LOGNET: ENERGY-EFFICIENT NEURAL NETWORKS USING LOGARITHMIC COMPUTATION

Edward H. Lee[1], *Daisuke Miyashita*[1,2], *Elaina Chai*[1], *Boris Murmann*[1], *S. Simon Wong*[1]

Stanford University[1], Toshiba[2]

ABSTRACT

We present the concept of logarithmic computation for neural networks. We explore how logarithmic encoding of non-uniformly distributed weights and activations is preferred over linear encoding at resolutions of 4 bits and less. Logarithmic encoding enables networks to 1) achieve higher classification accuracies than fixed-point at low resolutions and 2) eliminate bulky digital multipliers. We demonstrate our ideas in the hardware realization, LogNet, an inference engine using only bitshift-add convolutions and weights distributed across the computing fabric. The opportunities from hardware work in synergy with those from the algorithm domain.

Index Terms— Neural networks, logarithmic computation, approximate computing, hardware-software co-design

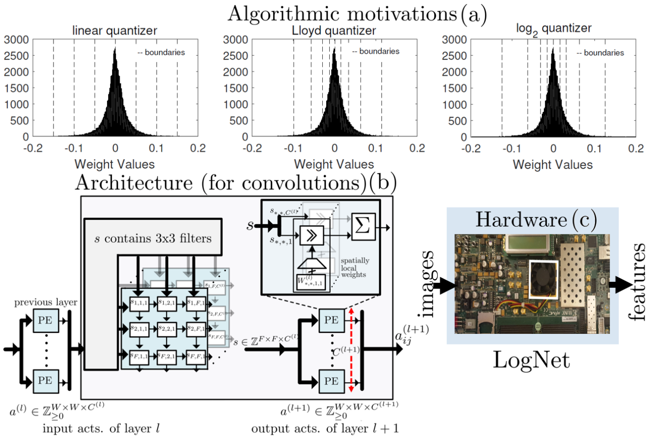


Fig. 1. Low-precision encoding of `conv2_1` layer in VGG16 (a) that illustrates conventional linear (left) versus MSE-optimal (middle) and proposed logarithmic (right) encoding boundaries. The LogNet engine (b), and the hardware realization on the VC707 FPGA board (c).

1. INTRODUCTION

Convolutional neural networks (CNN) have demonstrated state-of-the-art performance in image classification [1, 2, 3] but have steadily grown in computational complexity. For example, *VGG16* [2] and *Deep Residual Learning* [3] networks require 563 and 230 MB of memory to store weights in 32-bit resolution. In order for these large networks to run inference in mobile and real-time embedded systems, it is often times necessary to quantize weights and activations.

Recently, researchers have deployed networks that compute using 8-bit fixed-point representation [4, 5] and have successfully trained networks with 16-bit fixed point [6]. The works of [7, 8, 4, 9] analyzed the effects of quantizing the trained weights for inference. For example, [10] shows that convolutional layers in AlexNet [1] can be encoded to as little as 5 bits without a significant accuracy penalty. There has also been recent work in training using low precision arithmetic. For example, [6] proposes a stochastic rounding scheme to help train networks using 16-bit fixed-point, and [11] proposes quantized back-propagation and ternary connect. However, their method does not completely eliminate all multiplications end-to-end. The work [12] illustrates advantages of encoding activations in the $\log$ domain. Our work extends these ideas by showing that $\log$ is a superior encoding scheme to linear that achieves high classification performance and low hardware complexity for resolutions of 5 bits and lower.

There have been a few but significant advances in the development of specialized hardware of large networks. For example, the works of [13, 14, 15, 16] developed Field-Programmable Gate Arrays (FPGA) and application-specific integrated circuit (ASIC) implementations for convolutional layers. The EIE ASIC [17] exploits sparse-weights for fully-connected layers. Many of these works use fixed and floating-point operations and access data using off-chip DRAM, which can be $1000\times$ more expensive in energy consumption than a memory access to on-chip SRAM [18]. Our hardware realizations exploits low-precision $\log_2 4$b encoded weights, low-precision activations, spatial locality for convolutions, and distributed weights stored across the FPGA fabric.

