END-TO-END STREAMING KEYWORD SPOTTING

Raziel Alvarez, Hyun-Jin Park

Google Inc.
{raziel,hjpark}@google.com

ABSTRACT

We present a system for keyword spotting that, except for a front-end component for feature generation, it is entirely contained in a deep neural network (DNN) model trained "end-to-end" to predict the presence of the keyword in a stream of audio. The main contributions of this work are, first, an efficient memoized neural network topology that aims at making better use of the parameters and associated computations in the DNN by holding a memory of previous activations distributed over the depth of the DNN. The second contribution is a method to train the DNN, end-to-end, to produce the keyword spotting score. This system significantly outperforms previous approaches both in terms of quality of detection as well as size and computation.

***Index Terms*—** deep neural networks, keyword spotting, audio processing, embedded speech recognition

1. INTRODUCTION

Keyword detection is like searching for a needle in a haystack: the detector must listen to continuously streaming audio, ignoring nearly all of it, yet still triggering correctly and instantly. In the last few years, with the advent of voice assistants, keyword spotting has become a common way to initiate a conversation with them (e.g. "Ok Google", "Alexa", or "Hey Siri"). As the assistant use cases spread through a variety of devices, from mobile phones to home appliances and further into the internet-of-things (IoT) –many of them battery powered or with restricted computational capacity, it is important for the keyword spotting system to be both high-quality as well as computationally efficient.

Neural networks are core to the state-of-the-art keyword spotting systems. These solutions, however, are not developed as a single deep neural network (DNN). Instead, they are traditionally comprised of different subsystems, independently trained, and/or manually designed. For example, a typical system is composed by three main components: 1) a signal processing frontend, 2) an acoustic encoder, and 3) a separate decoder. Of those components, it is the last two that make use of DNNs along with a wide variety of decoding implementations. They range from traditional approaches that make use of a Hidden Markov Model (HMM) to characterize acoustic features from a DNN into both "keyword" and "background" (i.e. non-keyword speech and noise) classes [1, 2, 3, 4, 5]. Simpler derivatives of that approach perform a temporal integration computation that verifies the outputs of the acoustic model are high in the right sequence for the target keyword in order to produce a single detection likelihood score [6, 7, 8, 9, 10]. Other recent systems make use of CTC-trained DNNs –typically recurrent neural networks (RNNs) [11], or even sequence-to-sequence trained models that rely on beam search decoding [12]. This last family of systems is the closest to be considered end-to-end, however they are generally too computationally complex for many embedded applications.

Optimizing independent components, however, creates added complexities and is suboptimal in quality compared to doing it suboptimal jointly. Deployment also suffers due to the extra complexity, making it harder to optimize resources (e.g. processing power and memory consumption). The system described in this paper addresses those concerns by learning both the encoder and decoder components into a single deep neural network, jointly optimizing to directly produce the detection likelihood score. This system could be trained to subsume the signal processing frontend as well as in [3, 13], but it is typically computationally costly to replace highly optimized fast Fourier transform implementations with a neural network of equivalent quality. However, it is something we consider exploring in the future. Overall, we find this system provides state-of-the-art quality across a number of audio and speech conditions compared to a traditional, non end-to-end baseline system described in [14]. Moreover, the proposed system significantly reduces the resource requirements for deployment by cutting computation and size over five times compared to the baseline system.

The rest of the paper is organized as follows. In Section 2 we present the architecture of the keyword spotting system; in particular the two main contributions of this work: the neural network topology, and the end-to-end training methodology. Next, in Section 3 we describe the experimental setup, and the results of our evaluations in Section 4, where we compare against the baseline approach of [14]. Finally, we conclude with a discussion of our findings in Section 5.

2. END-TO-END SYSTEM

This paper proposes a new end-to-end keyword spotting system that by subsuming both the encoding and decoding components into a single neural network can be trained to produce directly an estimation (i.e. score) of the presence of a keyword in streaming audio. The following two sections cover the efficient memoized neural network topology being utilized, as well as the method to train the end-to-end neural network to directly produce the keyword spotting score.

2.1. Efficient memoized neural network topology

We make use of a type of neural network layer topology called SVDF (single value decomposition filter), originally introduced in [15] to approximate a fully-connected layer with a low rank approximation. As proposed in [15] and depicted in Equation 1, the activation a for each node m in the rank-1 SVDF layer at a given inference step t can be interpreted as performing a mix of selectivity in time ($\alpha^{(m)}$) with selectivity in the feature space ($\beta^{(m)}$) over a sequence of input vectors $\mathbf{x}_t = [\mathbf{X}_{t-T}, \cdots, \mathbf{X}_t]$ of size F.

