2018 IEEE International Conference on Big Data (Big Data)

CompactNet: High Accuracy Deep Neural Network Optimized for On-Chip Implementation

Abhinav Goel
Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
goel39@purdue.edu

Zeye Liu
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA, USA
zeyel@andrew.cmu.edu

Ronald D. Blanton
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA, USA
rblanton@andrew.cmu.edu

***Abstract*—Recent research has focused on Deep Neural Networks (DNNs) implemented directly in hardware. However, larger DNNs require significant energy and area, thereby limiting their wide adoption. We propose a novel DNN quantization technique and a corresponding hardware solution, CompactNet that optimizes the use of hardware resources even further, through dynamic allocation of memory for each parameter. Experimental results for the MNIST and CIFAR-10 datasets, show that CompactNet reduces the memory requirement by over 80%, the energy requirement by 12-fold, and the area requirement by 7-fold, when compared to the conventional DNN. This is achieved with minimal degradation to the classification accuracy. We demonstrate that, CompactNet provides pareto-optimal designs to make trade-offs between accuracy and resource requirement. The applications of CompactNet can be extended to datasets like ImageNet, and into models like MobileNet.**

***Index Terms*—approximate computing, deep neural networks, low power design, asic**

I. Introduction

The tremendous increases in computational power have enabled Deep Neural Networks (DNNs) to set new standards of performance for numerous tasks that include, object recognition, speech processing, and automatic image captioning [1]. New technologies incorporated in virtual assistants [2], driverless cars [3], and indoor navigation systems [4][5] also use DNNs for prediction and classification. In order to achieve a high classification accuracy, the size of the DNN, along with energy, and memory required is large [6]. Hence, most practical applications for DNNs are trained on large server clusters or GPUs [7]. Now, with machine learning algorithms being deployed on mobile and embedded devices, frequent access to off-chip DRAM for data results in inordinate amounts of energy consumption, which is not feasible for a device designed for low-power consumption and a long battery life [8].

To meet the latency constraints for implementing DNNs for real-time responses, even the fastest CPUs and GPUs are insufficient [3][9]. However, when DNNs are implemented on application-specific integrated circuits (ASICs), DNNs can take full advantage of the hardware customizability and minute requirement for control logic to eliminate redundancies that enables faster run time than software. Google's *Tensor Processing Unit* (TPU) proposed in [9] is an example of dedicated hardware built for DNN implementation that can replace CPUs and GPUs.

Overcoming the need for the large floating-point/fixed-point Multiply-Accumulate (MAC) units, and the associated memory requirement for storing floating-point/fixed-point values is critical for hardware solutions to become more commonplace [10]. CompactNet introduces a new quantization technique, and an updated training algorithm that limits the memory requirement for each parameter (weights and biases of the DNN). Each parameter is represented with only 4 or 8 bits, depending on its *relevance* to the output of the DNN. The inputs, outputs, and activations are limited to a 8-bit fixed-point format to reduce the complexity of the arithmetic computation units on hardware. The amalgamation of these techniques ensures that the hardware costs are significantly reduced.

The rest of the paper is organized as follows. Related prior work in the area is briefly summarized in Section II. In Section III, we contrast the existing standard of *lightweight* DNN implementations with CompactNet and explain its characteristics and the training process. Section IV describes the experiments conducted and analyzes the results in detail. This paper is concluded in Section V and a summary is provided.

II. Related Work

The first part of this section talks about different techniques to reduce data communication, and computations. The second part is an overview of existing *lightweight* DNN implementations.

To address the challenges associated with custom hardware implementation of a DNN, prior research has primarily concentrated on individually reducing data communication or the computational complexity. Through the work in [11], the authors use hashing to exploit the redundancies in DNNs to limit frequent access to memory. *ApproxANN* [12] characterizes the impact of neurons based on their contribution to the output and uses approximation in certain less-critical neurons. The work in [13] shows that reducing the word size has a negligible impact on the accuracy of the system, but reduces memory access drastically. Work in [14] optimizes the number of memory accesses by including cost penalties for the utilization hardware resources. Different methods to reduce the energy consumption by proposing approximate

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

arithmetic circuits for DNNs that can tolerate a certain degree of inaccuracy are proposed in [15][16]. These works mainly address issues related to memory, computation, energy, or area requirements individually, but do not delve deep making DNNs easier to implement holistically.