2. THEORY

Convolutional and fully-connected (FC) layers in a network performs the atomic operations $y = \sigma(w^T x)$, where $x \in \mathbb{R}^n$ is the input (the activations from the previous layer), $w \in \mathbb{R}^n$ the weights for the specified inputs, σ the nonlinear activation function (e.g. `ReLU`), and y the activation to the next

Financial thanks to NSFGRFP and TI SGF. Contact email: edhlee@stanford.edu, Supplementals and models: https://github.com/edhlee/

978-1-5090-4117-6/17/$31.00 ©2017 IEEE

layer. In hardware, this requires n multiply-add operations. For energy-efficient inference, linear fixed-point at 8b (int8) is commonly used; any further decrease in resolution usually results in large classification errors. In the range of resolutions between 1 and 5b in the context of inference for neural networks, we argue that logarithmic base-2 encoding ($\log_2$) is preferred over linear fixed-point.

For any modality, an ideal number $w \in \mathbb{R}$ is encoded as a discrete element in a finite set of size 2^N, where N is the resolution or bitwidth of w. For example using linear fixed-point, the discrete set is arranged with 2^N equidistant boundaries. A conversion from w to its fixed-point representation is described by the operation $\text{LinearQuant}(w) = \mathbf{proj}_{\Omega}(w)$ that projects $w \in \mathbb{R}^n$ onto $\Omega = \{\texttt{min}, \texttt{min} + \Delta_{\texttt{LSB}}, \ldots, \texttt{max}\}^n$, a lattice with 2^N boundaries per element in w. The variables min and max are user-defined, and least-significant bit step-size is denoted as $\Delta_{\texttt{LSB}} = (\texttt{max} - \texttt{min})/2^N$. We also define the full-scale range as $\text{FSR} = \log_2\{\texttt{max} - \texttt{min}\}$ in logarithmic scale for notational convenience. To quantize the non-negative activations, we define quantization of x that is used throughout our experiments specifically as $\text{LinearQuant}(x) = \text{ClipLinear}\left(\text{Round}\left(\frac{x}{\Delta_{\texttt{LSB}}}\right) \times \Delta_{\texttt{LSB}}, 0, 2^{\text{FSR}} - \Delta_{\texttt{LSB}}\right)$, in which $\Delta_{\texttt{LSB}} = 2^{\text{FSR}-N}$, and $\text{ClipLinear}(x, \texttt{min}, \texttt{max}) = \texttt{min} \cdot \mathbb{1}(x < \texttt{min}) + \texttt{max} \cdot \mathbb{1}(x \geq \texttt{max}) + x \cdot \mathbb{1}(\texttt{min} \leq x \leq \texttt{max})$. $\text{Round}(x)$ means to round x to the nearest integer, and $\mathbb{1}(a) = 1$ if the condition a is true and 0 else. Logarithmic quantization of x throughout this paper is described by $\text{LogQuant}(x) = \alpha \cdot \text{Sign}(x)2^{\tilde{x}} \cdot \mathbb{1}(x \neq 0)$, where $\tilde{x} = \text{ClipLog}\left(\text{Round}(\log_2(|x|)), \text{FSR} - 2^N, \text{FSR} - 1\right)$ and $\text{ClipLog}(x, \texttt{min}, \texttt{max}) = \texttt{max} \cdot \mathbb{1}(x \geq \texttt{max}) + x \cdot \mathbb{1}(\texttt{min} \leq x \leq \texttt{max})$. The use of α is used to adjust the dynamic range of the sum of multiply-add operations for the activations if needed; because this constant scaling acts unilaterally on all elements of x, α can be easily absorbed to the next layer (until the very end) in the network or simply removed.

In our paper, the simulations and hardware currently add in the linear domain even when multiplied in $\log$. However, if non-negative activations are encoded in $\log$, we can compute the output activations completely in $\log$. For example, suppose the output activation that we wish to compute is $s_n = w_1 x_1 + \ldots + w_n x_n$, and $\tilde{s}_n = \log_2(s_n)$, $\tilde{p}_i = \tilde{w}_i + \tilde{x}_i$. Assume the weights are also positive. When $n = 2$, $\tilde{s}_2 = \log_2\left(\sum_{i=1}^{2} \text{Bitshift}\,(1, \tilde{p}_i, 1)\right) \simeq \max\,(\tilde{p}_1, \tilde{p}_2) + \text{Bitshift}\,(1, -|\tilde{p}_1 - \tilde{p}_2|, 1)$. For n in general, the summation is monotonically non-decreasing with n and is approximated by $\tilde{s}_n \simeq \max\,(\tilde{s}_{n-1}, \tilde{p}_n) + \text{Bitshift}\,(1, -|\lfloor \tilde{s}_{n-1} \rfloor - \tilde{p}_n|, 1)$. Since $x_i \geq 0$, we only need to ensure that the weights in s_n are of the same sign, otherwise we require two isolated computations: 1) negative weights: $\tilde{s}_{n-} = \log_2(w^T \Sigma_{(-)} x)$ where $\Sigma_{(-)} = \text{diag}(\mathbb{1}(w_1 < 0)\,, \cdots, \mathbb{1}(w_n < 0))$ and 2) positive weights: $\tilde{s}_{n+} = \log_2(w^T \Sigma_{(+)} x)$ before computing $s_{n+} + s_{n-}$ by a logarithmic approximation.

