Convolutional Neural Networks for Small-footprint Keyword Spotting

Tara N. Sainath, Carolina Parada

Google, Inc. New York, NY, U.S.A

{tsainath, carolinap}@google.com

10.21437/Interspeech.2015-352

Abstract

We explore using Convolutional Neural Networks (CNNs) for a small-footprint keyword spotting (KWS) task. CNNs are attractive for KWS since they have been shown to outperform DNNs with far fewer parameters. We consider two different applications in our work, one where we limit the number of multiplications of the KWS system, and another where we limit the number of parameters. We present new CNN architectures to address the constraints of each applications. We find that the CNN architectures offer between a 27-44% relative improvement in false reject rate compared to a DNN, while fitting into the constraints of each application.

1. Introduction

With the rapid development of mobile devices, speech-related technologies are becoming increasingly popular. For example, Google offers the ability to search by voice [1] on Android phones, while personal assistants such as Google Now, Apple's Siri, Microsoft's Cortana and Amazon's Alexa, all utilize speech recognition to interact with these systems. Google has enabled a fully hands-free speech recognition experience, known as "Ok Google" [2], which continuously listens for specific keywords to initiate voice input. This keyword spotting (KWS) system runs on mobile devices, and therefore must have a small memory footprint and low computational power.

The current KWS system at Google [2] uses a Deep Neural Network (DNN), which is trained to predict sub keyword targets. The DNN has been shown to outperform a Keyword/Filler Hidden Markov Model system, which is a commonly used technique for keyword spotting. In addition, the DNN is attractive to run on the device, as the size of the model can be easily adjusted by changing the number of parameters in the network.

However, we believe that alternative neural network architecture might provide further improvements for our KWS task. Specifically, Convolutional Neural Networks (CNNs) [3] have become popular for acoustic modeling in the past few years, showing improvements over DNNs in a variety of small and large vocabulary tasks [4, 5, 6].

CNNs are attractive compared to DNNs for a variety of reasons. First, DNNs ignore input topology, as the input can be presented in any (fixed) order without affecting the performance of the network [3]. However, spectral representations of speech have strong correlations in time and frequency, and modeling local correlations with CNNs, through weights which are shared across local regions of the input space, has been shown to be beneficial in other fields [7]. Second, DNNs are not explicitly designed to model translational variance within speech signals, which can exist due to different speaking styles [3]. More specifically, different speaking styles lead to formants being shifted in the frequency domain. These speaking styles require us to apply various speaker adaptation techniques to reduce feature variation. While DNNs of sufficient size could indeed capture translational invariance, this requires large networks with lots of training examples. CNNs on the other hand capture translational invariance with far fewer parameters by averaging the outputs of hidden units in different local time and frequency regions.

We are motivated to look at CNNs for KWS given the benefits CNNs have shown over DNNs with respect to improved performance and reduced model size [4, 5, 6]. In this paper, we look at two applications of CNNs for KWS. First, we consider the problem where we must limit the overall computation of our KWS system, that is parameters and multiplies. With this constraint, typical architectures that work well for CNNs and pool in frequency only [8], cannot be used here. Thus, we introduce a novel CNN architecture which does not pool but rather strides the filter in frequency, to abide within the computational constraints issue. Second, we consider limiting the total number of parameters of our KWS system. For this problem, we show we can improve performance by pooling in time and frequency, the first time this has been shown to be effective for speech without using multiple convolutional blocks [5, 9].

We evaluate our proposed CNN architectures on a KWS task consisting of 14 different phrases. Performance is measured by looking at the false reject (FR) rate at the operating threshold of 1 false alarm (FA) per hour. In the task where we limit multiplications, we find that a CNN which strides filters in frequency gives over a 27% relative improvement in FR over the DNN. Furthermore, in the task of limiting parameters, we find that a CNN which pools in time offers over a 41% improvement in FR over the DNN and 6% over the traditional CNN [8] which pools in frequency only.