Compared to a conventional DNN, a *lightweight* DNN has less storage, simpler computation logic, faster inference speeds and lower energy consumption. One popular approach to obtain a *lightweight* DNN complexity is to quantize the floating-point parameters and activations into discrete levels. Binarized neural networks (BNNs) [17]-[19] constrain the parameters to 1 or -1, implying each parameter is only a single bit. Similarly, XNOR-Net [20] constrains parameters, inputs, and activations to 1 or -1, meaning that only XNOR operators are required in each neuron. In Trained Ternary Quantization (TTQ) [21], the parameters are constrained to zero, a positive level, or a negative level. However, the large number of neurons and layers required by BNNs, XNOR-Net, and TTQ to achieve the desired accuracy, makes them unfeasible for most practical applications. Deep Compression [22] constrains the parameters to several quantized levels with linear or density-based quantization to save storage. DoReFa-Net [23] uses fixed-point parameters, activations and gradients with fewer than 32 bits. LightNN [24] constrains the parameters to be either a power of two or a sum of powers of two, thus reducing the storage, and replacing the multiplications with either a single shift or combination of shifts and adds. Overall, these quantized DNNs constrain the parameters from the $\mathbb{R}^d$ to $\mathbb{A}^b$, where d and b are the number of parameters, and $b < d$, $\mathbb{R}$ is the set of real numbers, and $\mathbb{A}$ is the constrained space ($\mathbb{A} \subset \mathbb{R}$). Methods presented in [25] provide a platform to quantize a trained DNN through a retraining algorithm, while CompactNet discusses a new algorithm, which trains a *hardware-friendly* network from scratch.

The quantization method used in CompactNet furthers the gains of prior work by ensuring the classification accuracy is maintained to a great extent, with a new approach to the training process. The novel dynamic memory quantization algorithm used in CompactNet to reduce the off-chip memory accesses and computational complexity can be employed to improve the latency, energy and area requirements. This algorithm dynamically selects a custom bit-sequence for the parameter, based on its relevance to the output of the DNN.

Our contributions through CompactNet include:

1) Most importantly, dynamic quantization for each parameter depending on its relevance to the output.
2) The use of custom 4 and 8-bit sequences to represent the parameters. This provides an advantage in terms of circuit complexity, and memory requirement. This is different from prior methods, which require a selection of a bit-sequence before the network is trained [24].
3) A DNN hardware solution that furthers the gains of prior work [13][24], by using a customized MAC unit that replaces all the multipliers with shift-and-add circuit elements. To further improve performance, an 8-bit fixed-point format is used to represent the activations. The CompactNet model, would blend in seamlessly with high-speed, efficient architectures like MobileNet [26], and SqueezeNet [27].

III. Neural Network Implementations

A. CompactNet Architecture

Each neuron of a DNN is a non-linear function of a weighted sum of inputs. Conventional DNNs predominantly work with single precision floating-point parameters and activations for high precision and accuracy. However, working with floating-point requires significant computation for performing basic operations. The storage requirement is also commensurately large. By using a fixed-point representation, the storage requirement for a DNN can be significantly reduced. The self-correcting nature of DNNs offsets the approximation errors and ensures similar classification accuracy as the floating-point representation [13]. An extension of fixed-point as described in [13] is a form of dynamic fixed-point. Dynamic fixed-point uses small word sizes to represent the insignificant values in a DNN, leading to large savings in the total storage. However, the complexity of the basic operators is considerably higher because of the need for an extra decoder circuit to account for operands of several different sizes.

CompactNet uses an 8-bit fixed-point format for representing the inputs and the outputs of the DNN, and the activations of every neuron. This is done to avoid the computational complexity associated with arithmetic of floating-point values. At the end of every forward-pass each parameter is then quantized, and approximated to the nearest power of two, or to the nearest a sum of powers of two dynamically (depending on the relevance of the parameter to the output of the DNN). By doing so, it is possible to replace all the bulky multiplier units with shift-and-add circuit elements, thus improving area utilization.

The appropriate word size for each parameter is assigned depending on its relevance to the output of the DNN. This dynamic selection of 4-bit or 8-bit word sizes for the quantized parameters, coupled with the usage of a very minimal 8-bit fixed-point format to represent the inputs, outputs, and activations is a novel approach to reduce hardware costs.

