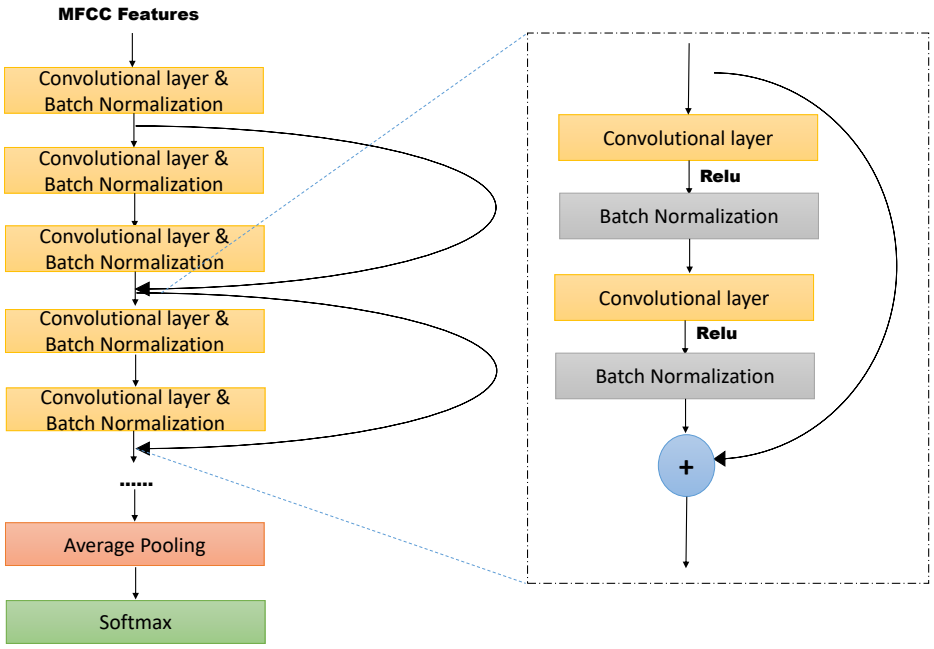

Fig. 2.6. Architecture of residual learning approach for keyword spotting

rameters used for the models are illustrated in Table. 2.3, Table. 2.4 and Table. 2.5, where m, r, n are the width, the height and the number of features maps, respectively, and d_w, d_h are convolution dilation.

layer type	m	r	n	d_w	d_h
conv	3	3	45	-	-
residual block $\times 6$	3	3	45	$2^{\frac{i}{3}}$	$2^{\frac{i}{3}}$
conv	3	3	45	16	16
batch normalization	-	-	45	-	-
average-pool	-	-	45	-	-
softmax	-	-	12	-	-
Number of Parameters	238K				

TABLE 2.3
PARAMETERS FOR RES15

In sum, Tang first proposed the use of dilation convolutional layers and ResNet for keyword spotting, and achieved the state-of-the-art results. And we found the following improvement points for a better keyword spotting model [2].

layer type	m	r	n
conv	3	3	19
average-pool	4	3	19
residual block $\times 3$	3	3	19
average-pool	-	-	19
softmax	-	-	12
Number of Parameters	20K		

TABLE 2.4
PARAMETERS FOR RES8-NARROW

layer type	m	r	n
conv	3	3	45
average-pool	2	2	45
residual block $\times 12$	3	3	45
average-pool	-	-	45
softmax	-	-	12
Number of Parameters	438K		

TABLE 2.5
PARAMETERS FOR RES26

- In addition to the dilated convolutional layer, consider applying a variety of advanced convolutional layers, such as transposed convolutional layer, separable convolutional layer, etc.

- Don't fix the combination of layers. For example, in Sainath's model, the output of each convolutional layer is always passed to a max pooling layer, while in Tang's model, the output of one residual block is always passed to one residual block. This design method is likely to cause too many parameters. In this thesis, between each two nodes in the neural network architecture, we set enough candidate operations with different degrees of complexity, which could be a simple single max pooling layer or a complex residual block.

- In the whole architecture, there should be a possibility of connection between every two nodes. Take Fig. 2.7 as an example. This is part of the residual learning model. By definition, there are six nodes in this architecture, numbered Node1 to Node6. In this architecture, disconnected nodes pairs are: $[1, 3]$, $[1, 4]$, $[1, 5]$, $[1, 6]$, $[2, 5]$, $[2, 6]$, $[3, 5]$ and $[3, 6]$. This makes some of the possibilities of a better performing neural network architecture lost.

  In this thesis, we search each candidate operation for the edge between any two nodes. On the other hand, we also allow no connection between two nodes. One of the candidate operations is "skip connections".

In summary, our search space (candidate operations) is designed as follows.

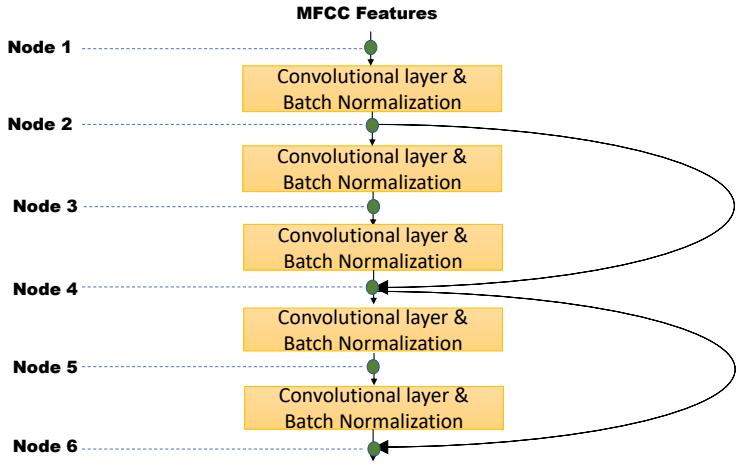

Fig. 2.7. Schematic diagram of nodes in a neural network

- **Skip Connections**: There is no operation on this edge.
- **Identity**: There is the identity operation on this edge.
- **Average Pooling & Max Pooling**: The aim of pooling is to turn the joint representation of features into a more functional one which keeps important information while discarding irrelevant information [25]. In image recognition, pooling can greatly help the model to identify the same object at different locations (ignoring the irrelevant information of location). Similarly, in keyword spotting, since we perform random time shift to each sample in order to increase the difficulty of recognition, the MFCC features have similar characteristic, as the width of it represents the time dimension. Thus, pooling layers should be candidate operations of our model.

