A Fixed-Point Neural Network For Keyword Detection on Resource Constrained Hardware

Mohit Shah*, Jingcheng Wang†, David Blaauw†, Dennis Sylvester†, Hun-Seok Kim†, and Chaitali Chakrabarti*
*School of Electrical, Computer and Energy Engineering
Arizona State University, Tempe, Arizona 85287
Email: {mohit.shah, chaitali}@asu.edu
†Department of Electrical Engineeering and Computer Science
University of Michigan, Ann Arbor, MI 48109
Email: {jiwang,blaauw,dennis,hunseok}@umich.edu

***Abstract*—Keyword detection is typically used as a front-end to trigger automatic speech recognition and spoken dialog systems. The detection engine needs to be continuously listening, which has strong implications on power and memory consumption. In this paper, we devise a neural network architecture for keyword detection and present a set of techniques for reducing the memory requirements in order to make the architecture suitable for resource constrained hardware. Specifically, a fixed-point implementation is considered; aggressively scaling down the precision of the weights lowers the memory compared to a naive floating-point implementation. For further optimization, a node pruning technique is proposed to identify and remove the least active nodes in a neural network. Experiments are conducted over 10 keywords selected from the Resource Management (RM) database. The trade-off between detection performance and memory is assessed for different weight representations. We show that a neural network with as few as 5 bits per weight yields a marginal and acceptable loss in performance, while requiring only 200 kilobytes (KB) of on-board memory and a latency of 150 ms. A hardware architecture using a single multiplier and a power consumption of less than 10mW is also presented.**

I. Introduction

Keyword detection refers to the task of identifying selected keywords embedded in a stream of words. This system is typically used as a front-end for automatic speech recognition (ASR) and spoken dialog systems (SDS). A device is triggered to wake up if a specific keyword is detected and enters a fully operational mode, where it performs speech recognition and provides appropriate responses. Hence, the speech recognition engine does not need to be operated continuously, reducing the power consumption by a huge margin. However, the keyword detection system still needs to be *always on*, i.e. continuously listening, which has strong implications on power consumption. As a result, there is a strong need to develop an architectural framework for keyword detection with minimal power consumption and memory footprint.

There is a vast amount of literature identifying various methods for keyword detection. Existing methods can be broadly classified as follows - (i) perform complete speech recognition over the phrase and then detect the keyword by looking at the transcriptions provided [1]–[3], (ii) train separate models for the keyword and out-of-vocabulary (OOV) words, and detect keywords based on the likelihood over each model. The first method requires the entire phrase to be uttered completely, i.e. offline. It also requires a complete ASR system, which is computationally intensive because of the exhaustive search required to perform transcription. The second method is relatively simple and can be performed in an online setting. It is more suited for applications where the set of keywords to be detected is known beforehand. There are multiple techniques available for performing keyword detection using the latter approach. Until recently, techniques based on Gaussian Mixture Models (GMM) for acoustic modeling and Hidden Markov Models (HMM) for modeling the sequence of words were quite common [4]–[8]. The OOV words were modeled using a garbage or a filler model, while a separate GMM-HMM was trained for each keyword. The most likely state sequence was then identified using the Viterbi algorithm. GMMs can be easily implemented in a parallel fashion, however, the Viterbi step is inherently sequential, which increases the control overhead and latency.

Recently, neural network (NN) based methods have shown tremendous success on speech recognition tasks. This success has come after advances made in the field of deep learning, which allows for efficient training of a network with many hidden layers and a large number of neurons (nodes) per layer [9], [10]. These networks are well-suited to capture the complex, non-linear patterns from the acoustic properties of speech. Detection is again straightforward; a matrix-vector multiplication step followed by a non-linear operation at each layer. Such operations can be easily extended for parallel implementations, thus offering a lower latency and a uniform architecture compared to the aforementioned HMM-based methods. One such approach for keyword detection was presented in [10]. In spite of the low-latency algorithm and highly accurate detection performance, the network is quite large, requiring upto a few million multiplications every few milliseconds as well as large memory banks for storing these weights. Mobile devices are often constrained in the amount of available hardware resources, making this approach less suited for practical applications.