CompactNet uses a technique called *k*-ones approximation [24] through which each parameter is represented as $parameter = sign(parameter) \cdot (2^{n_1} + 2^{n_2} + \cdots + 2^{n_k})$, where *k* is to be selected. The operations involving the activations of the layer $L-1$ (input of the layer L) and the parameters, in each neuron, can be seen in equation (1), where, $value \ll n$ corresponds to an n-bit left-shift to the *value*. Similarly, a right shift can also be used for negative values of n. From prior research, it is clear that *k* = 2 is sufficient for high classification accuracy [24].

$$\begin{aligned} out_L &= parameter \cdot input_L \\ &= \text{sign(parameter)} \cdot (2^{n_1} + 2^{n_2} + \cdots + 2^{n_k}) \cdot input_L \\ &= \text{sign(parameter)} \cdot (input_L \ll n_1 + input_L \ll n_2 \\ &\quad + \cdots + input_L \ll n_k) \end{aligned} \tag{1}$$

	k = 1		k = 2				
Example E	**Encode** E	**Value stored**	**Upper nibble** U	**Difference** $D = E - U$	**Encode** D **into Lower Nibble**	**Lower nibble** L	**Value stored** $U + L$
11	0011	8	8	3	0011 0001	2	10
-7	1011	-8	-8	1	1011 0000	1	-7
0.74	0101	0.5	0.5	0.24	0101 0110	0.25	0.75
0	0000	1	1	-1	0000 1000	-1	0

Table 1: Examples of parameter encoding when they are represented with both k =1, and k = 2.

Through the novelty of CompactNet we show that, k = 1 can be used along with k = 2, with the 8-bit fixed-point operations. This allows us to achieve a higher classification accuracy, while improving the DNN's performance on hardware. This is an improvement on prior work, where only either k = 1 or k = 2 is used, and the selection of k has to be made before the network is trained. Equations (2) and (3) represent the values of the parameters after performing k-ones approximation, for $k = 1$ and $k = 2$, respectively.

$$parameter_{k=1} = 2^n \tag{2}$$

$$parameter_{k=2} = 2^{n_1} + 2^{n_2} \tag{3}$$

B. Proposed Quantization

All the parameters in CompactNet are quantized to two levels to reduce the memory requirement. The relevance of a parameter to the output, is quantified by a term we call the *Impact of Approximation* (IoA). The IoA, described in equation (4), measures the normalized difference between the highest degree of approximation (using, k = 1) and the actual parameter.

$$IoA = \frac{|parameter_{k=1} - parameter|}{parameter} \tag{4}$$

As seen in equation (5), if the IoA for a parameter lies within a predefined threshold, ϕ, then k = 1 is selected to represent the parameter sufficiently. Otherwise, k = 2 is selected to give a closer approximation. This is done to set an upper bound for the error in the approximated parameter, when $k = 1$ is used. The threshold value is selected depending on the size of the DNN and the dataset used, empirically. A larger threshold value will lead to a greater fraction of parameters being represented with $k = 1$, which leads to lower memory requirements at the expense of classification accuracy.

$$k = \begin{cases} 1, & IoA \leq \phi \\ 2, & otherwise \end{cases} \tag{5}$$

Each quantized parameter is stored using just four and eight bits for k = 1 and k = 2, respectively. This constrains the number of shifts (n in equation (2), n_1 and n_2 in equation (3)) to a maximum of three shifts in either direction. Thus, the reasons for using IoA are two-fold:

1) The parameters with large values, contribute more significantly to the output of the DNN. IoA ensures large values can be represented with more precision, and need not be scaled down, or scaled up to the nearest power of two.
2) From prior work [22], we can infer that a significant percentage of the parameters, lie within the range $[-0.01, 0.01]$. These values get significantly misrepresented, when they are approximated with k = 1, which is undesirable.

The use of only 4 and 8-bit representations ensures there is no need for a complex decoder circuit, which is generally associated with architectures that have large variations in operand sizes [13]. The 4-bit encoding format includes a *Sign bit*: to signify the polarity of the parameter, a *Shift bit*: to indicate an integer parameter (left shift) or a fractional parameter (right shift), and the remaining bits indicate the number of shifts required on the activation so as to obtain a similar result as the multiplication: $activation_{L-1} \times parameter_L$. The 8-bit encoding format is the arithmetic sum of two 4-bit encoding formats, corresponding to $k = 2$. The lower nibble of the 8-bit format is the difference between the true value and the approximation made in the upper nibble. This encoding optimizes the memory utilization, as all the parameters stored in memory are reduced from 32-bit floating point to just 4 or 8 bits.

