Convolutional Neural Networks for Speech Recognition

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu

***Abstract*—Recently, the hybrid deep neural network (DNN)-hidden Markov model (HMM) has been shown to significantly improve speech recognition performance over the conventional Gaussian mixture model (GMM)-HMM. The performance improvement is partially attributed to the ability of the DNN to model complex correlations in speech features. In this paper, we show that further error rate reduction can be obtained by using convolutional neural networks (CNNs). We first present a concise description of the basic CNN and explain how it can be used for speech recognition. We further propose a limited-weight-sharing scheme that can better model speech features. The special structure such as local connectivity, weight sharing, and pooling in CNNs exhibits some degree of invariance to small shifts of speech features along the frequency axis, which is important to deal with speaker and environment variations. Experimental results show that CNNs reduce the error rate by 6%-10% compared with DNNs on the TIMIT phone recognition and the voice search large vocabulary speech recognition tasks.**

***Index Terms*—Convolution, convolutional neural networks, Limited Weight Sharing (LWS) scheme, pooling.**

I. Introduction

THE aim of automatic speech recognition (ASR) is the transcription of human speech into spoken words. It is a very challenging task because human speech signals are highly variable due to various speaker attributes, different speaking styles, uncertain environmental noises, and so on. ASR, moreover, needs to map variable-length speech signals into variable-length sequences of words or phonetic symbols. It is well known that hidden Markov models (HMMs) have been very successful in handling variable length sequences as well as modeling the temporal behavior of speech signals using a sequence of states, each of which is associated with a particular probability distribution of observations. Gaussian mixture models (GMMs) have been, until very recently, regarded as the most powerful model for estimating the probabilistic distribution of speech signals associated with each of these HMM states. Meanwhile, the generative training methods of GMM-HMMs have been well developed for ASR based on the popular expectation maximization (EM) algorithm. In addition, a plethora of discriminative training methods, as reviewed in [1], [2], [3], are typically employed to further improve HMMs to yield the state-of-the-art ASR systems.

Very recently, HMM models that use artificial neural networks (ANNs) instead of GMMs have witnessed a significant resurgence of research interest [4], [5], [6], [7], [8], [9], initially on the TIMIT phone recognition task with mono-phone HMMs for MFCC features [10], [11], [12], and shortly thereafter on several large vocabulary ASR tasks with triphone HMM models [6], [7], [13], [14], [15], [16]; see an overview of this series of studies in [17]. In retrospect, the performance improvements of these recent attempts have been ascribed to their use of "deep" learning, a reference both to the number of hidden layers in the neural network as well as to the abstractness and, by some accounts, psychological plausibility of representations obtained in the layers furthest removed from the input, which hearkens back to the appeal of ANNs to cognitive scientists thirty years ago. A great many other design decisions have been made in these alternative ANN-based models to which significant improvements might have been attributed.

Even without deep learning, ANNs are powerful discriminative models that can directly represent arbitrary classification surfaces in the feature space without any assumptions about the data's structure. GMMs, by contrast, assume that each data sample is generated from one hidden expert (i.e., a Gaussian) and a weighted sum of those Gaussian components is used to model the entire feature space. ANNs have been used for speech recognition for more than two decades. Early trials worked on static and limited speech inputs where a fixed-sized buffer was used to hold enough information to classify a word in an isolated speech recognition scheme [18], [19]. They have been used in continuous speech recognition as feature extractors, in both the TANDEM approach [20], [21] and in so-called bottleneck feature methods [22], [23], [24], and also as nonlinear predictors to aid the recognition of speech units [25], [26]. Their first successful application to continuous speech recognition, however, was in a manner that almost exactly parallels the use of GMMs now, i.e., as sources of HMM state posterior probabilities, given a fixed number of feature frames [27].

How do the recent ANN-HMM hybrids differ from earlier approaches? They are simply much larger. Advances in computing hardware over the last twenty years have played a signif-

Manuscript received October 11, 2013; revised February 04, 2014; accepted July 05, 2014. Date of publication July 16, 2014; date of current version nulldate. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Haizhou Li.

O. Abdel-Hamid and H. Jiang are with the Department of Electrical Engineering and Computer Science, Lassonde School of Engineering, York University, Toronto, ON M3J 1P3, Canada (e-mail: ossama@cse.yorku.ca; hj@cse.yorku.ca).

A.-r. Mohamed and G. Penn are with the Computer Science Department, University of Toronto, Toronto, ON M5S, Canada (e-mail: asamir@cs.utoronto.ca; gpenn@cs.utoronto.ca).

L. Deng and D. Yu are with Microsoft Research, Redmond, WA 98052 USA (e-mail: deng@microsoft.com; dongyu@microsoft.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASLP.2014.2339736

icant role in the advance of ANN-based approaches to acoustic modeling because training ANNs with so many hidden units on so many hours of speech data has only recently become feasible. The recent trend towards ANN-HMM hybrids began with using restricted Boltzmann machines (RBMs), which can take (temporally) subsequent context into account. Comparatively recent advances in learning through minimizing "contrastive divergence" [28] enable us to approximate learning with RBMs. Compared to conventional GMM-HMMs, ANNs can easily leverage highly correlated feature inputs, such as those found in much wider temporal contexts of acoustic frames, typically 9-15 frames. Hybrid ANN-HMMs also now often directly use log mel-frequency spectral coefficients without a decorrelating discrete cosine transform [29], [30], DCTs being largely an artifact of the decorrelated mel-frequency cepstral coefficients (MFCCs) that were popular with GMMs. All of these factors have had a significant impact upon performance.

This historical deconstruction is important because the premise of the present paper is that very wide input contexts and domain-appropriate representational invariance are so important to the recent success of neural-network-based acoustic models that an ANN-HMM architecture embodying these advantages can in principle outperform other ANN architectures of potentially unlimited depth for at least some tasks. We present just such a novel architecture below, which is based upon convolutional neural networks (CNNs) [31]. CNNs are among the oldest deep neural-network architectures [32], and have enjoyed great popularity as a means for handwriting recognition. A modification of CNNs will be presented here, called *limited weight sharing*, however, which to some extent impairs their ability to be stacked unboundedly deep. We moreover illustrate the application of CNNs to ASR in detail, and provide additional experimental results on how different CNN configurations may affect final ASR performance (Section V).

CNNs have been applied to acoustic modeling before, notably by [33] and [34], in which convolution was applied over windows of acoustic frames that overlap in time in order to learn more stable acoustic features for classes such as phone, speaker and gender. Weight sharing over time is actually a much older idea that dates back to the so-called time-delay neural networks (TDNNs) [35] of the late 1980s, but TDNNs had emerged initially as a competitor with HMMs for modeling time-variation in a "pure" neural-network-based approach. That purity may be of some value to the aforementioned cognitive scientists, but it is less so to engineers. As far as modeling time variations is concerned, HMMs do relatively well at this task; convolutional methods, i.e., those that use neural networks endowed with weight sharing, local connectivity and pooling (properties that will be defined below), are probably overkill, in spite of the initially positive results of [35]. We will continue to use HMMs in our model for handling variation along the time axis, but then apply convolution on the *frequency* axis of the spectrogram. This endows the learned acoustic features with a tolerance to small shifts in frequency, such as those that may arise from differing vocal tract lengths, and has led to a significant improvement over DNNs of similar complexity on TIMIT speaker-independent phone recognition, with a relative phone error rate reduction of about 8.5%. Learning invariant representations over frequency (or time) are notoriously more difficult for standard DNNs.

Deep architectures have considerable merit. They enable a model to handle many types of variability in the speech signal. The work of [29], [36] shows that the feature representations used in the upper hidden layers of DNNs are indeed more invariant to small perturbations in the input, regardless of their putative deep structural insight or abstraction, and in a manner that leads to better model generalization and improved recognition performance, especially under speaker and environmental variations. The more crucial question we have undertaken to answer is whether even better performance might be attainable if some representational knowledge that arises from a careful study of the empirical domain can be used to explicitly handle the variations in question.[1] Vocal tract length normalization (VTLN) is another very good example of this. VTLN warps the frequency axis based on a single learnable warping factor to normalize speaker variations in the speech signals, and has been shown [41], [16] to further improve the performance of DNN-HMM hybrid models when applied to the input features. More recently, the deep architecture taking the form of recurrent neural networks, even with unstacked single-layer variants, have been reported with very competitive error rates [42].

We first review the DNN and its use within the hybrid DNN-HMM architecture (Section II). Section III explains and elaborates upon the CNN architecture and its uses in speech recognition. Section IV presents limited weight sharing and the new CNN structure that incorporates it.

II. Deep Neural Networks: A Review

Generally speaking, a *deep neural network (DNN)* refers to a feedforward neural network with more than one hidden layer. Each hidden layer has a number of units (or neurons), each of which takes all outputs of the lower layer as input, multiplies them by a weight vector, sums the result and passes it through a non-linear activation function such as *sigmoid* or *tanh* as follows:

$$o_i^{(l)} = \sigma\left(\sum_j o_j^{(l-1)} w_{j,i}^{(l)} + w_{0,i}^{(l)}\right) \tag{1}$$

where $o_i^{(l)}$ denotes the output of the i-th unit in the l-th layer, $w_{j,i}^{(l)}$ denotes the connecting weight from the j-th unit in the layer $l-1$ to the i-th unit in the l-th layer, $w_{0,i}^{(l)}$ is a bias added to the i-th unit, and $\sigma(x)$ is the non-linear activation function. In this paper, we only consider the sigmoid function, i.e., $\sigma(x) = 1/(1+\exp(-x))$. For simplicity of notation, we can represent the above computation in the following vector form:

$$o_i^{(l)} = \sigma(\mathbf{o}^{(l-1)} \cdot \mathbf{w}_i^{(l)}) \tag{2}$$

where the bias term is absorbed in the column weight vector $\mathbf{w}_i^{(l)}$ by expanding the vector $\mathbf{o}^{(l-1)}$ with an extra dimension

[1]Portions of this research program have appeared in [37], [38] and [39]. There have also been important extensions of this work to larger vocabulary speech recognition tasks and to deep-learning models that retain some of the advantages presented here [39], [40].

of 1. Furthermore, all neuron activations in each layer can be represented in the following matrix form:

$$\mathbf{o}^{(l)} = \sigma(\mathbf{o}^{(l-1)}\mathbf{W}^{(l)}) \quad (l = 1, 2, \ldots, L-1) \tag{3}$$

where $\mathbf{W}^{(l)}$ denotes the weight matrix of the l-th layer, with ith column $\mathbf{w}_i^{(l)}$ for any i.

The first (bottom) layer of the DNN is the input layer and the topmost layer is the output layer. For a multi-class classification problem, the posterior probability of each class can be estimated using an output softmax layer:

$$y_i = \frac{\exp(o_i^{(L)})}{\sum_j \exp(o_j^{(L)})} \tag{4}$$

where $o_i^{(L)}$ is computed as $o_i^{(L)} = \mathbf{o}^{(L-1)} \cdot \mathbf{w}_i^{(L)}$.

In the hybrid DNN-HMM model, the DNN replaces the GMMs to compute the HMM state observation likelihoods. The DNN output layer computes the state posterior probabilities which are divided by the states' priors to estimate the observation likelihoods. In the training stage, forced alignment is first performed to generate a reference state label for every frame. These labels are used in supervised training to minimize the cross-entropy function, $Q(\{\mathbf{W}^{(l)}\}) = -\sum_i d_i \log y_i$, shown here for one training frame with i ranging over all target labels. The cross-entropy objective function aims at minimizing the discrepancy between the reference target $\mathbf{d}$ and the softmax DNN prediction $\mathbf{y}$.

The derivative of Q with respect to each weight matrix, $\mathbf{W}^{(l)}$, can be efficiently computed based on the well-known error back-propagation algorithm. If we use the stochastic gradient descent algorithm to minimize the objective function, for each training sample or mini-batch, each weight matrix update can be computed as:

$$\Delta\mathbf{W}^{(l)} = \epsilon \cdot \left(\mathbf{o}^{(l-1)}\right)' \mathbf{e}^{(l)} \quad (l = 1, 2, \ldots, L) \tag{5}$$

where ϵ is the learning rate and the error signal vector in the l-th layer, $\mathbf{e}^{(l)}$, is computed backwards from the sigmoid hidden unit as follows:

$$\mathbf{e}^{(L)} = \mathbf{d} - \mathbf{y} \tag{6}$$

$$\mathbf{e}^{(l)} = \left(\mathbf{e}^{(l+1)}\left(\mathbf{W}^{(l+1)}\right)'\right) \bullet \mathbf{o}^{(l)} \bullet \left(\mathbf{1} - \mathbf{o}^{(l)}\right) \quad (l = L-1, \ldots, 2, 1) \tag{7}$$

where $\bullet$ represents element-wise multiplication of two equally sized matrices or vectors.

Because of the increased model complexity of DNNs, a pretraining algorithm is often needed, which initializes all weight matrices prior to the above back-propagation algorithm, especially when the amount of training data is limited and when no constraints are imposed on the DNN weights (see [43] for more detailed discussions). One popular method to pretrain DNNs uses the restricted Boltzmann machine (RBM) as a building block. An RBM is a generative model that models the data's probability distribution. An RBM has a set of hidden units that are used to compute a better feature representation of the input data. After learning, all RBM weights can be used as a good initialization for one DNN layer. The weights are learned one layer at a time starting from the bottom hidden layer. The hidden activations computed using the learned weights are sent as input to another RBM that can be used to initialize another layer on top. The *contrastive divergence* algorithm is normally used to learn RBM weights; see [13] for more details.

III. Convolutional Neural Networks and Their Use in ASR

The convolutional neural network (CNN) can be regarded as a variant of the standard neural network. Instead of using fully connected hidden layers as described in the preceding section, the CNN introduces a special network structure, which consists of alternating so-called *convolution* and *pooling* layers.

A. Organization of the Input Data to the CNN

In using the CNN for pattern recognition, the input data need to be organized as a number of *feature maps* to be fed into the CNN. This is a term borrowed from image-processing applications, in which it is intuitive to organize the input as a two-dimensional (2-D) array, being the pixel values at the x and y (horizontal and vertical) coordinate indices. For color images, RGB (red, green, blue) values can be viewed as three different 2-D feature maps. CNNs run a small window over the input image at both training and testing time, so that the weights of the network that looks through this window can learn from various features of the input data regardless of their absolute position within the input. *Weight sharing*, or to be more precise in our present situation, *full weight sharing* refers to the decision to use the same weights at every positioning of the window. CNNs are also often said to be *local* because the individual units that are computed at a particular positioning of the window depend upon features of the local region of the image that the window currently looks upon.

In this section, we discuss how to organize speech feature vectors into feature maps that are suitable for CNN processing. The input "image" in question for our purposes can loosely be thought of as a spectrogram, with static, delta and delta-delta features (i.e., first and second temporal derivatives) serving in the roles of red, green and blue, although, as described below, there is more than one alternative for how precisely to bundle these into feature maps.

In keeping with this metaphor, we need to use inputs that preserve locality in both axes of frequency and time. Time presents no immediate problem from the standpoint of locality. Like other DNNs for speech, a single window of input to the CNN will consist of a wide amount of context (9–15 frames). As for frequency, the conventional use of MFCCs does present a major problem because the discrete cosine transform projects the spectral energies into a new basis that may not maintain locality. In this paper, we shall use the log-energy computed directly from the mel-frequency spectral coefficients (i.e., with no DCT), which we will denote as *MFSC features*. These will be used to represent each speech frame, along with their deltas and delta-deltas, in order to describe the acoustic energy distribution in each of several different frequency bands.

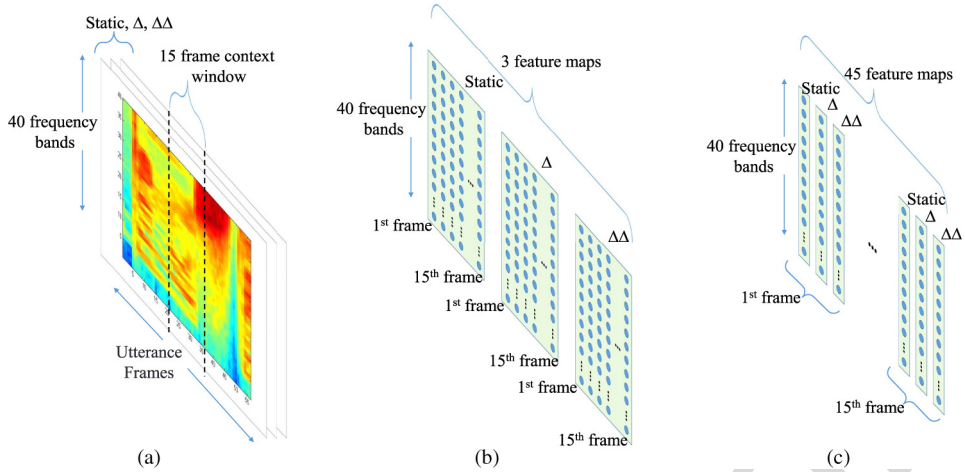


Fig. 1. Two different ways can be used to organize speech input features to a CNN. The above example assumes 40 MFSC features plus first and second derivatives with a context window of 15 frames for each speech frame.

There exist several different alternatives to organizing these MFSC features into maps for the CNN. First, as shown in Fig. 1(b), they can be arranged as three 2-D feature maps, each of which represents MFSC features (static, delta and delta-delta) distributed along both frequency (using the frequency band index) and time (using the frame number within each context window). In this case, a two-dimensional convolution is performed (explained below) to normalize both frequency and temporal variations simultaneously. Alternatively, we may only consider normalizing frequency variations. In this case, the same MFSC features are organized as a number of one-dimensional (1-D) feature maps (along the frequency band index), as shown in Fig. 1(c). For example, if the context window contains 15 frames and 40 filter banks are used for each frame, we will construct 45 (i.e., 15 times 3) 1-D feature maps, with each map having 40 dimensions, as shown in Fig. 1(c). As a result, a one-dimensional convolution will be applied along the frequency axis. In this paper, we will only focus on this latter arrangement found in Fig. 1(c), a one-dimensional convolution along frequency.