Authors thank Patrick Violette for his help with the training infrastructure to make this work possible.

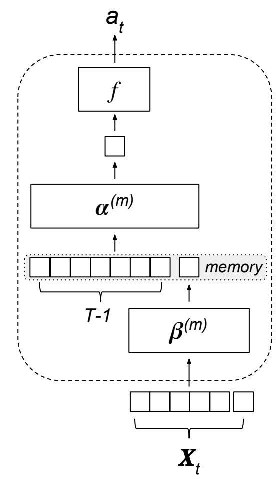


Fig. 1. A single node (m) in the SVDF layer.

$$a_t^{(m)} = f\left(\sum_{i=0}^{T-1} \alpha_i^{(m)} \sum_{j=1}^{F} \beta_j^{(m)} x_{(t-T+i),j}\right) \quad (1)$$

This is equivalent to performing, on an SVDF layer of N nodes, $N \times T$ 1-D convolutions of the feature filters β (by "sliding" each of the N filters on the input feature frames, with a stride of F), and then filtering each of the N output vectors (of size T) with the time filters α.

A more general and efficient interpretation, depicted in Figure 1, is that the layer is just processing a single input vector $\mathbf{X}_t$ at a time. Thus for each node m, the input $\mathbf{X}_t$ goes through the feature filter $\beta^{(m)}$, and the resulting scalar output gets concatenated to those $T-1$ computed in previous inference steps. The memory is initialized to zero during training for the first T inferences. Finally the time filter $\alpha^{(m)}$ is applied to them. This is how stateful networks work, where the layer is able to memorize the past within its state. Different from other stateful approaches [16], and typical recurrent layers, the SVDF does not recur the outputs into the state (memory), nor rewrites the entirety of the state with each inference. Instead, the memory keeps each inference's state isolated from subsequent runs, just pushing new entries and popping old ones based on the memory size T configured for the layer. This also means that by stacking SVDF layers we are extending the receptive field of the network. For example, a DNN with D stacked layers, each with a memory of T, means that the DNN is taking into account inputs as old as $\mathbf{X}_{t-D\times(T-1)}$. This approach works very well for streaming execution, like in speech, text, and other sequential processing, where we constantly process new inputs from a large, possibly infinite sequence but do not want to attend to all of it. An implementation is available at [17].

This layer topology offers a number of benefits over other approaches. Compared with the convolutions used in [14], it allows finer-grained control of the number of parameters and computation, given that the SVDF is composed by several relatively small filters. This is useful when selecting a tradeoff between quality, size and computation. Moreover, because of this characteristic, the SVDF allows creating very small networks that outperform other topologies which operate at larger granularity (e.g. our first stage, always-on network has about 13K parameters [8]). The SVDF also pairs very well with linear bottleneck layers to significantly reduce the parameter count as in [18, 19], and more recently in [10]. And because it allows for creating evenly sized deep networks, we can insert them throughout the network as in Figure 3. Another benefit is that due to the explicit sizing of the receptive field, it allows for a fine grained control over how much to remember from the past. This has resulted in SVDF outperforming RNN-LSTMs, which do not benefit from, and are potentially hurt by, paying attention to theoretically infinite past. It also avoids having complex logic to reset the state every few seconds as in [12].

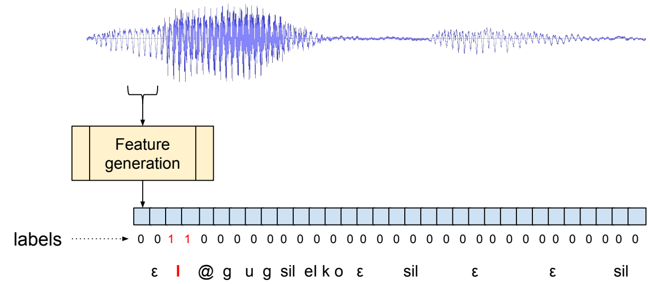


Fig. 2. Input sequence generation for "Ok google".

