

Locally-Connected and Convolutional Neural Networks for Small Footprint Speaker Recognition

Yu-hsin Chen, Ignacio Lopez-Moreno, Tara N. Sainath,
Mirkó Visontai, Raziél Alvarez, Carolina Parada

Google Inc., USA

yjc@google.com, elnota@google.com, tsainath@google.com,
mirkov@google.com, raziel@google.com, carolinap@google.com

Abstract

This work compares the performance of deep Locally-Connected Networks (LCN) and Convolutional Neural Networks (CNN) for text-dependent speaker recognition. These topologies model the local time-frequency correlations of the speech signal better, using only a fraction of the number of parameters of a fully connected Deep Neural Network (DNN) used in previous works. We show that both a LCN and CNN can reduce the total model footprint to 30% of the original size compared to a baseline fully-connected DNN, with minimal impact in performance or latency. In addition, when matching parameters, the LCN improves speaker verification performance, as measured by equal error rate (EER), by 8% relative over the baseline without increasing model size or computation. Similarly, a CNN improves EER by 10% relative over the baseline for the same model size but with increased computation.

1. Introduction

Speaker Verification (SV) is the process of verifying, based on a speaker's known utterances, whether an utterance belongs to the speaker. When the lexicon of the spoken utterances is constrained to a single word or phrase across all users, the process is referred to as *global* password Text-Dependent Speaker Verification (TD-SV). By constraining the lexicon, TD-SV compensates for phonetic variability, which poses a significant challenge in SV [1]. At Google, we target a global password TD-SV where the spoken password is given by: “*Ok Google*”. This particularly short, approximately 0.6 seconds long global password was chosen as it relates to the Google Keyword Spotting system [2] and Google VoiceSearch [3], facilitating the combination of all three systems.

Our goal is to create a small footprint TD-SV system that can run in real-time in space-constrained mobile platforms. Our constraints are *a*) total number of model parameters must be small (e.g. 0.8M parameters), and *b*) total number of operations must be small (e.g. 1.5M multiplications), in order to keep latency below 40ms on most platforms. Previous work [4] introduced our baseline system and compared it to the more standard i-vector approach. This system used a fully-connected Deep Neural Network (DNN) to extract a speaker-discriminative feature, “d-vector”, from each utterance. Utterance d-vectors were incrementally computed frame by frame, and improved latency by avoiding the computational costs associated with the latent variables of a factor analysis model [5], which occurred after utterance completion.

In this paper, we explore alternative architectures to the fully-connected feed-forward DNN architecture used to com-

pute d-vectors, with the goal of improving the equal error rate (EER) of the SV system while limiting and even reducing the number of parameters and latency. We explore locally-connected (LCN) and convolutional neural network (CNN) [6] architectures; these architectures focus on exploiting the local correlations of the speech signal. Both LCNs and CNNs are based on local receptive fields (i.e. patches), whose characteristic shape is sparse but locally dense. LCNs and CNNs have been widely used in image processing [7] and more recently in speech processing too [8, 9, 10]. Unlike in previous works, this paper uses LCNs and CNNs to directly compute speaker discriminative features while simultaneously constraining the size and latency of the model. In this paper we show that LCNs and CNNs can reduce the number of parameters in the first hidden layer by an order of magnitude with minimal performance degradation. We also show that for the same number of parameters, LCNs and CNNs can achieve better performance than fully-connected layers. Finally, we propose applying LCNs over CNNs in our global password TD-SV system because LCNs have lower latency.

This paper is organized as follows: Section 2 describes the baseline fully-connected d-vector system. Section 3 describes the LCNs and CNNs that are explored in this paper. Section 4 presents the results of two experiments: the first experiment compares models that differ only in the first hidden layer, while the second experiment compares models of the same size. Section 5 concludes the paper.