Once the input feature maps are formed, the convolution and pooling layers apply their respective operations to generate the activations of the units in those layers, in sequence, as shown in Fig. 2. Similar to those of the input layer, the units of the convolution and pooling layers can also be organized into maps. In CNN terminology, a pair of convolution and pooling layers in Fig. 2 in succession is usually referred to as one CNN "layer." A deep CNN thus consists of two or more of these pairs in succession. To avoid confusion, we will refer to convolution and pooling layers as convolution and pooling *plies*, respectively.

B. Convolution Ply

As shown in Fig. 2, every input feature map (assume I is the total number), $O_i (i = 1, \ldots, I)$, is connected to many feature maps (assume J in the total number), $Q_j (j = 1, \ldots, J)$, in the convolution ply based on a number of local weight matrices ($I \times J$ in total), $\mathbf{w}_{i,j} (i = 1, \ldots, I; j = 1, \ldots, J)$. The mapping can be represented as the well-known convolution operation in

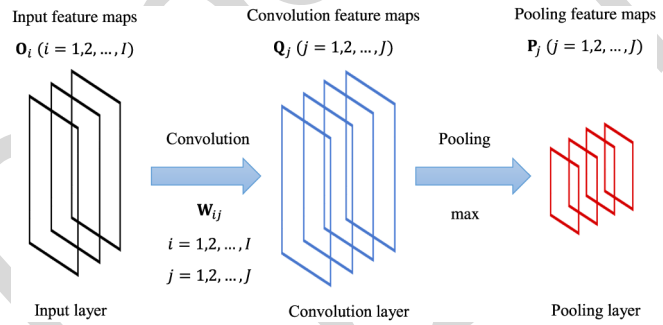


Fig. 2. An illustration of one CNN "layer" consisting of a pair of a convolution ply and a pooling ply in succession, where mapping from either the input layer or a pooling ply to a convolution ply is based on eq. (9) and mapping from a convolution ply to a pooling ply is based on eq. (10).

signal processing. Assuming input feature maps are all one dimensional, each unit of one feature map in the convolution ply can be computed as:

$$q_{j,m} = \sigma\left(\sum_{i=1}^{I}\sum_{n=1}^{F} o_{i,n+m-1} w_{i,j,n} + w_{0,j}\right), \quad (j = 1, \ldots, J) \tag{8}$$

where $o_{i,m}$ is the m-th unit of the i-th input feature map O_i, $q_{j,m}$ is the m-th unit of the j-th feature map Q_j in the convolution ply, $w_{i,j,n}$ is the nth element of the weight vector, $\mathbf{w}_{i,j}$, which connects the ith input feature map to the jth feature map of the convolution ply. F is called the *filter size*, which determines the number of frequency bands in each input feature map that each unit in the convolution ply receives as input. Because of the locality that arises from our choice of MFSC features, these feature maps are confined to a limited frequency range of the speech signal. Equation (8) can be written in a more concise matrix form using the convolution operator $*$ as:

$$Q_j = \sigma\left(\sum_{i=1}^{I} O_i * \mathbf{w}_{i,j}\right) \quad (j = 1, \ldots, J), \tag{9}$$

where O_i represents the i-th input feature map and $\mathbf{w}_{i,j}$ represents each local weight matrix, flipped to adhere to the convolution operation's definition. Both O_i and $\mathbf{w}_{i,j}$ are vectors if one dimensional feature maps are used, and are matrices if

two dimensional feature maps are used (where 2-D convolution is applied to the above equation), as described in the previous section. Note that, in this presentation, the number of feature maps in the convolution ply directly determines the number of local weight matrices that are used in the above convolutional mapping. In practice, we will constrain many of these weight matrices to be identical. It is also important to remember that the windows through which we view the input and apply one of these weight matrices will generally overlap. The convolution operation itself produces lower-dimensional data—each dimension decreases by filter size F minus one—but we can pad the input with dummy values (both dummy time frames and dummy frequency bands) to preserve the size of the feature maps. As a result, there could in principle be as many locations in the feature map of the convolution ply as there are in the input.

A convolution ply differs from a standard, fully connected hidden layer in two important aspects, however. First, each convolutional unit receives input only from a local area of the input. This means that each unit represents some features of a local region of the input. Second, the units of the convolution ply can themselves be organized into a number of feature maps, where all units in the same feature map share the same weights but receive input from different locations of the lower layer.

C. Pooling Ply

As shown in Fig. 2, a pooling operation is applied to the convolution ply to generate its corresponding pooling ply. The pooling ply is also organized into feature maps, and it has the same number of feature maps as the number of feature maps in its convolution ply, but each map is smaller. The purpose of the pooling ply is to reduce the resolution of feature maps. This means that the units of this ply will serve as generalizations over the features of the lower convolution ply, and, because these generalizations will again be spatially localized in frequency, they will also be invariant to small variations in location. This reduction is achieved by applying a pooling function to several units in a local region of a size determined by a parameter called *pooling size*. It is usually a simple function such as *maximization* or *averaging*. The pooling function is applied to each convolution feature map independently. When the max-pooling function is used, the pooling ply is defined as:

$$p_{i,m} = \max_{n=1}^{G} \quad q_{i,(m-1)\times s+n} \tag{10}$$

where G is the pooling size, and s, the *shift size*, determines the overlap of adjacent pooling windows. Similarly, if the average function is used, the output is calculated as:

$$p_{i,m} = r \sum_{n=1}^{G} \quad q_{i,(m-1)\times s+n} \tag{11}$$

where r is a scaling factor that can be learned. In image recognition applications, under the constraint that $G = s$, i.e., in which the pooling windows do not overlap and have no spaces between them, it has been claimed that max-pooling performs better than average-pooling [44]. In this work we will adjust G and s independently. Moreover, a non-linear activation function can be applied to the above $p_{i,m}$ to generate the final output. Fig. 3 shows a pooling ply with a pooling size of three. Each pooling unit receives input from three convolution ply units in the same feature map. If $G = s$, then the pooling ply would be one-third of the size of the convolution ply.

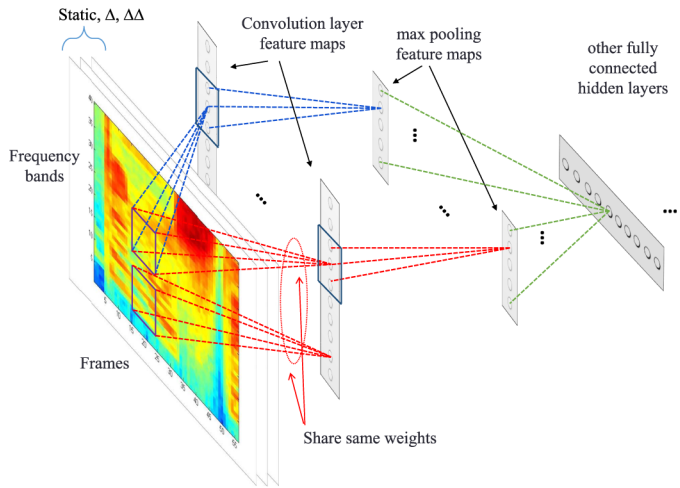


Fig. 3. An illustration of the regular CNN that uses so-called full weight sharing. Here, a 1-D convolution is applied along frequency bands.

D. Learning Weights in the CNN

All weights in the convolution ply can be learned using the same error back-propagation algorithm but some special modifications are needed to take care of sparse connections and weight sharing. In order to illustrate the learning algorithm for CNN layers, let us first represent the convolution operation in eq. (9) in the same mathematical form as the fully connected ANN layer so that the same learning algorithm in Section II can be similarly applied.

When one-dimensional feature maps are used, the convolution operations in eq. (9) can be represented as a simple matrix multiplication by introducing a large sparse weight matrix $\hat{\mathbf{W}}$ as shown in Fig. 4, which is formed by replicating a basic weight matrix $\mathbf{W}$ as in Fig. 4(a). The basic matrix $\mathbf{W}$ is constructed from all of the local weight matrices, $\mathbf{w}_{i,j}$, as follows:

$$\mathbf{W} = \begin{bmatrix} w_{1,1,1} & w_{1,2,1} & \cdots & w_{1,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{I,1,1} & w_{I,2,1} & \cdots & w_{I,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{I,1,2} & w_{I,2,2} & \cdots & w_{I,J,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{I,1,F} & w_{I,2,F} & \cdots & w_{I,J,F} \end{bmatrix}_{I\cdot F\times J} \tag{12}$$

where $\mathbf{W}$ is organized as $I \cdot F$ rows, where again F denotes filter size, each band contains I rows for I input feature maps, and $\mathbf{W}$ has J columns representing the weights of J feature maps in the convolution ply.

Meanwhile, the input and the convolution feature maps are also vectorized as row vectors $\hat{\mathbf{o}}$ and $\hat{\mathbf{q}}$. One single row vector $\hat{\mathbf{o}}$ is created from all of the input feature maps O_i $(i = 1, \ldots, I)$ as follows:

$$\hat{\mathbf{o}} = [\mathbf{v}_1 | \mathbf{v}_2 | \ \ldots \ | \mathbf{v}_M], \tag{13}$$

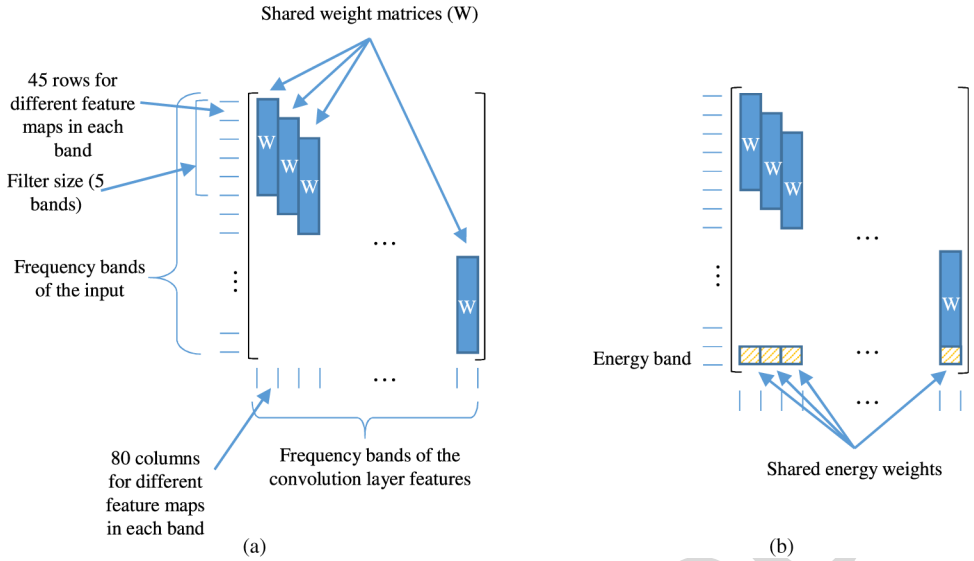


Fig. 4. All convolution operations in each convolution ply can be equivalently represented as one large matrix multiplication involving a sparse weight matrix, where both local connectivity and weight sharing can be represented in the structure of this sparse weight matrix. This figure assumes a filter size of 5, 45 input feature maps and 80 feature maps in the convolution ply. Sub-figure b shows an additional vector consisting of energy bands.

where $\mathbf{v}_m$ is a row vector containing the values of the mth frequency band along all I feature maps, and M is the number of frequency bands in the input layer. Therefore, the convolution ply outputs computed in eq. (9) can be equivalently expressed as a weight vector:

$$\hat{\mathbf{q}} = \sigma(\hat{\mathbf{o}}\hat{\mathbf{W}}) \tag{14}$$

This equation has the same mathematical form as a regular fully connected hidden layer as in eq. (2). Therefore, the convolution ply weights can be updated using the back-propagation algorithm as in eq. (5). The update for $\hat{\mathbf{W}}$ is similarly calculated as:

$$\Delta\hat{\mathbf{W}} = \epsilon \cdot \hat{\mathbf{o}}'\mathbf{e}. \tag{15}$$

The treatment of shared weights in the convolution ply is slightly different from the fully-connected DNN case where there is no weight sharing. The difference is that for the shared weights here, we sum them in their updates according to:

$$\Delta w_{i,j,n} = \sum_m \Delta\hat{\mathbf{W}}_{i+(m+n-2)\times I, j+(m-1)\times J} \tag{16}$$

where I and J are the number of feature maps in the input layer and convolution ply, respectively. Moreover, the above error vector $\mathbf{e}$ is either computed in the same way as in eq. (6) or back-propagated to the lower layer using the sparse matrix, $\hat{\mathbf{W}}$, as in eq. (7). Similarly, the biases can be handled by adding one row to the $\hat{\mathbf{W}}$ matrix to hold the bias values replicated among all convolution ply bands and adding one element with a value of one to the vector $\hat{\mathbf{o}}$.

Since the pooling ply has no weights, no learning is needed here. However, the error signals should be back-propagated to lower plies through the pooling function. In the case of max-pooling, the error signal is passed backwards only to the most active (largest) unit among each group of pooled units. That is, the error signal reaching the lower convolution ply can be computed as:

$$e^{\text{low}}_{i,n} = \sum_m e_{i,m} \cdot \delta(u_{i,m} + (m-1) \times s - n), \tag{17}$$

where $\delta(x)$ is the delta function and it has the value of 1 if x is 0 and zero otherwise, and $u_{i,m}$ is the index of the unit with the maximum value among the pooled units and is defined as:

$$u_{i,m} = \underset{n=1}{\overset{G}{\text{argmax}}} \; q_{i,(m-1)\times s+n} \tag{18}$$

E. Pretraining CNN Layers

RBM-based pretraining improves DNN performance especially when the training set is small. Pretraining initializes DNN weights to a proper range that leads to better optimization and regularization. For convolutional structure, a convolutional RBM (CRBM) has been proposed in [45]. Similar to RBMs, the training of the CRBM aims to maximize the likelihood function of the full training data according to an approximate contrastive divergence (CD) algorithm. In CRBMs, the convolution ply activations are stochastic. CRBMs define a multinomial distribution over each pool of hidden units in a convolution ply. Hence, at most one unit in each pooled set of units can be active. This requires either having no overlap between pooled units (i.e., $G = s$) or attaching different convolution units to each pooling unit as in the limited weight sharing described below in Sec. IV. Refer to [45] for more details on CRBM-based pretraining.

F. Treatment of Energy Features

In ASR, log-energy is usually calculated per frame and appended to other spectral features. In a CNN, it is not suitable to treat energy the same way as other filter bank energies since it is the sum of the energy in all frequency bands and so does not depend on frequency. Instead, the log-energy features

should be appended as extra inputs to all convolution units as shown in Fig. 4(b). Other non-localized features can be similarly treated. The experimental results in Section V show a consistent improvement in overall system performance by using the log-energy feature. There has been some question as to whether this improvement holds in larger-scale ASR tasks [40]. Nevertheless, these experiments at least show that nothing in principle prevents frequency-independent features such as log-energy from being accommodated within a CNN architecture when they stand to improve performance.

G. The Overall CNN Architecture

The building block of the CNN contains a pair of hidden plies: a convolution ply and a pooling ply. The input contains a number of localized features organized as a number of feature maps. The size (resolution) of feature maps gets smaller at upper layers as more convolution and pooling operations are applied. Usually one or more fully connected hidden layers are added on top of the final CNN layer in order to combine the features across all frequency bands before feeding to the output layer.

In this paper, we follow the hybrid ANN-HMM framework, where we use a softmax output layer on top of the topmost layer of the CNN to compute the posterior probabilities for all HMM states. These posteriors are used to estimate the likelihood of all HMM states per frame by dividing by the states' prior probabilities. Finally, the likelihoods of all HMM states are sent to a Viterbi decoder to recognize the continuous stream of speech units.

H. Benefits of CNNs for ASR

The CNN has three key properties: locality, weight sharing, and pooling. Each one of them has the potential to improve speech recognition performance. Locality in the units of the convolution ply allows more robustness against non-white noise where some bands are cleaner than the others. This is because good features can be computed locally from cleaner parts of the spectrum and only a smaller number of features are affected by the noise. This gives a better chance to higher layers of network to handle this noise because they can combine higher level features computed for each frequency band. This is clearly better than simply handling all input features in the lower layers as in standard, fully connected neural networks. Moreover, locality reduces the number of network weights to be learned.

Weight sharing can also improve model robustness and reduce overfitting as each weight is learned from multiple frequency bands in the input instead of just from one single location. It reduces the number of weights to learn in the network, moreover. Both locality and weight sharing are needed for the property of pooling. In pooling, the same feature values computed at different locations are pooled together and represented by one value. This leads to minimal differences in the features extracted by the pooling ply when the input patterns are slightly shifted along the frequency dimension, especially when max-pooling is used. This is very helpful in handling small frequency shifts that are common in speech signals. These frequency shifts may result from differences in vocal tract lengths among different speakers. Even for the same speaker, small frequency shifts may often occur. These shifts are difficult to handle within other models such as GMMs and DNNs, where many Gaussians and hidden units are needed to handle all possible pattern shifts. Moreover, it is difficult to learn such an operation as max-pooling in a standard ANN.

The same difficulty applies to temporal differences in the speech features as well. In a hybrid ANN-HMM, a number of frames within a context window are usually processed simultaneously by the ANN. The temporal variability due to varying speaking rate may be difficult to handle. CNNs, however, can handle this type of variability naturally when convolution is applied along the contextual window frames. On the other hand, since the CNN is required to compute an output for each frame for decoding, pooling or shift size may affect the fine resolution seen by higher layers of the CNN, and a large pooling size may affect state labels' localizations. This may cause phonetic confusion, especially at segment boundaries. Hence, a suitable pooling size must be chosen.