  The max pooling method selects the largest element in each pooling region as: [25]

$$y_{kij} = \max_{(p,q)\in R_{ij}} x_{kpq} \tag{2.6}$$

  while the average pooling method takes the arithmetic mean of the elements

in each pooling region as:

$$y_{kij} = \frac{1}{|R_{ij}|} \sum_{(p,q) \in R_{ij}} x_{kpq} \tag{2.7}$$

where R_{ij} is the pooling region, $|R_{ij}|$ is the size of the pooling region, y_{kij} is the output of the pooling operator for the kth feature map and x_{kpq} is the element at location (p, q) within the pooling region [25].

- **Dilated Convolution**: Dilated convolution is an advanced convolution operation with a wider kernel. Let F be a discrete function and k be a discrete filter [26]. The original discrete convolution operator * is defined as [26]:

$$(F * k)(p) = \sum_{s+t=p} F(s)k(t) \tag{2.8}$$

And the dilated convolution operator $*_l$ is defined as:

$$(F *_l k)(p) = \sum_{s+lt=p} F(s)k(t) \tag{2.9}$$

Essentially, ordinary convolution operators are a special case of dilated convolution operators ($l = 1$), as shown in Fig. 2.8. Dilated Convolution has been shown to be effective in identifying object contours [26][27]. Correspondingly, in keyword spotting, it can help identify the higher energy part of the MFCC map, that is, the pronunciation part. We add 3×3, 5×5, 7×7 and 9×9 dilated convolutions to the candidate operations. Each dilated convolution operation follows ReLU-DilatedConv-BatchNormalization order.

- **Separable Convolution**: Separable convolution, which is also called depthwise separable convolution, is a special convolution structure. It consists of a spatial convolution conducted independently over each channel of an input, followed by a point-sensitive convolution, i.e. a 1×1 convolution, projecting

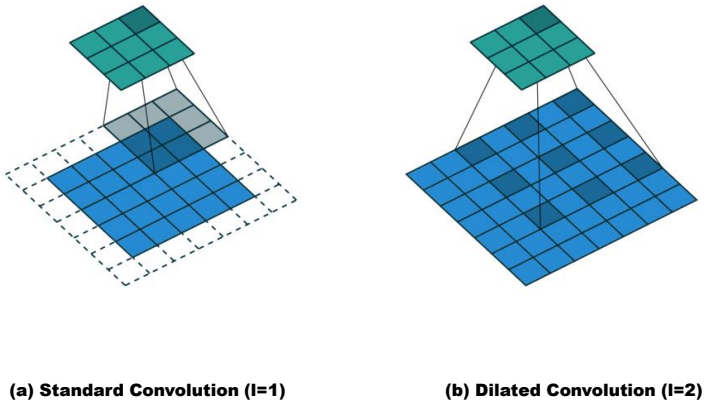

Fig. 2.8. Dilated Convolution [3]

the channels output onto a new channel space by the depth-sensitive convolution [28][29].

Compared to the standard convolution, separable convolution can save up a lot on computational power. A standard convolution's computation cost is [23]:

$$Cost_{Conv} = D_K \times D_K \times M \times N \times D_F \times D_F \tag{2.10}$$

where M, N are the number of input channels and output channels respectively, $D_K \times D_K$ is the kernel size and $D_F \times D_F$ is the feature map size.

On the other hand, the depthwise separable convolution has two layers, one for filtering and one for combining.The filtering computation cost is $D_K \times D_K \times M \times D_F \times D_F$, and the combination computation cost is $M \times N \times D_F \times D_F$. Thus, depthwise separable convolutions cost is [23]:

$$Cost_{SepConv} = D_K \times D_K \times M \times D_F \times D_F + M \times N \times D_F \times D_F \tag{2.11}$$

And it causes a reduction in computation of [23]:

$$\frac{D_K \times D_K \times M \times D_F \times D_F + M \times N \times D_F \times D_F}{D_K \times D_K \times M \times N \times D_F \times D_F} = \frac{1}{N} + \frac{1}{D_K^2} \quad (2.12)$$

This reduction in computation is very useful for the keyword spotting on mobile devices. We add 3×3, 5×5, 7×7 and 9×9 separable convolutions to the candidate operations and each separable convolution is always applied twice.

- **Residual Block**: The characteristics and structure of the Residual block have been explained in detail above. We add the residual block shown in the dotted box in Fig. 2.6 to the candidate operations.

In this section, we analyze various convolutional layers, and combinations of convolutional layers and other layers in detail to determine candidate operations for our model. In the next section, we will introduce the search algorithm for how to find the best operation from candidate operations.

2.3.3 Differentiable Architecture Search Algorithm

In order to perform a differentiable architecture search using gradient descent, the search space need to be continuous. Thus, we need perform a continuous relaxation on all operations (e.g. convolution, max pooling, average pooling). Let S be a set of all candidate operations. To make the search space continuous, over all possible operations, we relax the discrete option of an operation to a softmax function [4]:

$$\overline{o}^{(i,j)}(x) = \sum_{o \in S} \frac{exp({\alpha_o}^{(i,j)})}{\sum_{o' \in S} exp({\alpha_{o'}}^{(i,j)})} o(x) \quad (2.13)$$

where $\overline{o}^{(i,j)}$ is mixed operation and $\alpha^{(i,j)}$ is a parameter vector, storing the operation mixing weights for a pair of nodes (i, j). We called α mixing probabilities or architecture weights. At the end of the search, we select the α with the largest value and use its corresponding operation as the operation between these two nodes [4].

After continuous relaxation, there are two types of weights, architecture weights α and network weights w (weights of convolutional neural networks). α is the weights used to decide which operation to choose, but there is no way to learn α directly. Instead, the value of α must be learnt and optimized by training the neural network corresponding to α and comparing the validation accuracy. Thus, α and w need to be learnt jointly, using gradient descent [4].

We denote the training loss by Los_{train} and the validation loss by Los_{val}. The joint optimization for α and w requires following rules.

When training a neural network, the weights w are determined when the training loss is minimized. Thus, the optimal weights w are [4]:

$$w^* = argmin_w Los_{train}(w, \alpha^*) \tag{2.14}$$

On the other hand, the lowest validation loss means the neural architecture is great, corresponding to the optimized α.The optimization process of weights α is[4]:

$$\begin{aligned} \min_{\alpha} \quad & Los_{val}(w^*(\alpha), \alpha) \\ \text{s.t.} \quad & w^*(\alpha) = argmin_w Los_{train}(w, \alpha) \end{aligned} \tag{2.15}$$

This is a bilevel optimization problem with α as the upper-level variable and w as the lower-level variable [4].

