Efficient Memory Compression in Deep Neural Networks Using Coarse-Grain Sparsification for Speech Applications

Deepak Kadetotad[1], Sairam Arunachalam[2], Chaitali Chakrabarti[1], Jae-sun Seo[1]
[1]School of Electrical, Computer and Energy Engineering
[2]School of Computing, Informatics, Decision Systems Engineering
Arizona State University, Tempe, AZ 85281
{deepak.kadetotad, sarunac4, chaitali, jaesun.seo}@asu.edu

ABSTRACT
Recent breakthroughs in deep neural networks have led to the proliferation of its use in image and speech applications. Conventional deep neural networks (DNNs) are fully-connected multi-layer networks with hundreds or thousands of neurons in each layer. Such a network requires a very large weight memory to store the connectivity between neurons. In this paper, we propose a hardware-centric methodology to design low power neural networks with significantly smaller memory footprint and computation resource requirements. We achieve this by judiciously dropping connections in large blocks of weights. The corresponding technique, termed coarse-grain sparsification (CGS), introduces hardware-aware sparsity during the DNN training, which leads to efficient weight memory compression and significant computation reduction during classification without losing accuracy. We apply the proposed approach to DNN design for keyword detection and speech recognition. When the two DNNs are trained with 75% of the weights dropped and classified with 5-6 bit weight precision, the weight memory requirement is reduced by 95% compared to their fully-connected counterparts with double precision, while maintaining similar performance in keyword detection accuracy, word error rate, and sentence error rate. To validate this technique in real hardware, a time-multiplexed architecture using a shared multiply and accumulate (MAC) engine was implemented in 65nm and 40nm low power (LP) CMOS. In 40nm at 0.6V, the keyword detection network consumes 7μW and the speech recognition network consumes 103μW, making this technique highly suitable for mobile and wearable devices.

Categories and Subject Descriptions
1.1 [**System Design**]: HW/SW Co-design, Co-optimization

Keywords
Speech recognition; keyword detection; memory compression; low power design; deep neural networks.

ICCAD '16, November 07-10, 2016, Austin, TX, USA
 ISBN 978-1-4503-4466-1/16/11$15.00
DOI: http://dx.doi.org/10.1145/2966986.2967028

1. INTRODUCTION
Automatic speech recognition (ASR), which converts words spoken by a person into readable text, has been a popular area of research and development for the past decades [1]. The goal of ASR is to allow a machine to understand continuous speech in real-time with high accuracy, independent of speaker characteristics, noise, or temporal variations. Nowadays, ASR technology can be easily found in a number of commercial products, including "Google Now" (Google), "Siri" (Apple), and "Echo" (Amazon). In these systems, speech recognition is activated by specific keywords such as "Ok Google", "Hey Siri", and "Alexa". More systems now perform such wake-up keyword detection tasks in an 'always-on' mode, always listening to surrounding acoustics without a dedicated start control. Minimizing the power consumption of such always-on operations that can detect multiple keywords is crucial for mobile and wearable devices. The speech recognition task that follows keyword detection is much more compute-/memory-intensive such that it is typically offloaded to the cloud. A number of commercial systems do not allow speech recognition if the device is not connected to the internet. To expand the usage scenarios, it is important that the speech recognition engine also has low hardware complexity and consumes low power.

One of the widely used approaches for speech recognition employs a Hidden Markov Model (HMM) for modeling the sequence of words/phonemes and uses a Gaussian Mixture Model (GMM) for acoustic modeling [2]. The most likely sequence can be determined from the HMMs by employing the Viterbi algorithm. For keyword detection, a separate GMM-HMM could be trained for each keyword [3, 4], while the out-of-vocabulary (OOV) words are modeled using a garbage or filler model. In recent years, employing DNNs in conjunction with HMM models for keyword detection and speech recognition have shown substantial improvements in classification accuracy [5, 6]. Other prior works in ASR feature recurrent neural networks [7] or convolutional neural networks [8].

Although enhanced accuracy in word detection and recognition algorithms have been demonstrated, implementing such DNNs in hardware requires a large memory footprint and high computation power, due to the large number of parameters (100s of thousands for keyword detection, millions for speech recognition). Considering that (1) mobile and wearable devices have constraints on embedded memory and computational resources and (2) majority of the memory of DNNs comes from the weights between neurons, reducing the number of weight parameters without affecting the accuracy is a necessity for efficient hardware implementation. A number of prior works exist in weight memory

reduction for DNNs, including weight and node pruning [9], fixed-point representation [10], selective weight approximation [11], and quantized weight sharing with Huffman coding [12]. Most of these approaches reduce or drop weights on an element-by-element basis, and thus the resulting memory generally cannot be efficiently compressed onto hardware. This is because the address information of which weights are dropped need to be stored as well, which can offset the savings achieved from the reduced number of weights.

In this paper, we propose a coarse-grain sparsification technique on the weights of DNNs, which enables efficient compression of weight memory in an ASIC implementation. The key idea is that we drop the weight connections in large blocks (i.e., 64×64), and we enforce the coarse-grain weight drop in a static manner throughout the entire process of training. As a result, the final set of weights can be efficiently mapped onto SRAM arrays with minimal address information for classification. Upon training initialization, large blocks are randomly dropped with a certain probability (i.e., 75%), and only the remaining weights are subject to training. This is in contrast to Dropout [11] or DropConnect [13] techniques that are commonly used in deep learning algorithms to prevent overfitting. These techniques drop nodes or weights in a dynamic manner using a different random selection in every training iteration. Since the weights that are dropped are different each time, the weight size cannot actually be reduced, and the fully-connected matrix weights are required for the classification phase. On the other hand, partially connected neural networks that have sparse connection matrices show better storage per connection than its fully-connected counterpart and the storage per connection increases as the weight connections become sparser and more random [14].