IV. CNN with Limited Weight Sharing for ASR

A. Limited Weight Sharing (LWS)

The weight sharing scheme in Fig. 3, as described in the previous section, is *full weight sharing (FWS)*. This is the standard for CNNs as used in image processing, since the same patterns may appear at any location in an image. The properties of the speech signal typically vary over different frequency bands, however. Using separate sets of weights for different frequency bands may be more suitable since it allows for detection of distinct feature patterns in different filter bands along the frequency axis. Fig. 5 shows an example of the *limited weight sharing* (LWS) scheme for CNNs, where only the convolution units that are attached to the same pooling unit share the same convolution weights. These convolution units need to share their weights so that they compute comparable features, which may then be pooled together. In other words, each frequency band can be considered as a separate subnet with its own convolution weights. We call each of these subnets a *section* for notational convenience. Each section contains a number of feature maps in the convolution ply. Each of these feature maps is produced by using one weight vector to scan all input dimensions in this section to determine the existence or absence of this feature. The pooling size determines the number of applications of this weight vector to neighboring locations in the input space, i.e., the size of each feature map in the convolution ply equals the pooling size. Each pooling unit in this section summarizes an entire convolution feature map into one number using a pooling function, such as maximization or averaging. In mathematical terms, the convolution ply activations can be computed as:

$$q_{k,j,m} = \sigma\left(\sum_{i}\sum_{n=1}^{F} o_{i,(k-1)\times s+n+m-1} \cdot w_{k,i,j,n} + w_{k,0,j}\right) \tag{19}$$

where $w_{k,i,j,n}$ denotes the n-th convolution weight, mapping from the i-th input feature map to the j-th convolution map in the k-th section, where m ranges from 1 up to G (pooling size). The pooling ply activations in this case can be computed using:

$$p_{k,j} = \max_{m=1}^{G} q_{k,j,m}. \tag{20}$$

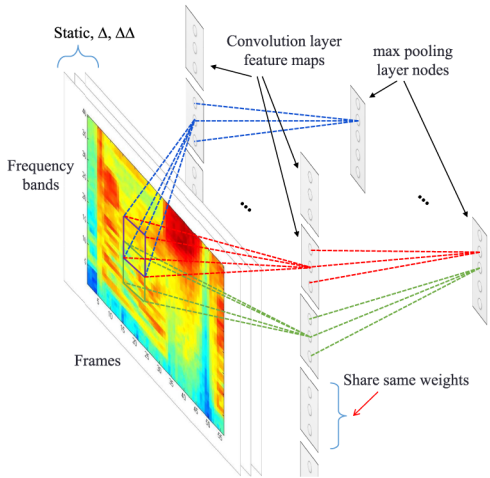


Fig. 5. An illustration of a CNN with limited weight sharing. 1-D convolution is applied along the frequency bands.

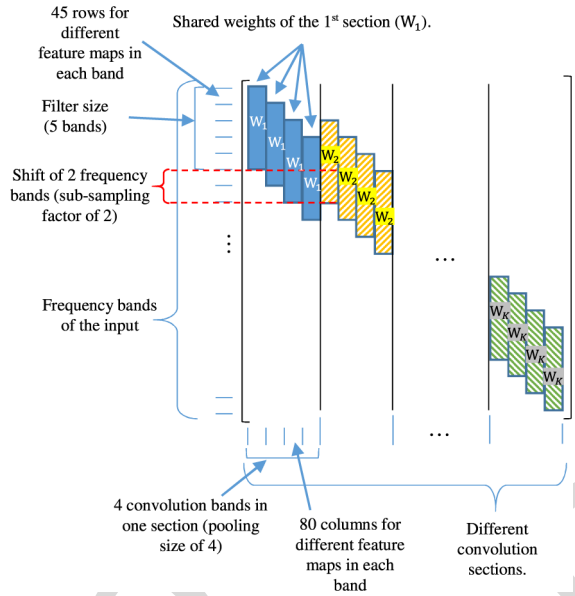


Fig. 6. The CNN layer using limited weight sharing (LWS) can also be represented as matrix multiplication using a large sparse matrix where local connectivity and weight sharing are represented in matrix form. The above figure assumes a filter size of 5, a pooling size of 4, 45 input feature maps, and 80 feature maps in the convolution ply.

Similarly, the above LWS convolution ply can also be represented with matrix multiplication using a large sparse matrix as in eq. (14) but both $\hat{\mathbf{o}}$ and $\hat{\mathbf{W}}$ need to be constructed in a slightly different way. First of all, the sparse matrix $\hat{\mathbf{W}}$ is constructed as in Fig. 6, where each $\mathbf{W}_k$ is formed based on local weights, $w_{k,i,j,n}$, as follows:

$$\mathbf{W}_k = \begin{bmatrix} w_{k,1,1,1} & w_{k,1,2,1} & \cdots & w_{k,1,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,I,1,1} & w_{k,I,2,1} & \cdots & w_{k,I,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,I,1,2} & w_{k,I,2,2} & \cdots & w_{k,I,J,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,I,1,F} & w_{k,I,2,F} & \cdots & w_{k,I,J,F} \end{bmatrix}_{I \cdot F \times J} \quad (k=1, 2, \ldots, K) \tag{21}$$

where these matrices $\mathbf{W}_k$ differ by section and the same weight matrix is replicated G times within each section. Secondly, the convolution ply input is vectorized as described in eq. (13), and the computed feature maps are organized as a large row vector $\hat{\mathbf{q}}$ by concatenating all values in each section as follows:

$$\hat{\mathbf{q}} = [\mathbf{v}_{1,1}| \ \ldots \ |\mathbf{v}_{1,G}| \ \ldots \ |\mathbf{v}_{K,1}| \ \ldots \ |\mathbf{v}_{K,G}], \tag{22}$$

where K is the total number of sections, G is the pooling size and $\mathbf{v}_{k,m}$ is a row vector containing the values of the units in the m-th band of the k-th section across all feature maps of the convolution ply:

$$\hat{\mathbf{v}}_{k,m} = [q_{k,1,m}, \ q_{k,2,m}, \ldots q_{k,I,m}], \tag{23}$$

where I is the total number of input feature maps within each section.

Learning the weights, in the case of limited weight sharing, can be done using the same eqs. (14) and (15) with $\hat{\mathbf{W}}$ and $\hat{\mathbf{q}}$ as defined above. Meanwhile, error vectors are propagated through the max pooling function as follows:

$$\mathbf{e}^{\text{low}}_{k,i,n} = \mathbf{e}_{k,i} \cdot \delta(u_{k,i} - n) \tag{24}$$

with:

$$u_{k,i} = \underset{m=1}{\overset{G}{\operatorname{argmax}}} \ q_{k,i,m}. \tag{25}$$

LWS also helps to reduce the total number of units in the pooling ply because each frequency band uses special weights that consider only the patterns appearing in the corresponding frequency range. As a result, a smaller number of feature maps per band should be sufficient. On the other hand, the LWS scheme does not allow for the addition of further convolution plies on top of the pooling ply since the features in different pooling-ply sections in LWS are unrelated and cannot be convolved locally. An LWS convolution ply on top of a regular full weight sharing one would be possible, however.

B. Pretraining of LWS-CNN

In this section, we propose to modify the CRBM model in [45] for pretraining the CNN with LWS as discussed in the preceding subsection. For learning the CRBM parameters, we need to define the conditional probabilities of the states of the hidden units given the visible ones and vice versa. The conditional probability of the activation for a hidden unit, $h_{k,j,m}$, which represents the state of the m-th frequency band of the j-th feature map from the k-th section, given the CRBM input $\mathbf{v}$, is defined as the following softmax function:

$$P(h_{k,j,m} = 1|\mathbf{v}) = \frac{\exp(I(h_{k,j,m}))}{\sum_{n=1}^{p} \exp(I(h_{k,j,n}))}, \tag{26}$$

where $I(h_{k,j,m})$ is the sum of the weighted signal reaching unit $h_{k,j,m}$ from the input layer and is defined as:

$$I(h_{k,j,m}) = \sum_{i} \sum_{n=1}^{f} v_{i,(k-1)\times s+n+m-1} w_{k,i,j,n} + w_{k,i,j,0} \tag{27}$$

The conditional probability distribution of $v_{i,n}$, which is the visible unit at the nth frequency band of the ith feature map, given the hidden unit states, can be computed by the following Gaussian distribution:

$$P(v_{i,n}|\mathbf{h}) = \mathcal{N}\left(v_{i,n}; \sum_{j,(k,m)\in\mathbf{C}(i,n)} h_{k,j,m} w_{k,i,j,f(n,k,m)}, \sigma^2\right) \quad (28)$$

where the above mean is the sum of the weighted signal arriving from the hidden units that are connected to the visible units, $\mathbf{C}(i,n)$ represents these connections as the set of indices of convolution bands and sections that receive input from the visible unit $v_{i,n}$, $w_{k,i,j,f(n,k,m)}$ is the weight on the link from the n-th band of the i-th input feature map to the m-th band of the j-th feature map of the k-th convolution section, $f(n,k,m)$ is a mapping function from the indices of connected nodes to the corresponding index of the filter element, and σ^2 is the variance of the Gaussian distribution and it is a fixed model parameter.

Based on the above two conditional probabilities, all connection weights of the above CRBM can be iteratively estimated by using the regular contrastive divergence (CD) algorithm. The weights of the trained CRBMs can be used as good initial values for the convolution ply in the LWS scheme. After the first convolution ply weights are learned, they are used to compute the convolution and pooling ply outputs using eqs. (19) and (20). The outputs of the pooling ply are used as inputs to continuously pretrain the next layer as done in deep belief network training [46].

V. Experiments

The experiments of this section have been conducted on two speech recognition tasks to evaluate the effectiveness of CNNs in ASR: small-scale phone recognition in TIMIT and large vocabulary voice search (VS) task. There have been extensions of the work described in this paper to other larger vocabulary speech recognition tasks that lend further support to the value of this approach [39], [40].

A. Speech Data and Analysis

The method of speech analysis is similar in the two datasets. Speech is analyzed using a 25-ms Hamming window with a fixed 10-ms frame rate. Speech feature vectors are generated by Fourier-transform-based filter-bank analysis, which includes 40 log energy coefficients distributed on a mel scale, along with their first and second temporal derivatives. All speech data were normalized so that each vector dimension has a zero mean and unit variance.

B. TIMIT Phone Recognition Results

For TIMIT, we used the standard 462-speaker training set and removed all SA records, since they may bias the results. A separate development set of 50 speakers was used for tuning all meta-parameters including the learning schedule and multiple learning rates. Results are reported using the 24-speaker core test set, which has no overlap with the development set. In addition to the log MFSC features, we added a log energy feature per frame. The log energy was normalized per utterance to have a maximum value of one, and then normalized to have zero mean and unit variance over the whole training data set. The energy feature is handled within a CNN as described in Section III.

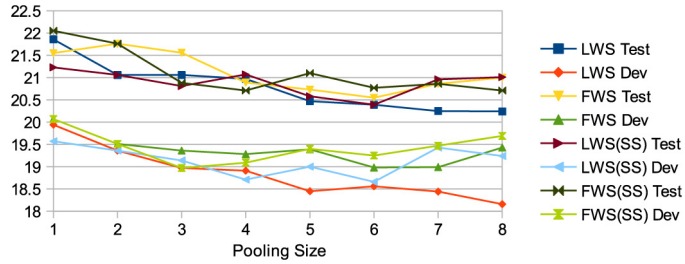


Fig. 7. Effects of different CNN pooling sizes on Phone Error Rate (PER in %) for both local weight sharing (LWS) and full weight sharing (FWS). Dev set and core test set accuracies are plotted separately. The convolution and pooling plys use a filter size of 8, 150 feature maps for FWS, and 80 feature maps per frequency band for LWS. A shift size of 2 is used with LWS and FWS while a shift size equal to the pooling size is used for LWS(SS) and FWS(SS).

We used 183 target class labels, i.e., 3 states for each HMM of 61 phones. After decoding, the original 61 phone classes were mapped to a set of 39 classes as in [47] for final scoring. In our experiments, a bigram language model over phones, estimated from the training set, was used in decoding. To prepare the ANN targets, a mono-phone HMM model was trained on the training data set, and it was used to generate state-level labels based on forced alignment. For neural-network training, learning rate annealing, in which the learning rate is steadily decreased over successive iterations, and early stopping strategies, in which a held-out development set is used to determine when overfitting has started, were utilized, as in [46].

We conducted many experiments on CNNs using both full weight sharing (FWS) and limited weight sharing (LWS) schemes. In this section, we first evaluate the ASR performance of CNNs under different settings of the CNN parameters. We normally fix all parameters except one and show how recognition performance varies with the remaining parameter. In these experiments we used one convolution ply, one pooling ply and two fully connected hidden layers on the top. The fully connected layers had 1000 units in each. The convolution and pooling parameters were: pooling size of 6, shift size of 2, filter size of 8, 150 feature maps for FWS, and 80 feature maps per frequency band for LWS. In all experiments, we fixed a random number generation seed for both weight initialization and order randomization of the training data. In the last table, we report the average of 3 runs with different seeds to compare recognition performance among DNNs, FWS-CNNs and LWS-CNNs.

1) Effects of Varying CNN Parameters: In this section, we analyze the effects of changing different CNN parameters. Figs. 7, 8, 9, and 10 show the results of these experiments on both the core test set (Test) and the development set (Dev). The figures show that both the pooling size and the number of feature maps have the most significant impact on the final ASR performance. Fig. 7 shows that all configurations yield better performance with increasing pooling size up to 6. LWS yields better performance with bigger pooling sizes. Figs. 7 and 8 show that overlapping pooling windows do not produce a clear performance gain, and that using the same value for both the pooling size and the shift size produces a similar performance while decreasing the model complexity. Fig. 9 shows that a larger number of feature maps usually leads to better performance, especially with

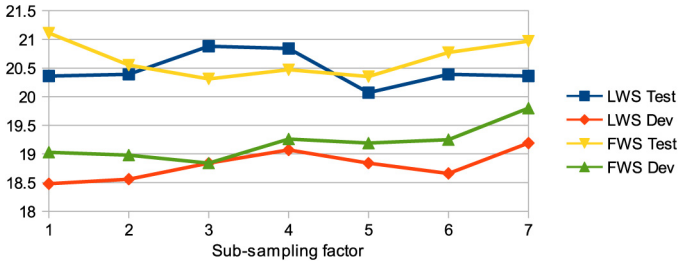


Fig. 8. Effects of different CNN shift sizes on Phone Error Rate (PER in %) for LWS and FWS. The convolution and pooling plys use a pooling size of 6, filter size of 8, 150 feature maps for FWS, and 80 feature maps per frequency band for LWS.

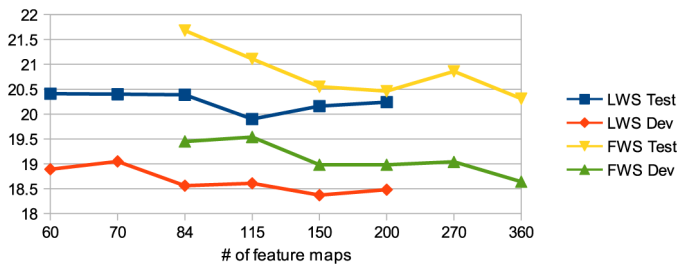


Fig. 9. Effects of different numbers of feature maps on Phone Error Rate (PER in %) for LWS and FWS. The convolution and pooling plys used a pooling size of 6, shift size of 2, filter size of 8.

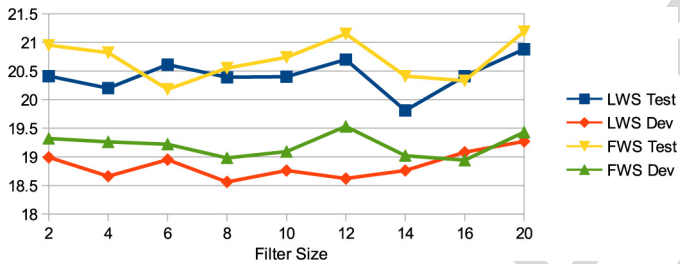


Fig. 10. Effects of different filter sizes on Phone Error Rate (PER in %) for LWS and FWS. The convolution and pooling plys use a pooling size of 6, shift size of 2, 150 feature maps for FWS, and 80 feature maps per frequency band for LWS.

TABLE I
EFFECTS OF USING ENERGY FEATURES ON PERCENT PER FOR THE TEST SET. THE CONVOLUTION AND POOLING PLYS USE A POOLING SIZE OF 6, SHIFT SIZE OF 2, FILTER SIZE OF 8, 150 FEATURE MAPS FOR FWS, AND 80 FEATURE MAPS PER FREQUENCY BAND FOR LWS

	No Energy	Energy
LWS	20.61%	20.39%
FWS	21.19%	20.55%

FWS. It also shows that LWS can achieve better performance with a smaller number of feature maps than FWS due to its ability to learn different feature patterns for different frequency bands. This indicates that the LWS scheme is more efficient in terms of the number of hidden units.

2) Effects of Energy Features: Table I shows the benefit of using energy features, producing a significant accuracy improvement, especially for FWS. While the energy features can be easily derived from other MFSC features, adding them as separate inputs to the convolution filters results in more discriminative power as it provides a way to compare the local frequency bands processed by the filter with the overall spectrum.

TABLE II
EFFECTS OF POOLING FUNCTIONS ON PERCENT PER. THE EXPERIMENTAL SETTING IS THE SAME AS TABLE I

	Average	Max
Development Set	19.63%	18.56%
Test Set	21.6%	20.39%

3) Effects of Pooling Functions: Table II shows that the max-pooling function performs better than the average function with the LWS scheme. These results are consistent with what has been observed in image recognition applications [44].