In this paper, we devise a neural network architecture for keyword detection and present a set of techniques for reducing the memory overhead of such architectures. Specifically, a fixed-point implementation is considered. Aggressively scaling down the precision of the weights can significantly lower the memory requirements compared to a naive floating-point implementation. For further optimization, a node pruning technique is proposed. According to this technique, the nodes in

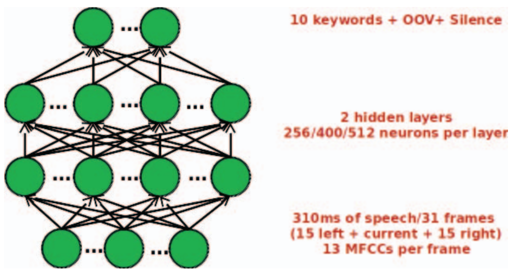
10 keywords + OOV+ Silence
2 hidden layers
256/400/512 neurons per layer
310ms of speech/31 frames
(15 left + current + 15 right)
13 MFCCs per frame

Fig. 1. A neural network architecture for keyword detection consisting of 1 input layer, 2 hidden layers and an output layer. The number of nodes in each hidden layer range from 256 to 512.

each hidden layer are first assigned a probability based on their activation patterns. This is followed by pruning away the least active nodes based on a specific threshold over the activation probabilities. To evaluate these techniques, experiments are conducted over 10 keywords selected from the Resource Management (RM) database [11]. The trade-off between detection performance and memory requirements is assessed for different precisions and different thresholds for pruning. We show that a neural network with as few as 5 bits per weight yields a marginal and acceptable loss in performance, while requiring only 200 kilobytes (KB) of on-board memory and a latency of 150 ms. A hardware architecture using a single multiplier and memory bank, and a power consumption of less than 10mW is also presented.

The remainder of this paper is organized as follows. An overview of the proposed approach and memory reduction techniques is described in Section II. Our experimental results are presented in Section III. Finally, the conclusions are presented in Section IV.

II. Proposed Approach

A. Preprocessing

The Resource Management (RM) database [11] consists of phrases recorded for scenarios pertaining to the naval forces. Speech is processed at a frame rate of 100 frames/second, i.e. a window size of 25ms and step size of 10ms. The first 13 Mel frequency coefficients (MFCC) are extracted for each frame. These features are augmented with MFCCs of the 15 previous frames and 15 future frames to form a 403-D feature vector per frame. This corresponds to 31 frames (13 MFCCs/frame) of 310ms of speech; the average word duration for this database was 300ms, hence, this choice was deemed to be appropriate for modeling words or sub-word units. Ten keywords - *ships*, *list*, *chart*, *display*, *fuel*, *show*, *track*, *submarine*, *latitude* and *longitude* were selected in this work. Forced-alignment was performed using the Kaldi speech recognition toolkit [12] in order to obtain the word boundaries. Each frame is then labeled as either one of the 10 keywords or OOV or silence. The speaker-independent train and test partitions are already specified with the database; there are 109 and 59 speakers in the training and test set, respectively. The speech features are z-normalized to zero mean and unit variance for each speaker.

B. Neural Network

The feedforward neural network used in this study is shown in Figure 1. The network consists of an input layer, two hidden layers and an output layer. The input layer consists of 403 nodes corresponding to the MFCCs extracted above. Denoting the input layer as x_i, where $i = 1, 2, ..., N$ is the number of nodes in the input layer, the computations involved for the input layer to the first hidden layer (h^1) are given below -

$$z_j^1 = \sum_{i=1}^{N} W_{ij}^1 x_i + b_j^1 \tag{1}$$

Here W^1 and b^1 refer to the weights and biases of this layer. A non-linear, rectified linear operation [13] is then applied over these intermediate values. Rectified linear (ReLU) units have attained popularity as opposed to the conventional sigmoid/logistic function as they capture more detailed information. Furthermore, they are relatively straightforward to implement in hardware as they require only a comparison operation, according to Eq. 2. In comparison, a sigmoid operation is implemented using a Taylor series expansion and is quite costly.