We introduce an algorithm-hardware co-design scheme for coarse-grain sparsification in order to significantly reduce the memory footprint as well the number of computations. The proposed coarse-grain sparsification technique was evaluated by training DNNs for keyword detection and speech recognition using the DARPA Resource Management (RM) database [15] and also by hardware implementations of the corresponding DNNs in 65nm LP and 40nm LP CMOS technology. Overall, this paper makes the following contributions.

- We propose static coarse-grain sparsification that can substantially reduce the memory footprint as well as necessary computations when the DNNs are implemented onto hardware, with minimal degradation in accuracy.
- Using the proposed coarse sparsification scheme throughout training and classification, weight memory is compressed by 4× for keyword and speech recognition DNNs. Employing low-precision of 5-6 bits for weights leads to an additional reduction of ~5×. Overall, the weight memory of the sparse low-precision DNNs is compressed by ~20× from that in fully-connected floating-point DNNs.
- The proposed scheme is implemented and evaluated in 65nm and 40nm CMOS. The DNN implementation of the keyword detection network resulted in 12.61μW and 7.01μW power consumption in 65nm LP and 40nm LP CMOS, respectively. The speech recognition network is larger and had higher power consumption of 255.87μW and 102.29μW in 65nm LP and 40nm LP CMOS, respectively.

The remainder of the paper is organized as follows. In Section 2, we describe the DNN processing operations for the two speech applications. Section 3 describes the proposed coarse-grain sparsification technique and how the training and classification is performed. In Section 4, the architecture of the custom deep neural network hardware with memory compression is described. Section 5 presents the experimental results in 65nm LP and 40nm LP CMOS, and the paper is concluded in Section 6.

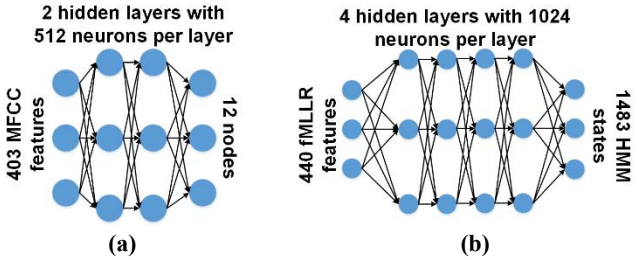

Figure 1. DNNs for (a) keyword detection and (b) speech recognition.

2. DNN PROCESSING IN SPEECH APPLICATIONS

2.1 DNNs for Keyword Detection and Speech Recognition

We designed two separate DNNs for keyword detection and speech recognition tasks, which are illustrated in Fig. 1. The keyword detection DNN is similar to that in [10] and consists of two hidden layers with 512 neurons per layer (Fig. 1(a)). There are 403 input nodes corresponding to 31 frames (15 previous, 15 future and 1 current) with 13 Mel frequency coefficients (MFCC) features per frame. The output layer consists of 12 nodes, where 10 nodes are for the 10 selected keywords, 1 node is for Out of Vocabulary (OOV) words and 1 node is for silence. The output of the neural network represents the probability estimate of each of the nodes in the output layer. These probabilities are calculated using the softmax activation function.

The DNN designed for speech recognition system, as shown in Fig. 1(b), consists of 4 hidden layers with 1,024 neurons per layer. There are 440 input nodes corresponding to 11 frames (5 previous, 5 future and 1 current) with 40 fMLLR (feature-space Maximum Likelihood Linear Regression) features per frame. The output layer consists of 1,483 probability estimates that are sent to the Hidden Markov Model (HMM) unit to determine the best sequence of phonemes. The transcription of the words and sentences for the particular set of phonemes is done using the Kaldi toolkit [16].

2.2 Training DNNs

Training the neural network is performed by minimizing the cross-entropy error, as described in Eq. (1).

$$E = -\sum_{i=1}^{N} t_i * \ln(y_i) \quad (1)$$

Here N is the size of the output layer, y_i is the i^{th} output node and t_i is the i^{th} target value or label. The mini-batch stochastic gradient method [17] is used to train the network. The weight W_{ij} is updated in the $(k+1)^{\text{th}}$ iteration, using Eq. (2).

$$(W_{ij})_{k+1} = (W_{ij})_k + C_{ij} * lr * \{(\Delta W_{ij})_k + m * (\Delta W_{ij})_{k-1}\} \quad (2)$$

C_{ij} is the binary connection coefficient between two subsequent neural network layers, which is introduced for the proposed CGS, and only the weights in the network corresponding to C_{ij}=1 are updated. m is the momentum and lr is the learning rate. The change

in weight for each iteration is the differential of the cost function with respect to the weight value, as shown in Eq. (3).

$$\Delta W = \frac{\delta E}{\delta W} \tag{3}$$

2.3 Feedforward Classification of DNNs

Once training is completed, feed-forward computations are performed in the two DNNs as follows. Let the input layer be denoted as x_i $(i = 1, 2, \dots, N)$, where N is the number of input features. The computation in the first hidden layer h^1 is given in Eq. (4).

$$z_j^1 = \sum_{i=1}^{N} C_{ij} W_{ij}^1 x_i + b_j^1, \tag{4}$$

where z is the output neuron, W is the weight matrix and b is the bias for the hidden layer. For both networks, the non-linear activation function that is applied at the end of each hidden layer is ReLU (Rectified Linear Unit) [18]. Other hidden layer neurons are computed similarly.