2.2. Method to train the end-to-end neural network

The goal of our end-to-end training is to optimize the network to produce the likelihood score, and to do so as precisely as possible. This means to have a high score right at the place where the last bit of the keyword is present in the streaming audio, and not before and particularly not much after (i.e. a "spiky" behaviour is desirable). This is important since the system is bound to an operating point defined by a threshold (between 0 and 1) that is chosen to strike a balance between false-accepts and false-rejects, and a smooth likelihood curve would add variability to the firing point. Moreover, any time in between the true end of the keyword and the point where the score meets the threshold will become latency in the system (e.g. the "assistant" will be slow to respond) –a common drawback of CTC-trained RNNs [20] we aim to avoid.

2.2.1. *Label generation*

We generate input sequences composed of pairs <$\mathbf{X}_t$, c>. Where $\mathbf{X}$ is a 1D tensor corresponding to log-mel filter-bank energies produced by a front-end as in [6, 15, 14], and c is the class label (one of $\{0, 1\}$). Each tensor $\mathbf{X}$ is first force-aligned from annotated audio utterances, using a large LVCSR system, to break up the components of the keyword [21]. For example, "ok google" is broken into: "ou", "k", "eI", "`<silence>`", "g", "u", "g", "@", "l". Then we assign labels of 1 to all sequence entries, part of a true keyword utterance, that correspond to the last component of the keyword ("l" in our "Ok google" example). All other entries are assigned a label of 0, including those that are part of the keyword but that are not its last component. See Figure 2. Additionally, we tweak the label generation by adding a fixed amount of entries with a label of 1, starting from the first vector $\mathbf{X}_t$ corresponding to the final keyword component (e.g. "l"). This is with the intention of balancing the amount of negative and positive examples, in the same spirit as [1]. This proved important to make training stable, as otherwise the amount of negative examples overpowered the positive ones.

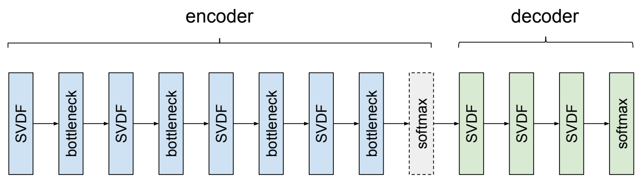


Fig. 3. End-to-end topology trained to predict the keyword likelihood score. Bottleneck layers reduce parameters and computation. The intermediate softmax is used in encoder+decoder training only.

2.2.2. *Training recipe*

The end-to-end training uses a simple frame-level cross-entropy (CE) loss that for the feature vector $\mathbf{X}_t$ is defined by $\lambda_t(\mathbf{W}) = -\log y_{c_t}(\mathbf{X}_t, \mathbf{W})$, where $\mathbf{W}$ are the parameters of the network, $y_i(\mathbf{X}_t, \mathbf{W})$ the ith output of the final softmax. Our training recipe uses asynchronous stochastic gradient descent (ASGD) to produce a single neural network that can be fed streaming input features and produce a detection score. We propose two variants of this recipe:

Encoder+decoder. A two stage training procedure where we first train an acoustic encoder, as in [6, 15, 14], and then a decoder from the outputs of the encoder (rather than filterbank energies) and the labels from 2.2.1. We do this in a single DNN by creating a final topology that is composed of the encoder and its pre-trained parameters (including the softmax), followed by the decoder. See Figure 3. During the second stage of training the encoder's parameters are frozen, such that only the decoder is trained. This recipe is useful on models that tend to overfit to subsets of the entire training dataset.

End-to-end. In this option, we train the DNN end-to-end directly, with the sequences from 2.2.1. The DNN may be any topology, but we use that of the encoder+decoder, except for the intermediate encoder softmax (now an unnecessary information bottleneck). See Figure 3. Similar to the encoder+decoder recipe, we can also initialize the encoder section with a pre-trained model, and use an adaptation rate $[0-1]$ to tune how much the encoder section is being adjusted (e.g. a rate of 0 is equivalent to the encoder+decoder recipe). This end-to-end pipeline, where the entirety of the topology's parameters are adjusted, tends to outperform the encoder+decoder one, particularly in smaller sized models which do not tend to overfit.

3. EXPERIMENTAL SETUP

In order to determine the effectiveness of our approach, we compare against a known keyword spotting system proposed in [14]. This section describes the setups used in the results section.

3.1. Front-end

Both setups use the same front-end, which generates 40-dimensional log-mel filter-bank energies out of 30ms windows of streaming audio, with overlaps of 10ms. The front-end can be queried to produce a sequence of contiguous frames centered around the *current* frame $\mathbf{x}_t = [\mathbf{X}_{t-C_l}, \cdots, \mathbf{X}_t, \cdots, \mathbf{X}_{t+C_r}]$. Older frames are said to form the left context C_l, and newer frames form the right context C_r. Additionally, the sequences can be requested with a given stride σ.