$$h_j^1 = \max(0, z_j^1) \tag{2}$$

The computations from the first hidden layer to the second hidden layer are the same as Eqs (1) and (2). The output layer is modeled as a softmax layer with $K+2$ nodes. Out of these, K nodes correspond to the K pre-defined keywords that are to be detected and the remaining 2 nodes correspond to OOV and silence. The softmax output yields a probability estimate for each of the K possible outputs for the current frame.

Training is performed by minimizing the cross-entropy error cost function. Backpropagation is applied to iteratively update the weights and biases of each layer. Mini-batch stochastic gradient with a batchsize of 500 samples is used for optimization. The network is trained for a total of 10 epochs with a learning rate of 0.001 and a momentum of 0.8. In our experiments, the number of layers was varied from 1 to 3, while the number of nodes for each hidden layer was selected from 256, 400 or 512. The optimal values were determined via validation on a randomly selected subset of the training set.

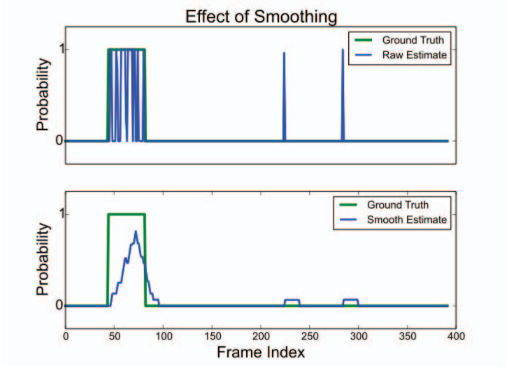

Fig. 2. An example of the smoothing process for detection of *ships* in the phrase *do any ships that are in pacific fleet have SPS-48.* (Top) Frame-wise, raw posterior probability estimates, and (Bottom) Frame-wise, smoothed probability estimates. The window size is $W = 15$.

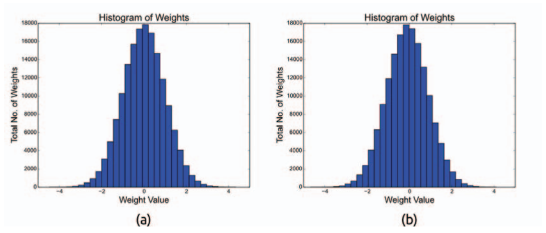

Fig. 3. Histogram of weights for (a) input to hidden layer 1, and (b) hidden layer 1 to hidden layer 2.

C. Post-Processing

The output layer returns a posterior probability estimate for each frame, i.e. every 10ms. To reduce the inherent noise in such estimates, the latter are smoothed using a symmetrical moving average window of W frames centered around the current frame -

$$\hat{y}_j = \frac{1}{W} \sum_{i=j-(W-1)/2}^{j+(W-1)/2} y_i \tag{3}$$

where, y_i is the probability estimate obtained from the final softmax layer, and $\hat{y}_j$ is the smoothed estimate.

This helps eliminate noisy bursts and reduce the false alarm rate. The window size W was varied from 10 to 50 frames (100ms to 500ms) in our experiments. An example highlighting the effect of smoothing is shown in Figure 2. Here, we can observe that the frame-wise probabilities returned at the output are quite noisy and smoothing suppresses the noise by combining estimates from the past and the future.

The overall goal is to determine whether a specific keyword is present in the entire phrase, hence, the output should either be 1, if the keyword is present, and 0 otherwise. To obtain this phrase-level decision, an additional post-processing step is applied over the smoothed estimates. Using a sliding window of size C frames, if the average probability estimate within this window exceeds a certain threshold, then a keyword is said to be present in the phrase. The window size C is usually dependent on the expected duration of a keyword and was varied from 10 to 30 (100ms to 300ms) in our experiments.