3. DNN WEIGHT COMPRESSION USING SPARSITY AND PRECISION REDUCTION

In this section, two methods that contribute to the overall weight memory reduction are discussed. First, coarse-grain sparsification (CGS) of fully-connected weight matrix is described for both training and classification of DNNs. In addition, to further reduce the memory footprint during classification, the precision of weights is minimized while maintaining the word detection and recognition accuracy.

3.1 Coarse-Grain Sparsification (CGS)

A considerable volume of literature exists on the so-called partially connected neural networks [14], where weight connections between layers are dropped. Previous approaches drop weights based on some specific criteria (e.g., weight value is lower than a threshold). This is performed primarily on an element-by-element basis, which prevents efficient mapping onto hardware memory arrays. To overcome this inefficiency, we propose to randomly drop weights in DNNs in a coarse-grained block-by-block basis.

The fully-connected weight connections in DNNs are first divided into a number of large square blocks (e.g., 64×64), and a number of blocks are randomly dropped with a certain probability (e.g., 50%, 75%) prior to training. Throughout training and classification, the dropped blocks remain at zero and do not contribute to the physical memory footprint. In this paper, we focus on power/area reduction for classification hardware, but we postulate that CGS will also lead to low-power acceleration for training.

A sample weight matrix of size 512×512 is shown in Fig. 2(a), where the weight matrix is broken down in blocks of size 64×64. In this example, 8 blocks exist along any given row or column. The colored squares represent regions where eligible connections are present, and the white squares represent regions with absence of connections. Note that dropping all blocks along a column or row is not allowed, to prevent any neuron between two layers from being completely disconnected. Fig. 2(b) illustrates the compressed (along the row) coarse-grained sparse matrix that only stores regions with active connections.

If the DNN training is done using fully-connected weights, it is much more difficult to sparsify or compress the memory during

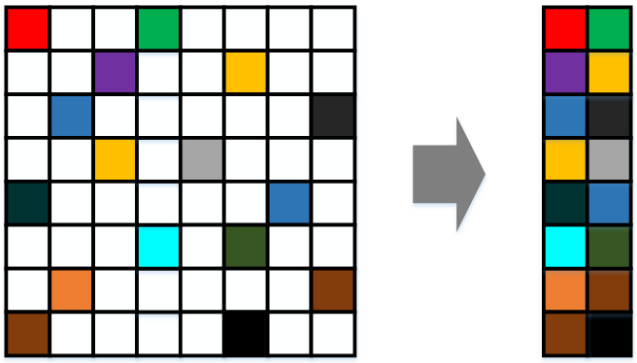

Figure 2. Example of weight matrix compression. (a) Sample weight matrix of size 512×512 with block size of 64×64. Only 25% of the blocks are selected (colored); white blocks represent zero connections. (b) Compressed matrix is of size 512×128.

classification. Therefore, we investigate constraining the weight matrix during training, such that only a small portion (25% in this specific application) of blocks is subject to update. This way, once the training is completed, it becomes straightforward to store only a small number of blocks (25%) with non-zero weights in a compressed form.

Finding the optimal block size that balances hardware overhead and detection/recognition accuracy is crucial. For the same weight drop rate, if the block size is too small, the number of non-zero blocks becomes large, which requires substantial amount of address information and additional combinational logic. On the other hand, if the block size is too large, the training performance suffers due to the inability to represent fine-grain weight connectivity. Orthogonal to the block size, the drop rate is an important design parameter that directly dictates the maximum allowable compression of memory. We investigated the effect of drop rate and block size independently for keyword detection and speech recognition DNNs. Through simulations, we found that 75% weight drop rate still yielded sufficient accuracy for both keyword detection and speech recognition tasks; the results are presented in Section 5.

3.2 Low-Precision Classification

The aforementioned training procedure in Section 2.2 is performed on a GPU using weights with floating-point representation. Although more recent works investigated low-precision training [19, 20], combining such low precision in the training phase remains as future work. Once our DNN training is completed off-line, for classification, we represent the weights as well as neurons using a low-precision fixed-point format to further reduce the memory footprint as well as computation cost. For the trained DNN, we first quantize the weights with floating-point neurons down to the precision where the DNN accuracy is acceptable. Then, we operate the DNN with reduced precision on the weights and choose reduced precision for the neurons, while still achieving acceptable accuracy. This methodology most likely leads to a higher precision for the neurons compared to that of the weights, but this is an acceptable trade-off since weights have a larger impact on the overall memory footprint as well as power consumption. Throughout the paper, we denote fixed-point precision 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.