The rest of this paper is as follows. In Section 2 we give an overview of the KWS system used in this paper. Section 3 presents different CNN architectures we explore, when limiting computation and parameters. The experimental setup is described in Section 4, while results comparing CNNs and DNNs is presented in Section 5. Finally, Section 6 concludes the paper and discusses future work.

2. Keyword Spotting Task

A block diagram of the DNN KWS system [2] used in this work is shown in Figure 1. Conceptually, our system consists of three components. First, in the feature extraction module, 40 dimensional log-mel filterbank features are computed every 25ms with a 10ms frame shift. Next, at every frame, we stack 23 frames to the left and 8 frames to the right, and input this into the DNN.

The baseline DNN architecture consists of 3 hidden layers with 128 hidden units/layer and a softmax layer. Each hidden layer uses a rectified linear unit (ReLU) nonlinearity. The softmax output layer contains one output target for each of the

words in the keyword phrase to be detected, plus a single additional output target which represents all frames that do not belong to any of the words in the keyword (denoted as 'filler' in Figure 1). The network weights are trained to optimize a cross-entropy criterion using distributed asynchronous gradient descent [10]. Finally, in the posterior handling module, individual frame-level posterior scores from the DNN are combined into a single score corresponding to the keyword. We refer the reader to [2] for more details about the three modules.

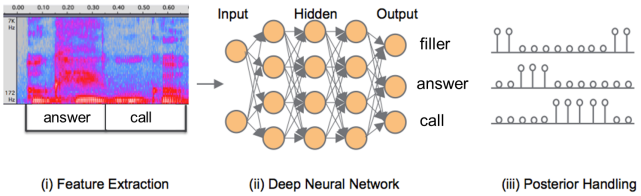


Figure 1: Framework of Deep KWS system, components from left to right: (i) Feature Extraction (ii) Deep Neural Network (iii) Posterior Handling

3. CNN Architectures

In this section, we describe CNN architectures as an alternative to the DNN described in Section 2. The feature extraction and posterior handling stages remain the same as Section 2.

3.1. CNN Description

A typical CNN architecture is shown in Figure 2. First, we are given an input signal $\mathbf{V} \in \Re^{t \times f}$, where t and f are the input feature dimension in time and frequency respectively. A weight matrix $\mathbf{W} \in \Re^{(m \times r) \times n}$ is convolved with the full input V. The weight matrix spans across a small local time-frequency patch of size $m \times r$, where $m <= t$ and $r <= f$. This weight sharing helps to model local correlations in the input signal. The weight matrix has n hidden units (i.e., feature maps). The filter can stride by a non-zero amount s in time and p in frequency. Thus, overall the convolutional operation produces n feature maps of size $\frac{(t-m+1)}{s} \times \frac{(f-r+1)}{v}$.

After performing convolution, a max-pooling layer helps to remove variability in the time-frequency space that exists due to speaking styles, channel distortions, etc. Given a pooling size of $p \times q$, pooling performs a sub-sampling operation to reduce the time-frequency space. For the purposes of this paper, we consider non-overlapping pooling as it has not shown to be helpful for speech [8]. After pooling, the time-frequency space has dimension $\frac{(t-m+1)}{s \cdot p} \times \frac{(f-r+1)}{v \cdot q}$.

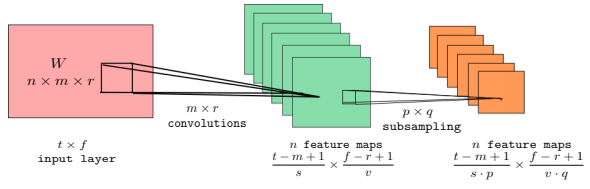


Figure 2: Diagram showing a typical convolutional network architecture consisting of a convolutional and max-pooling layer.

3.2. Typical Convolutional Architecture

An typical convolutional architecture that has been heavily tested and shown to work well on many LVCSR tasks [6, 11] is to use two convolutional layers. Assuming that the log-mel input into the CNN is $t \times f = 32 \times 40$, then typically the first layer has a filter size in frequency of $r = 9$. The architecture is less sensitive to the filter size in time, though a common practice is to choose a filter size in time which spans $2/3$ of the overall input size in time, i.e. $m = 20$. Convolutional multiplication is performed by striding the filter by $s = 1$ and $v = 1$ across both time and frequency. Next, non-overlapping max-pooling in frequency only is performed, with a pooling region of $q = 3$. The second convolutional filter has a filter size of $r = 4$ in frequency, and no max-pooling is performed.