D. Fixed-Point Implementation

The aforementioned training procedure is implemented using a floating-point representation. The optimized weights, when stored in floating point require a lot of memory. For instance, storing each weight in 32-bit floating-point format would require 2 MBs for a network with 512 nodes per hidden layer. Often, hardware on mobile devices is constrained in the amount of memory available, such as a few KBs only. Hence, a fixed point implementation is necessary to reduce the memory footprint. A histogram of the weights for each layer is shown in Figure 3. The weights are normally distributed, and so we can use different linear or non-linear quantization schemes. We follow a simple linear quantization scheme owing to its simplicity and generalizability. Throughout the paper, we denote fixed-point using a Q$A.B$ format, where A denotes the number of bits assigned to the integer part and B denotes the number of bits assigned to the fractional part. Unless mentioned otherwise, an additional sign bit is assumed.

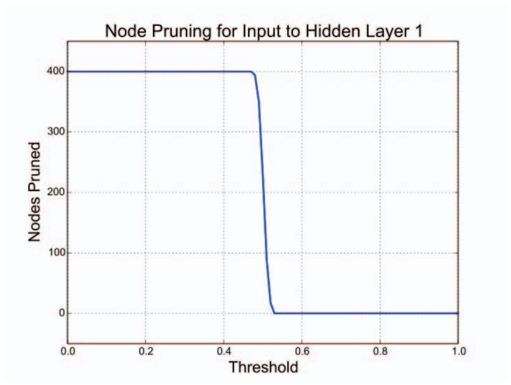

Fig. 4. Node pruning in hidden layer 1 for different threshold values. There are 400 nodes in the hidden layer.

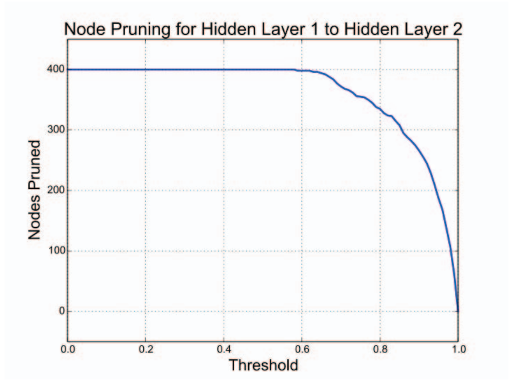

Fig. 5. Node pruning in hidden layer 2 for different threshold values. There are 400 nodes in the hidden layer.

The input nodes and intermediate hidden layers are also stored in a fixed-point format to further reduce the accumulator size during multiplication operations. The former are represented using 16 bits in a Q2.13 format. The latter are represented using 24 or 32 bits, i.e. a Q8.16 or Q16.16 format. The hidden layer nodes are always positive, hence, a sign bit is not required.

E. Node Pruning

Depending on the size of the neural network, there may be a few nodes in the hidden layers that are rarely or never active. If such nodes can be identified, then they can be pruned away, thus reducing both memory and multiplications. Here, we propose one such approach to identify inactive nodes, which is described below.

First, we evaluate the network on the training data and the weights learnt from backpropagation during training. For each node in the hidden layers, we identify the nodes which are zero and maintain a count. This count is averaged over all the training examples to yield a probability estimate for each node, i.e. $p(\text{node is zero})$. Using a threshold value $t \in (0, 1)$, we remove the nodes that have $p > t$. The number of nodes pruned decrease as t approaches 1.

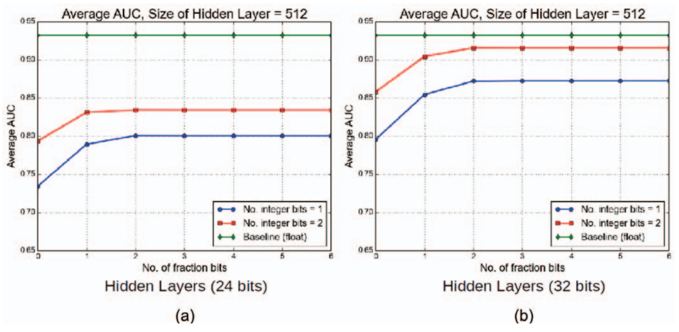

Fig. 6. The effect of different fixed-point representations for the weights on the overall AUC performance. The input is represented using 16 bits. Hidden layer nodes are represented using (a) 24 bits, and (b) 32 bits.