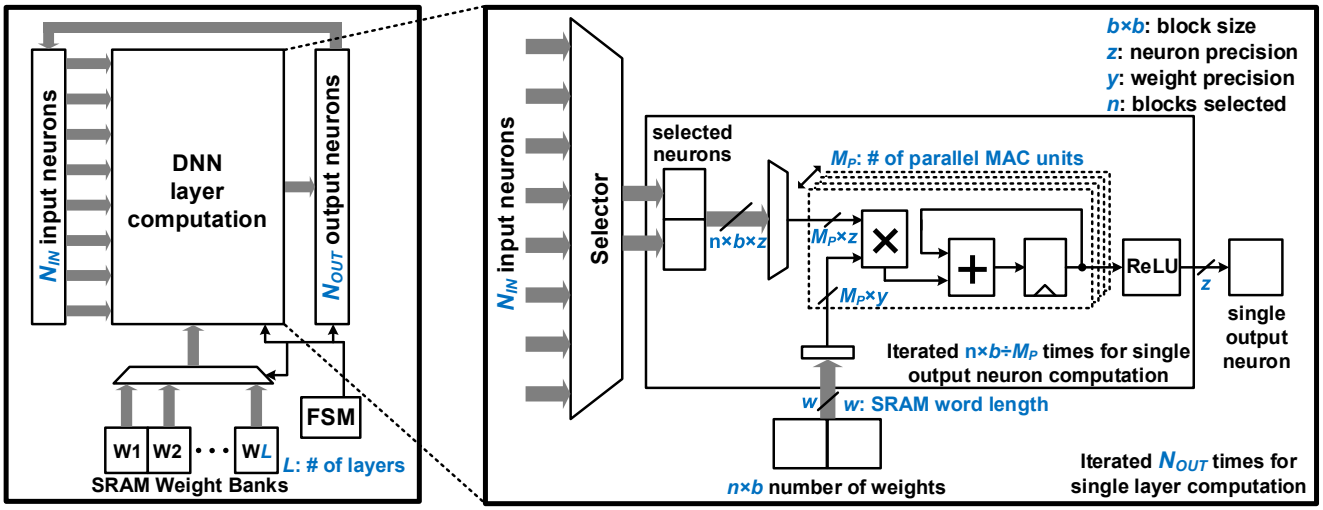

Figure 3. Block diagram of the DNN-based classification system.

4. HARDWARE ARCHITECTURE AND CMOS IMPLEMENTATION

The architecture of the CGS based keyword detection and speech recognition DNNs is shown in Fig. 3. The main computation block (shown on the right) operates on one layer at a time. It consists of M_p MAC units that operate in parallel on data that is forwarded by the selector. The weights of the network are stored in L SRAM banks while the neuron inputs and outputs are stored in registers. The finite state machine (FSM) coordinates the data movement and re-use of the computation blocks.

4.1 Keyword Detection DNN Implementation

The keyword detection network described in Section 2.1 has 403 input neurons and 512 neurons in the two hidden layers, which all have 16-b precision. The network was trained such that the CGS block size is the same for all the layers. The block size used for keyword detection is 64×64, hence the 512 input neurons of the compute block are divided into 8 blocks with 64 inputs per block. Due to the sparsity of the weight matrix, there exists only 128 non-zero weight values with 5-bit precision. The Selector module chooses two blocks out of the eight input blocks and directs them to the MAC unit. The Selector is implemented using two 8:1 block multiplexers, each of which is controlled by a 3-bit select signal. The select signal contains the spatial location of the blocks in the uncompressed weight matrix and is used to route the data to the MAC. For the keyword detection DNN, the parameters in Fig. 3 are N_{IN} = 512, b = 64, z = 16, y = 5 and n = 2.

The MAC unit operates on inputs from the SRAM that contains stored weights, and from the Selector module that forwards the neuron values corresponding to the chosen weights. After investigating architectures with different number of MAC units, we chose an implementation with 16 parallel MAC units (M_p = 16) as it consumes the least amount of power, as shown in Section 5.2. The MAC unit for the keyword system computes eight sum of products in parallel. After the sum of products computation is completed, ReLU activation is performed, and then conveyed to the output. The output of the MAC unit is a single output neuron, which is used as an input to the next layer's computations. The 4 parallel MAC architecture takes 4,096 cycles to compute on one layer of the keyword detection network.

4.2 Speech Recognition DNN Implementation

We employed the same architecture shown in Fig. 3 for the speech recognition system. The speech recognition DNN described in Section 2.1 has 440 input neurons and 1,024 neurons in the next four hidden layers, all of which have 16-b precision. The network was trained to allow re-use of the computation blocks for all the layers. The block size used for speech recognition is 64×64, hence the 1,024 input neurons of the compute block is divided into 16 blocks of 64 inputs. The Selector module chooses 4 out of the 16 input blocks and directs them to the MAC units. For the speech recognition DNN, the parameters in Fig. 3 are N_{IN} = 1,024, b = 64, z = 16, y = 6 and n = 4.

The MAC unit for the speech recognition system receives 256 inputs from the SRAM and Selector blocks. After investigating architectures with different number of MAC units, we chose an architecture with 20 MAC units (M_p = 20) since it has the lowest power consumption. The number of cycles required to finish the computation for a layer with 20 MAC units is 13,108 cycles.

4.3 Finite State Machine

The finite state machine (FSM) has 5 and 7 distinct states for the keyword and speech system, respectively. In state 0, all the functional blocks are inactive as the weights are loaded onto the SRAM and the system is waiting to compute. Other states control different steps of the computation of the deep neural network layers. For instance, it counts the number of total computations performed by the functional blocks and accordingly feeds it the correct data.

5. EXPERIMENTAL RESULTS

In this section, we present the software and hardware evaluation results for keyword detection and speech recognition. We compare the performance of the proposed sparsely connected network architecture with fully-connected fixed-point and double-precision floating-point architectures. All software evaluations were conducted on the RM database [15]. The hardware logic and memory are implemented using TSMC 65nm LP and 40nm LP CMOS technology. The supply voltage of both the memory and logic of DNNs is scaled down to 0.7V (65nm) and 0.6V (40nm) to achieve enhanced power efficiency. The software and hardware evaluation setups for the two networks are described in Section 5.1.