For example, in our task if we want to keep the number of parameters below 250K, a typical architecture CNN architecture is shown in Table 1. We will refer to this architecture as `cnn-trad-fpool3` in this paper. The architecture has 2 convolutional, one linear low-rank and one DNN layer. In Section 5, we will show the benefit of this architecture for KWS, particularly the pooling in frequency, compared to a DNN.

However, a main issue with this architecture is the huge number of multiplies in the convolutional layers, which get exacerbated in the second layer because of the 3-dimensional input, spanning across time, frequency and feature maps. This type of architecture is infeasible for power-constrained small-footprint KWS tasks where multiplies are limited. Furthermore, even if our application is limited by parameters and not multiplies, other architectures which pool in time might be better suited for KWS. Below we present alternative CNN architectures to address the tasks of limiting parameters or multiplies.

type	m	r	n	p	q	Par.	Mul.
conv	20	8	64	1	3	10.2K	4.4M
conv	10	4	64	1	1	164.8K	5.2M
lin	-	-	32	-	-	65.5K	65.5K
dnn	-	-	128	-	-	4.1K	4.1K
softmax	-	-	4	-	-	0.5K	0.5K
Total	-	-	-	-	-	244.2K	9.7M

Table 1: CNN Architecture for `cnn-trad-fpool3`

3.3. Limiting Multiplies

Our first problem is to find a suitable CNN architecture where we limit the number of multiplies to 500K. After experimenting with several architectures, one solution to limit the number of multiplies is to have one convolutional layer rather than two, and also have the time filter span all of time. The output of this convolutional layer is then passed to a linear low-rank layer and then 2 DNN layers. Table 2, show a CNN architecture with only one convolutional layer, which we refer to as `cnn-one-fpool3`. For simplicity, we have omitted $s = 1$ and $v = 1$ from the Table. Notice by using one convolutional layer, the number of multiplies after the first convolutional layer is cut by a factor of 10, compared to `cnn-trad-fpool3`.

type	m	r	n	p	q	Params	Mult
conv	32	8	54	1	3	13.8K	456.2K
linear	-	-	32	-	-	19.8K	19.8K
dnn	-	-	128	-	-	4.1K	4.1K
dnn	-	-	128	-	-	16.4K	16.4K
softmax	-	-	4	-	-	0.5K	0.5K
Total	-	-	4	-	-	53.8K	495.6K

Table 2: CNN Architecture for `cnn-one-fpool3`

Pooling in frequency ($q = 3$) requires striding the filter by $v = 1$, which also increases multiplies. Therefore, we compare architectures which do not pool in frequency but rather stride the filter in frequency[1]. Table 3 shows the CNN architecture when we have a frequency filters of size $r = 8$ and stride the filter by $v = 4$ (i.e., 50% overlap), as well as when we stride by $v = 8$ (no overlap). We will refer to these as `cnn-one-fstride4` and `cnn-one-fstride8` respectively. For simplicity, we have omitted the linear and DNN layers, as they are the same as Table 2. Table 3 shows that if we stride the filter by $v > 1$ we reduce multiplies, and can therefore increase the number of hidden units n to by 3-4 times larger than the `cnn-one-fpool3` architecture in Table 2.

model	m	r	n	s	v	Params	Mult
(a)	32	8	**186**	1	4	47.6K	428.5K
(b)	32	8	**336**	1	8	86.6K	430.1K

Table 3: CNN for (a) `cnn-one-fstride4` and (b) `cnn-one-fstride8`

3.4. Limiting Parameters

One of the issue with the models presented in the previous section was that when keeping multiplies fixed, the number of parameters of the model remains much smaller than 250K. However, increasing CNN parameters often leads to further improvements [6]. In other applications, we would like to design a model where we keep the number of parameters fixed, but allow multiplications to vary. In this section, we explore CNN architectures different than `cnn-trad-fpool3` where we limit model size to be 250K but do not limit the multiplies.