2. d-vector Baseline Model

Figure 1 contains the complete topology of the baseline fully-connected DNN and its position in the SV pipeline. Let x^t be the input features of the input layer at time t . x^t is formed by stacking q -dimensional mel-filterbank vectors by l contextual vectors to the left and r contextual vectors to the right; the total number of stacked frames is $l + r + 1$. Therefore, there are $v = q(l + r + 1)$ visible units per input x^t . The hidden layers contain units with a rectified linear unit (ReLU) activation. Each hidden layer contains k units.

The output of the DNN is a softmax layer which corresponds to the number of speakers in the development set, N . Each input has a target label, which is an integer corresponding to speaker identity. See [4] for details. The DNN is trained using the cross-entropy criterion.

For enrollment, the parameters of the DNN are fixed. We derive the d-vector speaker feature from output activations of the last hidden layer (before the softmax layer). To compute the d-vector, for every input x^t of a given utterance, we compute

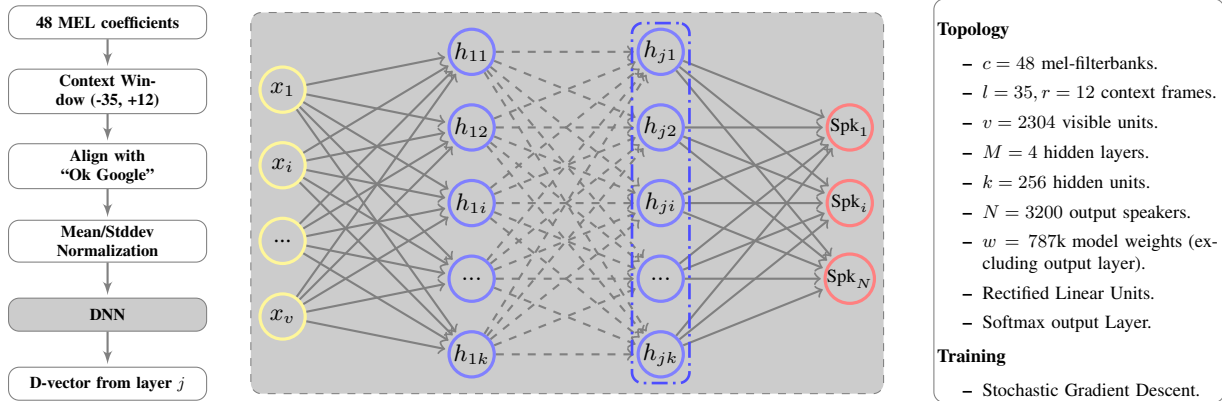


Figure 1: Pipeline process from the waveform to the final score (left). DNN topology (middle). DNN description (right).

the output activations h_j^t of the last hidden layer j , using standard feed-forward propagation. Then we take the element-wise maximum of activations to form the compact representation of that utterance, the d-vector \vec{d} . Thus, the i^{th} component of the k -dimensional d-vector \vec{d} is given by:

$$\vec{d}_i = \max_t (h_{ji}^t) \quad (1)$$

Note that in the computation of \vec{d} we do not use any of the parameters in the output layer, which can be discarded. Thus, for M hidden layers, the number of total weights w in real-time system is given by:

$$w = vk + (M - 1)k^2 \quad (2)$$

Each utterance generates exactly one d-vector. For enrollment, a speaker provides a few utterances of the global password; the d-vector from each of these utterances is averaged together to form a speaker model that is used for speaker verification, similar to the original i-vector model [11].

During evaluation, the scoring function is the cosine distance between the speaker model d-vector and the d-vector of an evaluation utterance.

3. Optimizing Local Connections

In order for our SV system to run in real-time on space-constrained platforms, the size of the DNN feature extractor must be small. However, in a fully-connected model with large number of visible units v , the term vk dominates over the rest of terms in Eq. 2; the first hidden layer accounts for most of the parameters. For example, our baseline model is a fully-connected DNN model with $v = 48 \times 48$ input elements and $k = 256$ hidden nodes in each of $M = 4$ hidden layers, such that the input layer accounts for the 75% of the model parameters. Similarly, in the previous work [4], the input layer accounted for 70% of the network parameters. Direct methods to reduce DNN size include reducing the number of hidden layers, reducing the input size by using fewer stacked context frames, and reducing the number of hidden nodes per layer; however, Table 1 shows that reducing the number of layers, context size, or hidden units strongly hurts performance. Therefore, in order to limit model size, this paper focuses on reducing the size of the first hidden layer using alternative architectures.