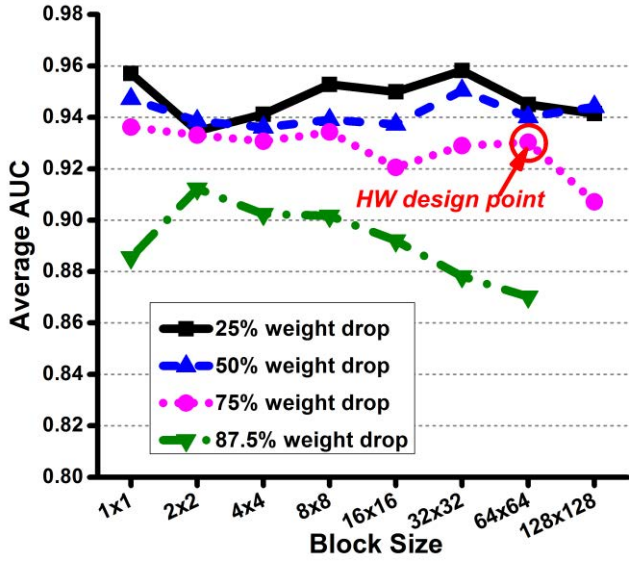

Figure 4. Effect of block size and percentage drop on average AUC of keyword detection DNN.

5.1 Experimental Setup

5.1.1 Software Evaluation Setup

For the keyword detection system, there are cases when the system predicts a keyword that is present (true positive) or the system predicts a keyword that is absent (false alarm). A good metric to determine the performance of the system is given by area under the receiver operator characteristics (ROC), referred to as the area under the curve (AUC) [21]. We evaluate different neural network architectures with respect to their AUCs. The training of network is done using a learning rate of 0.001, momentum of 0.8 and batch size of 500. The network is trained for 6 epochs.

For the speech recognition system, the 40 fMMLR features are extracted from the speech waveform by applying transforms to the MFCC features as described in [22]. The fMLLR features of the 5 previous frames and 5 future frames are added to obtain a 440-dimension feature vector for each frame. The baseline GMM-HMM model is trained using the script provided for the RM database. The DNN training is performed using the PDNN toolkit [23] on the GTX 650 GPU. For training this network, we use the "newbob" technique using an initial learning rate of 0.08, momentum of 0.5 and batch size of 256. If the validation error between two epochs differs by less than 0.2%, the learning rate is scaled by a factor of 0.5. This scaling is performed at every subsequent epoch until training is complete. The training is stopped when the validation error between two consecutive epochs differs by less than 0.2%.

Table I. Comparison of memory requirements and AUC for keyword detection networks.

Architecture	Memory size	Average AUC
Floating-point (fully-connected)	1.81 MB	0.945
Fixed-point (fully-connected)	290.32 KB	0.940
Proposed CGS (block size: 64x64, 75% weight drop)	84.38 KB	0.912

5.1.2 Hardware Evaluation Setup

The power consumption results of the compute logic are obtained from PrimeTime simulation of the synthesized functional blocks with data switching activity information, while the SRAM power computations are calculated with read current values in the commercial memory compiler datasheet for 65nm LP and 40nm LP CMOS. The power consumption values are first obtained at the nominal voltage and temperature of 1.2V/1.1V (65nm/40nm) and 25°C, and then the scaling ratios are applied to calculate the power and performance at supply voltages down to 0.7V/0.6V (65nm/40nm), which are assumed as the minimum operating supply voltage (*Vmin*) of the 6-T SRAM arrays.

To scale the supply voltage, we use different approaches for logic and memory. For logic, we conduct SPICE simulations to get the voltage scaling ratio for both technology nodes. We obtain this ratio by extracting the reduction in output frequency and power consumption from simulation results of a 7-stage ring oscillator (RO) that consists of FO4 inverters at various supply voltage values. For 65nm LP and 40nm LP, the RO is simulated with the foundry transistor models. With the information on reference RO frequency and power at corresponding technology nodes, we scale the voltage down such that timing slack of the critical paths in keyword detection and speech recognition design decreases correspondingly. For memory active power, we use the same scaling factors from the RO simulation as done in logic. For memory leakage power, we simulate the leakage power of a SRAM bitcell from nominal voltage down to 0.7V/0.6V, and apply the leakage scaling ratio to project the leakage of SRAM arrays for different supply voltages.

5.2 Keyword Detection DNN

5.2.1 Software Results

Fig. 4 shows the effect of block size and percentage of the dropped connections on the AUC performance of the floating-point keyword detection DNN. The percentage of the dropped connections applies for the weights of the two hidden layers. We do not drop connections in the last layer since it consists of only 12 nodes and is relatively sensitive to detection accuracy. In addition, the weights of the last layer contribute to ~1% of the total weights in the system, thus any reduction in this layer does not result in substantial reduction in the overall system memory requirement. From Fig. 4, we observe that, for the same block size, increasing the percentage of dropped connections adversely affects the AUC performance, as expected. When the drop in connection is less than 50%, there is little change in the AUC performance even when the block size is large. However, for larger drop rates, the AUC performance becomes sensitive to the block size. For instance, the performance of a system with 75% of its weights dropped has an AUC performance loss of up to 0.029 when the block size is 128×128. Since the AUC performance loss is only 0.015 when the block size is 64×64, this configuration is selected as the hardware design point for our sparse network.

For the fully-connected fixed-point architecture, the weights and biases are represented using Q2.2 format. The inputs and hidden layers are represented by 15 bits in Q2.13 and Q10.5, respectively, which correspond to 16 bits when the sign bit is included. We use the same set of precisions for the proposed sparse architecture with block size of 64×64 and 75% dropout.