On way to improve CNN performance is to increase feature maps. If we want to increase feature maps but keep parameters fixed, we must explore sampling in time and frequency. Given that we already pool in frequency in `cnn-trad-fpool3`, in this section we explore sub-sampling in time.

Conventional pooling in time has been previously explored for acoustic modeling [4, 8], but has not shown promise. Our rationale is that in acoustic modeling, the sub-word units (i.e., context-dependent states) we want to classify occur over a very short time-duration (i.e., 10-30ms). Therefore, pooling in time is harmful. However, in KWS the keyword units occur over a much longer time-duration (i.e., 50-100ms). Thus, we explore if we can improve over `cnn-trad-fpool3` by sub-sampling the signal in time, either by striding or pooling. It should be noted that pooling in time helps when using multiple convolutional sub-networks [5, 9]. However, this type of approach increases number of parameters and is computationally expensive for our KWS task. To our knowledge, this is the first exploration of conventional sub-sampling in time with longer acoustic units.

3.4.1. *Striding in Time*

First, we compare architectures where we stride the time filter in convolution by an amount of $s > 1$. Table 4 shows different CNN architectures where we change the time filter stride s. We will refer to these architectures as `cnn-tstride2`, `cnn-tstride4` and `cnn-tstride8`. For simplicity, we have omitted the DNN layer and certain variables held constant for all experiments, namely frequency stride $v = 1$ and pool in time $p = 1$. One thing to notice is that as we increase the time filter stride, we can increase the number of feature maps n such that the total number of parameters remains constant. Our hope is that sub-sampling in time will not degrade performance, while increasing the feature maps will improve performance.

model	layer	m	r	n	s	q	Params
`cnn-tstride2`	conv	16	8	78	**2**	3	10.0K
	conv	9	4	78	1	1	219.0K
	lin	-	-	32	-	-	20.0K
`cnn-tstride4`	conv	16	8	100	**4**	3	12.8K
	conv	5	4	78	1	1	200.0K
	lin	-	-	32	-	-	25.6K
`cnn-tstride8`	conv	16	8	126	**8**	3	16.1K
	conv	5	4	78	1	1	190.5K
	lin	-	-	32	-	-	32.2K

Table 4: CNNs for Striding in Time

3.4.2. *Pooling in Time*

An alternative to striding the filter in time is to pool in time, by a non-overlapping amount. Table 5 shows configurations as we vary the pooling in time p. We will refer to these architectures as `cnn-tpool2` and `cnn-tpool4`. For simplicity, we have omitted certain variables held constant for all experiments, namely time and frequency stride $s = 1$ and $v = 1$. Notice that by pooling in time, we can increase the number of feature maps n to keep the total number of parameters constant.

model	layer	m	r	n	p	q	Params
`cnn-tpool2`	conv	21	8	94	**2**	3	5.6M
	conv	6	4	94	1	1	1.8M
	lin	-	-	32	-	-	65.5K
`cnn-tpool3`	conv	15	8	94	**3**	3	7.1M
	conv	6	4	94	1	1	1.6M
	lin	-	-	32	-	-	65.5K

Table 5: CNNs for Pooling in Time

4. Experimental Details

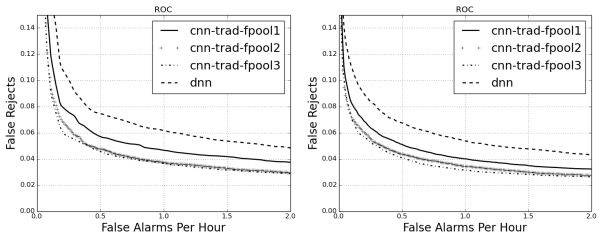
In order to compare the proposed CNN approaches to a baseline DNN KWS system, we selected fourteen phrases[2] and collected about 10K–15K utterances containing each of these phrases. We also collected a much larger set of approximately 396K utterances which do not contain any of the keywords and are thus used as 'negative' training data. The utterances were then randomly split into training, development, and evaluation sets in the ratio of 80:5:15, respectively.

