Convolutional Neural Networks for Acoustic Modeling of Raw Time Signal in LVCSR

Pavel Golik[1], *Zoltán Tüske*[1], *Ralf Schlüter*[1], *Hermann Ney*[1,2]

[1]Human Language Technology and Pattern Recognition, Computer Science Department, RWTH Aachen University, 52056 Aachen, Germany
[2]Spoken Language Processing Group, LIMSI CNRS, Paris, France
{golik,tuske,schlueter,ney}@cs.rwth-aachen.de

Abstract

In this paper we continue to investigate how the deep neural network (DNN) based acoustic models for automatic speech recognition can be trained without hand-crafted feature extraction. Previously, we have shown that a simple fully connected feed-forward DNN performs surprisingly well when trained directly on the raw time signal. The analysis of the weights revealed that the DNN has learned a kind of short-time time-frequency decomposition of the speech signal. In conventional feature extraction pipelines this is done manually by means of a filter bank that is shared between the neighboring analysis windows.

Following this idea, we show that the performance gap between DNNs trained on spliced hand-crafted features and DNNs trained on raw time signal can be strongly reduced by introducing 1D-convolutional layers. Thus, the DNN is forced to learn a short-time filter bank shared over a longer time span. This also allows us to interpret the weights of the second convolutional layer in the same way as 2D patches learned on critical band energies by typical convolutional neural networks.

The evaluation is performed on an English LVCSR task. Trained on the raw time signal, the convolutional layers allow to reduce the WER on the test set from 25.5% to 23.4%, compared to an MFCC based result of 22.1% using fully connected layers.
Index Terms: acoustic modeling, raw time signal, convolutional neural networks

1. Introduction

In our previous work [1] we have shown how a fully connected DNN can be trained from scratch without any feature extraction directly on the raw time signal. We also presented an analysis of the weights of the first hidden layer that revealed that the DNN tends to learn a filter bank of band pass filters. This finding nicely confirmed the time-frequency decomposition being the basic principle of popular short-term feature extraction pipelines such as MFCC [2], PLP [3] or Gammatone [4].

Since band pass filters can be expressed naturally by means of convolution in time, we decided to extend our investigations to convolutional neural networks (CNNs). The questions that we try to answer in this paper are: can we improve the DNN acoustic model trained on raw time signal if the learning of the filter bank is simplified by employing convolutional layers? How close can we then come to the conventional feature extraction methods? What can we learn from the parameters of the convolutional layers?

Several attempts have already been made to employ data-driven learning to improve feature extraction (e.g. [5, 6]), and indeed this approach leads to a better parametrization of the feature extraction pipeline. An artificial neural network is a suitable framework since it combines feature processing and the classification mechanisms, allowing to optimize both jointly. Training CNNs on the raw time signal has also been investigated in [7, 8] and has shown competitive results on smaller tasks. In this paper we perform the evaluation on a real-world LVCSR task.

This paper is structured as follows. Section 2 formally defines the convolutional layers that will be used in this work. Section 3 gives an overview of the evaluation task. The experimental results are presented in Section 4 followed by an analysis of the learned weights in Section 5. Conclusions are drawn in Section 6.

2. Convolutional neural networks

The convolutional neural networks (CNNs, [9, 10]) have become state-of-the-art models in the computer vision community [11]. In speech recognition, the CNNs have been applied in the same fashion to critical band energies (CRBE), obtained by a short-term filter bank [12, 13, 14]. We first consider a fully connected hidden layer. Given a sequence of D-dimensional observations arranged in a mini-batch of length N, the output of a hidden layer y at node i and time n is calculated from the input $x_n \in \mathbb{R}^D$ as

$$y_{i,n} = \sigma\Big(\sum_{j=0}^{D-1} w_{i,j} x_{j,n} + b_i\Big), \quad 0 \leq n < N \tag{1}$$

where $w_{i,j}$ are the weights, b_i is the bias and σ is a non-linear function such as sigmoid or rectified linear unit [15]. The free parameters $(w_{(i,\bullet)}, b_i)$ depend on the identity of the neuron i. In a typical convolutional layer that operates on both time and frequency dimensions (i.e. columns and rows of the mini-batch matrix), the weights are arranged as a 2D-patch and the parameters $(w_{(i,\bullet,\bullet)}, b_i)$ also depend on the identity of the convolutional unit i. The output of such a neuron can be evaluated at different positions (m, n):