To avoid expensive inner optimization, we follow DARTS' approximation approach on the gradient of the validation loss [4]:

$$\nabla_\alpha Loss_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha Loss_{val}(w - \xi \nabla_w Loss_{train}(w, \alpha), \alpha) \tag{2.16}$$

where w is the current neural network weights and ξ is the learning rate for one step of inner optimization. Based on the Equation. 2.16, we can approximate $w^*(\alpha)$ using only one single step, instead of spending a lot of time in the inner optimization in Equation. 2.15. And according to the experiment practice of DARTS, we should set ξ equal to the learning rate for optimizing w [4].

Algorithm 2.1 Differentiable Architecture Search algorithm for Keyword Spotting

1: Input training set and testing set. Each set contains MFCC features and corresponding labels.
2: Input candidate operations
3: Determine the structure of the cell (number of nodes)
4: For each edge (i, j), create a mixed operation parametrized by the architecture weights $\alpha^{(i,j)}$.
5: **while** not converged **do**
6: Update the architecture weights α by descending $\nabla_\alpha Loss_{val}(w - \xi\nabla_w Loss_{train}(w, \alpha), \alpha)$
7: Update the neural network weights w by descending $Los_{train}(w, \alpha)$
8: Determine the final architecture based on α

The outline of Differentiable Architecture Search algorithm for Keyword Spotting (DAS_KWS) is shown in Alg. 2.1.

There is also a simplified version of the architecture search approach, which is to set ξ to zero.From Equation. 2.16, we can find that when ξ is equal to zero, $\xi\nabla_w Loss_{train}(w, \alpha) = 0$. The architecture gradient becomes [4]:

$$\nabla_\alpha Loss_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha Loss_{val}(w, \alpha) \tag{2.17}$$

This is equivalent to assuming that the current w equals to $w^*(\alpha)$. This is a less accurate approximation method, but it can greatly speed up the operation. We follow DARTS's naming of these two approximation methods. When ξ is equal to zero, it is a first-order approximation. When ξ is greater than zero, it is a second-order approximation [4].

After running the architecture search algorithm shown by Alg. 2.1, all α values are obtained. In the last step, we need to export the architecture by the value of α. An intuitive method is to select the operation corresponding to the largest α value on each edge, and form the final architecture. However, this method leads to a problem: the architecture found is very cluttered because each node is connected to all previous nodes. Therefore, here we follow the previous neural network architecture search method. For each node in the architecture, we only keep the top two best operations that came from the previous nodes [11][12][4]. We define the quality of

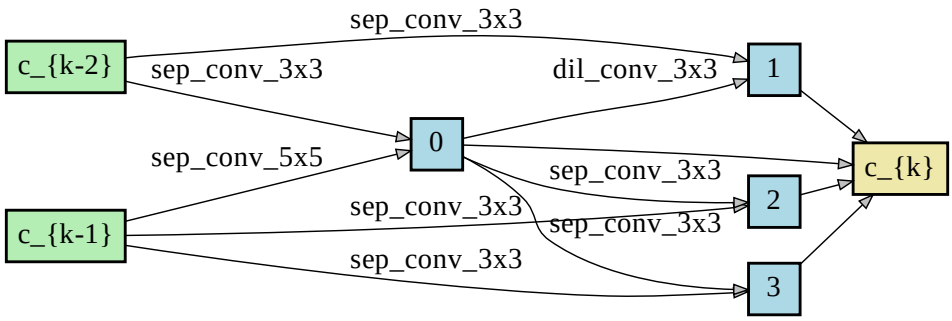

Fig. 2.9. Example of a cell that only top two best operations are kept

an operation like this (a greater value means the operation is better):

$$\frac{exp({\alpha_o}^{(i,j)})}{\sum_{o' \in S} exp({\alpha_{o'}}^{(i,j)})} \tag{2.18}$$

Take Fig. 2.9 as an example to illustrate this process more clearly. This cell has a total of 7 nodes, c_{k-1}, c_{k-2} are input nodes, node0, node1, node2, node3 are intermediate nodes, and c_{k} is the output node [4]. It can be seen that, for each intermediate node, only two operations passed from the previous node are retained.For example, for node3, only the connection between node0 and c_{k-1} is preserved. If this filtering method is not adopted, node3 will be connected at all previous nodes, which will make the architecture too chaotic.

So far, we have introduced the Differentiable Architecture Search algorithm for Keyword Spotting in detail, summarized it in Alg. 2.1, and made a detailed description of each step in Alg. 2.1.

2.4 Concluding Remarks

This Chapter mainly introduces the entire process of our model implementation, including the macro design of the entire model, the pre-processing of input data, the design of candidate operations, and how to search the neural network architecture.

Chapter 3

Model Search and Design

Our overall work of the next stage is divided into two steps: searching out the model and evaluating the model (training and testing the model). This chapter mainly introduces the work of the first part, mainly introduce how to use the DAS_KWS algorithm to search out specific models for keyword spotting.

The remainder of this chapter is organized as follows. Section 4.1 specifically introduces the setup process before the experiment, including application scenarios, data set split, data preprocessing, etc. Section 3.2 specifically introduces the design of the top-level architecture of our model and the structural design inside the cell. Section 3.3 shows eight convolutional neural network models we designed for keyword spotting, and details the model architecture, the number of various layers, and the total parameter size of the model.

3.1 Experimental Setup

We use Google Commands Dataset to search the neural network architecture [16]. Google Commands Dataset is a dataset published by Google specifically for keyword spotting. It contains 65,000 one-second audio utterances and contains 30 different words. For each word, 2,000 to 3,000 different people's pronunciations were collected. These thirty words include: 'bed', 'bird', 'cat', 'dog', 'down', 'eight', 'five', 'four', 'go', 'happy', 'house', 'left', 'marvin', 'nine', 'no', 'off',

'on', 'one', 'right', 'seven', 'Sheila', 'six', 'stop', 'three', 'tree', 'two', 'up', 'wow', 'yes', 'zero'. In addition, the data set also includes background noise samples, a total of six types: noise when doing the dishes, noise of miaowing, noise when exercising bike, pink noise, noise of running tap and white noise.

Our task is to discriminate among 12 classes: "yes," "no," "up," "down," "left," "right," "on," "off," "stop," "go", unknown, or silence. These words are the vocabulary most commonly used for voice control on mobile devices and have great practical significance. We divide the entire Google Commands Dataset into three parts, 40% for the training set, 40% for the validation set, and 20% for the test set. The three data sets are used as follows: In this chapter, we use the training set (40%) and the validation set (40%) to search the model's architecture to obtain several specific models. The specific use of training set and validation set has been explained in detail in the algorithm of Chapter2. After obtaining the specific models, the training set (40%) and the validation set (40%) are combined into a new training set (80%) to train the model, and the test set is used to evaluate the models. Note that when searching the neural network architecture or training the neural network, the test set is not involved.