The number of nodes pruned for different threshold values t, and for both hidden layers (400 nodes each) is shown in Figures 4 and 5. Here, we can observe that for the first hidden layer, there is a sharp change in the number of nodes pruned at $t = 0.5$. For $t < 0.5$, all nodes are pruned away, while for $t > 0.5$, all nodes are retained. In this case, all nodes in the first hidden layer are equally informative and node pruning is not helpful. On the other hand, for the second hidden layer, we can see that the transition is smoother, especially for $0.7 < t < 1$. If we set the threshold in this range, we can expect to prune nodes for only a marginal loss in performance.

Besides node pruning, singular value decomposition (SVD) was also considered for reducing the memory footprint as described in [14]. Accordingly, the weight matrix is represented as a product of two low-rank matrices. This technique helps lowers the memory, however, at the cost of increasing the number of multiplications. In our experiments, a significant drop in performance was observed using this technique, possibly due to the relatively smaller network compared to [14]. The degradation was even higher when SVD was combined with a fixed-point representation.

III. Experimental Results

The experiments and results for fixed-point keyword detection using the RM database are described in this section. For the baseline, we consider the performance obtained using a simple floating-point representation for all nodes and weights. The optimal values for window sizes, W and C, during post-processing were found to be 50 and 25 frames respectively. We use two metrics to compare different methods. First, we consider the total area under the curve (AUC) [15], which returns the area under the receiver operating characteristics (ROC) curve of true positive rate (TPR) vs. false alarm rate (FAR). Secondly, we consider the equal error rate (EER) [16], which indicates the rate at which the FAR is the same as the false reject rate (FRR). Here, the FRR is related to the TPR as follows: FRR=1-TPR.

A. *Floating-Point vs. Fixed-Point*

A comparison between floating and fixed-point implementations is shown in Figure 6. For fixed-point implementation, the input is represented using 16 bits (Q2.13). Figure 6 (a) and (b) show the performance with 24 bits (Q8.16) and 32 bits (Q16.16), respectively, for hidden layer nodes. For the

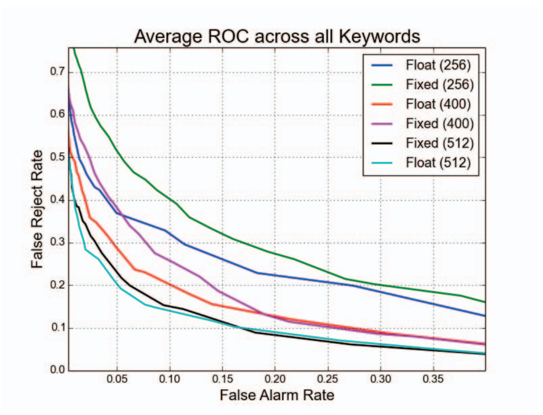

Fig. 7. A comparison of the average ROC across all keywords between networks with different hidden layer sizes and precisions. In case of fixed-point, the weights are stored in a Q2.2 format.

weights stored in a Q$A.B$ format, here, $A \in \{1, 2\}$ and $B \in \{1, 2, 3, 4, 5, 6\}$. First, we can see that the performance is significantly better when using 32 bits for the hidden layer nodes. Secondly, reserving 2 bits for the integer part yields a better AUC compared to just 1 bit. For the fractional part, we observe that increasing the resolution beyond 2 bits does not lead to any significant increase.