$$y_{i,m,n} = \sigma\Big(\sum_{j=m}^{m+k-1} \sum_{h=n}^{n+k'-1} w_{i,j-m,h-n} x_{j,h} + b_i\Big) \tag{2}$$

where the weight patch $w \in \mathbb{R}^{k \times k'}$ $(k < D, k' < N)$ is multiplied with a part of the input covering the direct neighborhood of the position (m, n). The set of outputs of a convolutional unit i at all positions $\{y_{i,\bullet,\bullet}\}$ is referred to as "feature map" [10] and can be thought of as a feature stream, extracted by this unit. For brevity, we omit the discussion on patch symmetry and

boundary handling. However, within the framework of frame-wise training it is convenient to end up with a feature map that has the same number of columns as the mini-batch.

Convolutional layers need to support processing of multiple feature streams for two reasons. First, the input can consist of pre-processed features like log-Mel energies and their first/second temporal derivatives. Second, once we stack more than one convolutional layer, the output of each filter from the first layer $l-1$ is considered an input stream for the following layer l. If the R input streams are also arranged in mini-batches of the same dimension, the weight patch and the input both get an additional dimension that needs to be summed over in Eq. 2:

$$y_{i,m,n} = \sigma\Big(\sum_{j=m}^{m+k-1}\sum_{h=n}^{n+k'-1}\sum_{r=0}^{R-1} w_{i,j-m,h-n,r}\, x_{j,h,r} + b_i\Big) \quad (3)$$

Convolving the raw speech signal in time is a one-dimensional operation on a single input stream, which simplifies Eq. 3 as follows. First, the mini-batch is now constructed from consecutive signal intervals of length D (e.g. $D = 170\text{ms} \cdot 16\text{kHz} = 2720$). The output of a convolutional unit i is then defined as

$$y_{i,m} = \sigma\Big(\sum_{j=m}^{m+k-1} w_{i,j-m} x_j + b_i\Big) \quad (4)$$

where m is the position within the output vector and $w \in \mathbb{R}^k$ is a weight vector. Evaluating the inner product in Eq. 4 for each sample $m = 0, 1, 2, ...$ would be very expensive and the output would be highly correlated. For this reason, it is useful to proceed with a step size $s > 1$, such that the output feature stream is only calculated for a subset of positions: $\{y_{i,m\cdot s} : 0 \le m \cdot s < D - k\}$.

If $s = 1$, the hidden unit i performs the convolution of the input vector with the mirrored weight vector (denoted as $\tilde{w}_i$) $\sigma(x * \tilde{w}_i + b_i)$, producing an output sequence of length $D-k+1$. If $s > 1$, the output of the hidden unit i is sub-sampled by this factor after evaluating convolution. Stacking multiple 1D-convolutional layers is possible by introducing the additional dimension to the weight and the input vector analogously to Eq. 3, such that $x^{(l)}_{\bullet,r} = y^{(l-1)}_{r,\bullet}$ for two consecutive layers $l-1$ and l. A convolutional layer that consists of G filters has only $G \cdot (Rk+1)$ trainable parameters, which is usually far less than in a fully connected layer.

3. Experimental setup

The training of the DNN-HMM acoustic models is performed w.r.t. the frame-wise cross entropy criterion on 50 hours of speech from the Quaero [16] English database *train11*. The development and evaluation sets consist of ca. 3.5 hours of speech each. The recognition is performed using a 4-gram language model (LM).

In all experiments we use fully connected hidden layers with 2000 rectified linear units (ReLUs). The output layer with 4500 nodes corresponds to the generalized triphones tied by a phonetic classification and regression tree. The weights are initialized *randomly* and pre-trained discriminatively in a layer-wise fashion [17]. The mini-batches are drawn from the shuffled training utterances.

The ASR baseline system is a conventional GMM/HMM based model trained on the same data w.r.t. the maximum likelihood criterion. We apply linear discriminant analysis (LDA) to 9 consecutive MFCC frames to obtain the final 45-dimensional features. The GMM with a globally pooled diagonal covariance matrix consists of approx. 660k densities, which corresponds to about 30M trainable parameters. For acoustic training and recognition we used the RASR toolkit [18, 19].