3.2. Baseline model setup

Our baseline system (Baseline_1850K) is taken from [14]. It consists of a DNN trained to predict subword targets within the keywords. The input to the DNN consists of a sequence with $C_l = 30$ frames of left and $C_r = 10$ frames of right context; each with a stride of $\sigma = 3$. The topology consists of a 1-D convolutional layer with 92 filters (of shape 8x8 and stride 8x8), followed by 3 fully-connected layers with 512 nodes and a rectified linear unit activation each. A final softmax output predicts 9 subword targets ("k" and "h" share a label for "Ok/Hey Google" detection), obtained from the same forced alignment process described in 2.2.1. This results in the baseline DNN containing 1.7M parameters, and performing 1.8M multiply-accumulate operations per inference (every 30ms of streaming audio). A keyword spotting score between 0 and 1 is computed by first smoothing the posterior values, averaging them over a sliding window of the previous 100 frames with respect to the current t; the score is then defined as the largest product of the smoothed posteriors in the sliding window as originally proposed in [7].

3.3. End-to-end model setup

The end-to-end system (prefix E2E) uses the DNN topology depicted in Figure 3, and all SVDF layers are of $rank = 1$. We present results with 3 distinct size configurations (infixes 700K, 318K, and 40K) each representing the approximate number of parameters, and the 2 training recipe variants (suffixes 1stage and 2stage) corresponding to end-to-end and encoder+decoder respectively, as described in 2.2.2. The input to all DNNs consists of a sequence with $C_l = 1$ frame of left and $C_r = 1$ frame of right context; each with a stride of $\sigma = 2$. More specifically, the E2E_700K model uses $N = 1280$ nodes in the first 4 SVDF layers, each with a memory $T = 8$, with intermediate bottleneck layers each of size 64; the following 3 SVDF layers have $N = 32$ nodes, each with a memory $T = 32$. This model performs 350K multiply-accumulate operations per inference (every 20ms of streaming audio). The E2E_318K model uses $N = 576$ nodes in the first 4 SVDF layers, each with a memory $T = 8$, with intermediate bottleneck layers each of size 64; the remainder layers are the same as E2E_700K. This model performs 159K multiply-accumulate operations per inference. Finally, the E2E_40K model uses $N = 96$ nodes in the first 4 SVDF layers, each with a memory $T = 8$, with intermediate bottleneck layers each of size 32; the remainder layers are the same as the other two models. This model performs 20K multiply-accumulate operations per inference.

3.4. Dataset

The training data for all experiments consists of 1 million anonymized hand-transcribed utterances of the keywords "Ok Google" and "Hey Google", with an even distribution. To improve robustness, we create "multi-style" training data by synthetically distorting the utterances, simulating the effect of background noise and reverberation. 8 distorted utterances are created for each original utterance; noise samples used in this process are extracted from environmental recordings of everyday events, music, and YouTube videos. Results are reported on four sets representative of various environmental conditions: *Clean non-accented* contains 170K non-accented English utterances of the keywords in "clean" conditions, plus 64K samples without the keywords (1K hours); *Clean accented* has 153K English utterances of the keywords with Australian, British, and Indian accents (also in "clean" conditions), plus 64K samples without the keywords (1K hours); *High pitched* has 1K high pitched utterances of the keywords, and 64K samples without them (1K hours); *Query logs* contains 110K keyword and 21K non-keyword utterances, collected from anonymized voice search queries. This last set contains background noises from real usage conditions.