We compare these $\log$ and linear quantizers in their abilities to encode weights and activations of a CNN. We show in Fig. 1(a) the 8 quantization boundaries of linear fixed-point superimposed with the ideal, unquantized weights of a convolutional layer in the fully-trained VGG16 network. The optimal quantization strategy depends upon the distribution of w. For most machine learning algorithms including CNNs, we usually add L_2 regularization in training, which 1) encourages gradients to push weights closer to 0 with force proportional to $|w|$ and 2) is equivalent to setting a Gaussian prior on weights w. Other regularizers such as L_1 (Laplacian prior) also distributes the weights non-uniformly and encourages sparse weights. This idea hints at ways to efficiently encode non-uniform weights rather than conventional linear encoding.

3. SIMULATIONS

To evaluate the rate of distortion induced by quantization, we compute the metric $||W - \text{Quant}(W)||$ for various quantization strategies. The optimal strategy that minimizes $||W - \text{Quant}(W)||_2$ is Lloyd quantization and is shown in Fig. 1(a,middle). The MSE-optimal boundaries are densely-packed near 0 to encode regions with higher occurrence. However, arbitrarily-variable step-sizes are difficult to realize in digital hardware. Using $\log_2$ representation in Fig. 1(a,right) on the other hand, we can achieve decision boundaries that are both close to MSE-optimal but can also be easily realized in digital. However, it is not always the case that $\log_2$ is always better than linear. The optimal strategy strongly depends on the resolution. As shown in Fig. 2, $\log$ encoding leads to both lower normalized MSE (NMSE) and less accuracy degradation in VGG16 (with all other layers encoded in floating-point) than does linear at 4b and lower. For higher resolutions however, linear encoding is more accurate in representing values that are close to the full-scale range of the quantizer.

We evaluate linear and logarithmic encoding strategies on the weights and activations and use classification accuracy (top-5) as the main performance metric using the classification task of ILSVRC-2012 [19] using Tensorflow [5], Caffe [20], and Chainer [21]. For these experiments, we use published models (AlexNet [1], VGG16 [2]) from the caffe model zoo [20]. We quantize each layer of VGG16 to 4b and measure the normalized mean-square error (NMSE) of each layer and its effect on top-5 accuracy with all other layers encoded in unquantized floating-point. The 4b results are shown in Fig. 3 with each layers' unquantized distributions in (a). In (b), both quantization error and accuracy are higher for $\log_2$ 4b than linear 4b except for the conv1_1 and fc8. We perform the same procedure on AlexNet, and we observe that for all layers, $\log_2$ performs better than linear in accuracy and quantization error in NMSE.

Quantizing these layers to 4b also sets many weights to

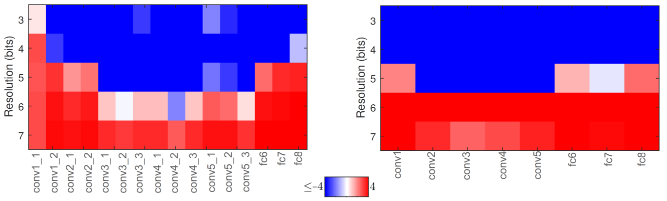


Fig. 2. Heatmap of ΔNMSE $=$ NMSE$_{\text{linear}}$ $-$ NMSE$_{\text{log}}$ of quantized networks for VGG16 and AlexNet. Log encoding is superior below $\approx$ 6b.