Table I compares the memory requirements and the performance of the system for different keyword detection networks. We see that there is a small drop in the performance of proposed CGS scheme compared to the fully-connected fixed-point and floating-point

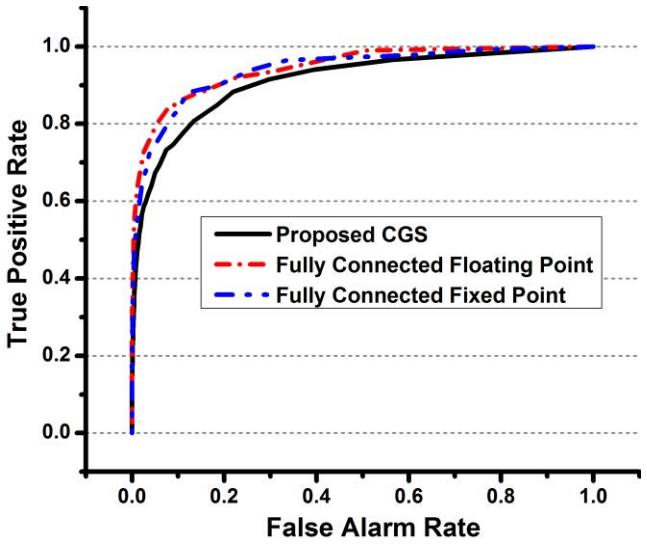

Figure 5. ROC Curve of different implementations for keyword detection DNN.

architectures. The ROC curves for the three different architectures shown in Fig. 5 are largely similar. From these results, we conclude that our coarsely sparsified DNN performs at a level similar to the floating-point architecture, while requiring only 4.13% of the memory required by the fully-connected floating-point DNN.

5.2.2 Hardware Results

The keyword detection system contains three SRAM memory banks: two 40KB banks for the hidden layer, and one 3.75KB bank for the final layer. The system must operate at a minimum of 1.56 MHz to meet the latency requirement of 10ms per input (for a single MAC architecture).

Fig. 6 shows the design space exploration of the keyword detection network, which depicts the change in power consumption and logic area as a function of the number of MAC units. The increase in the number of MAC units allows for reducing the operating frequency, which in turn increases the timing slack of critical paths. The increase in slack is compensated by scaling down the supply voltage resulting in reduction in power. Also the 4× reduction in computation facilitated by the proposed CGS enables keyword classification in real-time. For both 65nm LP and 40nm LP CMOS technology nodes, it is observed that the lowest power consumption is obtained when the number of parallel MAC units is 16.

During the neural network computations, as only one weight bank performs read operation, the other two are kept in standby mode consuming leakage power. Overall, the power consumption is

Table II. Power and area for keyword detection networks.

	Area (μm²)		Power (μW)	
	65nm	**40nm**	**65nm**	**40nm**
Selector	38,045	13,592	1.55	0.24
Compute Unit	12,524	5,233	1.16	0.19
FSM	32,682	14,372	2.39	0.64
SRAM (leakage)	882,414	472,418	7.49 (4.40)	5.93 (5.19)
Total	**965,665**	**505,615**	**12.61**	**7.01**

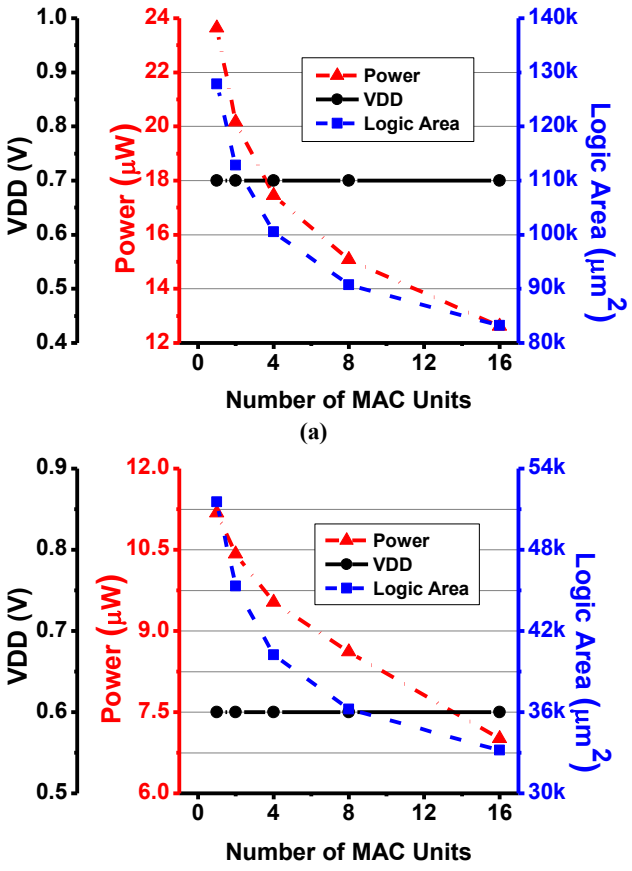

Figure 6. For the same real-time performance, supply voltage, system power, and area results for keyword detection DNN are shown for different number of parallel MAC units in (a) 65nm LP and (b) 40nm LP CMOS.

12.61μW when operating at 0.7V for 65nm LP and 7.01μW at 0.6V for 40nm LP CMOS. Table II lists the area and power contributions of the 16 MAC architecture. We also observe that the SRAM power consumption dominates, consuming >70% of the total system power, and that the SRAM leakage power consumes a higher percentage of the total power in 40nm, compared to that in 65nm.