Figure 7 shows a comparison between NN architectures with different hidden layer sizes and precisions. In case of fixed-point implementation, the input, hidden layer nodes and weights are stored in Q2.13, Q16.16 and Q2.2 formats, respectively. We can observe that hidden layers with 512 nodes each obtain the best performance. Here, an EER of 12% is obtained using floating-point compared to 13% for fixed-point precision. In comparison, using only 256 nodes per layer may reduce the memory footprint at the cost of a significant drop in performance. Here, an EER of 21% and 24% is obtained using floating and fixed-point precision, respectively.

A summary of the memory requirements is shown in Table I. The input, hidden layer nodes and weights are stored in Q2.13, Q16.16 and Q2.2 formats, respectively. For a network with 512 nodes per hidden layer, a total of 283.6 KB is required for storing the network weights. For 400 nodes per layer, we can see that there is only a marginal loss in performance; an AUC of 0.9098 compared to a floating point representation of 0.9201, while requiring only 195 KBs of memory.

Our results are not directly comparable with the results reported in earlier works [10], [17] since the databases are completely different. However, an AUC performance of 0.90 and an EER of approximately 15% is quite commonly observed for small to medium sized databases. In these aspects, the detection performance obtained here is well within the acceptable range.

B. *Node Pruning*

The performance after node pruning is shown in Figure 8. In this case, only the nodes of the second hidden layer were pruned, as per the procedure described earlier. The performance is analysed for different threshold values $t \in [0.75, 1.0]$. We observe that as the threshold increases, the number of nodes pruned decreases and the performance

TABLE I. COMPARISON OF AUC AND MEMORY REQUIREMENTS BETWEEN FLOATING AND FIXED POINT IMPLEMENTATIONS FOR NETWORKS WITH DIFFERENT HIDDEN LAYER CONFIGURATIONS.

Hidden Layer Width	AUC Floating-Point	AUC Fixed-Point	# of Weights	Memory KB
256	0.8520	0.8038	172300	102.7
350	0.8960	0.8428	268462	160.1
400	0.9201	0.9098	326812	194.8
512	0.9321	0.9153	475660	283.6

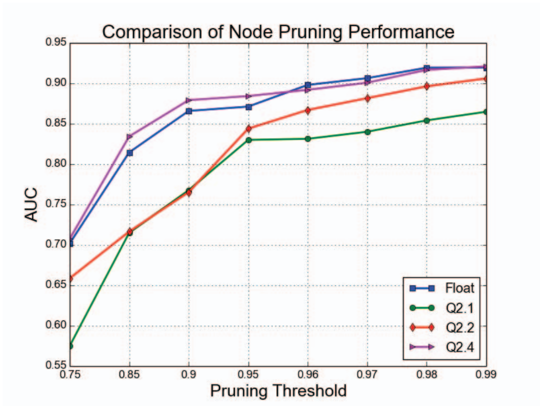

Fig. 8. A performance comparison between floating and fixed-point implementations for different pruning thresholds.

improves. Furthermore, the figure also shows a comparison between different fixed-point and floating-point representations for the weights. For weights represented in a Q2.2 format, and $t \in [0.95, 1.0]$, the loss in performance is not significant. The memory requirements for a network with 400 nodes per hidden layer and a threshold $t \gg 0.75$ are shown in Table II. For $t = 0.99$, the AUC is 0.9060 compared to 0.9098 obtained without pruning. The memory, in this case, reduces by a relative factor of 5%. Similarly, for $t = 0.98$, the AUC is 0.8962 with a relative decrease of 11.4% in memory. Hence, node pruning can be a useful technique to further optimize the neural network and reduce its on-board memory requirements.

C. Hardware Architecture

Most of the recently proposed neural network hardware accelerators target a very high performance [18], [19] and consume over 200mW of power. However, keyword detection, when used as a front-end trigger in mobile devices, is required to be always on and thus has a very tight power budget less than 10mW.