Table 1 contains examples of the encoding scheme using k-ones approximation. Each example is encoded using k = 1 and k = 2, for illustration. For the case when k = 1 is used, the examples are encoded directly with the aforementioned encoding format. As the range of values for k = 1 is $[-8, 8]$ (three shifts in either direction), values like 11 can not be adequately represented with k = 1, thus requiring k = 2. In the case of k = 2, the closest possible approximation is made in the upper nibble of the encoding format. The difference between the true value and the initial approximation is then encoded in the lower nibble. The range of values for k = 2 is the arithmetic sum of [-8,8] for the upper nibble and [-8,8] for the lower nibble.

Along with each encoded value, a *header bit* is stored in memory. This bit is used to differentiate between 4-bit and 8-bit sequences. For example, if the *header bit* $= 1$, then the next eight bits contain relevant data and will be read from memory. The ninth bit is treated as the *header bit* for the next sequence. Similarly, 4-bit sequences are read from memory, if the *header bit* $= 0$. These *header bits* are accounted for in the results pertaining to the storage, and memory requirement.

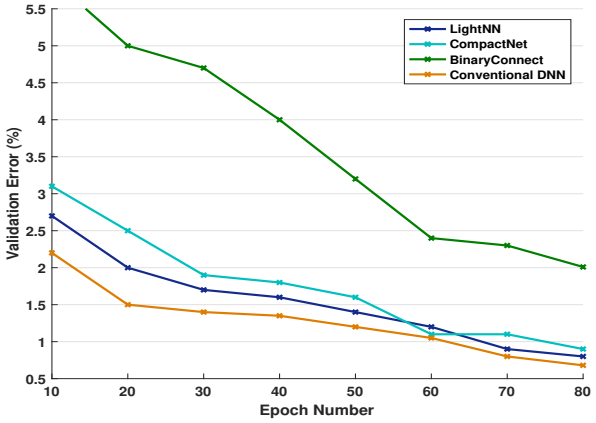

Figure 1: Results from the MNIST 3hidden configuration show that the training time required fairly similar across different techniques.

C. CompactNet Training

Training of CompactNet involves more operations compared to a conventional DNN. However, the network can be trained on a high performance computer, and then be transferred to an embedded system for inferencing. There isn't a need for the embedded system to train the network, as the convergence of the problem is more important than the run time of the training algorithm. The value of k is calculated and assigned to every parameter before each backward pass.

Algorithm CompactNet training

input: Training dataset (x, y), where x is input and y is label; Parameters after the $i-1^{th}$ iteration: p_{i-1} (weights + biases); DNN forward computation function $f(x, p)$; Learning rate η; Threshold value ϕ

output: Updated parameters p_i

for each entry in (x, y) **do**

1. **forward pass:** Compute the activations and output of the loss function l in 8-bit fixed-point format, using the existing parameters p_{i-1} and the function f.
2. **backward pass:** Compute the derivatives of the loss function l wrt p_{i-1} and obtain updated parameters p_i.
3. ***k*-ones approximation and quantization:** Calculate $p_{ik=1} = 2^{n_1}$ and $p_{ik=2} = 2^{n_1} + 2^{n_2}$ for each individual parameter.
4. **if** $IoA \leq \phi$ **then** $p_i = p_{ik=1}$ **else** $p_i = w_{ik=2}$

end for

With the updated value of k, each parameter is approximated to either $parameter_{k=1}$ or $parameter_{k=2}$. The updated values of the parameters are then used in the next forward pass step to calculate the layer activations and the loss function. In order to qualify the area and energy costs of arithmetic units, the activations are accumulated in 8-bit fixed-point as opposed to 32-bit floating-point. The algorithm described is used for training in CompactNet. At the end of the training process the parameters are encoded into 4-bit and 8-bit sequences for k = 1 and k = 2, respectively. Figure 1 shows that the CompactNet training algorithm does not have issues in converging to the global minimum. It must also be noted that the the number of training epochs, and the training time required in CompactNet is not considerably higher compared to the conventional DNN.