Although the first hidden layer contains most of our baseline fully-connected DNN model’s weights, the weight matri-

Layers	Patch	Depth	Weights	Multiplies	EER
4			787k	787k	3.88
3	48×48	256	721k	721k	4.16
4	48×48	256	787k	787k	3.88
	20×48		442k	442k	4.05
4	5×48	256	258k	258k	5.04
	48×48		128	344k	344k

Table 1: Baseline results for various configurations of fully-connected networks: with variable number of layers (*top*), with variable context sizes (*middle*) and with variable number of nodes (*bottom*.) The “Weights” column is the number of weights in each model, and represents the model footprint. The “Multiplies” column corresponds to the number of multiplications required for computing the feed-forward neural net, and represents the latency impact.

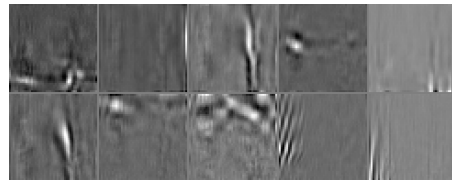


Figure 2: Weight matrices of first fully-connected layer in DNN. The weight matrices are sparse with well-localized non-zero weights.

ces of the first fully-connected hidden layer are very sparse and low-rank; Fig. 2 shows visualizations of the weight matrices from the first hidden layer. Previous works have taken note of DNN sparsity and attempted to train networks that are less sparse [12], or iteratively prune low-value weights [13]. We observe that the sparse non-zero weights are clumped close together, not scattered throughout the matrix, such that a small patch could span over the well-localized non-zero weights. This is important because we rely heavily on parallel SIMD operations (as in [14]) to efficiently compute neural nets using small dense matrices rather than large and sparse matrices. In this work, we seek to use LCN and CNN layers to take advantage of the sparse and local nature of the DNN to constrain the model size while improving performance.

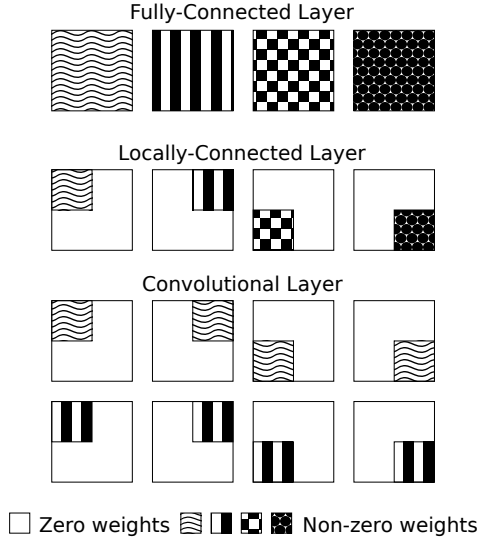


Figure 3: In a fully-connected input layer, each filter contains non-zero weights for each input element. In a LCN input layer, each filter is only non-zero for a subset of the input elements, and different filters may cover different subsets of the input. While each filter in a LCN layer covers only one patch of the input, each filter in a CNN layer covers all the patches in the input through convolution. Each patterned square corresponds to a filter matrix.

3.1. Locally Connected DNNs

To reduce the model size, we experiment with explicitly enforcing sparsity in the first hidden layer by using a LCN layer [6]. When using local connections, each of the hidden activations is the result of processing a locally-connected “patch” of v , rather than all of v as done in fully-connected DNNs. Fig. 3 compares the weight matrices of a fully-connected layer and a LCN layer, emphasizing how a LCN layer is equivalent to a sparse fully-connected layer.