Figure 9 shows the proposed hardware architecture that consists of one Processing Element (PE), three register files, a central memory for weights and one finite state machine (FSM) scheduler. Typical neural network hardware accelerators employ a group of PEs to form a systolic array or ring [20], while our design uses only one PE because of the relatively low throughput requirement of the application - 47 million multiplications per second. A 32-by-5 multiplier in TSMC40nm technology can run at 142MHz at a low voltage of 0.4V. Thus one multiplier is enough for the workload.

As shown in Figure 10, a PE contains one multiply accumulate (MAC) unit, a rectified linear (ReLU) module, and

TABLE II. MEMORY REQUIREMENTS AFTER NODE PRUNING FOR WEIGHTS STORED IN Q2.2 FORMAT.

Threshold	AUC	# of Weights	Memory (KB)
0.95	0.8438	249581	152.3
0.96	0.8669	257428	157.1
0.97	0.8816	269818	164.6
0.98	0.8962	282621	172.5
0.99	0.9060	301206	183.8
No pruning	0.9098	326812	194.8

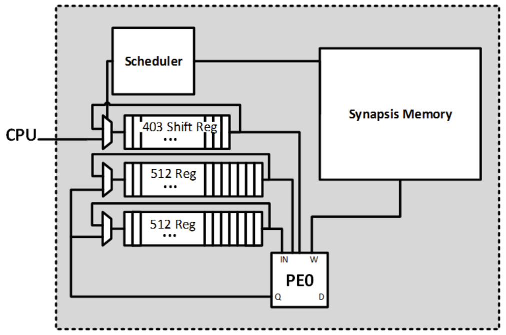

Fig. 9. Top level architecture of the feedforward neural network.

a sigmoid module. Rectified Linear module is implemented by a comparator. The sigmoid function can be implemented using a Look-Up Table (LUT). Three register file based FIFOs are used to hold input and output neurons. Every 10ms, 13 new inputs will be shifted into 403-word shift register file. Two 512-word register files store outputs from 2 hidden layers with 512 neurons each in maximum. Network weights are stored in a 256KB memory block made by the 8T SRAM compiler. Because of the relatively large memory requirement of the application, the chip space and power is dominated by this 256KB memory. 8T SRAM cell is chosen over 6T because it is more reliable under lower supply voltages. The proposed accelerator minimizes power consumption by scaling down the supply voltage to near threshold points (e.g. 0.5V-0.6V). Based on the synthesis result, the area and power estimation of each component at 0.6V is given in Table III. To meet the workload requirement, 50MHz clock frequency is assumed. In Table IV and Table V, the current design, Q2.2 with pruning, is compared with two other implementations, Q2.13 and floating-point without pruning. The results show that 5-bit fixed-point implementations for weight and node pruning are highly effective in saving area and power. To use the memory efficiently, six 5-bit weights are attached together to occupy one wordline in memory.

IV. CONCLUSIONS

A fully connected, feedforward neural architecture for spoken keyword detection was proposed in this work. A post-processing method to obtain phrase-level metrics using a sliding window approach was also described. To reduce the memory footprint for network weights, the latter were stored using a fixed-point representation. Experiments were conducted on 10 keywords selected from the RM corpus, and results show that there is only a marginal loss in performance when the weights are stored in a Q2.2 format, i.e. only 5 bits. The total memory required in this case is approximately 200 KBs,

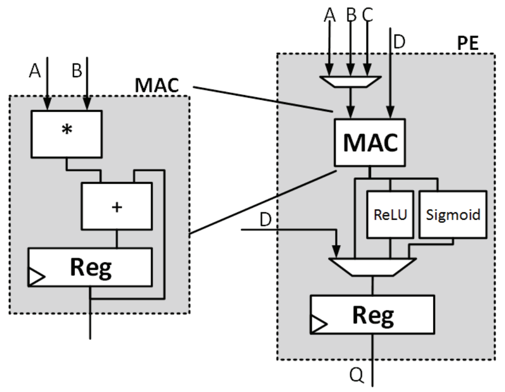

Fig. 10. Internal architecture of the processing element (PE) and multiply-accumulate unit (MAC).

TABLE III. AREA AND POWER ESTIMATION.