4) Overall Performance: Here we compare the overall performance of different CNN configurations with a baseline DNN system on the same TIMIT task. All results of the comparison are listed in Table III, along with the numbers of weight parameters and computations in each model. Average PERs were obtained over three runs with different random seeds. The first row shows the average PER obtained from a DNN that had three hidden layers. Its first hidden layer had 2000 units, to match the increased number of units in the CNN. The other two hidden layers had 1000 units in each. The second row reports the average PER from a similar DNN with 5 layers. The parameters of CNNs in rows 3 and 4 were chosen based on the performance obtained on the Dev set in the previous sections. Both had a filter size of 8, a pooling size of 6, and a shift size of 2. The number of feature maps was 150 for LWS and 360 for FWS. The results in Table III show that the CNN performance was much better than that of the corresponding DNN and that LWS was slightly better than FWS even with less than half the number of units in the pooling ply. Although the number of units in the LWS convolution ply was slightly larger than that of the FWS, LWS-CNN gives a much smaller model size since LWS results in far fewer weights in the upper, fully connected layers. The CNN with LWS gave more than an 8% relative reduction in PER over the DNN. The fifth row in Table III shows the performance of using two pairs of convolution and pooling plies with FWS in addition to two fully connected hidden layers on top. The sixth row shows the performance for the same model when the second convolution layer uses LWS. We coarsely tuned the two-layer parameters on the development set, and obtained a PER of 20.23% and 20.36% which show only minor differences to using one convolution layer. On the other hand, using two convolution layers tends to result in a smaller number of parameters as the fourth column shows.

C. Large Vocabulary Speech Recognition Results

In this section, we examine the recognition performance of CNNs on a large vocabulary ASR task. We used a voice search dataset containing 18 hours of speech data. Initially, a conventional state-tied triphone HMM was built. The HMM state labels were used as the targets in training both the DNNs and CNNs, which both followed the standard recipe. The first 15 epochs were run with a learning rate of 0.08, followed by 10 additional epochs with a reduced learning rate of 0.002. We investigated the effects of pretraining using an RBM for the fully connected layers and using a CRBM, as described in section 4-B, for the convolution and pooling plies. In this section, we used bigger hidden layers of 2000 units each. The DNN had three hidden

TABLE III
PERFORMANCE ON TIMIT OF DIFFERENT CNN CONFIGURATIONS, COMPARED WITH DNNS, ALONG WITH THE SIZE OF THE MODEL IN TOTAL NUMBER OF PARAMETERS, AND THE SPEED IN TOTAL NUMBER OF MULTIPLY-AND-ACCUMULATE OPERATIONS. AVERAGE PERS WERE COMPUTED OVER 3 RUNS WITH DIFFERENT RANDOM SEEDS AND SHOWN IN THE 3RD COLUMN, WHILE THE MINIMUM AND MAXIMUM PERS ARE SHOWN IN THE 4TH COLUMN. THE SECOND COLUMN SHOWS THE NETWORK STRUCTURE AND THE CONFIGURATION OF THE HIDDEN LAYERS ARE SHOWN WITHIN BRACES. THE NUMBER OF NODES OF A FULLY CONNECTED LAYER IS GIVEN DIRECTLY. FOR CNN LAYERS THE CNN LAYER PARAMETERS ARE GIVEN FOR FWS OR LWS IN BRACKETS WHERE: 'M' IS THE NUMBER OF FEATURE MAPS, 'P' IS THE POOLING SIZE, 'S' IS THE SHIFT SIZE, AND 'F' IS THE FILTER SIZE

ID	Network structure	Average PER	min-max PER	# param's	# op's
1	DNN {2000 + 2×1000}	22.02%	21.86-22.11%	6.9M	6.9M
2	DNN {2000 + 4×1000}	21.87%	21.68-21.98%	8.9M	8.9M
3	CNN {LWS(m:150 p:6 s:2 f:8) + 2×1000}	20.17%	19.92-20.41%	5.4M	10.7M
4	CNN {FWS(m:360 p:6 s:2 f:8) + 2×1000}	20.31%	20.16-20.58%	8.5M	13.6M
5	CNN {FWS(m:150 p:4 s:2 f:8) + **FWS**(m:300 p:2 s:2 f:6) + 2×1000}	20.23%	20.11-20.29%	4.5M	11.7M
6	CNN {FWS(m:150 p:4 s:2 f:8) + **LWS**(m:150 p:2 s:2 f:6) + 2×1000}	20.36%	19.91-20.61%	4.1M	7.5M

TABLE IV
PERFORMANCE ON THE VS LARGE VOCABULARY DATA SET IN PERCENT WER WITH AND WITHOUT PRETRAINING (PT). THE EXPERIMENTAL SETTING IS THE SAME AS TABLE I

	No PT	With PT
DNN	37.1%	35.4%
CNN	34.2%	33.4%

layers while the CNN had one pair of convolution and pooling plies in addition to two hidden fully connected layers. The CNN layer used limited weight sharing and had 84 feature maps per section. It had a filter size of 8, a pooling size of 6, and a shift size of 2. Moreover, the context window had 11 frames. Frame energy features were not used in these experiments.

Table IV shows that the CNN improves word error rate (WER) performance over the DNN regardless of whether pretraining is used. Similar to the TIMIT results, the CNN improves performance by about an 8% relative error reduction over the DNN in the VS task without pretraining. With pretraining, the relative word error rate reduction is about 6%. Moreover, the results show that pretraining the CNN can improve its performance, although the effect of pretraining for the CNN is not as strong as that for the DNN.

VI. CONCLUSIONS

In this paper, we have described how to apply CNNs to speech recognition in a novel way, such that the CNN's structure directly accommodates some types of speech variability. We showed a performance improvement relative to standard DNNs with similar numbers of weight parameters using this approach (about 6-10% relative error reduction), in contrast to the more equivocal results of convolving along the time axis, as earlier applications of CNNs to speech had attempted [33], [34], [35]. Our hybrid CNN-HMM approach delegates temporal variability to the HMM, while convolving along the frequency axis creates a degree of invariance to small frequency shifts, which normally occur in actual speech signals due to speaker differences.

In addition, we have proposed a new, limited weight sharing scheme that can handle speech features in a better way than the full weight sharing that is standard in previous CNN architectures such as those used in image processing. Limited weight sharing leads to a much smaller number of units in the pooling ply, resulting in a smaller model size and lower computational complexity than the full weight sharing scheme.

We observed improved performance on two ASR tasks: TIMIT phone recognition and a large-vocabulary voice search task, across a variety of CNN parameter and design settings. We determined that the use of energy information is very beneficial for the CNN in terms of recognition accuracy. Further, the ASR performance was found to be sensitive to the pooling size, but insensitive to the overlap between pooling units, a discovery that will lead to better efficiency in storage and computation. Finally, pretraining of CNNs based on convolutional RBMs was found to yield better performance in the large-vocabulary voice search experiment, but not in the phone recognition experiment. This discrepancy is yet to be examined thoroughly in our future work.

REFERENCES

[1] H. Jiang, "Discriminative training for automatic speech recognition: A survey," *Comput. Speech, Lang.*, vol. 24, no. 4, pp. 589–608, 2010.
[2] X. He, L. Deng, and W. Chou, "Discriminative learning in sequential pattern recognition—A unifying review for optimization-oriented speech recognition," *IEEE Signal Process. Mag.*, vol. 25, no. 5, pp. 14–36, Sep. 2008.
[3] L. Deng and X. Li, "Machine learning paradigms for speech recognition: An overview," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 21, no. 5, pp. 1060–1089, May 2013.
[4] G. E. Dahl, M. Ranzato, A. Mohamed, and G. E. Hinton, "Phone recognition with the mean-covariance restricted Boltzmann machine," *Adv. Neural Inf. Process. Syst.*, no. 23, 2010.
[5] A. Mohamed, T. Sainath, G. Dahl, B. Ramabhadran, G. Hinton, and M. Picheny, "Deep belief networks using discriminative features for phone recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2011, pp. 5060–5063.
[6] D. Yu, L. Deng, and G. Dahl, "Roles of pre-training and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2010.
[7] G. Dahl, D. Yu, L. Deng, and A. Acero, "Large vocabulary continuous speech recognition with context-dependent DBN-HMMs," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2011, pp. 4688–4691.
[8] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Proc. IEEE Workshop Autom. Speech Recognition Understand. (ASRU)*, 2011, pp. 24–29.
[9] N. Morgan, "Deep and wide: Multiple layers in automatic speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 7–13, Jan. 2012.
[10] A. Mohamed, G. Dahl, and G. Hinton, "Deep belief networks for phone recognition," in *Proc. NIPS Workshop Deep Learn. Speech Recognition Related Applicat.*, 2009.
[11] A. Mohamed, D. Yu, and L. Deng, "Investigation of full-sequence training of deep belief networks for speech recognition," in *Proc. Interspeech*, 2010, pp. 2846–2849.

[12] L. Deng, D. Yu, and J. Platt, "Scalable stacking and learning for building deep architectures," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process.*, 2012, pp. 2133–2136.
[13] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 30–42, Jan. 2012.
[14] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Interspeech*, 2011, pp. 437–440.
[15] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, and A. Mohamed, "Making deep belief networks effective for large vocabulary continuous speech recognition," in *IEEE Workshop Autom. Speech Recogn. Understand. (ASRU)*, 2011, pp. 30–35.
[16] J. Pan, C. Liu, Z. Wang, Y. Hu, and H. Jiang, "Investigation of deep neural networks (DNN) for large vocabulary continuous speech recognition: Why DNN surpasses GMMs in acoustic modeling," in *Proc. ISCSLP*, 2012.
[17] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
[18] T. Landauer, C. Kamm, and S. Singhal, "Learning a minimally structured back propagation network to recognize speech," in *Proc. 9th Annu. Conf. Cogn. Sci. Soc.*, 1987, pp. 531–536.
[19] D. Burr, "A neural network digit recognizer," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 1986.
[20] Q. Zhu, B. Chen, N. Morgan, and A. Stolcke, "Tandem connectionist feature extraction for conversational speech recognition," in *Machine Learning for Multimodal Interaction*. Berlin/Heidelberg, Germany: Springer , 2005, vol. 3361, pp. 223–231.
[21] H. Hermansky, D. P. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional HMM systems," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2000, vol. 3, pp. 1635–1638.
[22] F. Grézl, M. Karafiát, S. Kontár, and J. Cernocky, "Probabilistic and bottle-neck features for LVCSR of meetings," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2007, vol. 4, pp. 757–800.
[23] L. Deng, M. Seltzer, D. Yu, A. Acero, A. Mohamed, and G. Hinton, "Binary coding of speech spectrograms using a deep auto-encoder," in *Proc. Interspeech*, 2010.
[24] Y. Bao, H. Jiang, L.-R. Dai, and C. Liu, "Incoherent training of deep neural networks to de-correlate bottleneck features for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2013, pp. 6980–6984.
[25] D. Zhang, L. Deng, and M. Elmasry, *A Pipelined Neural Network Architecture For Speech Recognition, In Book: VLSI Artificial Neural Networks Engineering*. Norwell, MA, USA: Kluwer, 1994.
[26] L. Deng, K. Hassanein, and M. Elmasry, "Analysis of correlation structure for a neural predictive model with applications to speech recognition," *Neural Netw.*, vol. 7, no. 2, pp. 331–339, 1994.
[27] H. A. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*. Norwell, MA, USA: Kluwer, 1993.
[28] G. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, pp. 1771–1800, 2002.
[29] A. Mohamed, G. Hinton, and G. Penn, "Understanding how deep belief networks perform acoustic modelling," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2012, pp. 4273–4276.
[30] J. Li, D. Yu, J.-T. Huang, and Y. Gong, "Improving wideband speech recognition using mixed-bandwidth training data in CD-DNN-HMM," in *Proc. IEEE Spoken Lang. Technol. Workshop (SLT)*, 2012, pp. 131–136.
[31] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1995.
[32] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, pp. 193–202, 1980.
[33] H. Lee, P. Pham, Y. Largman, and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Proc. Adv. Neural Inf. Process. Syst. 22*, 2009, pp. 1096–1104.
[34] D. Hau and K. Chen, "Exploring hierarchical speech representations using a deep convolutional neural network," in *Proc. 11th UK Workshop Comput. Intell. (UKCI '11)*, Manchester, U.K., 2011.
[35] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 3, pp. 328–339, Mar. 1989.
[36] D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide, "Feature learning in deep neural networks - studies on speech recognition tasks," in *Proc. Int. Conf. Learn. Represent.*, 2013.
[37] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Mar. 2012, pp. 4277–4280.
[38] O. Abdel-Hamid, L. Deng, and D. Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition," in *Proc. Interspeech*, 2013.
[39] L. Deng, O. Abdel-Hamid, and D. Yu, "A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2013, pp. 6669–6673.
[40] T. N. Sainath, A.-R. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2013, pp. 8614–8618.
[41] A. Mohamed, T. Sainath, G. Dahl, B. Ramabhadran, G. Hinton, and M. Picheny, "Deep belief networks using discriminative features for phone recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2011, pp. 5060–5063.
[42] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," Tech. Rep. 14021128v1 [cs.NE], Feb. 2014, arXiv.
[43] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2013, pp. 8599–8603.
[44] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Proc. 20th Int. Conf. Artif. Neural Netw.: Part III*, Berlin/Heidelberg, Germany, 2010, pp. 92–101, Springer-Verlag ser. ICANN'10.
[45] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 609–616.
[46] A. Mohamed, G. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 14–22, Jan. 2012.
[47] K. F. Lee and H. W. Hon, "Speaker-independent phone recognition using hidden Markov models," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 37, no. 11, pp. 1641–1648, Nov. 1989.

Ossama Abdel-Hamid received his B.Sc. with honors in information technology from Cairo University, Egypt, in 2002, from where he received his M.Sc. in 2007. He is currently a Ph.D. candidate in the Computer Science Department, York University, Canada. He joined the speech research group at RDI, Egypt in the period from 2003 to 2007. Moreover, he had internships at IBM Watson research center in 2006, Google in 2008, and Microsoft Reseach in 2012. His current research focuses on improving automatic speech recognition performance using deep learning methods.

Abdel-rahman Mohamed is currently a post doctoral fellow at the University of Toronto. He finished his Ph.D. study at the University of Toronto in 2013 working on deep neural network (DNN) acoustic models for ASR. Before studying in Toronto, he received his B.Sc. and M.Sc. from the Electronics and Communication Engineering Department, Cairo University, in 2004 and 2007. Since 2004 he has been with the speech research group at the RDI Company, Egypt. Then he joined the ESAT-PSI speech group at the Katholieke Universiteit Leuven, Belgium. His research focuses on developing machine learning techniques for automatic speech recognition and understanding.

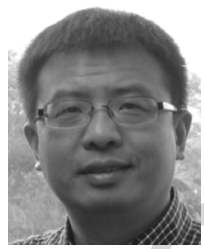
Hui Jiang (M'00–SM'11) received B.Eng. and M.Eng. degrees from the University of Science and Technology of China (USTC) and his Ph.D. degree from the University of Tokyo, Tokyo, Japan, in September 1998, all in electrical engineering.

From October 1998 to April 1999, he was a researcher in the University of Tokyo. From April 1999 to June 2000, he was with Department of Electrical and Computer Engineering, University of Waterloo, Canada, as a postdoctoral fellow. From 2000 to 2002, he was with Dialogue Systems Research, Multimedia Communication Research Lab, Bell Labs, Lucent Technologies Inc., Murray Hill, NJ. He joined Department of Computer Science and Engineering, York University, Toronto, Canada, as an Assistant Professor in fall 2002 and was promoted to Associate Professor in 2007 and to Full Professor in 2013. He served as an associate editor for IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING between 2009 and 2013. His current research interests lie in machine learning methods with applications to speech and language processing.

Li Deng received the Ph.D. from the University of Wisconsin-Madison. He was a tenured professor (1989-1999) at the University of Waterloo, Ontario, Canada, and then joined Microsoft Research, Redmond, where he is currently a Principal Research Manager of its Deep Learning Technology Center. Since 2000, he has also been an affiliate full professor at the University of Washington, Seattle, teaching computer speech processing. He has been granted over 60 U.S. or international patents, and has received numerous awards and honors bestowed by IEEE, ISCA, ASA, and Microsoft including the latest IEEE SPS Best Paper Award (2013) on deep neural nets for speech recognition. He authored or co-authored 4 books including the latest one on Deep Learning: Methods and Applications. He is a Fellow of the Acoustical Society of America, a Fellow of the IEEE, and a Fellow of the ISCA. He served as the Editor-in-Chief for IEEE SIGNAL PROCESSING MAGAZINE (2009-2011), and currently as Editor-in-Chief for IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING. His recent research interests and activities have been focused on deep learning and machine intelligence applied to large-scale text analysis and to speech/language/image multimodal processing, advancing his earlier work with collaborators on speech analysis and recognition using deep neural networks since 2009.

Gerald Penn (M'09–SM'10) is a Professor of Computer Science at the University of Toronto, where he has worked since 2001. He received his S.B. from the University of Chicago and his M.S. and Ph.D. from Carnegie Mellon University. From 1999 to 2001, he was a Member of Technical Staff in the Multimedia Communications Research Laboratory at Bell Labs. His research interests are in speech and natural language processing.

Dr. Penn is a senior member of AAAI, and a member of ACL and ISCA.



Dong Yu (M'97–SM'06) is a principal researcher at Microsoft Research - Speech and Dialog Research Group. He holds a Ph.D. degree in computer science from University of Idaho, an M.S. degree in computer science from Indiana University at Bloomington, an M.S. degree in electrical engineering from Chinese Academy of Sciences, and a B.S. degree (with honor) in electrical engineering from Zhejiang University (China). His current research interests include speech processing, robust speech recognition, discriminative training, and machine learning. He has published over 130 papers in these areas and is the inventor/coinventor of more than 50 granted/pending patents. His recent work on context-dependent deep neural network hidden Markov model (CD-DNN-HMM) has been seriously challenging the dominant position of the conventional GMM based system for large-vocabulary speech recognition and was recognized by the IEEE SPS 2013 best paper award.

Dr. Dong Yu is currently serving as a member of the IEEE Speech and Language Processing Technical Committee (2013-) and an associate editor of IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING (2011-). He has served as an associate editor of IEEE SIGNAL PROCESSING MAGAZINE (2008-2011) and the lead guest editor of IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING - special issue on deep learning for speech and language processing (2010-2011).