Previous works suggested that any reasonable tiling of the input space, including random patches, could be sufficient to obtain high performance [15, 16]. Thus, as more sophisticated approaches may not be necessary, we use LCN with square patches of size $p \times p$ that perfectly tile the input elements in a grid with no gaps. Let v be the number of input features, p the width and length of the square patch, $n = v/p^2$ the number of patches over the input and f_{lcnn} is the number of filters over each patch. Then, the total number of filters used by the LCN layer is given by nf_{lcnn} , while the number of weights in the network is:

$$w = vf_{\text{lcnn}} + nf_{\text{lcnn}}k + (M - 2)k^2 \quad (3)$$

Here k denotes the number of nodes of the rest of the hidden layers in the network. Note by comparing (2) and (3) that the variables f_{lcnn} and n offer finer control over the number of parameters in the network. The first two hidden layers are influenced by f_{lcnn} , while remaining hidden layers have k^2 weights. One interpretation of local connections is that they enforce patch-based sparse matrices when training; given the sparse filters in the first fully-connected hidden layer (Figure 3), local connections are a natural fit. By using a LCN layer, we are implementing a sparse-coding with hand-crafted bases.

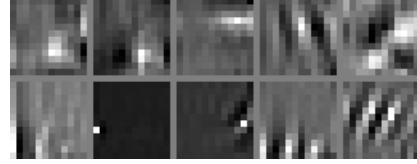


Figure 4: Filters from LCN layer with 12x12 patches.



Figure 5: Filters from CNN layer with 12x12 patches.

3.2. Convolutional Neural Networks

As Figure 4 shows, several LCN filters appear similar, suggesting further compression is possible. This motivates us to look at CNNs to reduce model size further. Like LCN, CNN also define a topology where local receptive fields, or patches, are used to model the local correlations in the input [7]. However, unlike LCN layers—where each filter is applied to a single patch in the input—in CNN layers, filters are convolved, such that all filters are applied to every input patch. (Figure 3). This approach may be interpreted as using a unique set of f_{cnn} filters repeated over all patches, versus using n sets of localized filters, each of size f_{lcnn} , as in LCN. As several LCN filters appeared similar in Figure 4, this strategy of sharing filters suggests that further compression is possible. Furthermore, previous work suggested that CNNs are particularly good in handling noisy or reverberant conditions [17, 18].

CNN layers take orders of magnitude more multiplications to compute than similarly sized fully-connected or LCN layers. In order to keep latency under 40ms on our target platforms, we limit our experiments to CNN configurations with 1.5M multiplications; under this constraint, the only configurations we consider are filters that shift with very large strides of size p when convolving. We do not use pooling layers, as they reduce speaker variance [19]. Given a 48×48 input, we present experiment results for CNN layers with $4 \times 24 \times 24$ patches, $16 \times 12 \times 12$ patches, or $64 \times 6 \times 6$ patches.

We compute the number of weights in a model with CNN first hidden layer as follows. Let v be the number of input features, p the width and length of square patch filter, $n = v/p^2$ the number of patches, f_{cnn} be the number filters from first hidden layer, and k be the number of nodes in the rest of the hidden layers; then the number of weights for a CNN model is

$$w = f_{\text{cnn}}p^2 + nf_{\text{cnn}}k + (M - 2)k^2$$

Unlike fully-connected and LCN models, the number of multiplications necessary to compute the CNN model is not equal to the number of model weights. The number of multiplications required to compute a CNN model is

$$vf_{\text{cnn}} + nf_{\text{cnn}}k + (M - 2)k^2$$

Some of the filters learned by CNN layer can be seen in Figure 5. The CNN filters appear to be smoother and sparser than the LCN filters in Figure 4.