To generate training data, we followed Tang's preprocessing approaches[2]. This is to allow a more rigorous comparison of the performance of our models, to ensure that the differences in model performance are not caused by different ways of processing the input data. First, at each epoch, we add background noise to the audio samples with 80% probability, and the choice of noise is completely random. Next, we perform random t millisecond time shifts on the audio samples to increase the difficulty of keyword spotting, where t satisfies a uniform distribution from -100 to 100. After that, we use a 20Hz / 4kHz filter to filter out frequency bands that the human ear cannot recognize. Finally, we follow the input preprocessing method introduced in Section. 2.2 to convert audio samples into $1 \times 101 \times 40$ MFCC features.

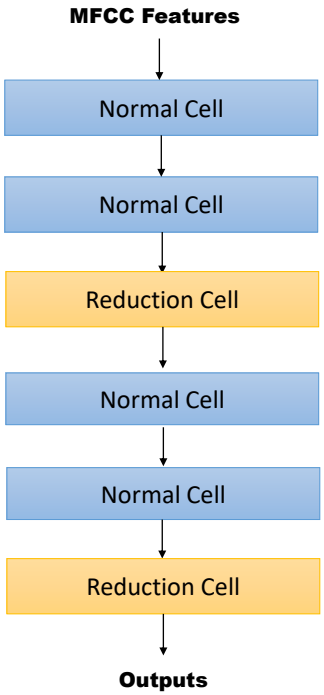

Fig. 3.1. Top-most architecture of DAS_KWS

3.2 Architecture Design

Our entire model architecture is designed like this. In order to control the size of the model, the top-most architecture is made up of six cells stacked, each two normal cells followed by a reduction cell, as shown in Fig. 3.1.

The definition of normal cells and reduction cells follows DARTS[4]. Normal cells ensure that their output spatial dimension is the same as the input, while reduction cells double the number of channels as their output spatial dimension. During the differentiable architecture search, the architecture weights α are kept as $(\alpha_{normal}, \alpha_{reduction})$, while α_{normal} is shared by all normal cells and $\alpha_{reduction}$ is shared by all reduction cells[4].

Both normal and reduction cells include 7 nodes, as shown in Fig. 3.2. The blue nodes c_{k-1}, c_{k-2} are input nodes, the green nodes node0, node1, node2, node3 are intermediate nodes, and the gray node c_{k} is the output node. For input nodes, the input is the cell outputs in the previous two layers. For output nodes, the

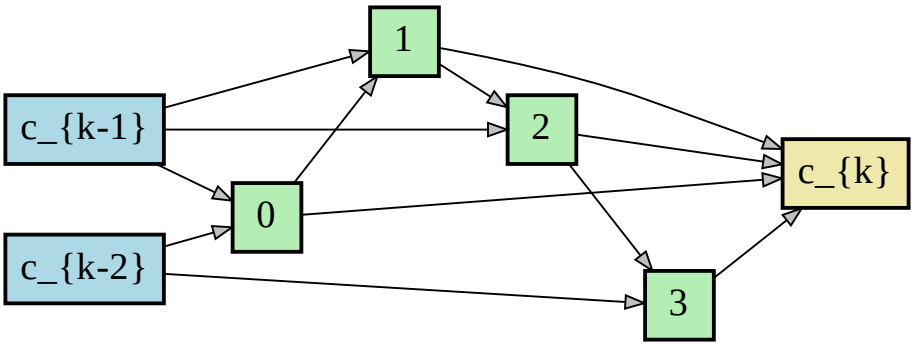

Fig. 3.2. Cell Design

output is concatenation of all the intermediate nodes. For the intermediate nodes, we will search for the edges between the node and all its previous nodes, determine the operation on each edge, and only retain the best two edges, according to the judgment method introduced in the Equation. 2.18.

All models in this thesis will be built according to the architecture described in this section. The structure of cells all follows Fig. 3.2. And after the searched cells are obtained, they are stacked using the method shown in Fig. 3.1 to obtain the final model.

3.3 Model search

In this section, we will show all our models searched by DAS_KWS algorithm. For architecture search, we consider the following 14 candidate operations for each edge: 3×3, 5×5, 7×7 and 9×9 separable convolutions, 3×3, 5×5, 7×7 and 9×9 dilated convolutions, 3×3 and 5×5 residual blocks, max pooling, average pooling, identity and skip connections. All operations are of stride one (if it has a stride). For each convolutional operation, we follow the order of Relu-Conv-BN. For each dilated convolution, the dilation parameter is set to 2. And each separable convolution should be applied twice to form an operation.

The searched normal cells and reductions will be stacked in the structure shown in Fig. 3.1 to get the final model. Therefore, in this section, we only show the specific structure of normal cells and reduction cells of each model.

We execute both first-order DAS_KWS and second-order DAS_KWS algorithms and get four models for keyword spotting respectively. In order to prove the robustness of our method, the model architecture search process does not contain a very extensive hyper parameters tuning process. In addition, these eight models were also selected at random, not based on test results. The internal schematic diagrams of the cells of these first-order models are shown from Fig. 3.3 to Fig. 3.6. For these first-order models, they are named as 'DASF1' to 'DASF4'. The detailed parameters of the first-order models are shown from Table. 3.1 to Table. 3.4. The number of each operation listed in the tables and the total number of parameters of the model refer to the model formed by six cells.

For these second-order models, they are named as 'DASS1' to 'DASS4', which are shown from Fig. 3.7 to Fig. 3.10. The detailed parameters of the second-order models are shown from Table. 3.5 to Table. 3.8.

layer type	m	r	n
sep-conv $\times 4$	5	5	16
dil-conv $\times 4$	3	3	16
max-pool $\times 40$	3	3	16
-	-	-	-
-	-	-	-
Number of Parameters	84K		

TABLE 3.1
PARAMETERS FOR DASF1

layer type	m	r	n
dil-conv $\times 4$	3	3	16
dil-conv $\times 2$	5	5	16
residual block $\times 2$	3	3	16
max-pool $\times 38$	3	3	16
sep-conv $\times 2$	5	5	16
Number of Parameters	148K		

TABLE 3.2
PARAMETERS FOR DASF2

layer type	m	r	n
sep-conv $\times 4$	3	3	16
dil-conv $\times 2$	3	3	16
dil-conv $\times 8$	5	5	16
max-pool $\times 34$	3	3	16
-	-	-	-
Number of Parameters	98K		