Convolutional Neural Networks for Speech Recognition

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu

***Abstract*—Recently, the hybrid deep neural network (DNN)-hidden Markov model (HMM) has been shown to significantly improve speech recognition performance over the conventional Gaussian mixture model (GMM)-HMM. The performance improvement is partially attributed to the ability of the DNN to model complex correlations in speech features. In this paper, we show that further error rate reduction can be obtained by using convolutional neural networks (CNNs). We first present a concise description of the basic CNN and explain how it can be used for speech recognition. We further propose a limited-weight-sharing scheme that can better model speech features. The special structure such as local connectivity, weight sharing, and pooling in CNNs exhibits some degree of invariance to small shifts of speech features along the frequency axis, which is important to deal with speaker and environment variations. Experimental results show that CNNs reduce the error rate by 6%-10% compared with DNNs on the TIMIT phone recognition and the voice search large vocabulary speech recognition tasks.**

***Index Terms*—Convolution, convolutional neural networks, Limited Weight Sharing (LWS) scheme, pooling.**

I. Introduction

THE aim of automatic speech recognition (ASR) is the transcription of human speech into spoken words. It is a very challenging task because human speech signals are highly variable due to various speaker attributes, different speaking styles, uncertain environmental noises, and so on. ASR, moreover, needs to map variable-length speech signals into variable-length sequences of words or phonetic symbols. It is well known that hidden Markov models (HMMs) have been very successful in handling variable length sequences as well as modeling the temporal behavior of speech signals using a sequence of states, each of which is associated with a particular probability distribution of observations. Gaussian mixture models (GMMs) have been, until very recently, regarded as the most powerful model for estimating the probabilistic distribution of speech signals associated with each of these HMM states. Meanwhile, the generative training methods of GMM-HMMs have been well developed for ASR based on the popular expectation maximization (EM) algorithm. In addition, a plethora of discriminative training methods, as reviewed in [1], [2], [3], are typically employed to further improve HMMs to yield the state-of-the-art ASR systems.

Very recently, HMM models that use artificial neural networks (ANNs) instead of GMMs have witnessed a significant resurgence of research interest [4], [5], [6], [7], [8], [9], initially on the TIMIT phone recognition task with mono-phone HMMs for MFCC features [10], [11], [12], and shortly thereafter on several large vocabulary ASR tasks with triphone HMM models [6], [7], [13], [14], [15], [16]; see an overview of this series of studies in [17]. In retrospect, the performance improvements of these recent attempts have been ascribed to their use of "deep" learning, a reference both to the number of hidden layers in the neural network as well as to the abstractness and, by some accounts, psychological plausibility of representations obtained in the layers furthest removed from the input, which hearkens back to the appeal of ANNs to cognitive scientists thirty years ago. A great many other design decisions have been made in these alternative ANN-based models to which significant improvements might have been attributed.

Even without deep learning, ANNs are powerful discriminative models that can directly represent arbitrary classification surfaces in the feature space without any assumptions about the data's structure. GMMs, by contrast, assume that each data sample is generated from one hidden expert (i.e., a Gaussian) and a weighted sum of those Gaussian components is used to model the entire feature space. ANNs have been used for speech recognition for more than two decades. Early trials worked on static and limited speech inputs where a fixed-sized buffer was used to hold enough information to classify a word in an isolated speech recognition scheme [18], [19]. They have been used in continuous speech recognition as feature extractors, in both the TANDEM approach [20], [21] and in so-called bottleneck feature methods [22], [23], [24], and also as nonlinear predictors to aid the recognition of speech units [25], [26]. Their first successful application to continuous speech recognition, however, was in a manner that almost exactly parallels the use of GMMs now, i.e., as sources of HMM state posterior probabilities, given a fixed number of feature frames [27].

How do the recent ANN-HMM hybrids differ from earlier approaches? They are simply much larger. Advances in computing hardware over the last twenty years have played a signif-

Manuscript received October 11, 2013; revised February 04, 2014; accepted July 05, 2014. Date of publication July 16, 2014; date of current version nulldate. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Haizhou Li.

O. Abdel-Hamid and H. Jiang are with the Department of Electrical Engineering and Computer Science, Lassonde School of Engineering, York University, Toronto, ON M3J 1P3, Canada (e-mail: ossama@cse.yorku.ca; hj@cse.yorku.ca).

A.-r. Mohamed and G. Penn are with the Computer Science Department, University of Toronto, Toronto, ON M5S, Canada (e-mail: asamir@cs.utoronto.ca; gpenn@cs.utoronto.ca).

L. Deng and D. Yu are with Microsoft Research, Redmond, WA 98052 USA (e-mail: deng@microsoft.com; dongyu@microsoft.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASLP.2014.2339736

icant role in the advance of ANN-based approaches to acoustic modeling because training ANNs with so many hidden units on so many hours of speech data has only recently become feasible. The recent trend towards ANN-HMM hybrids began with using restricted Boltzmann machines (RBMs), which can take (temporally) subsequent context into account. Comparatively recent advances in learning through minimizing "contrastive divergence" [28] enable us to approximate learning with RBMs. Compared to conventional GMM-HMMs, ANNs can easily leverage highly correlated feature inputs, such as those found in much wider temporal contexts of acoustic frames, typically 9-15 frames. Hybrid ANN-HMMs also now often directly use log mel-frequency spectral coefficients without a decorrelating discrete cosine transform [29], [30], DCTs being largely an artifact of the decorrelated mel-frequency cepstral coefficients (MFCCs) that were popular with GMMs. All of these factors have had a significant impact upon performance.

This historical deconstruction is important because the premise of the present paper is that very wide input contexts and domain-appropriate representational invariance are so important to the recent success of neural-network-based acoustic models that an ANN-HMM architecture embodying these advantages can in principle outperform other ANN architectures of potentially unlimited depth for at least some tasks. We present just such a novel architecture below, which is based upon convolutional neural networks (CNNs) [31]. CNNs are among the oldest deep neural-network architectures [32], and have enjoyed great popularity as a means for handwriting recognition. A modification of CNNs will be presented here, called *limited weight sharing*, however, which to some extent impairs their ability to be stacked unboundedly deep. We moreover illustrate the application of CNNs to ASR in detail, and provide additional experimental results on how different CNN configurations may affect final ASR performance (Section V).

CNNs have been applied to acoustic modeling before, notably by [33] and [34], in which convolution was applied over windows of acoustic frames that overlap in time in order to learn more stable acoustic features for classes such as phone, speaker and gender. Weight sharing over time is actually a much older idea that dates back to the so-called time-delay neural networks (TDNNs) [35] of the late 1980s, but TDNNs had emerged initially as a competitor with HMMs for modeling time-variation in a "pure" neural-network-based approach. That purity may be of some value to the aforementioned cognitive scientists, but it is less so to engineers. As far as modeling time variations is concerned, HMMs do relatively well at this task; convolutional methods, i.e., those that use neural networks endowed with weight sharing, local connectivity and pooling (properties that will be defined below), are probably overkill, in spite of the initially positive results of [35]. We will continue to use HMMs in our model for handling variation along the time axis, but then apply convolution on the *frequency* axis of the spectrogram. This endows the learned acoustic features with a tolerance to small shifts in frequency, such as those that may arise from differing vocal tract lengths, and has led to a significant improvement over DNNs of similar complexity on TIMIT speaker-independent phone recognition, with a relative phone error rate reduction of about 8.5%. Learning invariant representations over frequency (or time) are notoriously more difficult for standard DNNs.

Deep architectures have considerable merit. They enable a model to handle many types of variability in the speech signal. The work of [29], [36] shows that the feature representations used in the upper hidden layers of DNNs are indeed more invariant to small perturbations in the input, regardless of their putative deep structural insight or abstraction, and in a manner that leads to better model generalization and improved recognition performance, especially under speaker and environmental variations. The more crucial question we have undertaken to answer is whether even better performance might be attainable if some representational knowledge that arises from a careful study of the empirical domain can be used to explicitly handle the variations in question.[1] Vocal tract length normalization (VTLN) is another very good example of this. VTLN warps the frequency axis based on a single learnable warping factor to normalize speaker variations in the speech signals, and has been shown [41], [16] to further improve the performance of DNN-HMM hybrid models when applied to the input features. More recently, the deep architecture taking the form of recurrent neural networks, even with unstacked single-layer variants, have been reported with very competitive error rates [42].

We first review the DNN and its use within the hybrid DNN-HMM architecture (Section II). Section III explains and elaborates upon the CNN architecture and its uses in speech recognition. Section IV presents limited weight sharing and the new CNN structure that incorporates it.

II. Deep Neural Networks: A Review

Generally speaking, a *deep neural network (DNN)* refers to a feedforward neural network with more than one hidden layer. Each hidden layer has a number of units (or neurons), each of which takes all outputs of the lower layer as input, multiplies them by a weight vector, sums the result and passes it through a non-linear activation function such as *sigmoid* or *tanh* as follows:

$$o_i^{(l)} = \sigma\left(\sum_j o_j^{(l-1)} w_{j,i}^{(l)} + w_{0,i}^{(l)}\right) \tag{1}$$

where $o_i^{(l)}$ denotes the output of the i-th unit in the l-th layer, $w_{j,i}^{(l)}$ denotes the connecting weight from the j-th unit in the layer $l-1$ to the i-th unit in the l-th layer, $w_{0,i}^{(l)}$ is a bias added to the i-th unit, and $\sigma(x)$ is the non-linear activation function. In this paper, we only consider the sigmoid function, i.e., $\sigma(x) = 1/(1+\exp(-x))$. For simplicity of notation, we can represent the above computation in the following vector form:

$$o_i^{(l)} = \sigma(\mathbf{o}^{(l-1)} \cdot \mathbf{w}_i^{(l)}) \tag{2}$$

where the bias term is absorbed in the column weight vector $\mathbf{w}_i^{(l)}$ by expanding the vector $\mathbf{o}^{(l-1)}$ with an extra dimension

[1]Portions of this research program have appeared in [37], [38] and [39]. There have also been important extensions of this work to larger vocabulary speech recognition tasks and to deep-learning models that retain some of the advantages presented here [39], [40].

of 1. Furthermore, all neuron activations in each layer can be represented in the following matrix form:

$$\mathbf{o}^{(\mathbf{l})} = \sigma(\mathbf{o}^{(l-1)}\mathbf{W}^{(l)}) \quad (l = 1, 2, \ldots, L-1) \tag{3}$$

where $\mathbf{W}^{(l)}$ denotes the weight matrix of the l-th layer, with ith column $\mathbf{w}_i^{(l)}$ for any i.

The first (bottom) layer of the DNN is the input layer and the topmost layer is the output layer. For a multi-class classification problem, the posterior probability of each class can be estimated using an output softmax layer:

$$y_i = \frac{\exp(o_i^{(L)})}{\sum_j \exp(o_j^{(L)})} \tag{4}$$

where $o_i^{(L)}$ is computed as $o_i^{(L)} = \mathbf{o}^{(L-1)} \cdot \mathbf{w}_i^{(L)}$.

In the hybrid DNN-HMM model, the DNN replaces the GMMs to compute the HMM state observation likelihoods. The DNN output layer computes the state posterior probabilities which are divided by the states' priors to estimate the observation likelihoods. In the training stage, forced alignment is first performed to generate a reference state label for every frame. These labels are used in supervised training to minimize the cross-entropy function, $Q(\{\mathbf{W}^{(l)}\}) = -\sum_i d_i \log y_i$, shown here for one training frame with i ranging over all target labels. The cross-entropy objective function aims at minimizing the discrepancy between the reference target $\mathbf{d}$ and the softmax DNN prediction $\mathbf{y}$.

The derivative of Q with respect to each weight matrix, $\mathbf{W}^{(l)}$, can be efficiently computed based on the well-known error back-propagation algorithm. If we use the stochastic gradient descent algorithm to minimize the objective function, for each training sample or mini-batch, each weight matrix update can be computed as:

$$\Delta\mathbf{W}^{(l)} = \epsilon \cdot \left(\mathbf{o}^{(l-1)}\right)' \mathbf{e}^{(l)} \quad (l = 1, 2, \ldots, L) \tag{5}$$

where ϵ is the learning rate and the error signal vector in the l-th layer, $\mathbf{e}^{(l)}$, is computed backwards from the sigmoid hidden unit as follows:

$$\mathbf{e}^{(L)} = \mathbf{d} - \mathbf{y} \tag{6}$$

$$\mathbf{e}^{(l)} = \left(\mathbf{e}^{(l+1)}\left(\mathbf{W}^{(l+1)}\right)'\right) \bullet \mathbf{o}^{(l)} \bullet \left(\mathbf{1} - \mathbf{o}^{(l)}\right) \quad (l = L-1, \ldots, 2, 1) \tag{7}$$

where $\bullet$ represents element-wise multiplication of two equally sized matrices or vectors.

Because of the increased model complexity of DNNs, a pretraining algorithm is often needed, which initializes all weight matrices prior to the above back-propagation algorithm, especially when the amount of training data is limited and when no constraints are imposed on the DNN weights (see [43] for more detailed discussions). One popular method to pretrain DNNs uses the restricted Boltzmann machine (RBM) as a building block. An RBM is a generative model that models the data's probability distribution. An RBM has a set of hidden units that are used to compute a better feature representation of the input data. After learning, all RBM weights can be used as a good initialization for one DNN layer. The weights are learned one layer at a time starting from the bottom hidden layer. The hidden activations computed using the learned weights are sent as input to another RBM that can be used to initialize another layer on top. The *contrastive divergence* algorithm is normally used to learn RBM weights; see [13] for more details.

III. Convolutional Neural Networks and Their Use in ASR

The convolutional neural network (CNN) can be regarded as a variant of the standard neural network. Instead of using fully connected hidden layers as described in the preceding section, the CNN introduces a special network structure, which consists of alternating so-called *convolution* and *pooling* layers.

A. Organization of the Input Data to the CNN

In using the CNN for pattern recognition, the input data need to be organized as a number of *feature maps* to be fed into the CNN. This is a term borrowed from image-processing applications, in which it is intuitive to organize the input as a two-dimensional (2-D) array, being the pixel values at the x and y (horizontal and vertical) coordinate indices. For color images, RGB (red, green, blue) values can be viewed as three different 2-D feature maps. CNNs run a small window over the input image at both training and testing time, so that the weights of the network that looks through this window can learn from various features of the input data regardless of their absolute position within the input. *Weight sharing*, or to be more precise in our present situation, *full weight sharing* refers to the decision to use the same weights at every positioning of the window. CNNs are also often said to be *local* because the individual units that are computed at a particular positioning of the window depend upon features of the local region of the image that the window currently looks upon.

In this section, we discuss how to organize speech feature vectors into feature maps that are suitable for CNN processing. The input "image" in question for our purposes can loosely be thought of as a spectrogram, with static, delta and delta-delta features (i.e., first and second temporal derivatives) serving in the roles of red, green and blue, although, as described below, there is more than one alternative for how precisely to bundle these into feature maps.

In keeping with this metaphor, we need to use inputs that preserve locality in both axes of frequency and time. Time presents no immediate problem from the standpoint of locality. Like other DNNs for speech, a single window of input to the CNN will consist of a wide amount of context (9–15 frames). As for frequency, the conventional use of MFCCs does present a major problem because the discrete cosine transform projects the spectral energies into a new basis that may not maintain locality. In this paper, we shall use the log-energy computed directly from the mel-frequency spectral coefficients (i.e., with no DCT), which we will denote as *MFSC features*. These will be used to represent each speech frame, along with their deltas and delta-deltas, in order to describe the acoustic energy distribution in each of several different frequency bands.

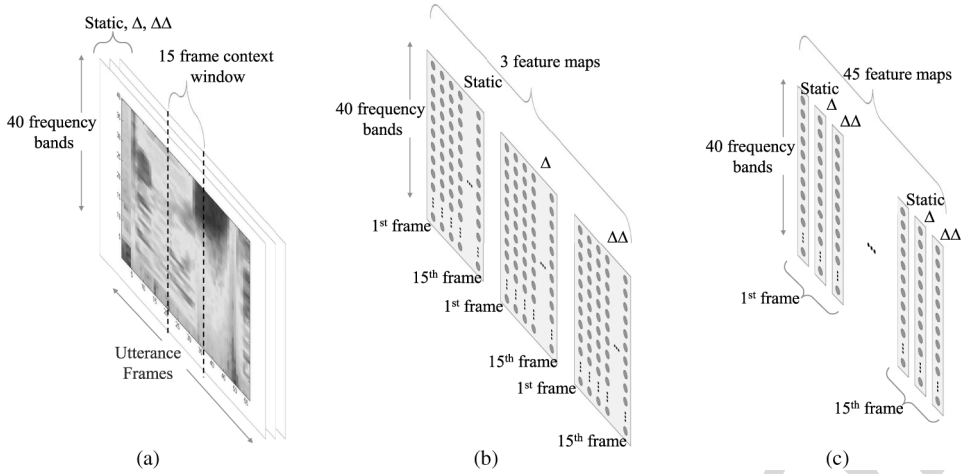


Fig. 1. Two different ways can be used to organize speech input features to a CNN. The above example assumes 40 MFSC features plus first and second derivatives with a context window of 15 frames for each speech frame.

There exist several different alternatives to organizing these MFSC features into maps for the CNN. First, as shown in Fig. 1(b), they can be arranged as three 2-D feature maps, each of which represents MFSC features (static, delta and delta-delta) distributed along both frequency (using the frequency band index) and time (using the frame number within each context window). In this case, a two-dimensional convolution is performed (explained below) to normalize both frequency and temporal variations simultaneously. Alternatively, we may only consider normalizing frequency variations. In this case, the same MFSC features are organized as a number of one-dimensional (1-D) feature maps (along the frequency band index), as shown in Fig. 1(c). For example, if the context window contains 15 frames and 40 filter banks are used for each frame, we will construct 45 (i.e., 15 times 3) 1-D feature maps, with each map having 40 dimensions, as shown in Fig. 1(c). As a result, a one-dimensional convolution will be applied along the frequency axis. In this paper, we will only focus on this latter arrangement found in Fig. 1(c), a one-dimensional convolution along frequency.