Technique	Weights	Activation function	Activation output
Conventional DNN	floating-point	ReLU	floating-point
BinaryConnect	± 1	ReLU	floating-point
BinaryNet	± 1	Sign	± 1
LightNN$_{k=2}$	$\pm(2^{n_1}+2^{n_2})$	ReLU	floating-point
LightNN$_{k=1}$	$\pm 2^{n_1}$	ReLU	floating-point
CompactNet	$\pm(2^{n_1}+2^{n_2})$ and $\pm 2^{n_1}$	ReLU	fixed-point

Table 2: Weight representation and computation comparisons for various approaches to DNN hardware implementation.

IV. Experiments and Results

CompactNet is tested against prior techniques: conventional DNN, LightNN$_{k=2}$ [24], LightNN$_{k=1}$ [24], BinaryNet [19] and BinaryConnect [17]. Table 2 highlights the differences among the various implementation techniques. All the experiments are trained and tested with two datasets: MNIST and CIFAR-10. MNIST contains 70,000 grayscale images of handwritten digits, each of size 28×28 pixels. CIFAR-10 contains 60,000 color images belonging to ten different categories, each image in CIFAR-10 is of size 32×32 pixels.

The approach for training includes the use of the dropout layers in the DNN to avoid overfitting. The loss function used is categorical cross entropy, which gives optimal results for the classification problem at hand. The techniques are prototyped on the Theano platform for Python. The ADAM learning rule is employed in all the experiments. The same number of training epoch is applied to each technique for fair result comparison. The results presented in this paper are obtained after averaging the test error of the epochs with the lowest

Dataset	Config.	Description
MNIST	1hidden	One hidden layer with 100 neurons.
	2conv	Two convolutional layers and two fully-connected layers.
	3hidden	Three hidden layers with 4096 neurons each.
CIFAR-10	3conv	Three convolutional layers and one fully-connected layer.
	6conv	Six convolutional layers and three fully-connected layers.

Table 3: The list of DNN configurations used in the experiments conducted.

	MNIST						CIFAR-10			
	1hidden		2conv		3hidden		3conv		6conv	
Configuration	**Test Error**	**Memory Reduction Percentage**	**Test Error**	**Memory Reduction Percentage**	**Test Error**	**Memory Reduction Percentage**	**Test Error**	**Memory Reduction Percentage**	**Test Error**	**Memory Reduction Percentage**
Conventional DNN	1.75%	0%	0.8%	0%	0.71%	0%	20.19%	0%	9.30%	0%
BinaryConnect	5.6%	95%	4.15%	95%	2.01%	95%	31.04%	95%	13.71%	95%
BinaryNet	7.10%		6.5%		1.9%		61%		21.56%	
LightNN$_{k=1}$	2.00%	87.5%	1.90%	87.5%	0.90%	87.5%	25.31%	87.5%	9.11%	87.5%
LightNN$_{k=2}$	1.93%	75%	1.31%	75%	0.80%	75%	24.60%	75%	8.21%	75%
CompactNet	1.96%	83.60%	1.48%	83.80%	0.84%	84.10%	25.03%	81.60%	8.39%	84.80%

Table 4: Comparison of the test error, and the corresponding memory requirement of different techniques and configurations.

validation error over several iterations. The validation set is used for selecting this best epoch in every iteration. Table 3 lists the various DNN configurations used for each dataset.

It must be noted that, all the results put forward are obtained after training on the same CPU till the training error begins to saturate, to maintain a fair comparison. This is the reason for the difference (if any) from the results published in the original work.

The general trend is that the accuracy increases with increasing precision of the parameters, however there are some exceptions to this rule due to the regularization effects observed due to constraining the parameters. The activation function has a significant impact on the accuracy, that is, ReLU performs considerably better than Sign activation function. Overfitting is frequent in large DNNs; use of dropout layers and regularization is key to avoid entrapment of the loss function in a local minimum. The classification error has been reported in Table 4. The average accuracy decreases in the order: Conventional DNN, LightNN$_{k=2}$, CompactNet, LightNN$_{k=1}$, BinaryConnect, and BinaryNet. From this we can infer that the accuracy drop in CompactNet is very similar, if not better, than other *lightweight* techniques.