4. Experiment results

The experiments are performed on a small footprint global password TD-SV task. The spoken password in all our datasets is given by “Ok Google” and samples are collected from anonymized real traffic from the Google KWS system [2]. The training set for our neural networks contains 3,200 anonymized speakers speaking, with an average of ~ 745 repetitions per speaker. Repetitions are recorded in multiple sessions in a wide variety of environments, including multiple devices and languages. A non-overlapped set of 3,000 speakers are present for enrollment and evaluation. Each speaker in the evaluation set enrolls with 3 to 9 utterances and it is evaluated with 7 positive utterances. In our results, all possible trials were considered, leading to $\sim 21k$ target trials and $\sim 6.3M$ non-target trials. Results are reported in Equal Error Rate (EER). We found that relative differences with other operating points are preserved.

The hidden layers generally contain 256 nodes, except in Section 4.3, when our experiment calls for matching the number of parameters between different model architectures. The focus of this paper is on experimenting with variations of the first hidden layer, which processes the input frames.

4.1. Baseline system

Our baseline system is a fully-connected neural network with 4 fully-connected hidden layers of 256 nodes each, described in Figure 1 and Section 2. Our baseline system is similar to our DNN in previous work [4], but more recent experiments suggested the following optimizations: *a)* maxout layers have been replaced by fully-connected layers with rectified linear units, *b)* in Eq. 1, the dimension-wise max function replaces the average function used before *c)* visible input elements are given by matrices of 48×48 elements instead of 41×40 , which gives us more flexibility in the configuration of patches. Note that 48×48 facilitates the definition of square patches as it is divisible by 24, 12, 8, 6, 4, 3 and 2.

4.2. Compressing first hidden layer

We experiment with only modifying the first hidden layer, fixing the last three hidden layers as fully-connected layers with 256 nodes, 66k weight parameters each. For LCN layers and CNN layers, we experiment with three patch sizes: 24×24 , 12×12 , 6×6 . In order to achieve 256 output nodes from the first hidden layer, the depth of each layer is varied with the type of layer and patch size. For example, a fully-connected layer with depth of 256 would have 256 output nodes. A LCN layer with 24×24 patch size with depth of 64 would generate 4 patches with depth 64, for a total of 256 output nodes as well.

Table 2 shows the configuration and equal error rate (EER) for each experimental model, as well model footprint and latency information. This experiment shows that the baseline fully-connected first hidden layer can be reduced from 590k parameters to 37k (6% of baseline layer) parameters with about 4% increase in EER by using a LCN layer with 12×12 patches or a CNN layer with 24×24 patches. For 4% increase in EER, we have LCN and CNN models that are 30% the size of the baseline model; in this experiment, the best LCN model and the best CNN model have the same number of parameters and similar EER.

4.3. Improve performance given size constraint

Section 4.2 focuses on reducing model size, allowing the EER to increase above that of the baseline model. In this section, we

	Patch	Depth	Weights	Multiplies	EER
Fully	48×48	256	787k	787k	3.88
	24×24	64	345k	345k	4.11
	12×12	16	234k	234k	4.02
LCN	6×6	4	206k	206k	4.54
	24×24	64	234k	345k	4.04
	12×12	16	199k	234k	4.24
CNN	6×6	4	197k	206k	4.45

Table 2: Compare fully-connected, LCN, and CNN first hidden layer. First hidden layer has 256 outputs, while the remaining hidden layers have 256 inputs and 256 outputs. “Weights” corresponds to model size. “Multiplies” corresponds to latency.

focus on closely matching the model size across different experimental models and decreasing EER. The model size is important for resource-constrained platforms. To match the model size, the first hidden layer is no longer constrained to have 256 hidden units, allowing us to increase the depth of the LCN and CNN layers. The last two hidden layers are fully-connected, have 256 inputs and outputs, and contain 66k weights.

Table 3 shows the EER, number of weights (model size), and number of multiplications (latency) for each experimental model. When parameters are matched, every LCN and CNN experimental model has smaller EER than that of the baseline fully-connected model. With approximately the same number of weights and multiplications, LCN model with 12×12 patches has EER that is 8% lower than baseline model. With approximately the same number of weights and 90% more multiplications, CNN model with 24×24 patches has EER that is 10% lower than the baseline model. When the number of model parameters is held constant, CNN models have better performance than LCN models.