TABLE 3.3
PARAMETERS FOR DASF3

layer type	m	r	n
dil-conv $\times 12$	5	5	16
sep-conv $\times 4$	3	3	16
sep-conv $\times 8$	7	7	16
max-pool $\times 16$	3	3	16
average-pool $\times 8$	3	3	16
Number of Parameters	149K		

TABLE 3.4
PARAMETERS FOR DASF4

layer type	m	r	n
dil-conv $\times 14$	5	5	16
sep-conv $\times 4$	3	3	16
max-pool $\times 20$	3	3	16
skip-connect $\times 10$	-	-	-
-	-	-	-
Number of Parameters	105K		

TABLE 3.5
PARAMETERS FOR DASS1

layer type	m	r	n
sep-conv $\times 12$	7	7	16
sep-conv $\times 8$	5	5	16
sep-conv $\times 4$	3	3	16
average-pool $\times 16$	3	3	16
skip-connect $\times 8$	-	-	-
Number of Parameters	207K		

TABLE 3.6
PARAMETERS FOR DASS2

layer type	m	r	n
sep-conv $\times 4$	3	3	16
sep-conv $\times 12$	5	5	16
sep-conv $\times 10$	7	7	16
max-pool $\times 22$	3	3	16
-	-	-	-
-	-	-	-
Number of Parameters	197K		

TABLE 3.7
PARAMETERS FOR DASS3

layer type	m	r	n
dil-conv $\times 10$	5	5	16
dil-conv $\times 2$	3	3	16
sep-conv $\times 4$	5	5	16
sep-conv $\times 20$	7	7	16
max-pool $\times 10$	3	3	16
skip-connect $\times 2$	-	-	-
Number of Parameters	243K		

TABLE 3.8
PARAMETERS FOR DASS4

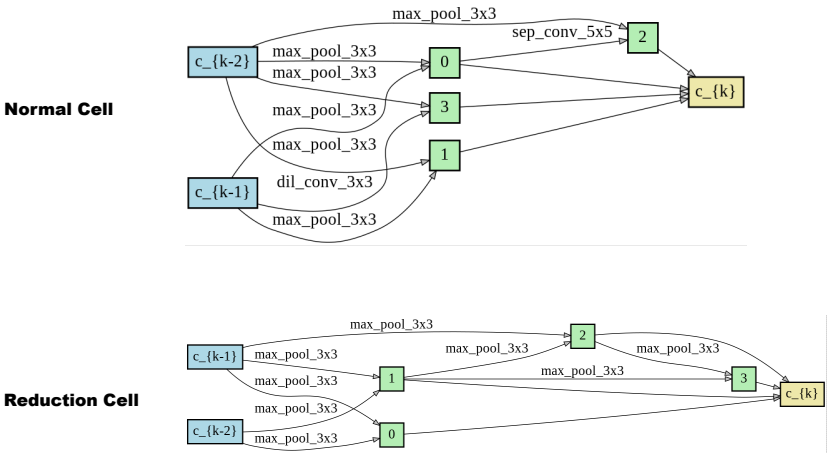

Fig. 3.3. Model DASF1 [4]

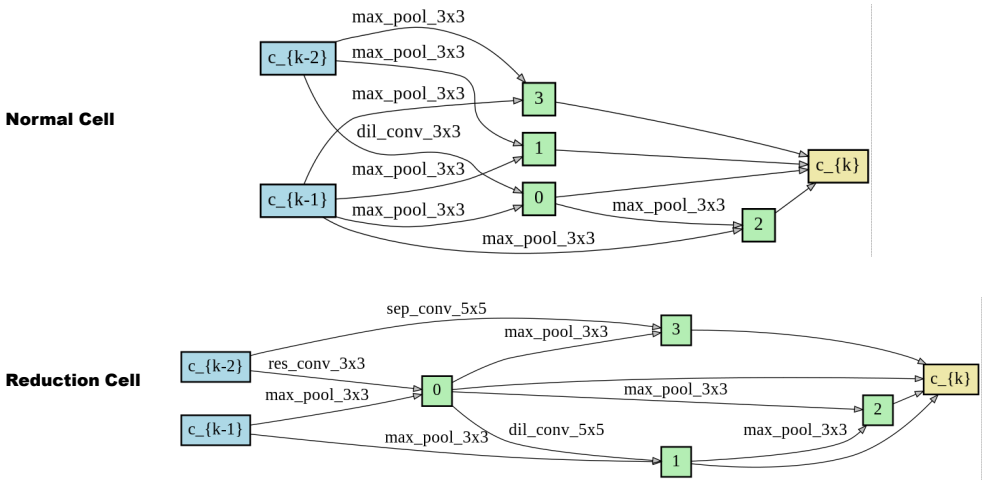

Fig. 3.4. Model DASF2 [4]

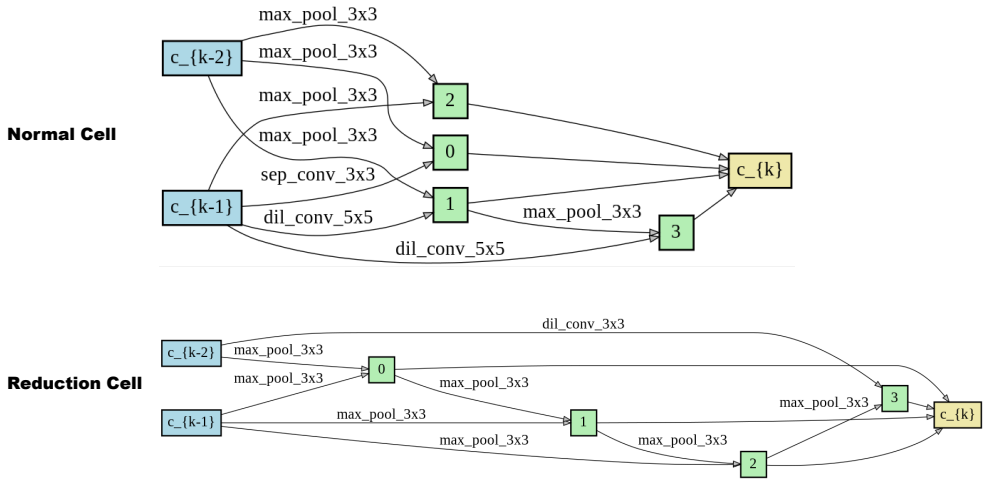

Fig. 3.5. Model DASF3 [4]