Next, we created noisy training and evaluation sets by artificially adding car and cafeteria noise at SNRs randomly sampled between [-5dB, +10dB] to the clean data sets. Models are trained in noisy conditions, and evaluated in both clean and noisy conditions.

KWS performance is measured by plotting a receiver operating curve (ROC), which calculates the false reject (FR) rate per false alarm (FA) rate. The lower the FR per FA rate is the better. The KWS system threshold is selected to correspond to 1 FA per hour of speech on this set.

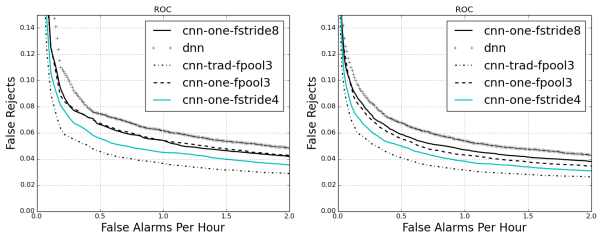
[1] Since the pooling region is small ($q = 3$), we have found that we cannot pool if we stride the frequency filter by $v > 1$

[2] The keyword phrases are: 'answer call', 'decline call', 'email guests', 'fast forward', 'next playlist', 'next song', 'next track', 'pause music', 'pause this', 'play music', 'set clock', 'set time', 'start timer', and 'take note'.



(a) Results on Clean (b) Results on Noisy

Figure 3: ROCs for DNN vs. CNNs with Pooling in Frequency



(a) Results on Clean (b) Results on Noisy

Figure 4: ROCs for DNN vs. CNN, Matching Multiplies

5. Results

5.1. Pooling in Frequency

First, we analyze how a typical CNN architecture, as described in Section 3.2 compares to a DNN for KWS. While the number of parameters is the same for both the CNN and DNN (250K), the number of multiplies for the CNN is 9M. To understand the behavior of frequency pooling for the KWS task, we compare CNN performance when we do not pool $p = 1$, as well as pool by $p = 2$ and $p = 3$, holding the number of parameters constant for all three experiments.

Figures 3a and 3b show that for both clean and noisy speech, CNN performance improves as we increase the pooling size from $p = 1$ to $p = 2$, and seems to saturate after $p = 3$. This is consistent with results observed for acoustic modeling [8]. More importantly, the best performing CNN (`cnn-trad-fpool3`) shows improvements of over 41% relative compared to the DNN in clean and noisy conditions at the operating point of 1 FA/hr. Given these promising results, we next compare CNN and DNN performance when we constrain multiplies and parameters.

5.2. Limiting Multiplies

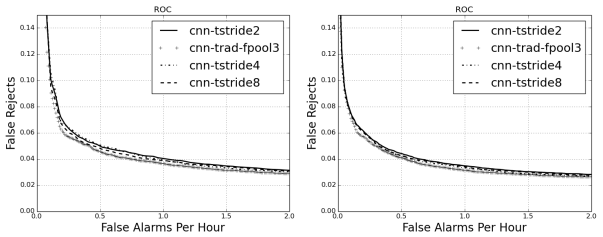
In this section, we compare various CNN architectures described in Section 3.3 when we limit the number of multiplies to be 500K. Figures 4a and 4b show results for both clean and noisy speech. The best performing system is `cnn-one-fstride4`, where we stride the frequency filter with 50% overlap but do not pool in frequency. This gives much better performance than `cnn-one-fstride8` which has a non-overlapping filter stride. Furthermore, if offers improvements over `cnn-one-fpool3`, which pools in frequency. While pooling in frequency is helpful, as demonstrated in Section 5.1, it is computationally expensive and thus we must reduce feature maps drastically to limit computation. Therefore, if we are in a situation where multiplies are limited, the preferred CNN architecture is to stride the filter with overlap.