In baseline experiments with MFCC features we feed 17 stacked frames into the DNN so that the overall length of the temporal context is approximately the same as in the experiments on the raw time signal. When processing audio sampled at 16 kHz in the same 10 ms steps as in the MFCC pipeline, a raw time signal "feature" frame consists of 160 samples from the PCM waveform. These rectangular windows are non-overlapping so that stacking 17 neighboring vectors does not result in discontinuities of the signal. The samples quantized with 16 bit need to be normalized to a numerically robust range by performing a global mean and variance normalization. This 1D operation can be interpreted as DC bias removal and loudness equalization and at the same time it serves numerical purposes to stabilize the DNN training with gradient descent.

4. Experimental results

In the first set of experiments, we trained a Gaussian Mixture Model (GMM) and two fully connected DNNs with 9 hidden layers. Table 1 shows that the DNN performs clearly better than GMM, even if trained on the raw time signal. Also, adding more layers slightly improves the DNN performance.

Table 1: *Baseline results. WER in %.*

Features	model	# hidden layers	dev	eval
MFCC	GMM	-	24.4	31.6
	DNN	9	16.9	22.1
time signal	DNN	9	20.7	26.3
	DNN	12	20.3	25.5

We start the investigation of the convolutional layers with the following topology: a single convolutional layer with a ReLU activation function is followed by a maximum pooling layer [12] with a non-overlapping pooling size of 4 and several fully connected hidden layers. Each input vector spans 170 ms of the time signal sampled at 16 kHz ($D = 2720$) and the consecutive input vectors in a mini-batch are obtained by shifting the time signal by 10 ms.

The convolutional layer has $G = 128$ filters of size $k = 256$ that are shifted by $s = 31$ samples such that every filter produces an output sequence of size $\lfloor (D-k)/s \rfloor + 1 = 80$. The pooling layer further reduces the total output dimension of this layer by factor 4 to $G \cdot 80/4 = 2560$, which has about the size of a typical fully connected hidden layer. Table 2 shows that the ASR performance saturates at 10 fully connected layers. Notably, such a CNN with just 5 fully connected layers already reaches the performance of the DNN with 12 layers (cf. Table 1).

Table 2: *Varying the number of fully connected layers in a CNN with a single convolutional layer. WER in %.*

	Fully connected layers						
	5	7	8	9	10	11	12
dev	20.3	19.5	19.1	18.9	18.6	18.7	18.5
eval	25.6	25.1	24.3	24.3	24.1	23.9	24.0

Now a filter of length $k = 256$ has an impulse response that corresponds to at most $256/16\text{kHz} = 16\text{ms}$ or a bandwidth of

at least $16\text{kHz}/256 = 62.5\text{Hz}$. Table 3 shows how the window length affects the recognition performance. It seems that 256 is a good choice and that the performance is not very sensitive to this parameter.

Table 3: *Varying the length k of the filters. WER in %.*

	Filter length k in samples			
	128	256	512	1024
dev	18.8	18.6	18.8	18.9
eval	24.2	24.1	24.1	24.3

Let us now move on to a setup with two convolutional layers. As discussed in Section 2, the input to the second hidden layer can be considered to consist of G feature streams, where every stream is an output of a filter from the first convolutional layer. Each filter in the first layer processes an interval of 2720 samples (170 ms) and outputs 80 samples (for the moment we ignore the sub-sampling by the pooling layer). While this sounds like a strong sub-sampling, in Section 5 we will see that the learned filters correspond to band passes, such that the output is band-limited and can thus be sub-sampled without too much loss of information. Similar to Eq. 4, the output of a convolutional unit in the second hidden layer can be expressed as

$$y_{i,m} = \sigma\Big(\sum_{j=m}^{m+k-1}\sum_{r=0}^{R-1} w_{i,j-m,r}x_{j,r} + b_i\Big) \qquad (5)$$

where the number of feature streams R equals the number of filters G in the first convolutional layer. So the input to every node in the second convolutional layer consists of 80 samples from each of the G filters, representing the same 170 ms of input signal. The weights are not shared between the streams, so that each filter in the second layer learns a different combination of the inputs.

For practical reasons, in the following we choose a slightly different parametrization of the CNN. Table 4 gives an overview of the important parameters.

Table 4: *Parameters of a CNN with two convolutional layers: input streams R, convolutional filters G, size of the filters k, step size s*

Layer number	R	G	k	s	pooling size
1	1	64	256	31	2
2	64	128	15	1	2

Here every filter in the second convolutional layer takes $80/2 = 40$ input samples from each of the 64 streams and produces $40 - 15 + 1 = 26$ samples. The overall output dimension of the second convolutional layer is then $128 \cdot 26/2 = 1664$. Table 5 compares the performance of the CNNs with one and two convolutional layers. Adding a second convolutional layer improves the recognition performance only slightly, but the CNN with a single convolutional layer cannot compensate the gap with more fully connected layers.