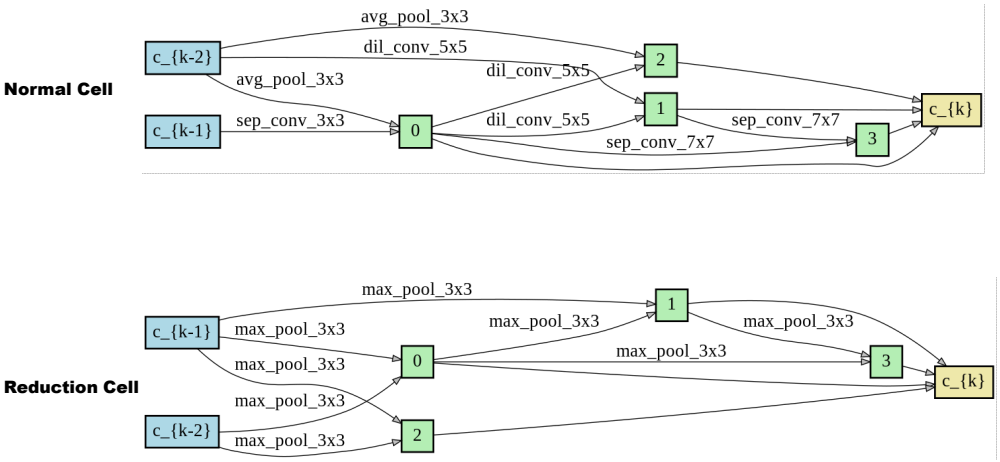

Fig. 3.6. Model DASF4 [4]

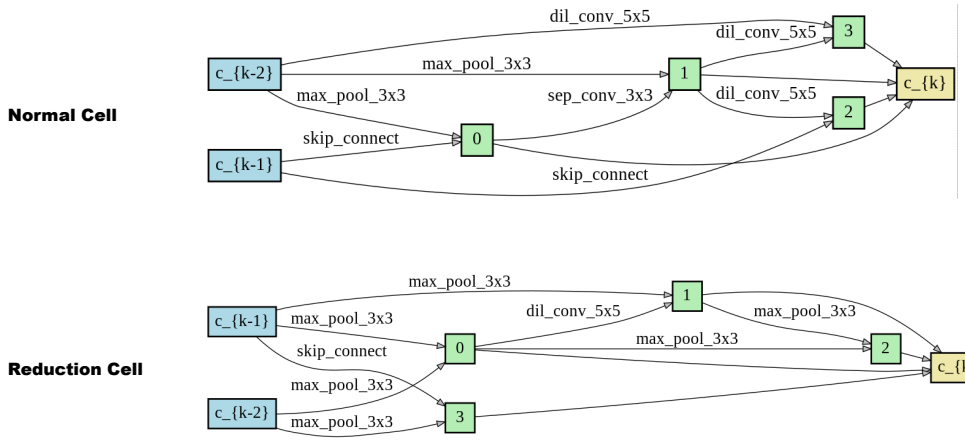

Fig. 3.7. Model DASS1 [4]

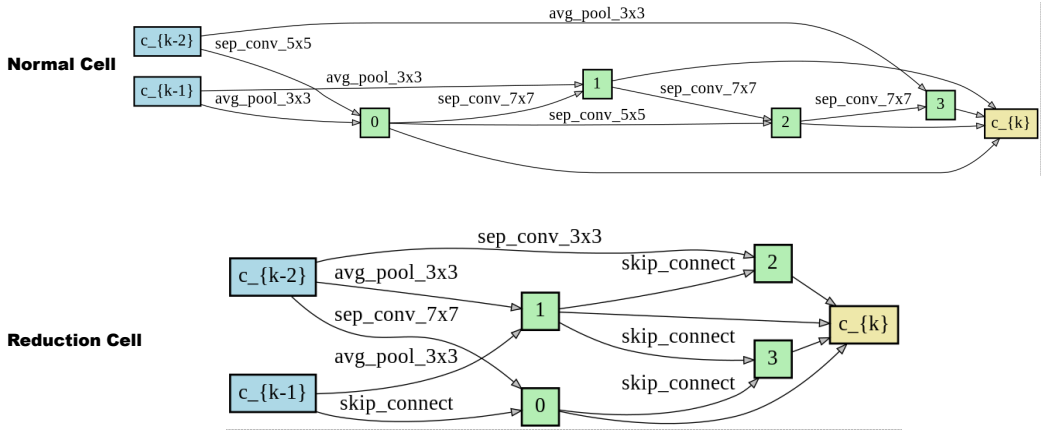

Fig. 3.8. Model DASS2 [4]

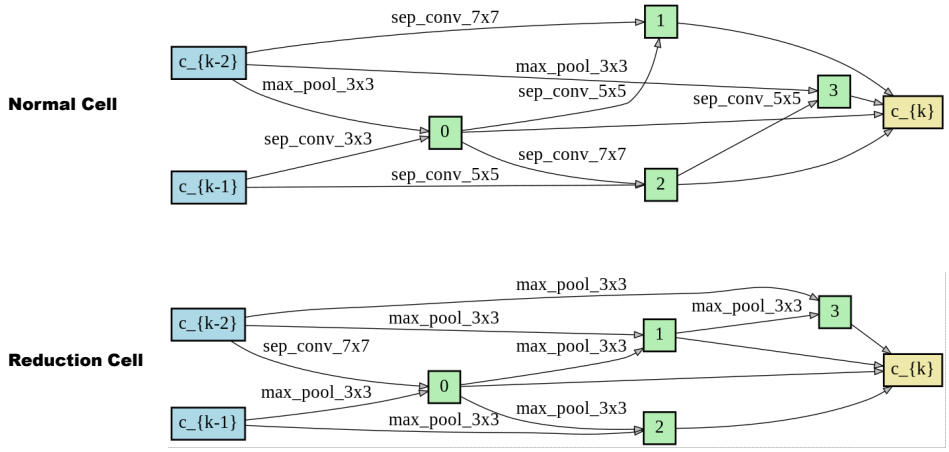

Fig. 3.9. Model DASS3 [4]