The choice of activation function, and the loss function has no bearing on the memory requirement. The conventional DNN, based on full floating-point requires the greatest amount of memory due to the use of 32 bits for each value. BNNs use only a single bit to represent the parameters, hence they require the least amount of memory. The memory requirement decreases in the order: Conventional DNN, LightNN$_{k=2}$, CompactNet, LightNN$_{k=1}$, and BNNs. This trend is depicted in Table 4. As CompactNet uses 4-bit and 8-bit sequences, the storage requirement is between LightNN$_{k=1}$ and LightNN$_{k=2}$ which require only 4-bit and 8-bit sequences, respectively. All the approaches, except CompactNet, provide a constant improvement on the memory requirement when compared with the conventional DNN. Due to the dynamic adaptions to the word size, the memory required in CompactNet varies with the dataset and configuration.

The circuit area requirement for large DNNs is calculated

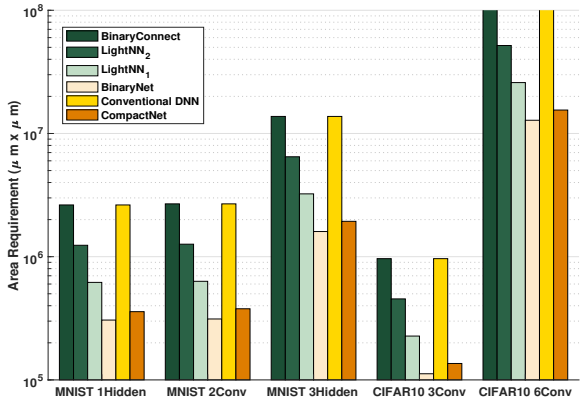

Figure 2: Circuit area required for each technique for various datasets and configurations.

using a pipelined approach, where a single instance of the largest neuron is repeated for each occurrence. Through this process, the worst-case area requirement is calculated. The number of large on-chip components is proportional to the area required. Multipliers present in the conventional DNNs are severalfold larger than the XOR gates needed in BinaryNet. Floating-point operations require much larger and complex components compared to fixed-point operations, thus occupying larger area. The general trend for the area required decreases in the order: Conventional DNN, BinaryConnect, LightNN$_{k=2}$, LightNN$_{k=1}$, CompactNet, and BinaryNet. Figure 2 compares the area required for the various architectures. CompactNet is at least $7\times$ smaller than the Conventional DNN, and only $1.2\times$ larger than BinaryNet in terms of circuit area. While, LightNN$_{k\,=\,1}$ is $4\times$ smaller than the conventional DNN, LightNN$_{k\,=\,2}$ is only $2\times$ smaller. BinaryConnect provides no improvement in terms of area requirement because of the use of floating-point operations. CompactNet shows these gains because of the use of approximate shift-and-add multipliers. The 8-bit representations for activation outputs

ensures that the shift-and-add elements are more compact.

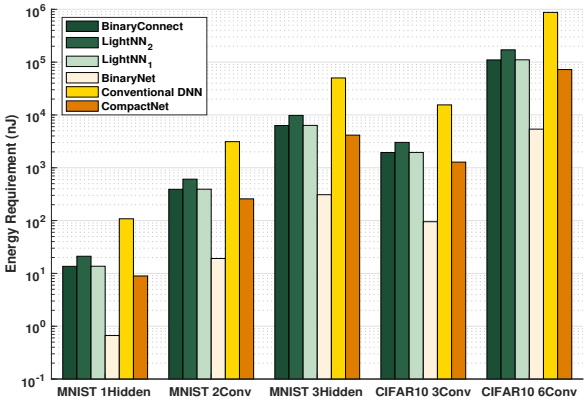

Figure 3: Total energy required for each approach for various datasets and configurations.

The energy requirement is computed as the sum of Switch, Leakage, Memory Access, and Internal energy consumption while running the model. Our results show that the activation functions have a significant impact on the energy requirement of the architecture. The Sign activation function in BinaryNet corresponds to the most energy-efficient logic. Furthermore, fewer bits for the parameters ensures less energy for each memory access. Due to the use of ReLU activation function and 32-bit floating point for the parameters, conventional DNNs require the most energy. From Figure 3 we see the general trend for the energy requirement decreases in the order: conventional DNN, LightNN$_{k=2}$, LightNN$_{k=1}$, BinaryConnect, CompactNet, and BinaryNet. CompactNet reduces the energy requirement by over $12\times$ from the conventional DNN. LightNN$_{k=1}$, LightNN$_{k=2}$, and BinaryConnect provide $7\times$, $5\times$, $8\times$ improvement on the conventional DNN, respectively. In comparison, CompactNet performs significantly better because of the reduced number of memory accesses. Due to the better classification accuracy possible with CompactNet, compared with BinaryNet, it is a more suitable option for most applications.