Table 5: *Results for CNNs with multiple convolutional layers.*

number of hidden layers		WER [%]	
convolutional	fully connected	dev	eval
1	10	18.6	24.1
	11	18.7	23.9
	12	18.5	24.0
2	10	18.3	23.6
	11	18.2	23.4

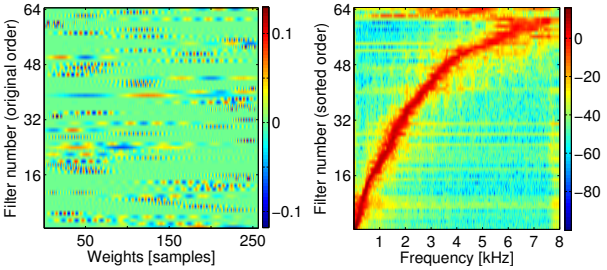

Figure 1: *Left: the weights learned by a convolutional layer. Right: magnitude spectrum of the rows on the left, sorted by the estimated center frequency.*

5. Analysis of the learned weights

5.1. First convolutional layer

Previously we proposed an analysis of the learned weights in a fully connected DNN trained on the raw time signal [1]. The basic idea was to perform a Fourier transform of each row of the first weight matrix $w_{(i,\bullet)}$ and to estimate the bandwidth f_b^i and the center frequency f_c^i of the coefficients learned by every hidden node i. Re-ordering the nodes by the estimated filter center frequency revealed that about 75% of the hidden nodes have learned a bank of relatively narrow band pass filters. This sorting is a permutation of indices $\pi(i)$. The filter bank has been found to resemble the audiologically parametrized time-frequency decomposition that is implemented in many conventional feature extraction pipelines like MFCC or PLP.

In case of a CNN trained on the raw time signal, the analysis of the weights in the first convolutional layer can be done in the same straight-forward manner as for DNNs, since the weight vector of each convolutional unit corresponds to a mirrored impulse response of a filter with finite impulse response (FIR). The weights of each filter $w_{(i,\bullet)} \in \mathbb{R}^k$ are zero-padded and a DFT is performed to obtain the magnitude spectrum of the filter response:

$$W_i = 20 \cdot \log_{10} |\text{DFT}\{w_{(i,\bullet)}\}| \in \mathbb{R}^{1\times 8000} \quad 0 \le i < G \quad (6)$$

The center frequency is estimated by a simple heuristic based on the position of the maximum in the magnitude spectrum [1]. Finally, the rows W_i are sorted by the estimated center frequency such that $f_c^{\pi(0)} \le f_c^{\pi(1)} \le \ldots \le f_c^{\pi(G-1)}$.

Figure 1 shows the weights $w_{(i,\bullet)}$ learned from a random initialization by 64 convolutional filters in the first hidden layer and the corresponding magnitude spectra with the sorted rows $W_{\pi(i)}$. This representation shows clearly that the CNN has learned a filter bank of relatively narrow band pass filters. It can also be seen that the filters are distributed non-linearly with about 78% of the filter centers below 4 kHz and a few filters (three topmost rows) that have a much larger bandwidth. Also, the center frequencies are sub-linear functions of the (sorted) filter index (cf. Mel/Bark scales).

5.2. Second convolutional layer

As discussed in Sections 2 and 4, the second convolutional layer operates on multiple input streams, one for each filter output from the filters in the first convolutional layer. In the previous

section we have seen that the learned filters in the first convolutional layer perform a time-frequency decomposition similar to an FIR filter bank. Having estimated the center frequencies of the band pass filters, we were able to reorder the hidden nodes such that neighboring filters operate on neighboring frequencies. This allows us to interpret the output of the first convolutional layer as follows: each node in the first hidden layer performs a band pass filtering of 170 ms of raw time signal, outputting 40 samples (because of the shift size $s = 31$ and the max-pooling layer with pooling size 2, cf. Table 4). Thus, for each input vector in the mini-batch, the first convolutional layer extracts a *spectrogram* or *critical band energies* $\{y_{\pi(i),\bullet}\}$ that can be naturally thought of as a matrix with 64 rows (one for each filter) and 40 columns (one for each output sample).