4. RESULTS

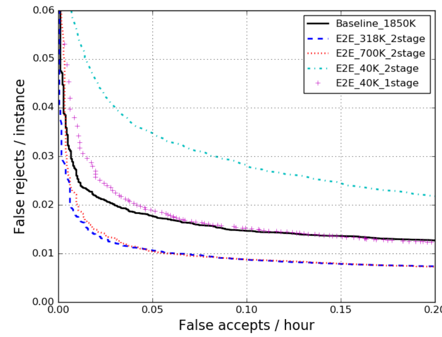
Our goal is to compare the effectiveness of the proposed approach against the baseline system described in [14]. Inference is floating point, though (unreported) TensorFlow Lite's quantization [22] numbers showed no meaningful degradation. We evaluate the false-reject (FR) and false-accept (FA) tradeoff across several end-to-end models of distinct sizes and computational complexities. As can be seen in the Receiver Operating Characteristic (ROC) curves in Figure 4, the 2 largest end-to-end models, with 2-stage training, significantly outperform the recognition quality of the much larger and complex Baseline_1850K system. More specifically, E2E_318K_2stage and E2E_700K_2stage show up to 60% relative FR rate reduction over Baseline_1850K in most test conditions. Moreover, E2E_318K_2stage uses only about 26% of the computations that Baseline_1850K uses (once normalizing their execution rates over time), but still shows significant improvements. We also explore end-to-end models at a size that, as described in [8], is small enough (in both size and computation) to be executed continuously with very little power consumption. These 2 models, E2E_40K_1stage and E2E_40K_2stage, also explore the capacity of end-to-end training (*1stage*) versus encoder+decoder training (*2stage*). The ROC curves show that *1stage* training outperforms *2stage* training on all conditions, but particularly on both "clean" environments where it gets fairly close to the performance of the baseline setup. That is a significant achievement considering E2E_40K_1stage has 2.3% the parameters and performs 3.2% the computations of Baseline_1850K. Table 1 compares the recognition quality of all setups by fixing on a very low false-accept rate of 0.1 FA per hour on a dataset containing only negative (i.e. non-keyword) utterances. Thus the table shows the false-reject rates at that operating point. Here we can appreciate similar trends as those described above: the 2 largest end-to-end models outperform the baseline across all datasets, reducing FR rate about 40% on the clean conditions, and 40%-20% on the other 2 sets depending on the model size. This table also shows how *1stage* outperforms *2stage* for small sized models, and presents similar FR rates as Baseline_1850K on clean conditions.

Table 1. FR rate over 4 test conditions at 0.1 FAh level.

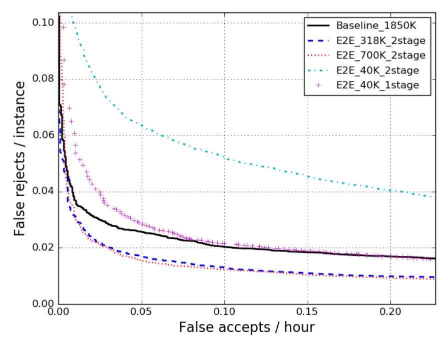
Models	Non Acc.	Accented	High P.	Q. Logs
Baseline_1850K	1.46%	2.03%	14.41%	12.13%
E2E_318K	0.87%	1.27%	11.39%	9.99%
E2E_700K	0.87%	1.22%	8.57%	8.90%
E2E_40K_2	2.80%	5.19%	26.54%	39.22%
E2E_40K_1	1.52%	2.09%	23.73%	35.32%

5. CONCLUSION

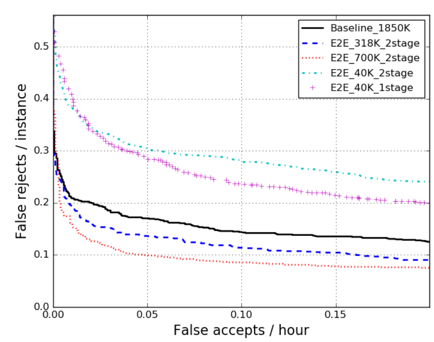
We presented a system for keyword spotting that by combining an efficient topology and two types of end-to-end training can significantly outperform previous approaches, at a much lower cost of size and computation. We specifically show how it beats the performance of a setup taken from [14] with models over 5 times smaller, and even get close to the same performance with a model over 40 times smaller. Our approach provides further benefits of not requiring anything other than a front-end and a neural network to perform the detection, and thus it is easier to extend to newer keywords and/or fine-tune with new training data. Future work includes exploring other loss-functions, as well as generalizing for multi-channel support.



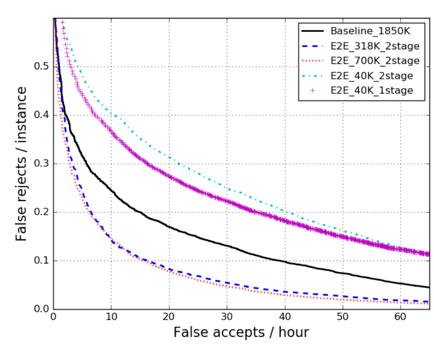
(a) Clean non-accented



(b) Clean accented



(c) High pitched voices



(d) Anonymous query logs

Fig. 4. ROC curves under different conditions.