Once the input feature maps are formed, the convolution and pooling layers apply their respective operations to generate the activations of the units in those layers, in sequence, as shown in Fig. 2. Similar to those of the input layer, the units of the convolution and pooling layers can also be organized into maps. In CNN terminology, a pair of convolution and pooling layers in Fig. 2 in succession is usually referred to as one CNN "layer." A deep CNN thus consists of two or more of these pairs in succession. To avoid confusion, we will refer to convolution and pooling layers as convolution and pooling *plies*, respectively.

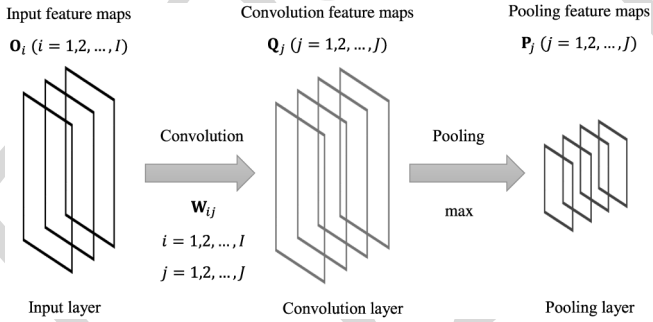


Fig. 2. An illustration of one CNN "layer" consisting of a pair of a convolution ply and a pooling ply in succession, where mapping from either the input layer or a pooling ply to a convolution ply is based on eq. (9) and mapping from a convolution ply to a pooling ply is based on eq. (10).

B. Convolution Ply

As shown in Fig. 2, every input feature map (assume I is the total number), $O_i(i = 1, \ldots, I)$, is connected to many feature maps (assume J in the total number), $Q_j(j = 1, \ldots, J)$, in the convolution ply based on a number of local weight matrices ($I \times J$ in total), $\mathbf{w}_{i,j}(i = 1, \ldots, I; j = 1, \ldots, J)$. The mapping can be represented as the well-known convolution operation in signal processing. Assuming input feature maps are all one dimensional, each unit of one feature map in the convolution ply can be computed as:

$$q_{j,m} = \sigma\left(\sum_{i=1}^{I}\sum_{n=1}^{F} o_{i,n+m-1} w_{i,j,n} + w_{0,j}\right), \quad (j = 1, \ldots, J) \tag{8}$$

where $o_{i,m}$ is the m-th unit of the i-th input feature map O_i, $q_{j,m}$ is the m-th unit of the j-th feature map Q_j in the convolution ply, $w_{i,j,n}$ is the nth element of the weight vector, $\mathbf{w}_{i,j}$, which connects the ith input feature map to the jth feature map of the convolution ply. F is called the *filter size*, which determines the number of frequency bands in each input feature map that each unit in the convolution ply receives as input. Because of the locality that arises from our choice of MFSC features, these feature maps are confined to a limited frequency range of the speech signal. Equation (8) can be written in a more concise matrix form using the convolution operator $*$ as:

$$Q_j = \sigma\left(\sum_{i=1}^{I} O_i * \mathbf{w}_{i,j}\right) \quad (j = 1, \ldots, J), \tag{9}$$

where O_i represents the i-th input feature map and $\mathbf{w}_{i,j}$ represents each local weight matrix, flipped to adhere to the convolution operation's definition. Both O_i and $\mathbf{w}_{i,j}$ are vectors if one dimensional feature maps are used, and are matrices if

two dimensional feature maps are used (where 2-D convolution is applied to the above equation), as described in the previous section. Note that, in this presentation, the number of feature maps in the convolution ply directly determines the number of local weight matrices that are used in the above convolutional mapping. In practice, we will constrain many of these weight matrices to be identical. It is also important to remember that the windows through which we view the input and apply one of these weight matrices will generally overlap. The convolution operation itself produces lower-dimensional data—each dimension decreases by filter size F minus one—but we can pad the input with dummy values (both dummy time frames and dummy frequency bands) to preserve the size of the feature maps. As a result, there could in principle be as many locations in the feature map of the convolution ply as there are in the input.

A convolution ply differs from a standard, fully connected hidden layer in two important aspects, however. First, each convolutional unit receives input only from a local area of the input. This means that each unit represents some features of a local region of the input. Second, the units of the convolution ply can themselves be organized into a number of feature maps, where all units in the same feature map share the same weights but receive input from different locations of the lower layer.

C. Pooling Ply

As shown in Fig. 2, a pooling operation is applied to the convolution ply to generate its corresponding pooling ply. The pooling ply is also organized into feature maps, and it has the same number of feature maps as the number of feature maps in its convolution ply, but each map is smaller. The purpose of the pooling ply is to reduce the resolution of feature maps. This means that the units of this ply will serve as generalizations over the features of the lower convolution ply, and, because these generalizations will again be spatially localized in frequency, they will also be invariant to small variations in location. This reduction is achieved by applying a pooling function to several units in a local region of a size determined by a parameter called *pooling size*. It is usually a simple function such as *maximization* or *averaging*. The pooling function is applied to each convolution feature map independently. When the max-pooling function is used, the pooling ply is defined as:

$$p_{i,m} = \max_{n=1}^{G} \quad q_{i,(m-1)\times s+n} \tag{10}$$

where G is the pooling size, and s, the *shift size*, determines the overlap of adjacent pooling windows. Similarly, if the average function is used, the output is calculated as:

$$p_{i,m} = r \sum_{n=1}^{G} \quad q_{i,(m-1)\times s+n} \tag{11}$$

where r is a scaling factor that can be learned. In image recognition applications, under the constraint that $G = s$, i.e., in which the pooling windows do not overlap and have no spaces between them, it has been claimed that max-pooling performs better than average-pooling [44]. In this work we will adjust G and s independently. Moreover, a non-linear activation function can be applied to the above $p_{i,m}$ to generate the final output. Fig. 3 shows a pooling ply with a pooling size of three. Each pooling unit receives input from three convolution ply units in the same feature map. If $G = s$, then the pooling ply would be one-third of the size of the convolution ply.

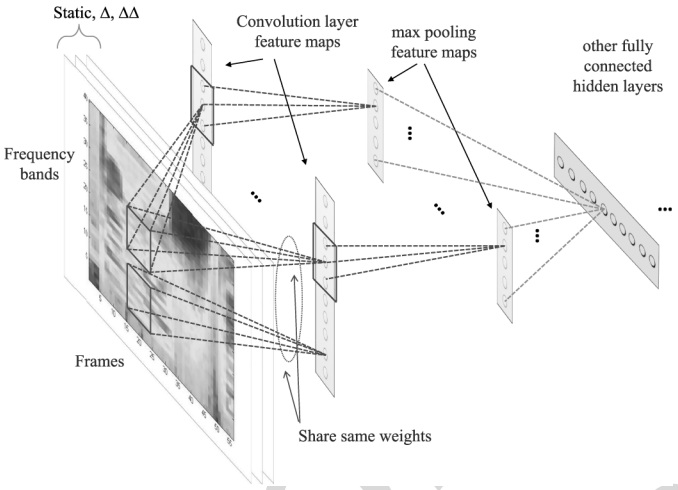


Fig. 3. An illustration of the regular CNN that uses so-called full weight sharing. Here, a 1-D convolution is applied along frequency bands.

D. Learning Weights in the CNN

All weights in the convolution ply can be learned using the same error back-propagation algorithm but some special modifications are needed to take care of sparse connections and weight sharing. In order to illustrate the learning algorithm for CNN layers, let us first represent the convolution operation in eq. (9) in the same mathematical form as the fully connected ANN layer so that the same learning algorithm in Section II can be similarly applied.

When one-dimensional feature maps are used, the convolution operations in eq. (9) can be represented as a simple matrix multiplication by introducing a large sparse weight matrix $\hat{\mathbf{W}}$ as shown in Fig. 4, which is formed by replicating a basic weight matrix $\mathbf{W}$ as in Fig. 4(a). The basic matrix $\mathbf{W}$ is constructed from all of the local weight matrices, $\mathbf{w}_{i,j}$, as follows:

$$\mathbf{W} = \begin{bmatrix} w_{1,1,1} & w_{1,2,1} & \cdots & w_{1,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{I,1,1} & w_{I,2,1} & \cdots & w_{I,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{I,1,2} & w_{I,2,2} & \cdots & w_{I,J,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{I,1,F} & w_{I,2,F} & \cdots & w_{I,J,F} \end{bmatrix}_{I\cdot F\times J} \tag{12}$$

where $\mathbf{W}$ is organized as $I \cdot F$ rows, where again F denotes filter size, each band contains I rows for I input feature maps, and $\mathbf{W}$ has J columns representing the weights of J feature maps in the convolution ply.

Meanwhile, the input and the convolution feature maps are also vectorized as row vectors $\hat{\mathbf{o}}$ and $\hat{\mathbf{q}}$. One single row vector $\hat{\mathbf{o}}$ is created from all of the input feature maps O_i $(i = 1, \ldots, I)$ as follows:

$$\hat{\mathbf{o}} = [\mathbf{v}_1 | \mathbf{v}_2 | \ \ldots \ | \mathbf{v}_M], \tag{13}$$

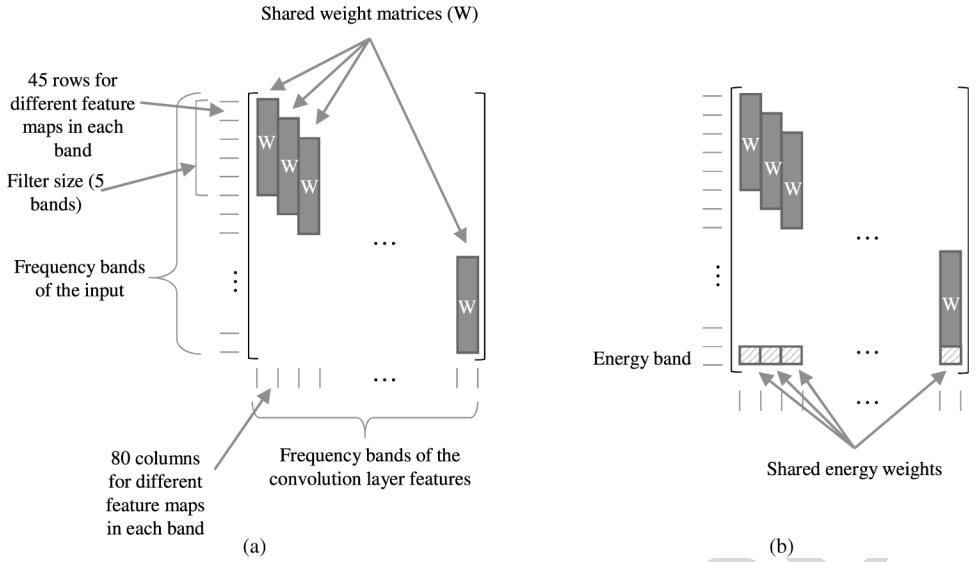


Fig. 4. All convolution operations in each convolution ply can be equivalently represented as one large matrix multiplication involving a sparse weight matrix, where both local connectivity and weight sharing can be represented in the structure of this sparse weight matrix. This figure assumes a filter size of 5, 45 input feature maps and 80 feature maps in the convolution ply. Sub-figure b shows an additional vector consisting of energy bands.

where $\mathbf{v}_m$ is a row vector containing the values of the mth frequency band along all I feature maps, and M is the number of frequency bands in the input layer. Therefore, the convolution ply outputs computed in eq. (9) can be equivalently expressed as a weight vector:

$$\hat{\mathbf{q}} = \sigma(\hat{\mathbf{o}}\hat{\mathbf{W}}) \tag{14}$$

This equation has the same mathematical form as a regular fully connected hidden layer as in eq. (2). Therefore, the convolution ply weights can be updated using the back-propagation algorithm as in eq. (5). The update for $\hat{\mathbf{W}}$ is similarly calculated as:

$$\Delta\hat{\mathbf{W}} = \epsilon \cdot \hat{\mathbf{o}}'\mathbf{e}. \tag{15}$$

The treatment of shared weights in the convolution ply is slightly different from the fully-connected DNN case where there is no weight sharing. The difference is that for the shared weights here, we sum them in their updates according to:

$$\Delta w_{i,j,n} = \sum_m \Delta\hat{\mathbf{W}}_{i+(m+n-2)\times I, j+(m-1)\times J} \tag{16}$$

where I and J are the number of feature maps in the input layer and convolution ply, respectively. Moreover, the above error vector $\mathbf{e}$ is either computed in the same way as in eq. (6) or back-propagated to the lower layer using the sparse matrix, $\hat{\mathbf{W}}$, as in eq. (7). Similarly, the biases can be handled by adding one row to the $\hat{\mathbf{W}}$ matrix to hold the bias values replicated among all convolution ply bands and adding one element with a value of one to the vector $\hat{\mathbf{o}}$.

Since the pooling ply has no weights, no learning is needed here. However, the error signals should be back-propagated to lower plies through the pooling function. In the case of max-pooling, the error signal is passed backwards only to the most active (largest) unit among each group of pooled units. That is, the error signal reaching the lower convolution ply can be computed as:

$$e^{\text{low}}_{i,n} = \sum_m e_{i,m} \cdot \delta(u_{i,m} + (m-1) \times s - n), \tag{17}$$

where $\delta(x)$ is the delta function and it has the value of 1 if x is 0 and zero otherwise, and $u_{i,m}$ is the index of the unit with the maximum value among the pooled units and is defined as:

$$u_{i,m} = \underset{n=1}{\overset{G}{\text{argmax}}} \quad q_{i,(m-1)\times s+n} \tag{18}$$

E. Pretraining CNN Layers

RBM-based pretraining improves DNN performance especially when the training set is small. Pretraining initializes DNN weights to a proper range that leads to better optimization and regularization. For convolutional structure, a convolutional RBM (CRBM) has been proposed in [45]. Similar to RBMs, the training of the CRBM aims to maximize the likelihood function of the full training data according to an approximate contrastive divergence (CD) algorithm. In CRBMs, the convolution ply activations are stochastic. CRBMs define a multinomial distribution over each pool of hidden units in a convolution ply. Hence, at most one unit in each pooled set of units can be active. This requires either having no overlap between pooled units (i.e., $G = s$) or attaching different convolution units to each pooling unit as in the limited weight sharing described below in Sec. IV. Refer to [45] for more details on CRBM-based pretraining.

F. Treatment of Energy Features

In ASR, log-energy is usually calculated per frame and appended to other spectral features. In a CNN, it is not suitable to treat energy the same way as other filter bank energies since it is the sum of the energy in all frequency bands and so does not depend on frequency. Instead, the log-energy features

should be appended as extra inputs to all convolution units as shown in Fig. 4(b). Other non-localized features can be similarly treated. The experimental results in Section V show a consistent improvement in overall system performance by using the log-energy feature. There has been some question as to whether this improvement holds in larger-scale ASR tasks [40]. Nevertheless, these experiments at least show that nothing in principle prevents frequency-independent features such as log-energy from being accommodated within a CNN architecture when they stand to improve performance.

G. The Overall CNN Architecture

The building block of the CNN contains a pair of hidden plies: a convolution ply and a pooling ply. The input contains a number of localized features organized as a number of feature maps. The size (resolution) of feature maps gets smaller at upper layers as more convolution and pooling operations are applied. Usually one or more fully connected hidden layers are added on top of the final CNN layer in order to combine the features across all frequency bands before feeding to the output layer.

In this paper, we follow the hybrid ANN-HMM framework, where we use a softmax output layer on top of the topmost layer of the CNN to compute the posterior probabilities for all HMM states. These posteriors are used to estimate the likelihood of all HMM states per frame by dividing by the states' prior probabilities. Finally, the likelihoods of all HMM states are sent to a Viterbi decoder to recognize the continuous stream of speech units.

H. Benefits of CNNs for ASR

The CNN has three key properties: locality, weight sharing, and pooling. Each one of them has the potential to improve speech recognition performance. Locality in the units of the convolution ply allows more robustness against non-white noise where some bands are cleaner than the others. This is because good features can be computed locally from cleaner parts of the spectrum and only a smaller number of features are affected by the noise. This gives a better chance to higher layers of network to handle this noise because they can combine higher level features computed for each frequency band. This is clearly better than simply handling all input features in the lower layers as in standard, fully connected neural networks. Moreover, locality reduces the number of network weights to be learned.

Weight sharing can also improve model robustness and reduce overfitting as each weight is learned from multiple frequency bands in the input instead of just from one single location. It reduces the number of weights to learn in the network, moreover. Both locality and weight sharing are needed for the property of pooling. In pooling, the same feature values computed at different locations are pooled together and represented by one value. This leads to minimal differences in the features extracted by the pooling ply when the input patterns are slightly shifted along the frequency dimension, especially when max-pooling is used. This is very helpful in handling small frequency shifts that are common in speech signals. These frequency shifts may result from differences in vocal tract lengths among different speakers. Even for the same speaker, small frequency shifts may often occur. These shifts are difficult to handle within other models such as GMMs and DNNs, where many Gaussians and hidden units are needed to handle all possible pattern shifts. Moreover, it is difficult to learn such an operation as max-pooling in a standard ANN.

The same difficulty applies to temporal differences in the speech features as well. In a hybrid ANN-HMM, a number of frames within a context window are usually processed simultaneously by the ANN. The temporal variability due to varying speaking rate may be difficult to handle. CNNs, however, can handle this type of variability naturally when convolution is applied along the contextual window frames. On the other hand, since the CNN is required to compute an output for each frame for decoding, pooling or shift size may affect the fine resolution seen by higher layers of the CNN, and a large pooling size may affect state labels' localizations. This may cause phonetic confusion, especially at segment boundaries. Hence, a suitable pooling size must be chosen.