Since the second convolutional layer also performs convolution in time and not in frequency (similar to a Time-Delay Neural Network [9]), every hidden unit i in the second layer has a weight vector $w_{i,\bullet,\bullet} \in \mathbb{R}^{k \times R}$ of length $k = 15$ for each of the $R = 64$ input frequency bands (cf. Eq. 5). With a shift size of $s = 1$, a node in the second convolutional layer outputs $40 - 15 + 1 = 26$ samples.

Figure 2 shows some of the 128 weight vectors $(w_{i,\bullet,\bullet})^T$ learned by the second convolutional layer from a random initialization. The rows in each subplot are reordered such that the neighboring rows represent input from neighboring frequency bands. The X-axis corresponds to the time axis, although the input samples are now the sub-sampled outputs of the first layer. The weight patches shown in the plot are thus "shifted" horizontally 26 times over the critical band energies extracted by the first hidden layer. After forwarding through the max-pooling layer, the output is arranged as a vector of size $128 \cdot 26/2 = 1664$ before being fed into the fully connected hidden layers.

The filters in Figure 2 show some recognizable patterns. First, the learned weights are blurred along the time axis, which is presumably because this is the direction of the convolution, i.e. the axis along which the weights are shared between multiple positions in the input vector. Second, the weights are close to zero (green area) everywhere except for relatively small regions. In those "active" regions within the time-frequency plane, the weights appear to represent a pattern that is non-stationary in both directions (e.g. $i \in \{0, 8, 10, 14\}$). Third, many of the patterns represent a difference (i.e. a "contrast") between the regions with negative (blue areas) and positive weights (red areas). This means these filters produce a strong output when they match a similar pattern in the input spectrogram, which can occur at the onset or offset of a phonetic event characterized e.g. by some dynamics of the formants.

Further, the weights in filters number $i \in \{1, 9, 13\}$ have learned to perform a Gaussian-like filtering in time in several bands that is similar to the MRASTA filtering [20]. In contrast, filter number $i = 7$ has learned a band pass that is more time-invariant. Similar patterns have been observed when analyzing the weights learned by a conventional 2D-CNN on manually extracted filter bank outputs [21].

6. Conclusions

Previously we have demonstrated that a hybrid DNN/HMM acoustic model trained on the raw time signal can achieve reasonable recognition performance on an LVCSR task [1]. In this work we have shown that the performance can be further improved by adopting the idea from classical feature extraction

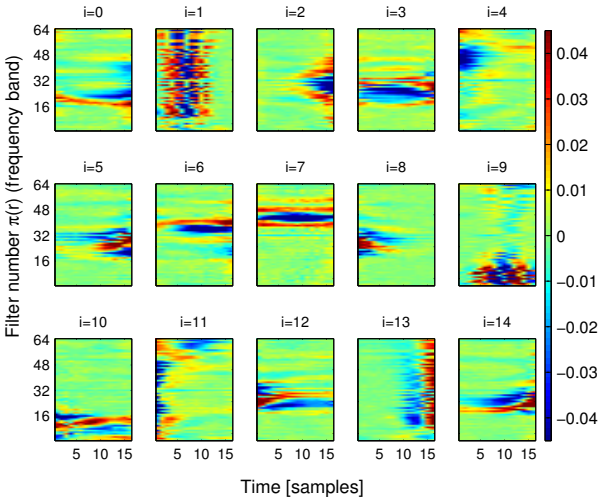

Figure 2: *Learned weights from the second convolutional layer. X-axis represents time and the Y-axis the frequency after a reordering of the rows by the filter center frequency (cf. Section 5.1). Please refer to Section 5.2 for a detailed description of the plots.*

pipelines of filter bank parameters shared across subsequent positions in time. We switched from conventional fully connected layers to one-dimensional convolutional layers that perform filtering in time. This is a natural choice for the raw time signal input that has several advantages. First, the weight sharing greatly reduces the number of trainable parameters, thus simplifying the optimization problem. Second, the learned weights can be interpreted as impulse responses of FIR filters more directly. By introducing convolutional layers to the raw time signal DNN, the WER on the test set dropped from 25.5% to 23.4%. This reduces the gap to an MFCC based result of 22.1% to only 6% relative increase in WER. We were able to repeat the analysis performed earlier on the fully connected input layer and observed the same trend: the first convolutional layer learned a filter bank of band pass filters that are non-linearly distributed in frequency. Further, we extended this analysis to the second convolutional layer by reordering the learned weights according to the estimated center frequency of the band passes from the first layer. This revealed the patterns that the filters in the second layer "look for" in the output of the first hidden layer. Some of those patterns can be interpreted as detectors of dynamic events in the time-frequency plane that are relevant for the state posterior estimation.