	Patch	Depth	Weights	Multiplies	EER
Fully	48×48	256	787k	787k	3.88
	24×24	197	787k	787k	3.71
LCN	12×12	102	784k	784k	3.60
	6×6	35	786k	786k	3.75
	24×24	411	789k	1499k	3.52
CNN	24×24	154	785k	1117k	3.75
	24×24	40	788k	879k	3.87

Table 3: Match total number of parameters, holding last 2 hidden layers constant while varying the first 2 hidden layers. “Weights” corresponds to model size. “Multiplies” corresponds to latency.

5. Conclusions

In this paper, we compare two alternative neural network layer architectures to a fully-connected baseline for small footprint text-dependent speaker verification. Both LCN and CNN layers can be used to shrink model size to 30% of baseline with a 4% relative increase in EER (Table 2). When model size is held constant, CNN model is preferred because it reduces baseline EER by 10% relatively, versus 8% for LCN model of the same size (Table 3). If latency, which corresponds to number of model multiplications, is constrained, then the LCN model is preferred because it uses 52% fewer multiplications than CNN model, though LCN model has slightly higher EER.

6. References

- [1] H. Aronowitz, R. Hoory, J. W. Pelecanos, and D. Nahamoo, "New developments in voice biometrics for user authentication." in *INTERSPEECH*, 2011, pp. 17–20.
- [2] R. Prabhavalkar, R. Alvarez, C. Parada, P. Nakkiran, and T. N. Sainath, "Automatic gain control and multi-style training for robust small-footprint keyword spotting with deep neural networks." in *to appear at ICASSP*, 2015.
- [3] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, ""your word is my command": Google search by voice: A case study," in *Advances in Speech Recognition*. Springer, 2010, pp. 61–90.
- [4] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, 2014, pp. 4052–4056.
- [5] T. Stafylakis, P. Kenny, P. Ouellet, J. Perez, M. Kockmann, and P. Dumouchel, "Text-dependent speaker recognition using plda with uncertainty propagation," *matrix*, vol. 500, p. 1, 2013.
- [6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [7] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2, 2004, pp. II–97.
- [8] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 8614–8618.
- [9] M. McLaren, Y. Lei, N. Scheffer, and L. Ferrer, "Application of convolutional neural networks to speaker recognition in noisy conditions," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.
- [10] Y. Lei, L. Ferrer, A. Lawson, M. McLaren, and N. Scheffer, "Application of convolutional neural networks to language identification in noisy conditions," in *Proc. Speaker Odyssey Workshop (submitted)*, 2014.
- [11] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-End Factor Analysis for Speaker Verification," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 19, no. 4, pp. 788 – 798, February 2011.
- [12] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting Sparseness In Deep Neural Networks For Large Vocabulary Speech Recognition," in *ICASSP 2012*. IEEE SPS, March 2012.
- [13] B. Hassibi, D. G. Stork, and S. C. R. Com, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, 1993, pp. 164–171.
- [14] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.
- [15] A. Coates and A. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ser. ICML '11, L. Getoor and T. Scheffer, Eds. New York, NY, USA: ACM, June 2011, pp. 921–928.
- [16] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *ICCV*. IEEE, 2009, pp. 2146–2153.
- [17] M. McLaren, Y. Lei, N. Scheffer, and L. Ferrer, "Application of convolutional neural networks to speaker recognition in noisy conditions," in *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, H. Li, H. M. Meng, B. Ma, E. Chng, and L. Xie, Eds. ISCA, 2014, pp. 686–690.
- [18] H. Lee, P. T. Pham, Y. Largman, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks." in *NIPS*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1096–1104.
- [19] T. N. Sainath, A. rahman Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr." in *ICASSP*. IEEE, 2013, pp. 8614–8618.