5.3 Speech Recognition DNN

5.3.1 Software Results

The performance of the speech recognition system is measured by two key metrics, the word error rate (WER) and the sentence error rate (SER). The word error rate is defined as WER=100*(S+D+I)/N, where *S* denotes the number of substitutions, *D* is the number of deletions, *I* is the number of insertions and *N* is the number of words in the reference. The sentence error rate is the percentage of sentences that has at least one error. While WER and SER are for the whole system (DNN+HMM), here we only analyze the effect of the DNN parameters on the error rates. Fig. 7 shows the effect of percentage of the dropped connections on the WER of the floating point system as a function of the block size. For up to 75% percentage of dropped weights at all layers of the network, the WER performance of the system is comparable to fully-connected floating-point DNNs. Increasing the drop rate to 87.5% for block sizes larger than 64×64 worsens the error rate significantly, and is not preferable. Based on

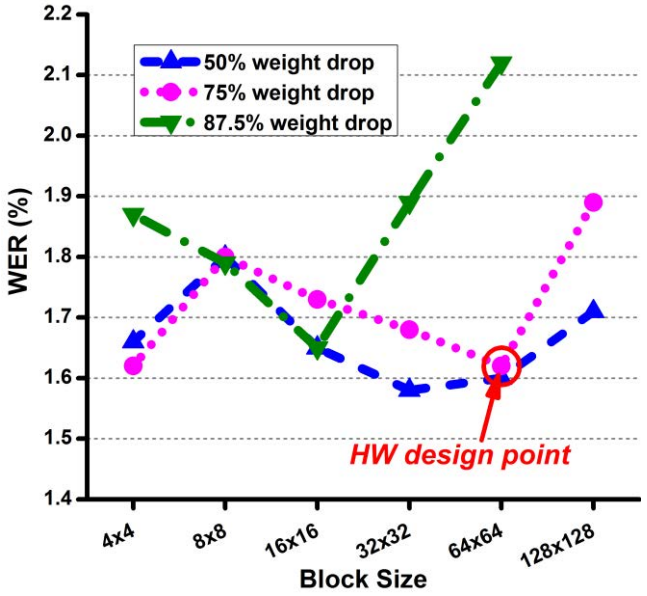

Figure 7: Effect of block size and percentage on dropped weights on average WER for speech recognition.

this analysis, we choose a drop rate of 75% across all layers with block size of 64×64.

In order to derive a fixed-point speech recognition DNN, we follow the same procedure as the keyword detection DNN. The histogram of the weights and biases is used to determine the precision of the integer bits and fractional precision so that the error rates are below 2%. In this case, the fractional precision was 5 bits and the weights are represented by Q0.5. The precision of the hidden layers was found to be Q10.5, and the input neurons are represented by Q4.11. Therefore, both keyword detection and speech recognition DNNs require 16 bits for the hidden layer nodes.

Table III. Comparison of memory, SER, and WER for speech recognition networks.

Architecture	WER	SER	Memory
Floating-point (fully=connected)	1.64%	10.89%	19.53MB
Fixed-point (fully-connected)	1.77%	11.10%	3.66MB
Proposed CGS (block size : 64x64, 75% dropout)	1.67%	10.96%	1.02MB

Table IV. Power and area for speech recognition networks.

	Area (μm²)		Power	
	65nm	40nm	65nm (μW)	40nm (μW)
Selector	119,481	61,221	22.50	4.74
Compute Unit	20,953	10,883	8.61	2.18
FSM	182,264	58,624	29.14	5.89
SRAM (leakage)	8,333,200	4,505,040	195.62 (48.90)	94.22 (59.70)
Total	**8,655,898**	**4,635,768**	**255.87**	**102.29**

Table III compares the performance of our system with the fully-connected floating-point and fixed-point architectures. The sparse fixed-point DNN using the proposed CGS technique with up to 75% of its connections dropped, has an WER close to that of the floating point fully-connected DNN. The proposed architecture requires memory size of only 1.02MB compared to 19.53MB of a fully-connected floating-point architecture. Thus, the sparsified fixed-point network is able to reduce ~95% of the memory requirement with minimal degradation in WER/SER performance.

5.3.2 Hardware Results

The speech recognition system with 20 MAC units operates at 6.7MHz to meet the 10ms latency requirement in order to function at real time. There are five SRAM banks that store the DNN weights, where the size of each bank is 224KB. Similar to the keyword detection network, only one SRAM bank is active at a time while four other banks will be in standby mode.

Fig. 8 shows the design space exploration of the speech recognition network. For both technology nodes, the 20 MAC unit architecture is found to be the optimal design point in terms of power consumption. The area and power values for the 20 MAC architecture are shown in Table IV.

Overall, the power consumption of the speech recognition system is only 255.87μW (at 0.7V) in 65nm LP and 102.29μW (at 0.6V) in 40nm LP. Similar to the keyword detection DNN, the power of