7. Acknowledgements

This work has received funding from the Quaero Programme funded by OSEO, French State agency for innovation. H. Ney was partially supported by a senior chair award from DIGITEO, a French research cluster in Île-de-France. Supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Defense U.S. Army Research Laboratory (DoD / ARL) contract no. W911NF-12-C-0012. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoD/ARL, or the U.S. Government.

8. References

[1] Z. Tüske, P. Golik, R. Schlüter, and H. Ney, "Acoustic modeling with deep neural networks using raw time signal for LVCSR," in *Proc. Interspeech*, Singapore, Sep. 2014, pp. 890–894.

[2] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, Aug. 1980.

[3] H. Hermansky, "Perceptual linear predictive (PLP) analysis of speech," *Journal of the Acoustical Society of America*, vol. 87, no. 4, pp. 1738–1752, 1990.

[4] R. Schlüter, I. Bezrukov, H. Wagner, and H. Ney, "Gammatone features and feature combination for large vocabulary speech recognition," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Honolulu, HI, USA, Apr. 2007, pp. 649–652.

[5] A. Biem and S. Katagiri, "Filter bank design based on discriminative feature extraction," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Adelaide, Australia, Apr. 1994, pp. 485–488.

[6] T. N. Sainath, B. Kingsbury, A.-r. Mohamed, and B. Ramabhadran, "Learning filter banks within a deep neural network framework," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Olomouc, Czech Republic, Dec. 2013, pp. 297–302.

[7] D. Palaz, R. Collobert, and M. Magimai.-Doss, "Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks," in *Proc. Interspeech*, Lyon, France, Aug. 2013, pp. 1766–1770.

[8] D. Palaz, M. Magimai.-Doss, and R. Collobert, "Convolutional neural networks-based continuous speech recognition using raw speech signal," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Brisbane, Australia, Apr. 2015, accepted for publication.

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

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2*, D. Touretzky, Ed., vol. 2. Denver, CO: Morgan Kaufman, 1990.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[12] 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. on Acoustics, Speech and Signal Processing*, Kyoto, Japan, Mar. 2012, pp. 4277–4280.

[13] H. Soltau, H. Kuo, L. Mangu, G. Saon, and T. Beran, "Neural network acoustic models for the DARPA RATS program," in *Proc. Interspeech*, Lyon, France, Aug. 2013, pp. 3092–3096.

[14] T. N. Sainath, B. Kingsbury, A. Mohamed, G. E. Dahl, G. Saon, H. Soltau, T. Beran, A. Y. Aravkin, and B. Ramabhadran, "Improvements to deep convolutional neural networks for LVCSR," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Olomouc, Czech Republic, Dec. 2013, pp. 315–320.

[15] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. of the 27th Int. Conf. on Machine Learning*, Haifa, Israel, Jun. 2010, pp. 807–814.

[16] (2013) Quaero Programme. Accessed: 2015-03-27. [Online]. Available: http://www.quaero.org

[17] F. Seide, G. Li, X. Chen, and D. Yu, "Feature engineering in context-dependent deep neural networks for conversational speech transcription," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Honolulu, HI, USA, Dec. 2011, pp. 24–29.

[18] D. Rybach, S. Hahn, P. Lehnen, D. Nolden, M. Sundermeyer, Z. Tüske, S. Wiesler, R. Schlüter, and H. Ney, "RASR - the RWTH Aachen University open source speech recognition toolkit," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Honolulu, HI, USA, Dec. 2011.

[19] S. Wiesler, A. Richard, P. Golik, R. Schlüter, and H. Ney, "RASR/NN: The RWTH neural network toolkit for speech recognition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Florence, Italy, May 2014, pp. 3313–3317.

[20] H. Hermansky and P. Fousek, "Multi-resolution RASTA filtering for TANDEM-based ASR," in *Proc. Interspeech*, Lisbon, Portugal, Sep. 2005, pp. 361–364.

[21] S. Chang and N. Morgan, "Robust CNN-based speech recognition with Gabor filter kernels," in *Proc. Interspeech*, Singapore, Sep. 2014, pp. 905–909.