IV. CNN with Limited Weight Sharing for ASR

A. Limited Weight Sharing (LWS)

The weight sharing scheme in Fig. 3, as described in the previous section, is *full weight sharing (FWS)*. This is the standard for CNNs as used in image processing, since the same patterns may appear at any location in an image. The properties of the speech signal typically vary over different frequency bands, however. Using separate sets of weights for different frequency bands may be more suitable since it allows for detection of distinct feature patterns in different filter bands along the frequency axis. Fig. 5 shows an example of the *limited weight sharing* (LWS) scheme for CNNs, where only the convolution units that are attached to the same pooling unit share the same convolution weights. These convolution units need to share their weights so that they compute comparable features, which may then be pooled together. In other words, each frequency band can be considered as a separate subnet with its own convolution weights. We call each of these subnets a *section* for notational convenience. Each section contains a number of feature maps in the convolution ply. Each of these feature maps is produced by using one weight vector to scan all input dimensions in this section to determine the existence or absence of this feature. The pooling size determines the number of applications of this weight vector to neighboring locations in the input space, i.e., the size of each feature map in the convolution ply equals the pooling size. Each pooling unit in this section summarizes an entire convolution feature map into one number using a pooling function, such as maximization or averaging. In mathematical terms, the convolution ply activations can be computed as:

$$q_{k,j,m} = \sigma\left(\sum_{i}\sum_{n=1}^{F} o_{i,(k-1)\times s+n+m-1} \cdot w_{k,i,j,n} + w_{k,0,j}\right) \tag{19}$$

where $w_{k,i,j,n}$ denotes the n-th convolution weight, mapping from the i-th input feature map to the j-th convolution map in the k-th section, where m ranges from 1 up to G (pooling size). The pooling ply activations in this case can be computed using:

$$p_{k,j} = \max_{m=1}^{G} q_{k,j,m}. \tag{20}$$

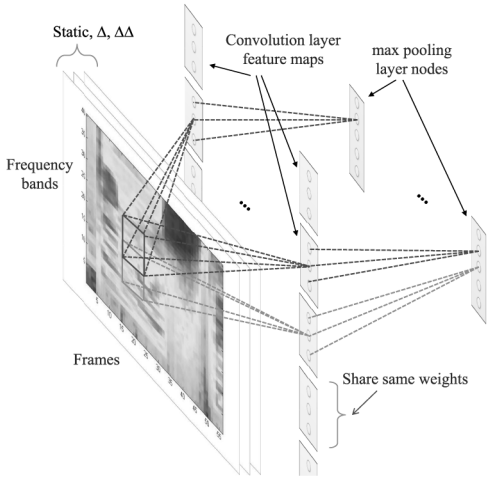


Fig. 5. An illustration of a CNN with limited weight sharing. 1-D convolution is applied along the frequency bands.

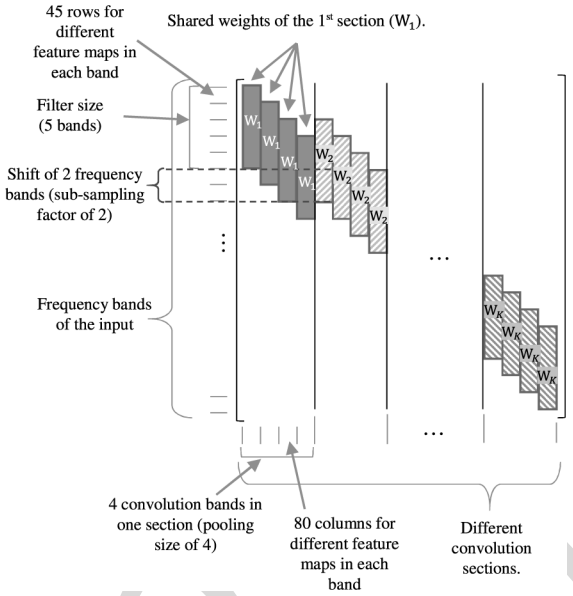


Fig. 6. The CNN layer using limited weight sharing (LWS) can also be represented as matrix multiplication using a large sparse matrix where local connectivity and weight sharing are represented in matrix form. The above figure assumes a filter size of 5, a pooling size of 4, 45 input feature maps, and 80 feature maps in the convolution ply.

Similarly, the above LWS convolution ply can also be represented with matrix multiplication using a large sparse matrix as in eq. (14) but both $\hat{\mathbf{o}}$ and $\hat{\mathbf{W}}$ need to be constructed in a slightly different way. First of all, the sparse matrix $\hat{\mathbf{W}}$ is constructed as in Fig. 6, where each $\mathbf{W}_k$ is formed based on local weights, $w_{k,i,j,n}$, as follows:

$$\mathbf{W}_k = \begin{bmatrix} w_{k,1,1,1} & w_{k,1,2,1} & \dots & w_{k,1,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,I,1,1} & w_{k,I,2,1} & \dots & w_{k,I,J,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,I,1,2} & w_{k,I,2,2} & \dots & w_{k,I,J,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,I,1,F} & w_{k,I,2,F} & \dots & w_{k,I,J,F} \end{bmatrix}_{I \cdot F \times J} \quad (k=1, 2, \dots, K) \tag{21}$$

where these matrices $\mathbf{W}_k$ differ by section and the same weight matrix is replicated G times within each section. Secondly, the convolution ply input is vectorized as described in eq. (13), and the computed feature maps are organized as a large row vector $\hat{\mathbf{q}}$ by concatenating all values in each section as follows:

$$\hat{\mathbf{q}} = [\mathbf{v}_{1,1}| \ \dots \ |\mathbf{v}_{1,G}| \ \dots \ |\mathbf{v}_{K,1}| \ \dots \ |\mathbf{v}_{K,G}], \tag{22}$$

where K is the total number of sections, G is the pooling size and $\mathbf{v}_{k,m}$ is a row vector containing the values of the units in the m-th band of the k-th section across all feature maps of the convolution ply:

$$\hat{\mathbf{v}}_{k,m} = [q_{k,1,m}, \ q_{k,2,m}, \dots q_{k,I,m}], \tag{23}$$

where I is the total number of input feature maps within each section.

Learning the weights, in the case of limited weight sharing, can be done using the same eqs. (14) and (15) with $\hat{\mathbf{W}}$ and $\hat{\mathbf{q}}$ as defined above. Meanwhile, error vectors are propagated through the max pooling function as follows:

$$\mathbf{e}^{\text{low}}_{k,i,n} = \mathbf{e}_{k,i} \cdot \delta(u_{k,i} - n) \tag{24}$$

with:

$$u_{k,i} = \underset{m=1}{\overset{G}{\text{argmax}}} \ q_{k,i,m}. \tag{25}$$

LWS also helps to reduce the total number of units in the pooling ply because each frequency band uses special weights that consider only the patterns appearing in the corresponding frequency range. As a result, a smaller number of feature maps per band should be sufficient. On the other hand, the LWS scheme does not allow for the addition of further convolution plies on top of the pooling ply since the features in different pooling-ply sections in LWS are unrelated and cannot be convolved locally. An LWS convolution ply on top of a regular full weight sharing one would be possible, however.

B. Pretraining of LWS-CNN

In this section, we propose to modify the CRBM model in [45] for pretraining the CNN with LWS as discussed in the preceding subsection. For learning the CRBM parameters, we need to define the conditional probabilities of the states of the hidden units given the visible ones and vice versa. The conditional probability of the activation for a hidden unit, $h_{k,j,m}$, which represents the state of the m-th frequency band of the j-th feature map from the k-th section, given the CRBM input $\mathbf{v}$, is defined as the following softmax function:

$$P(h_{k,j,m} = 1|\mathbf{v}) = \frac{\exp(I(h_{k,j,m}))}{\sum_{n=1}^{p} \exp(I(h_{k,j,n}))}, \tag{26}$$

where $I(h_{k,j,m})$ is the sum of the weighted signal reaching unit $h_{k,j,m}$ from the input layer and is defined as:

$$I(h_{k,j,m}) = \sum_{i} \sum_{n=1}^{f} v_{i,(k-1)\times s+n+m-1} w_{k,i,j,n} + w_{k,i,j,0} \tag{27}$$

The conditional probability distribution of $v_{i,n}$, which is the visible unit at the nth frequency band of the ith feature map, given the hidden unit states, can be computed by the following Gaussian distribution:

$$P(v_{i,n}|\mathbf{h}) = \mathcal{N}\left(v_{i,n}; \sum_{j,(k,m)\in\mathbf{C}(i,n)} h_{k,j,m} w_{k,i,j,f(n,k,m)}, \sigma^2\right) \quad (28)$$

where the above mean is the sum of the weighted signal arriving from the hidden units that are connected to the visible units, $\mathbf{C}(i,n)$ represents these connections as the set of indices of convolution bands and sections that receive input from the visible unit $v_{i,n}$, $w_{k,i,j,f(n,k,m)}$ is the weight on the link from the n-th band of the i-th input feature map to the m-th band of the j-th feature map of the k-th convolution section, $f(n,k,m)$ is a mapping function from the indices of connected nodes to the corresponding index of the filter element, and σ^2 is the variance of the Gaussian distribution and it is a fixed model parameter.

Based on the above two conditional probabilities, all connection weights of the above CRBM can be iteratively estimated by using the regular contrastive divergence (CD) algorithm. The weights of the trained CRBMs can be used as good initial values for the convolution ply in the LWS scheme. After the first convolution ply weights are learned, they are used to compute the convolution and pooling ply outputs using eqs. (19) and (20). The outputs of the pooling ply are used as inputs to continuously pretrain the next layer as done in deep belief network training [46].

V. Experiments

The experiments of this section have been conducted on two speech recognition tasks to evaluate the effectiveness of CNNs in ASR: small-scale phone recognition in TIMIT and large vocabulary voice search (VS) task. There have been extensions of the work described in this paper to other larger vocabulary speech recognition tasks that lend further support to the value of this approach [39], [40].

A. Speech Data and Analysis

The method of speech analysis is similar in the two datasets. Speech is analyzed using a 25-ms Hamming window with a fixed 10-ms frame rate. Speech feature vectors are generated by Fourier-transform-based filter-bank analysis, which includes 40 log energy coefficients distributed on a mel scale, along with their first and second temporal derivatives. All speech data were normalized so that each vector dimension has a zero mean and unit variance.

B. TIMIT Phone Recognition Results

For TIMIT, we used the standard 462-speaker training set and removed all SA records, since they may bias the results. A separate development set of 50 speakers was used for tuning all meta-parameters including the learning schedule and multiple learning rates. Results are reported using the 24-speaker core test set, which has no overlap with the development set. In addition to the log MFSC features, we added a log energy feature per frame. The log energy was normalized per utterance to have a maximum value of one, and then normalized to have zero mean and unit variance over the whole training data set. The energy feature is handled within a CNN as described in Section III.

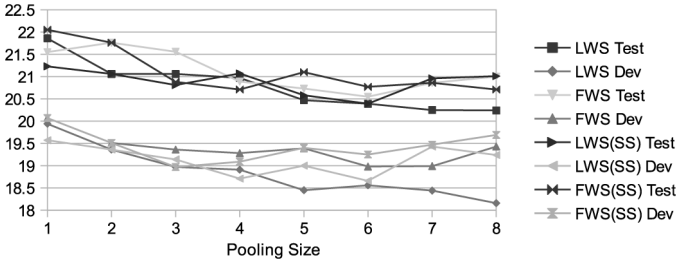


Fig. 7. Effects of different CNN pooling sizes on Phone Error Rate (PER in %) for both local weight sharing (LWS) and full weight sharing (FWS). Dev set and core test set accuracies are plotted separately. The convolution and pooling plys use a filter size of 8, 150 feature maps for FWS, and 80 feature maps per frequency band for LWS. A shift size of 2 is used with LWS and FWS while a shift size equal to the pooling size is used for LWS(SS) and FWS(SS).

We used 183 target class labels, i.e., 3 states for each HMM of 61 phones. After decoding, the original 61 phone classes were mapped to a set of 39 classes as in [47] for final scoring. In our experiments, a bigram language model over phones, estimated from the training set, was used in decoding. To prepare the ANN targets, a mono-phone HMM model was trained on the training data set, and it was used to generate state-level labels based on forced alignment. For neural-network training, learning rate annealing, in which the learning rate is steadily decreased over successive iterations, and early stopping strategies, in which a held-out development set is used to determine when overfitting has started, were utilized, as in [46].

We conducted many experiments on CNNs using both full weight sharing (FWS) and limited weight sharing (LWS) schemes. In this section, we first evaluate the ASR performance of CNNs under different settings of the CNN parameters. We normally fix all parameters except one and show how recognition performance varies with the remaining parameter. In these experiments we used one convolution ply, one pooling ply and two fully connected hidden layers on the top. The fully connected layers had 1000 units in each. The convolution and pooling parameters were: pooling size of 6, shift size of 2, filter size of 8, 150 feature maps for FWS, and 80 feature maps per frequency band for LWS. In all experiments, we fixed a random number generation seed for both weight initialization and order randomization of the training data. In the last table, we report the average of 3 runs with different seeds to compare recognition performance among DNNs, FWS-CNNs and LWS-CNNs.

1) Effects of Varying CNN Parameters: In this section, we analyze the effects of changing different CNN parameters. Figs. 7, 8, 9, and 10 show the results of these experiments on both the core test set (Test) and the development set (Dev). The figures show that both the pooling size and the number of feature maps have the most significant impact on the final ASR performance. Fig. 7 shows that all configurations yield better performance with increasing pooling size up to 6. LWS yields better performance with bigger pooling sizes. Figs. 7 and 8 show that overlapping pooling windows do not produce a clear performance gain, and that using the same value for both the pooling size and the shift size produces a similar performance while decreasing the model complexity. Fig. 9 shows that a larger number of feature maps usually leads to better performance, especially with

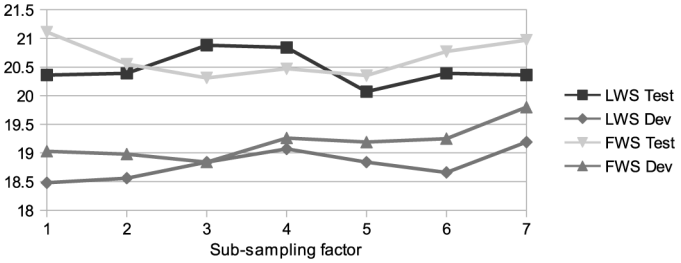


Fig. 8. Effects of different CNN shift sizes on Phone Error Rate (PER in %) for LWS and FWS. The convolution and pooling plys use a pooling size of 6, filter size of 8, 150 feature maps for FWS, and 80 feature maps per frequency band for LWS.

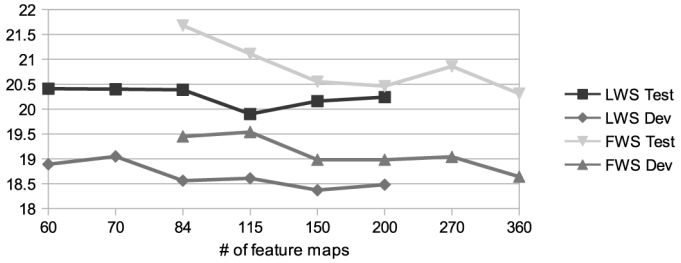


Fig. 9. Effects of different numbers of feature maps on Phone Error Rate (PER in %) for LWS and FWS. The convolution and pooling plys used a pooling size of 6, shift size of 2, filter size of 8.

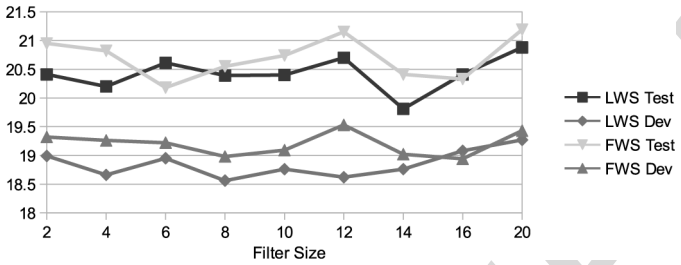


Fig. 10. Effects of different filter sizes on Phone Error Rate (PER in %) for LWS and FWS. The convolution and pooling plys use a pooling size of 6, shift size of 2, 150 feature maps for FWS, and 80 feature maps per frequency band for LWS.

TABLE I
EFFECTS OF USING ENERGY FEATURES ON PERCENT PER FOR THE TEST SET. THE CONVOLUTION AND POOLING PLYS USE A POOLING SIZE OF 6, SHIFT SIZE OF 2, FILTER SIZE OF 8, 150 FEATURE MAPS FOR FWS, AND 80 FEATURE MAPS PER FREQUENCY BAND FOR LWS

	No Energy	Energy
LWS	20.61%	20.39%
FWS	21.19%	20.55%

FWS. It also shows that LWS can achieve better performance with a smaller number of feature maps than FWS due to its ability to learn different feature patterns for different frequency bands. This indicates that the LWS scheme is more efficient in terms of the number of hidden units.

2) Effects of Energy Features: Table I shows the benefit of using energy features, producing a significant accuracy improvement, especially for FWS. While the energy features can be easily derived from other MFSC features, adding them as separate inputs to the convolution filters results in more discriminative power as it provides a way to compare the local frequency bands processed by the filter with the overall spectrum.

TABLE II
EFFECTS OF POOLING FUNCTIONS ON PERCENT PER. THE EXPERIMENTAL SETTING IS THE SAME AS TABLE I

	Average	Max
Development Set	19.63%	18.56%
Test Set	21.6%	20.39%

3) Effects of Pooling Functions: Table II shows that the max-pooling function performs better than the average function with the LWS scheme. These results are consistent with what has been observed in image recognition applications [44].