Through these experiments it is evident that CompactNet improves area and memory consumption significantly, while maintaining the accuracy to a great extent. The energy consumed during inferencing is notably lower than the conventional DNN, and similar to the existing standard of *lightweight* DNN implementations.

V. Summary and Conclusion

There is no single technique that surpasses all other techniques in terms of all factors: accuracy, storage, area, and energy. The tradeoffs are evaluated depending on the dataset, and the hardware resource constraints. Through the experiments the tradeoffs between the accuracy and costs are illustrated. CompactNet provides a greater number of pareto-optimal designs, refer Figure 4 and Figure 5, compared to other architectures and techniques.

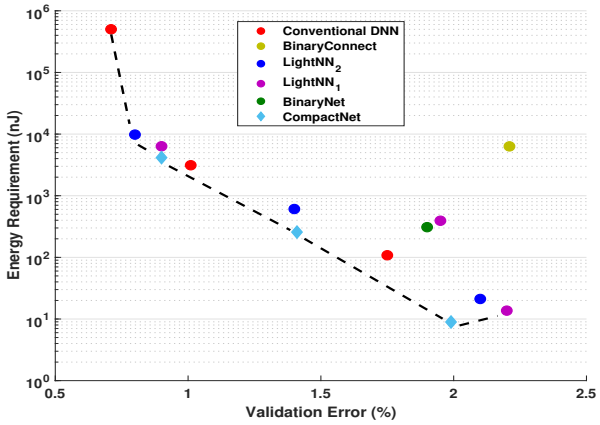

Figure 4: Pareto analysis for different techniques and configurations, based on Energy Requirement on the MNIST dataset. The dashed line represents the Pareto Frontier.

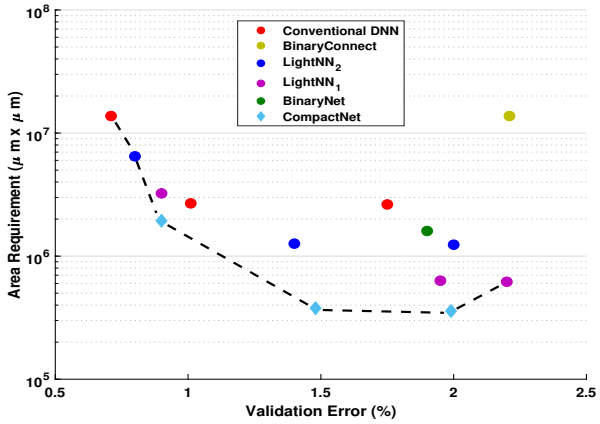

Figure 5: Pareto analysis for different teachniques and configurations, based on Area Requirement on the MNIST dataset. The dashed line represents the Pareto Frontier.

With CompactNet, we help bridge the gap between conventional DNNs and BNNs. The usage of fixed-point operators limits the complexity of the computation logic significantly. When used along with the quantized parameters and the shift-and-add multipliers, the resulting circuit is significantly more efficient, when compared to the conventional DNN. Dynamic encoding of the parameters also allows reduction in the off-chip DRAM accesses. CompactNet provides a suitable option for deployment of DNNs on hardware by reducing the memory, and area requirements while maintaining accuracy.

In our future work, we will employ the quantization method, and architecture described in this paper, with models like MobileNet. We would further improve the inferencing speed, with high accuracy, on the ImageNet dataset.

References

[1] A. Karpathy and L. Fei-Fei. "Deep visual-semantic alignments for generating image descriptions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3128–3137.

[2] J. Hauswald, M. A. Laurenzano, Y. Zhang, et al. “Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers”. In: *SIGPLAN Not.* 50.4 (Mar. 2015), pp. 223–238.

[3] M. Montemerlo, J. Becker, S. Bhat, et al. “Junior: The Stanford entry in the Urban Challenge”. In: *Journal of Field Robotics* 25.9 (2008), pp. 569–597.