	μm^2	mW
Scheduler	2855	0.176
Register	108870	0.58
PE	1548	0.11
SRAM	774054	2.64

making it highly suitable for resource constrained hardware devices. A node pruning technique was also presented to identify and remove the least active nodes in a neural network, thus, decreasing the memory requirements even further. For an acceptable loss in performance, an 11.4% reduction in memory is obtained after combining this technique with a fixed-point representation. Finally, a hardware architecture using a single multiplier and a memory bank was presented. These results demonstrate the applicability of the proposed approach for implementations with limited hardware resources.

TABLE IV. AREA COMPARISON (UNIT: μm^2).

	Q2.2 w/ pruning	Q2.13 w/o pruning	Floating-point w/o pruning
Register	108870	139353	139353
PE	1548	3327	6211
SRAM	774054	2802942	5605884
TOTAL	884472	2945622	5751448

TABLE V. POWER COMPARISON (UNIT: mW).

	Q2.2 w/ pruning	Q2.13 w/o pruning	Floating-point w/o pruning
Register	0.58	0.74	0.74
PE	0.11	0.21	2.14
SRAM	2.64	9.55	19.1
TOTAL	3.33	10.51	21.99

REFERENCES

[1] D. R. Miller, M. Kleber, C.-L. Kao, O. Kimball, T. Colthurst, S. A. Lowe, R. M. Schwartz, and H. Gish, "Rapid and accurate spoken term detection," in *Eighth Annual Conference of the International Speech Communication Association*, 2007.

[2] S. Parlak and M. Saraclar, "Spoken term detection for turkish broadcast news," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2008, pp. 5244–5247.

[3] J. Mamou, B. Ramabhadran, and O. Siohan, "Vocabulary independent spoken term detection," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 615–622.

[4] J. R. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous hidden markov modeling for speaker-independent word spotting," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1989, pp. 627–630.

[5] R. C. Rose and D. B. Paul, "A hidden markov model based keyword recognition system," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1990, pp. 129–132.

[6] J. Wilpon, L. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden markov modeling techniques," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1991, pp. 309–312.

[7] M.-C. Silaghi and H. Bourlard, "Iterative posterior-based keyword spotting without filler models," in *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding Workshop*. Citeseer, 1999, pp. 213–216.

[8] M.-C. Silaghi, "Spotting subsequences matching an hmm using the average observation probability criteria with application to keyword spotting," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 20. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, p. 1118.

[9] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Large vocabulary continuous speech recognition with context-dependent dbn-hmms," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2011, pp. 4688–4691.

[10] 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*. IEEE, 2014, pp. 4087–4091.

[11] P. Price, W. M. Fisher, J. Bernstein, and D. S. Pallett, "The darpa 1000-word resource management database for continuous speech recognition," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1988, pp. 651–654.

[12] D. Povey, A. Ghoshal, and et al., "The Kaldi speech recognition toolkit," in *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE, 2011.

[13] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean *et al.*, "On rectified linear units for speech processing," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 3517–3521.

[14] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition." in *Proceedings of INTERSPEECH*, 2013, pp. 2365–2369.

[15] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.

[16] M. G. Rahim, C.-H. Lee, and B.-H. Juang, "Discriminative utterance verification for connected digits recognition," *Speech and Audio Processing, IEEE Transactions on*, vol. 5, no. 3, pp. 266–277, 1997.

[17] J. Keshet, D. Grangier, and S. Bengio, "Discriminative keyword spotting," *Speech Communication*, vol. 51, no. 4, pp. 317–329, 2009.

[18] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, "A 1.93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," in *Proceedings of IEEE International Conference on Solid-State Circuits Conference*. IEEE, 2015, pp. 1–3.

[19] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2014, pp. 269–284.

[20] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 6.67 mW sparse coding ASIC enabling on-chip learning and inference," in *Symposium on VLSI Circuits Digest of Technical Papers*. IEEE, 2014, pp. 1–2.