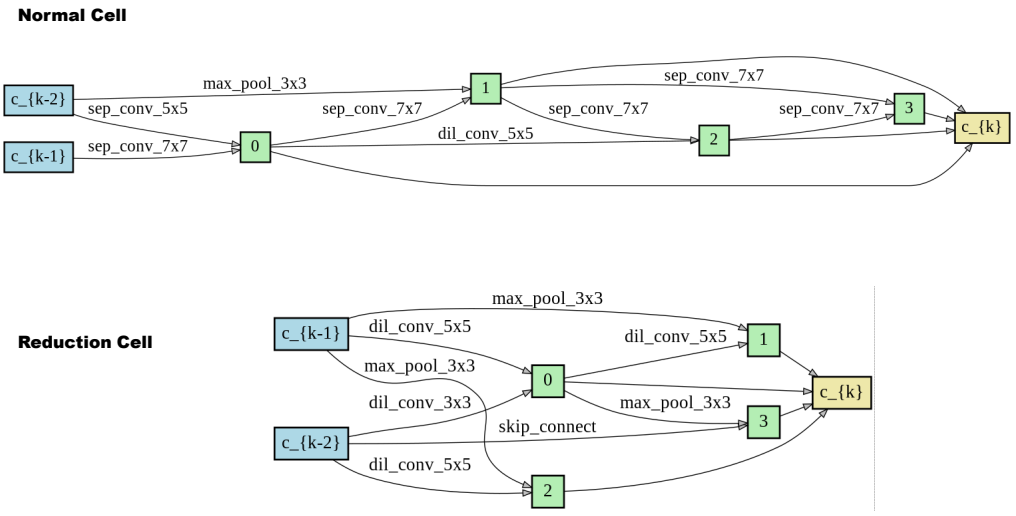

Fig. 3.10. Model DASS4 [4]

3.4 Conclusions

In this chapter, eight convolutional neural network models for keyword spotting are proposed, and their model architecture and parameter size are explained in detail. These eight models will be compared with the state-of-the-art models on keyword spotting.

Chapter 4

Evaluation

In Chapter 3, we propose eight convolutional neural network models for keyword spotting. In this chapter, we evaluate these models through tests.

The remainder of this chapter is organized as follows. Section 4.1 follows Section 3.1 of Chapter3, and introduces experimental setup in detail, including experimental motivation, data set selection, and so on. In section 4.2, we conduct evaluations on the Google Commands Dataset and compare the performance of our proposed approaches with other algorithms. We conclude this chapter in section 4.3.

4.1 Experimental Setup

The experimental setup follows the setup of the previous chapter. In the previous chapter, 10 vocabulary frequently used for voice control have been selected from Google Commands Dataset: “yes,” “no,” “up,” “down,” “left,” “right,” “on,” “off,” “stop,” “go”. This set of keywords is named "control keywords". These data is divided into a 40% training set, a 40% validation set, and a 20% test set. In the previous chapter, a training set and a validation set were used to design the architecture of the model. After obtaining the models, we merge the training set and the validation set into a new training set (80%), use this new training set to train DASF1 to DASF4 models and DASS1 to DASS4 models from scratch for 100 epochs, and

then use the test set (20%) to test. There are two reasons for this design:

- The test set should not be used at any step until the final test. So both when searching for the model architecture and when training the model, only the training set and validation set are used.

- When training a model, initialize the model's weights and start training from scratch. The purpose of this is: when searching the model structure, not only the architecture weights α are continuously optimized, but also the neural network weights w are learnt. What we want to evaluate is the designed convolutional neural network architecture, so we should ignore the w learnt before and train from scratch.

In addition to evaluating our models by keyword spotting accuracy, we also want to evaluate the transferability of our models. The DASF and DASS models are designed based on this set of "control keywords". We need to test whether the DASF and DASS models can be transferred and applied to the recognition of another set of keywords.

We selected another set of keywords from Google Commands Dataset, a total of ten, including: “zero,” “one,” “two,” “three,” “four,” “five,” “six,” “seven,” “eight,” “nine”[16]. This set of keywords is named "numerical keywords". These two sets of keywords, "control keywords" and "numerical keywords", are shown in Table 4.1.

We will also use the data set consisting of "numerical keywords" to evaluate the DASF and DASS models to prove the transferability of these models. The experimental method is similar to testing the "control keywords" data set. The "numerical keywords" data set is divided into a training set (80%) and a test set (20%). The training set is used to train the neural network from scratch, and then the test set is used to test.

The process of data generation and data preprocessing is exactly the same as explained in the previous chapter.

TABLE 4.1
KEYWORD SETS

Control Keywords	Numerical Keywords
yes	zero
no	one
up	two
down	three
left	four
right	five
on	six
off	seven
stop	eight
go	nine

4.2 Performance Evaluation

Based on the data set generated by 10 control keywords, we train DASF1 to DASF4, DASS1 to DASS4 models from scratch, and evaluate the model with test accuracy. For model training, we used gradient descent with a momentum of 0.9, learning rate of 0.025, batch size of 64 and weight decay of 3×10^{-4}. Our models are trained for a total of 100 epochs.

For the comparison model, we choose the classic CNN model of Sainath and the ResNet-based model of Tang, as shown in Table. 2.1, Table. 2.2, Table. 2.3, Table. 2.4 and Table. 2.5. The trad-fpool3 model is Sainath's base model; and tpool2 is the most accurate variant of Sainath's models [1]. The res15 and res26 model are top two most accurate variant of Tang's models, and res8-narrow is the most compact variant of Tang's models [2].

When training these baseline models, we fine-tuned the hyper parameters based on the open source code (https://github.com/castorini/honk). After getting the best-performing hyper parameters, we repeat the training-test process five times to get the final test results. Each baseline model is trained for 100 epochs. In addition, for easy comparison, we use the same data preprocessing method as Tang, so we use the experimental results listed in Tang's paper as the reference result as well [2].

All DASF models, DASS models, and baseline models are trained and tested for five times, so the reported test accuracy is the average of the five test results and shows a 95% confidence interval.

TABLE 4.2
TEST RESULTS ON CONTROL KEYWORDS

Model Architecture	Test Accuracy^{+} (%)	Parameters (K)
trad-fpool3*	$90.0\% \pm 0.31$	244
tpool2*	$93.1\% \pm 0.67$	7400
res15_retrain*	$93.2\% \pm 0.52$	238
res8-narrow_retrain*	$88.7\% \pm 0.98$	20
res26_retrain*	$92.9\% \pm 0.33$	438
res15**	$95.8\% \pm 0.48$	238
res8-narrow**	$90.1\% \pm 0.98$	20
res26**	$95.2\% \pm 0.18$	438
DASF1***	$96.3\% \pm 0.47$	84
DASF2***	$96.1\% \pm 0.35$	148
DASF3***	$95.9\% \pm 0.33$	98
DASF4***	$96.4\% \pm 0.35$	149
DASS1***	$96.6\% \pm 0.39$	105
DASS2***	$96.2\% \pm 0.37$	207
DASS3***	$96.4\% \pm 0.40$	197
DASS4***	$96.4\% \pm 0.37$	243

* Obtained by repeating training the model for 5 times with different random seeds using the code publicly released by the authors, after fine-tuning the hyperparameters

** Obtained from the results in the authors' paper

*** Obtained by repeating training our models for 5 times using different random seeds

$^{+}$ Test accuracy of each model with 95% confidence intervals (across five trials)

The test accuracies on control keywords of our models and baseline models are presented in Table. 4.2. Notably, all DASF and DASS models achieved results better than the state-of-the-art (res15 model). Moreover, except for model DASS4, all DASF and DASS models have a smaller parameter size than the state-of-the-art. Although the parameter size of the res8-narrow model is much smaller than our DASF and DASS models, our model beats it by 6 to 7 percentage points in test accuracy. In addition, the DASF and DASS models also defeated Sainath's

trad-fpool3 and tpool2 models in both test accuracy and parameter size.