4) Overall Performance: Here we compare the overall performance of different CNN configurations with a baseline DNN system on the same TIMIT task. All results of the comparison are listed in Table III, along with the numbers of weight parameters and computations in each model. Average PERs were obtained over three runs with different random seeds. The first row shows the average PER obtained from a DNN that had three hidden layers. Its first hidden layer had 2000 units, to match the increased number of units in the CNN. The other two hidden layers had 1000 units in each. The second row reports the average PER from a similar DNN with 5 layers. The parameters of CNNs in rows 3 and 4 were chosen based on the performance obtained on the Dev set in the previous sections. Both had a filter size of 8, a pooling size of 6, and a shift size of 2. The number of feature maps was 150 for LWS and 360 for FWS. The results in Table III show that the CNN performance was much better than that of the corresponding DNN and that LWS was slightly better than FWS even with less than half the number of units in the pooling ply. Although the number of units in the LWS convolution ply was slightly larger than that of the FWS, LWS-CNN gives a much smaller model size since LWS results in far fewer weights in the upper, fully connected layers. The CNN with LWS gave more than an 8% relative reduction in PER over the DNN. The fifth row in Table III shows the performance of using two pairs of convolution and pooling plies with FWS in addition to two fully connected hidden layers on top. The sixth row shows the performance for the same model when the second convolution layer uses LWS. We coarsely tuned the two-layer parameters on the development set, and obtained a PER of 20.23% and 20.36% which show only minor differences to using one convolution layer. On the other hand, using two convolution layers tends to result in a smaller number of parameters as the fourth column shows.

C. Large Vocabulary Speech Recognition Results

In this section, we examine the recognition performance of CNNs on a large vocabulary ASR task. We used a voice search dataset containing 18 hours of speech data. Initially, a conventional state-tied triphone HMM was built. The HMM state labels were used as the targets in training both the DNNs and CNNs, which both followed the standard recipe. The first 15 epochs were run with a learning rate of 0.08, followed by 10 additional epochs with a reduced learning rate of 0.002. We investigated the effects of pretraining using an RBM for the fully connected layers and using a CRBM, as described in section 4-B, for the convolution and pooling plies. In this section, we used bigger hidden layers of 2000 units each. The DNN had three hidden

TABLE III
PERFORMANCE ON TIMIT OF DIFFERENT CNN CONFIGURATIONS, COMPARED WITH DNNS, ALONG WITH THE SIZE OF THE MODEL IN TOTAL NUMBER OF PARAMETERS, AND THE SPEED IN TOTAL NUMBER OF MULTIPLY-AND-ACCUMULATE OPERATIONS. AVERAGE PERS WERE COMPUTED OVER 3 RUNS WITH DIFFERENT RANDOM SEEDS AND SHOWN IN THE 3RD COLUMN, WHILE THE MINIMUM AND MAXIMUM PERS ARE SHOWN IN THE 4TH COLUMN. THE SECOND COLUMN SHOWS THE NETWORK STRUCTURE AND THE CONFIGURATION OF THE HIDDEN LAYERS ARE SHOWN WITHIN BRACES. THE NUMBER OF NODES OF A FULLY CONNECTED LAYER IS GIVEN DIRECTLY. FOR CNN LAYERS THE CNN LAYER PARAMETERS ARE GIVEN FOR FWS OR LWS IN BRACKETS WHERE: 'M' IS THE NUMBER OF FEATURE MAPS, 'P' IS THE POOLING SIZE, 'S' IS THE SHIFT SIZE, AND 'F' IS THE FILTER SIZE

ID	Network structure	Average PER	min-max PER	# param's	# op's
1	DNN {2000 + 2×1000}	22.02%	21.86-22.11%	6.9M	6.9M
2	DNN {2000 + 4×1000}	21.87%	21.68-21.98%	8.9M	8.9M
3	CNN {LWS(m:150 p:6 s:2 f:8) + 2×1000}	20.17%	19.92-20.41%	5.4M	10.7M
4	CNN {FWS(m:360 p:6 s:2 f:8) + 2×1000}	20.31%	20.16-20.58%	8.5M	13.6M
5	CNN {FWS(m:150 p:4 s:2 f:8) + **FWS**(m:300 p:2 s:2 f:6) + 2×1000}	20.23%	20.11-20.29%	4.5M	11.7M
6	CNN {FWS(m:150 p:4 s:2 f:8) + **LWS**(m:150 p:2 s:2 f:6) + 2×1000}	20.36%	19.91-20.61%	4.1M	7.5M

TABLE IV
PERFORMANCE ON THE VS LARGE VOCABULARY DATA SET IN PERCENT WER WITH AND WITHOUT PRETRAINING (PT). THE EXPERIMENTAL SETTING IS THE SAME AS TABLE I

	No PT	With PT
DNN	37.1%	35.4%
CNN	34.2%	33.4%

layers while the CNN had one pair of convolution and pooling plies in addition to two hidden fully connected layers. The CNN layer used limited weight sharing and had 84 feature maps per section. It had a filter size of 8, a pooling size of 6, and a shift size of 2. Moreover, the context window had 11 frames. Frame energy features were not used in these experiments.

Table IV shows that the CNN improves word error rate (WER) performance over the DNN regardless of whether pretraining is used. Similar to the TIMIT results, the CNN improves performance by about an 8% relative error reduction over the DNN in the VS task without pretraining. With pretraining, the relative word error rate reduction is about 6%. Moreover, the results show that pretraining the CNN can improve its performance, although the effect of pretraining for the CNN is not as strong as that for the DNN.

VI. CONCLUSIONS

In this paper, we have described how to apply CNNs to speech recognition in a novel way, such that the CNN's structure directly accommodates some types of speech variability. We showed a performance improvement relative to standard DNNs with similar numbers of weight parameters using this approach (about 6-10% relative error reduction), in contrast to the more equivocal results of convolving along the time axis, as earlier applications of CNNs to speech had attempted [33], [34], [35]. Our hybrid CNN-HMM approach delegates temporal variability to the HMM, while convolving along the frequency axis creates a degree of invariance to small frequency shifts, which normally occur in actual speech signals due to speaker differences.

In addition, we have proposed a new, limited weight sharing scheme that can handle speech features in a better way than the full weight sharing that is standard in previous CNN architectures such as those used in image processing. Limited weight sharing leads to a much smaller number of units in the pooling ply, resulting in a smaller model size and lower computational complexity than the full weight sharing scheme.

We observed improved performance on two ASR tasks: TIMIT phone recognition and a large-vocabulary voice search task, across a variety of CNN parameter and design settings. We determined that the use of energy information is very beneficial for the CNN in terms of recognition accuracy. Further, the ASR performance was found to be sensitive to the pooling size, but insensitive to the overlap between pooling units, a discovery that will lead to better efficiency in storage and computation. Finally, pretraining of CNNs based on convolutional RBMs was found to yield better performance in the large-vocabulary voice search experiment, but not in the phone recognition experiment. This discrepancy is yet to be examined thoroughly in our future work.

REFERENCES

[1] H. Jiang, "Discriminative training for automatic speech recognition: A survey," *Comput. Speech, Lang.*, vol. 24, no. 4, pp. 589–608, 2010.
[2] X. He, L. Deng, and W. Chou, "Discriminative learning in sequential pattern recognition—A unifying review for optimization-oriented speech recognition," *IEEE Signal Process. Mag.*, vol. 25, no. 5, pp. 14–36, Sep. 2008.
[3] L. Deng and X. Li, "Machine learning paradigms for speech recognition: An overview," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 21, no. 5, pp. 1060–1089, May 2013.
[4] G. E. Dahl, M. Ranzato, A. Mohamed, and G. E. Hinton, "Phone recognition with the mean-covariance restricted Boltzmann machine," *Adv. Neural Inf. Process. Syst.*, no. 23, 2010.
[5] A. Mohamed, T. Sainath, G. Dahl, B. Ramabhadran, G. Hinton, and M. Picheny, "Deep belief networks using discriminative features for phone recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2011, pp. 5060–5063.
[6] D. Yu, L. Deng, and G. Dahl, "Roles of pre-training and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2010.
[7] G. Dahl, D. Yu, L. Deng, and A. Acero, "Large vocabulary continuous speech recognition with context-dependent DBN-HMMs," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2011, pp. 4688–4691.
[8] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Proc. IEEE Workshop Autom. Speech Recognition Understand. (ASRU)*, 2011, pp. 24–29.
[9] N. Morgan, "Deep and wide: Multiple layers in automatic speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 7–13, Jan. 2012.
[10] A. Mohamed, G. Dahl, and G. Hinton, "Deep belief networks for phone recognition," in *Proc. NIPS Workshop Deep Learn. Speech Recognition Related Applicat.*, 2009.
[11] A. Mohamed, D. Yu, and L. Deng, "Investigation of full-sequence training of deep belief networks for speech recognition," in *Proc. Interspeech*, 2010, pp. 2846–2849.

[12] L. Deng, D. Yu, and J. Platt, "Scalable stacking and learning for building deep architectures," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process.*, 2012, pp. 2133–2136.

[13] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 30–42, Jan. 2012.

[14] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Interspeech*, 2011, pp. 437–440.

[15] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak, and A. Mohamed, "Making deep belief networks effective for large vocabulary continuous speech recognition," in *IEEE Workshop Autom. Speech Recogn. Understand. (ASRU)*, 2011, pp. 30–35.

[16] J. Pan, C. Liu, Z. Wang, Y. Hu, and H. Jiang, "Investigation of deep neural networks (DNN) for large vocabulary continuous speech recognition: Why DNN surpasses GMMs in acoustic modeling," in *Proc. ISCSLP*, 2012.

[17] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[18] T. Landauer, C. Kamm, and S. Singhal, "Learning a minimally structured back propagation network to recognize speech," in *Proc. 9th Annu. Conf. Cogn. Sci. Soc.*, 1987, pp. 531–536.

[19] D. Burr, "A neural network digit recognizer," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 1986.

[20] Q. Zhu, B. Chen, N. Morgan, and A. Stolcke, "Tandem connectionist feature extraction for conversational speech recognition," in *Machine Learning for Multimodal Interaction*. Berlin/Heidelberg, Germany: Springer , 2005, vol. 3361, pp. 223–231.

[21] H. Hermansky, D. P. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional HMM systems," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2000, vol. 3, pp. 1635–1638.

[22] F. Grézl, M. Karafiát, S. Kontár, and J. Cernocky, "Probabilistic and bottle-neck features for LVCSR of meetings," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2007, vol. 4, pp. 757–800.

[23] L. Deng, M. Seltzer, D. Yu, A. Acero, A. Mohamed, and G. Hinton, "Binary coding of speech spectrograms using a deep auto-encoder," in *Proc. Interspeech*, 2010.

[24] Y. Bao, H. Jiang, L.-R. Dai, and C. Liu, "Incoherent training of deep neural networks to de-correlate bottleneck features for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2013, pp. 6980–6984.

[25] D. Zhang, L. Deng, and M. Elmasry, *A Pipelined Neural Network Architecture For Speech Recognition, In Book: VLSI Artificial Neural Networks Engineering*. Norwell, MA, USA: Kluwer, 1994.

[26] L. Deng, K. Hassanein, and M. Elmasry, "Analysis of correlation structure for a neural predictive model with applications to speech recognition," *Neural Netw.*, vol. 7, no. 2, pp. 331–339, 1994.

[27] H. A. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*. Norwell, MA, USA: Kluwer, 1993.

[28] G. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, pp. 1771–1800, 2002.

[29] A. Mohamed, G. Hinton, and G. Penn, "Understanding how deep belief networks perform acoustic modelling," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2012, pp. 4273–4276.

[30] J. Li, D. Yu, J.-T. Huang, and Y. Gong, "Improving wideband speech recognition using mixed-bandwidth training data in CD-DNN-HMM," in *Proc. IEEE Spoken Lang. Technol. Workshop (SLT)*, 2012, pp. 131–136.

[31] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1995.

[32] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, pp. 193–202, 1980.

[33] H. Lee, P. Pham, Y. Largman, and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Proc. Adv. Neural Inf. Process. Syst. 22*, 2009, pp. 1096–1104.

[34] D. Hau and K. Chen, "Exploring hierarchical speech representations using a deep convolutional neural network," in *Proc. 11th UK Workshop Comput. Intell. (UKCI '11)*, Manchester, U.K., 2011.

[35] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 3, pp. 328–339, Mar. 1989.

[36] D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide, "Feature learning in deep neural networks - studies on speech recognition tasks," in *Proc. Int. Conf. Learn. Represent.*, 2013.

[37] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Mar. 2012, pp. 4277–4280.

[38] O. Abdel-Hamid, L. Deng, and D. Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition," in *Proc. Interspeech*, 2013.

[39] L. Deng, O. Abdel-Hamid, and D. Yu, "A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2013, pp. 6669–6673.

[40] T. N. Sainath, A.-R. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2013, pp. 8614–8618.

[41] A. Mohamed, T. Sainath, G. Dahl, B. Ramabhadran, G. Hinton, and M. Picheny, "Deep belief networks using discriminative features for phone recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2011, pp. 5060–5063.

[42] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," Tech. Rep. 14021128v1 [cs.NE], Feb. 2014, arXiv.

[43] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2013, pp. 8599–8603.

[44] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Proc. 20th Int. Conf. Artif. Neural Netw.: Part III*, Berlin/Heidelberg, Germany, 2010, pp. 92–101, Springer-Verlag ser. ICANN'10.

[45] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 609–616.

[46] A. Mohamed, G. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 14–22, Jan. 2012.

[47] K. F. Lee and H. W. Hon, "Speaker-independent phone recognition using hidden Markov models," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 37, no. 11, pp. 1641–1648, Nov. 1989.

Ossama Abdel-Hamid received his B.Sc. with honors in information technology from Cairo University, Egypt, in 2002, from where he received his M.Sc. in 2007. He is currently a Ph.D. candidate in the Computer Science Department, York University, Canada. He joined the speech research group at RDI, Egypt in the period from 2003 to 2007. Moreover, he had internships at IBM Watson research center in 2006, Google in 2008, and Microsoft Reseach in 2012. His current research focuses on improving automatic speech recognition performance using deep learning methods.

Abdel-rahman Mohamed is currently a post doctoral fellow at the University of Toronto. He finished his Ph.D. study at the University of Toronto in 2013 working on deep neural network (DNN) acoustic models for ASR. Before studying in Toronto, he received his B.Sc. and M.Sc. from the Electronics and Communication Engineering Department, Cairo University, in 2004 and 2007. Since 2004 he has been with the speech research group at the RDI Company, Egypt. Then he joined the ESAT-PSI speech group at the Katholieke Universiteit Leuven, Belgium. His research focuses on developing machine learning techniques for automatic speech recognition and understanding.

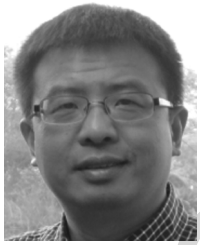
Hui Jiang (M'00–SM'11) received B.Eng. and M.Eng. degrees from the University of Science and Technology of China (USTC) and his Ph.D. degree from the University of Tokyo, Tokyo, Japan, in September 1998, all in electrical engineering.

From October 1998 to April 1999, he was a researcher in the University of Tokyo. From April 1999 to June 2000, he was with Department of Electrical and Computer Engineering, University of Waterloo, Canada, as a postdoctoral fellow. From 2000 to 2002, he was with Dialogue Systems Research, Multimedia Communication Research Lab, Bell Labs, Lucent Technologies Inc., Murray Hill, NJ. He joined Department of Computer Science and Engineering, York University, Toronto, Canada, as an Assistant Professor in fall 2002 and was promoted to Associate Professor in 2007 and to Full Professor in 2013. He served as an associate editor for IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING between 2009 and 2013. His current research interests lie in machine learning methods with applications to speech and language processing.

Li Deng received the Ph.D. from the University of Wisconsin-Madison. He was a tenured professor (1989-1999) at the University of Waterloo, Ontario, Canada, and then joined Microsoft Research, Redmond, where he is currently a Principal Research Manager of its Deep Learning Technology Center. Since 2000, he has also been an affiliate full professor at the University of Washington, Seattle, teaching computer speech processing. He has been granted over 60 U.S. or international patents, and has received numerous awards and honors bestowed by IEEE, ISCA, ASA, and Microsoft including the latest IEEE SPS Best Paper Award (2013) on deep neural nets for speech recognition. He authored or co-authored 4 books including the latest one on Deep Learning: Methods and Applications. He is a Fellow of the Acoustical Society of America, a Fellow of the IEEE, and a Fellow of the ISCA. He served as the Editor-in-Chief for IEEE SIGNAL PROCESSING MAGAZINE (2009-2011), and currently as Editor-in-Chief for IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING. His recent research interests and activities have been focused on deep learning and machine intelligence applied to large-scale text analysis and to speech/language/image multimodal processing, advancing his earlier work with collaborators on speech analysis and recognition using deep neural networks since 2009.

Gerald Penn (M'09–SM'10) is a Professor of Computer Science at the University of Toronto, where he has worked since 2001. He received his S.B. from the University of Chicago and his M.S. and Ph.D. from Carnegie Mellon University. From 1999 to 2001, he was a Member of Technical Staff in the Multimedia Communications Research Laboratory at Bell Labs. His research interests are in speech and natural language processing.

Dr. Penn is a senior member of AAAI, and a member of ACL and ISCA.



Dong Yu (M'97–SM'06) is a principal researcher at Microsoft Research - Speech and Dialog Research Group. He holds a Ph.D. degree in computer science from University of Idaho, an M.S. degree in computer science from Indiana University at Bloomington, an M.S. degree in electrical engineering from Chinese Academy of Sciences, and a B.S. degree (with honor) in electrical engineering from Zhejiang University (China). His current research interests include speech processing, robust speech recognition, discriminative training, and machine learning. He has published over 130 papers in these areas and is the inventor/coinventor of more than 50 granted/pending patents. His recent work on context-dependent deep neural network hidden Markov model (CD-DNN-HMM) has been seriously challenging the dominant position of the conventional GMM based system for large-vocabulary speech recognition and was recognized by the IEEE SPS 2013 best paper award.

Dr. Dong Yu is currently serving as a member of the IEEE Speech and Language Processing Technical Committee (2013-) and an associate editor of IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING (2011-). He has served as an associate editor of IEEE SIGNAL PROCESSING MAGAZINE (2008-2011) and the lead guest editor of IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING - special issue on deep learning for speech and language processing (2010-2011).