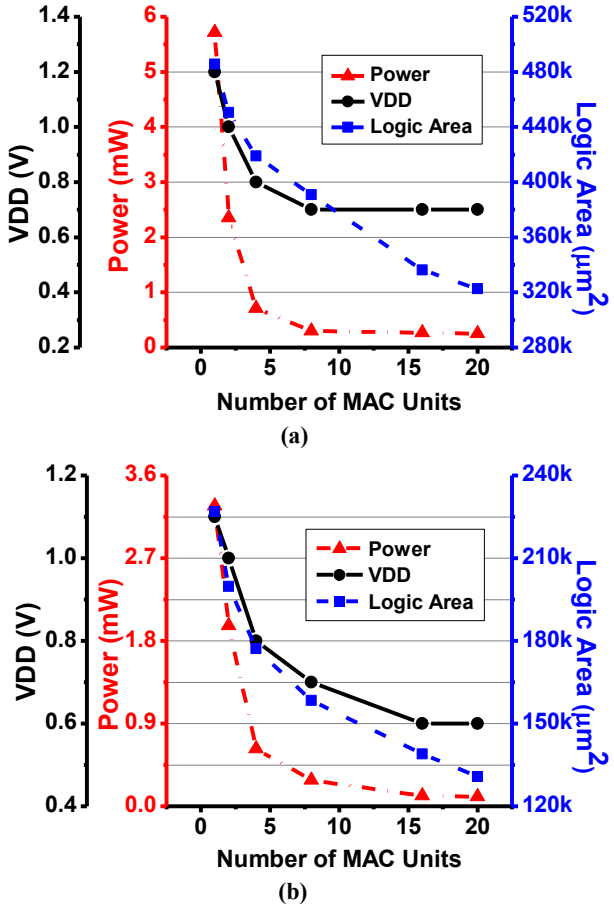

Figure 8. For the same real-time performance, supply voltage, system power, and area results for speech recognition DNN are shown for different number of parallel MAC units in (a) 65nm LP and (b) 40nm LP CMOS.

the speech recognition DNN is predominantly due to the SRAM, consuming >80% of total system power. Therefore, at lower voltages, as the frequency scales down with the increase in the number of MAC units, the leakage power consumes a larger portion of the total power, especially in 40nm LP (>50%).

6. CONCLUSION

In this paper, we proposed a software-hardware co-design scheme of deep neural networks where the weight memory is compressed by dropping connections in blocks, leading to more compact and low-power hardware. We show that a network with 75% of its connected network performs at a level similar to that of a fully connected network with similar architecture. We test this approach on two speech applications, namely keyword detection and speech recognition on the RM database. We implemented both networks in 65nm and 40nm LP CMOS, exploring the performance and power trade-offs between computation and memory. Using the proposed CGS technique together with the fixed-point precision weights, the total weight memory of DNNs are reduced by ~95% compared to that of a fully-connected floating-point DNN. Employing voltage scaling, the keyword detection and speech recognition network consumes 7.01μW and 102.29μW in 40nm, respectively. For future works, we intend to use the CGS architecture on recursive neural networks and convolutional neural networks to achieve comparable power and area savings demonstrated above.

7. ACKNOWLEDGEMENT

This work was partially supported by National Science Foundation grant 1535669 and Samsung Advanced Institute of Technology.

8. REFERENCES

[1] D. Yu and L. Deng, Automatic Speech Recognition, Springer Signal and communication technology, 2012.

[2] D. Su, X. Wu and L. Xu, "GMM-HMM acoustic model training by a two level procedure with Gaussian components determined by automatic model selection," *Proc. ICASSP,* pp. 4890-4893, 2010.

[3] J. R. Rohlicek, W. Russell, S. Roukos and H. Gish, "Continuous hidden Markov modeling for speaker-independent word spotting," *Proc. ICASSP,* pp. 627-630, 1989.

[4] J. G. Wilpon, L. G. Miller and P. Modi, "Improvements and applications for key word recognition using hidden Markov modeling techniques," *Proc. ICASSP,* pp. 309-312, 1991.

[5] L. Deng et al., "Recent advances in deep learning for speech research at Microsoft," *Proc. ICASSP,* pp. 8604-8608, 2013.

[6] L. Deng, G. Hinton and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: an overview," *Proc. ICASSP,* pp. 8599-8603, 2013.

[7] A. Graves, A. R. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks," *Proc. ICASSP,* pp. 6645-6649, 2013.

[8] T . Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," *Proc. Interspeech,* 2015.

[9] T. He, Y. Fan, Y. Qian, T. Tan and K. Yu, "Reshaping deep neural network for fast decoding by node-pruning," *Proc. ICASSP,* pp. 245-249, 2014.

[10] M. Shah, J. Wang, D. Blaauw, D. Sylvester, H.-S. Kim and C. Chakrabarti, "A fixed-point neural network for keyword detection on resource constrained hardware," *IEEE Workshop on SiPS,* pp. 1-6, 2015.

[11] S. Venkataramani, A. Ranjan, K. Roy and A. Raghunathan, "AxNN: energy-efficient neuromorphic systems using approximate computing," *Proc. ISLPED,* pp. 27-32, 2014.

[12] S. Han, H. Mao and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *Proc. ICLR,* 2016.

[13] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun and R. Fergus, "Regularization of Neural Networks using DropConnect," *Proc. ICML,* vol. 28, no. 3, pp. 1058-1066, 2013.

[14] A. Canning and E. Gardner, "Partially connected models of neural networks," *Journal of Physics A: Mathematical and General,* vol. 21, no. 15, pp. 3275-3284, 1988.

[15] P. Price, W. M. Fisher, J. Bernstein and D. S. Pallett, "The DARPA 1000-word resource management database for continuous speech recognition," *Proc. ICASSP,* pp. 651-654, 1988.

[16] D. Povey et al., The Kaldi speech recognition toolkit, 2011.

[17] W. Gardner, "Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique," *Signal Processing,* vol. 6, no. 2, pp. 113-133, 1984.

[18] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks," *Proc. NIPS,* pp. 1097-1105, 2012.

[19] M. Courbariaux, Y. Bengio and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," *Advances in Neural Information Processing Systems,* vol. 28, pp. 3105-3113, 2015.

[20] S. Gupta, A. Agrawal, K. Gopalakrishnan and P. Narayanan, "Deep learning with limited numerical precision," *Proc. ICML,* 2015.

[21] 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.

[22] S. P. Rath, D. Povey, K. Veselý and J. Cernocký, "Improved feature processing for deep neural networks.," in *Proc. Interspeech*, 2013.

[23] Y. Miao, "Kaldi+PDNN: Building DNN-based ASR Systems with Kaldi and PDNN," in *arXiv:1401.6984*, 2014.