We also test the transferability of the DASF and DASS models by changing the dataset from control keywords to numerical keywords. All experimental setup, data pre-processing process, and hyper parameters used for training are the same as before.

TABLE 4.3
TEST RESULTS ON NUMERICAL KEYWORDS

Model Architecture	Test Accuracy^{+} (%)	Parameters (K)
trad-fpool3*	$90.7\% \pm 0.50$	244
tpool2*	$93.2\% \pm 0.62$	7400
res15_retrain*	$92.8\% \pm 0.66$	238
res8-narrow_retrain*	$87.9\% \pm 0.95$	20
res26_retrain*	$92.3\% \pm 0.72$	438
DASF1***	$97.5\% \pm 0.32$	84
DASF2***	$97.3\% \pm 0.30$	148
DASF3***	$97.1\% \pm 0.35$	98
DASF4***	$97.2\% \pm 0.22$	149
DASS1***	$97.4\% \pm 0.38$	105
DASS2***	$96.9\% \pm 0.32$	207
DASS3***	$97.4\% \pm 0.30$	197
DASS4***	$97.3\% \pm 0.27$	243

* Obtained by repeating training the model for 5 times with different random seeds using the code publicly released by the authors, after fine-tuning the hyperparameters

*** Obtained by repeating training our models for 5 times using different random seeds

$^{+}$ Test accuracy of each model with 95% confidence intervals (across five trials)

The test accuracies on numerical keywords of our models and baseline models are presented in Table. 4.3. All DASF and DASS models beat five baseline models in performance. Notably, on this set of numerical keywords, all DASF and DASS models (with more than 96.9% test accuracy) outperformed their performance on the control keywords set. This proves that our model based on control keywords can be transferable to other keywords.

4.3 Conclusions

In this chapter, we evaluate 4 DASF models and 4 DASS models designed in Chapter 3 by experiments. Two datasets were used: the control keywords dataset and the numerical keywords dataset. At the same time, we also introduce five baseline models for comparison, trad-fpool3, tpool2, res15, res8-narrow and res26, where res15 is the state-of-the-art model for keyword spotting. The test results show that: 4 DASF models and 4 DASS models can outperform the state-of-the-art model on two datasets, with a smaller parameter size. And the DASF and DASS models have a good transferability, they could transfer to other keywords and have comparable great performance.

Chapter 5

Conclusion and Future Work

In this thesis, we mainly worked on how to improve the accuracy of keyword spotting with smaller parameter sizes. The previous methods are to stack convolutional layers to form a model. When the parameter size is too large, the model is compacted by reducing the model depth or the number of feature maps. I first proposed the Differential Architecture Search Approach to solve the problem of keyword spotting. This approach optimizes the model from the architecture itself, which not only makes the model compact, but also improves the recognition accuracy.

Our approach is very robust. The DASF and DASS models designed are very different from each other in structure, but they all have similar and very competitive test results, all of which beat the state-of-the-art results. In addition, the model I proposed has great generality and transferability, which achieved state-of-the-art results on completely different keywords.

The future work should be focusing on exploring the connection ways between cells. The improvement in accuracy of our model mainly depends on the liberation of the internal connection relationship within the neural network. Within each cell, allow any two points to be connected, instead of connecting them sequentially with layers. However, in our model, the connection method of cell to cell is very fixed. We stack cells in a fixed order to form the final model. We should explore more possibilities for the connection between cells, and optimize on the top-level architecture.

References

[1] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[2] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5484–5488.

[3] S.-H. Tsang, "Review:dilatednet-dilated convolution (semantic segmentation)," https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5, accessed Nov 17, 2018.

[4] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[5] J. R. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous hidden markov modeling for speaker-independent word spotting," in *International Conference on Acoustics, Speech, and Signal Processing,*. IEEE, 1989, pp. 627–630.

[6] J. Wilpon, L. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden markov modeling techniques," in *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1991, pp. 309–312.

[7] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 4087–4091.

[8] S. R. Eddy, "What is a hidden markov model?" *Nature biotechnology*, vol. 22, no. 10, pp. 1315–1316, 2004.

[9] A. Churbanov and S. Winters-Hilt, "Implementing em and viterbi algorithms for hidden markov model in linear memory," *BMC bioinformatics*, vol. 9, no. 1, p. 224, 2008.

[10] J. Keshet, D. Grangier, and S. Bengio, "Discriminative keyword spotting," *Speech Communication*, vol. 51, no. 4, pp. 317–329, 2009.

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[12] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[13] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *arXiv preprint arXiv:1711.00436*, 2017.

[14] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[15] T. Véniat, O. Schwander, and L. Denoyer, "Stochastic adaptive neural architecture search for keyword spotting," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 2842–2846.

[16] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[17] C. Shan, J. Zhang, Y. Wang, and L. Xie, "Attention-based end-to-end models for small-footprint keyword spotting," *arXiv preprint arXiv:1803.10916*, 2018.

[18] Y. Wang, P. Getreuer, T. Hughes, R. F. Lyon, and R. A. Saurous, "Trainable frontend for robust and far-field keyword spotting," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5670–5674.

[19] T. Ganchev, N. Fakotakis, and G. Kokkinakis, "Comparative evaluation of various mfcc implementations on the speaker verification task," in *Proceedings of the SPECOM*, vol. 1, no. 2005, 2005, pp. 191–194.

[20] L. Muda, M. Begam, and I. Elamvazuthi, "Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques," *arXiv preprint arXiv:1003.4083*, 2010.

[21] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition," *Speech communication*, vol. 54, no. 4, pp. 543–565, 2012.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[23] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[24] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[25] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International conference on rough sets and knowledge technology*. Springer, 2014, pp. 364–375.

[26] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[27] E. Strubell, P. Verga, D. Belanger, and A. McCallum, "Fast and accurate entity recognition with iterated dilated convolutions," *arXiv preprint arXiv:1702.02098*, 2017.

[28] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[29] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.