The best performing system `cnn-one-fstride4` gives a 27% relative improvement in clean and 29% relative improvement in noisy over the DNN at the operating point of 1 FA/hr.

5.3. Limiting Parameters

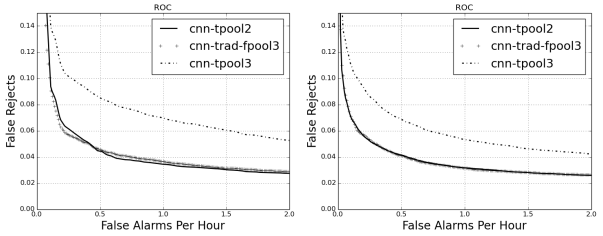
In this section, we compare CNN architectures where we match number of multiplies to the best performing system in Section 5.1, namely `cnn-trad-fpool3`. Figures 5a and 5b show performance of different architectures when we stride the convolutional filter in frequency, as described in Section 3.4.1. All architectures which stride the filter in time have slightly worse performance than `cnn-trad-fpool3` which does not stride the time filter.

In comparison, Figures 6a and 6b compare performance when we pool the convolutional filter in time. System `cnn-tpool2`, which pools in time by $p = 2$, is the best performing system. These results indicate that pooling in time, and therefore modeling the relationship between neighboring frames before sub-sampling, is more effective than striding in time which a-priori selects which neighboring frames to filter. In addition, when predicting long keyword units, pooling in time gives a 6% relative improvement over `cnn-trad-fpool3` in clean, but has a similar performance to `cnn-trad-fpool3` in noisy. In addition, `cnn-tpool2` shows a 44% relative improvement over the DNN in clean and 41% relative improvement in noisy. To our knowledge, this is the first time pooling in time without sub-networks has shown to be helpful for speech tasks.



(a) Results on Clean (b) Results on Noisy

Figure 5: ROC Curves comparing CNN with Striding in Time



(a) Results on Noisy (b) Results on Noisy

Figure 6: ROC Curves comparing CNN with Pooling in Time

6. Conclusions

In this paper, we explore CNNs for a KWS task. We compare CNNs to DNNs when we limit number of multiplies or parameters. When limiting multiplies, we find that shifting convolutional filters in frequency results in over a 27% relative improvement in performance over the DNN in both clean and noisy conditions. When limiting parameters, we find that pooling in time results in over a 41% relative improvement over a DNN in both clean and noisy conditions.

7. References

[1] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, ""Your word is my command": Google search by voice: A case study," in *Advances in Speech Recognition*, A. Neustein, Ed. Springer US, 2010, pp. 61–90.

[2] G. Chen, C. Parada, and G. Heigold, "Small-footprint Keyword Spotting using Deep Neural Networks," in *Proc. ICASSP*, 2014.

[3] Y. LeCun and Y. Bengio, "Convolutional Networks for Images, Speech, and Time-series," in *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[4] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn, "Applying Convolutional Neural Network Concepts to Hybrid NN-HMM Model for Speech Recognition," in *Proc. ICASSP*, 2012.

[5] L. Toth, "Combining Time-and Frequency-Domain Convolution in Convolutional Neural Network-Based Phone Recognition," in *Proc. ICASSP*, 2014.

[6] T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep Convolutional Neural Networks for LVCSR," in *Proc. ICASSP*, 2013.

[7] Y. LeCun, F. Huang, and L. Bottou, "Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting," in *Proc. CVPR*, 2004.

[8] T. Sainath, B. Kingsbury, G. Saon, H. Soltau, A. Mohamed, G. Saon, and B. Ramabhadran, "Deep Convolutional Networks for Large-Scale Speech Tasks," *Elsevier Special Issue in Deep Learning*, 2014.

[9] K. Vesely, M. Karafiat, and F. Grezel, "Convolutive Bottleneck Network Features for LVCSR," in *Proc. ASRU*, 2011.

[10] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng, "Large Scale Distributed Deep Networks," in *Proc. NIPS*, 2012.

[11] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks," in *to appear in Proc. ICASSP*, 2015.