6. REFERENCES

[1] S. Panchapagesan, M. Sun, A. Khare, S. Matsoukas, A. Mandal, B. Hoffmeister, and S. Vitaladevuni, "Multi-task learning and weighted cross-entropy for DNN-based keyword spotting," in *INTERSPEECH*, 2016.

[2] M. Sun, D. Snyder, Y. Gao, V. Nagaraja, M. Rodehorst, S. Panchapagesan, N. Strom, S. Matsoukas, and S. Vitaladevuni, "Compressed time delay neural network for small-footprint keyword spotting," in *INTERSPEECH*, 2017.

[3] K. Kumatani, S. Panchapagesan, M. Wu, M. Kim, N. Strom, G. Tiwari, and A. Mandal, "Direct modeling of raw audio with DNNs for wake word detection," *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 252–257, 2017.

[4] J. Guo, K. Kumatani, M. Sun, M. Wu, A. Raju, N. Strom, and A. Mandal, "Time-delayed bottleneck highway networks using a DFT feature for keyword spotting," in *ICASSP*. 2018, pp. 5489–5493, IEEE.

[5] M. Wu, S. Panchapagesan, M. Sun, J. Gu, R. Thomas, S.N.P. Vitaladevuni, B. Hoffmeister, and A. Mandal, "Monophone-based background modeling for two-stage on-device wake word detection," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5494–5498.

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

[7] R. Prabhavalkar, R. Alvarez, C. Parada, P. Nakkiran, and T. Sainath, "Automatic gain control and multi-style training for robust small-footprint keyword spotting with deep neural networks," in *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 4704–4708.

[8] A. Gruenstein, R. Alvarez, C. Thornton, and M. Ghodrat, "A cascade architecture for keyword spotting on mobile devices," in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

[9] Siri Team, "Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant," `https://machinelearning.apple.com/2017/10/01/hey-siri.html`, 2017, Accessed: 2018-10-06.

[10] G. Tucker, M. Wu, M. Sun, S. Panchapagesan, G. Fu, and S. Vitaladevuni, "Model compression applied to small-footprint keyword spotting," in *INTERSPEECH*. 2016, pp. 1878–1882, ISCA.

[11] Z. Wang, X. Li, and J. Zhou, "Small-footprint keyword spotting using deep neural network and connectionist temporal classifier," 09 2017.

[12] Y. He, R. Prabhavalkar, Rao. K., W. Li, A. Bakhtin, and I. McGraw, "Streaming small-footprint keyword spotting using sequence-to-sequence models," *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 474–481, 2017.

[13] T. Sainath, R. Weiss, A. Senior, K. Wilson, and O. Vinyals, "Learning the speech front-end with raw waveform CLDNNs," in *INTERSPEECH*, 2015.

[14] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting.," in *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2015, pp. 1478–1482.

[15] P. Nakkiran, R. Alvarez, R. Prabhavalkar, and C. Parada, "Compressing deep neural networks using a rank-constrained topology," in *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2015, pp. 1473–1477.

[16] S. Zhang, C. Liu, H. Jiang, S. Wei, L. Dai, and Y. Hu, "Feed-forward sequential memory networks: A new structure to learn long-term dependency," *CoRR*, vol. abs/1512.08301, 2015.

[17] TensorFlow Lite, "SVDF operation implementation," `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/lite/kernels/svdf.cc`, Accessed: 2018-10-16.

[18] F. Grézl and P. Fousek, "Optimizing bottle-neck features for LVCSR," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2008, pp. 4729–4732.

[19] T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Auto-encoder bottleneck features using deep belief networks," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, march 2012, pp. 4153–4156.

[20] A. Senior, H. Sak, F.C. Quitry, T. C. Sainath, and K. Rao, "Acoustic modelling with cd-ctc-smbr lstm rnns," in *ASRU*, 2015.

[21] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke, "Application of pretrained deep neural networks to large vocabulary speech recognition," in *Proceedings of Interspeech 2012*, 2012.

[22] R. Alvarez, R. Krishnamoorthi, S. Sivakumar, Y. Li, A. Chiao, P Warden, S. Shekhar, S. Sirajuddin, and Davis. T., "Introducing the Model Optimization Toolkit for TensorFlow," https://medium.com/tensorflow/introducing-the-model-optimization-toolkit-for-tensorflow-254aca1ba0a3, Accessed: 2019-02-17.