[4] Lin Zhang, Taoyun Zhou, and Baowang Lian. “Integrated IMU with Faster R-CNN Aided Visual Measurements from IP Cameras for Indoor Positioning”. In: 18 (Sept. 2018), p. 3134.

[5] S. Anup, A. Goel, and S. Padmanabhan. “Visual positioning system for automated indoor/outdoor navigation”. In: *TENCON 2017 - 2017 IEEE Region 10 Conference*. 2017, pp. 1027–1031.

[6] D. Silver, A. Huang, C. J. Maddison, et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (2016), pp. 484–503.

[7] Q. V. Le, R. Monga, M. Devin, et al. “Building high-level features using large scale unsupervised learning”. In: *CoRR* abs/1112.6209 (2011).

[8] Mitra, Ray, Chatterjee, et al. “Flood forecasting using Internet of things and artificial neural networks”. In: *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. 2016.

[9] N. P. Jouppi, C. Young, N. Patil, et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit”. In: *SIGARCH Comput. Archit. News* 45.2 (June 2017), pp. 1–12.

[10] L. Deng, G. Hinton, and B. Kingsbury. “New types of deep neural network learning for speech recognition and related applications: an overview”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. 2013, pp. 8599–8603.

[11] W. Chen, J. T. Wilson, S. Tyree, et al. “Compressing Neural Networks with the Hashing Trick”. In: *CoRR* abs/1504.04788 (2015).

[12] Q. Zhang, T. Wang, Y. Tian, et al. “ApproxANN: An approximate computing framework for artificial neural network”. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2015, pp. 701–706.

[13] M. Courbariaux, Y. Bengio, and J. P. David. “Low precision arithmetic for deep learning”. In: *CoRR* abs/1412.7024 (2014).

[14] R. Doshi, K. W. Hung, L. Liang, et al. “Deep learning neural networks optimization using hardware cost penalty”. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2016, pp. 1954–1957.

[15] S. Venkataramani, A. Ranjan, K. Roy, et al. “AxNN: Energy-efficient Neuromorphic Systems Using Approximate Computing”. In: *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*. ISLPED ’14. La Jolla, California, USA: ACM, 2014, pp. 27–32.

[16] S. S. Sarwar, S. Venkataramani, A. Raghunathan, et al. “Multiplier-less Artificial Neurons Exploiting Error Resiliency for Energy-efficient Neural Computing”. In: *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. DATE ’16. EDA Consortium, 2016, pp. 145–150.

[17] I. Hubara, M. Courbariaux, D. Soudry, et al. “Binarized Neural Networks”. In: *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016, pp. 4107–4115.

[18] M. Courbariaux and Y. Bengio. “BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1”. In: *CoRR* (2016).

[19] M Courbariaux, Y Bengio, and J. P. David. “BinaryConnect: Training Deep Neural Networks with binary weights during propagations”. In: 28 (Nov. 2015).

[20] M. Rastegari, V. Ordonez, J. Redmon, et al. “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”. In: *CoRR* abs/1603.05279 (2016).

[21] C. Zhu, S. Han, H. Mao, et al. “Trained Ternary Quantization”. In: *CoRR* abs/1612.01064 (2016).

[22] S. Han, H. Mao, and W. J. Dally. “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding”. In: *CoRR* abs/1510.00149 (2015). arXiv: 1510.00149.

[23] S. Zhou, Z. Ni, X. Zhou, et al. “DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients”. In: *CoRR* abs/1606.06160 (2016).

[24] R. Ding, Z. Liu, R. Shi, et al. “LightNN: Filling the Gap Between Conventional Deep Neural Networks and Binarized Networks”. In: *Proceedings of the on Great Lakes Symposium on VLSI 2017*. GLSVLSI ’17. ACM, 2017, pp. 35–40.

[25] A. Zhou, A. Yao, Y. Guo, et al. “Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights”. In: *CoRR* abs/1702.03044 (2017). arXiv: 1702.03044. URL: http://arxiv.org/abs/1702.03044.

[26] A. G. Howard, M. Zhu, B. Chen, et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (2017).

[27] F. N. Iandola, M. W. Moskewicz, K. Ashraf, et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size”. In: *CoRR* abs/1602.07360 (2016).