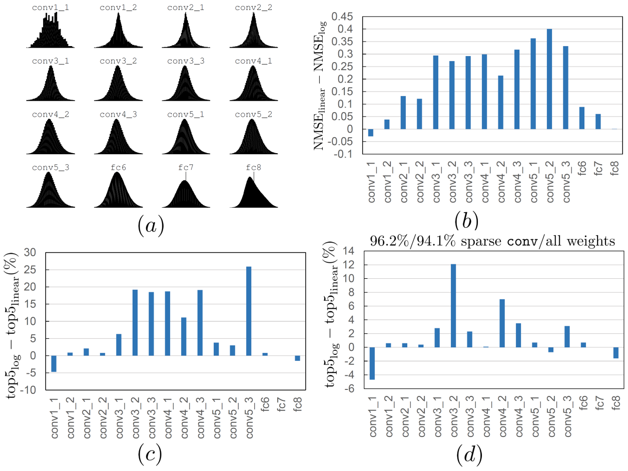
0; for example, the total percentage of zero or sparse weights can easily obtain $\geq 70\%$ for sparse-optimized workloads. To ensure a fair comparison between the two encoding strategies, we compare the performance of log and linear with the same level of sparsity of weights (%). Let Ω_{LSB} represent the set of indices for which the elements in quantized weights represented as $W^{(l)}$ equal Δ_{LSB}. In Ω_{LSB}, we randomly select n elements and set them to 0, where n represents the difference in the # of zero weights for linear and # of zero weights for log. We then set $W^{(l)}(\Omega_{\text{LSB}}) \rightarrow 0$ and perform this operation over all l, which sets the number of sparse weights per layer to be the same for $\log_2$ and linear. The accuracies are shown in (d), where we still observe higher accuracies for all but the two layers as before in (c) but now with the addition of `conv5_2`.

We now quantize all layers to 4 and 5-bits as shown in Table 1 without retraining and measure the classification accuracies; we keep activations in floating-point. These results confirm our expectations from the layer-wise quantization. We see performance drops from floating-point that are not as large for $\log_2$ 4b as they are for linear 4b. In order to improve accuracy degraded by 4b quantization, we retrain the layers using Alg. 1, where we quantize the weights after every forward pass using either LogQuant as shown or LinearQuant for linear encoding.

Algorithm 1 Simple retraining using weights from original quantized network. C is the softmax loss. $f(W^{(k)}, g_{W^{(k)}})$ updates the k-th layer weights $W^{(k)}$ based on SGD's error gradients $g_{W^{(k)}}$.

```
Require: a minibatch of inputs and targets $(a^{(0)}, a^{\texttt{label}})$, quantized weights $W$.
Ensure: updated quantized weights $W^{(k)}$ for $k = 1, \ldots, L$.
    for $k = 1$ to $L$ do $W^{(k)} \leftarrow \text{LogQuant}(W^{(k)})$, $a^{(k)} \leftarrow \text{ReLU}\left(W^{(k)} a^{(k-1)}\right)$ end for
    Compute $g_{a^{(L)}} = \frac{\partial C}{\partial a^{(L)}}$ knowing $a^{(L)}$ and $a^{\texttt{label}}$
    for $k = L$ to $1$ do $g_{a^{(k-1)}} \leftarrow g_{a^{(k)}} W^{(k)}$, $g_{W^{(k)}} \leftarrow g_{a^{(k)}}^{\top} a^{(k-1)}$, $W^{(k)} \leftarrow f(W^{(k)}, g_{W^{(k)}})$ end for
```

The classification accuracy of VGG16 with weights encoded in $\log_2$ 4b with or without is significantly higher than that of linear at 4b with retraining as shown in Tables 1 and 2.



(a)

(b)

(c)

(d)

Fig. 3. Comparison of $\log_2$ 4b and linear strategies to represent (a) layer-wise distributions on (b) quantization MSE, (c) difference in top-5 accuracies, and (d) difference in accuracies when $\log_2$ 4b with pruning is used. For (d), the # of sparse weights is equal for both linear and $\log_2$ 4b.

Table 1. Top-5 accuracies after linear and $\log_2$ encoding on all layers' weights without retraining.

Model	Float 32b	Lin. 4b	$\log_2$ 4b	Lin. 5b	$\log_2$ 5b
AlexNet	78.3%	1.6%	73.4%	71.0%	74.6%
VGG16	89.8%	0.5%	85.2%	83.2%	86.0%

Retraining of weights is necessary to enable a good model in linear 4b but unnecessary for $\log_2$ 4b to meet 85.2% performance. Furthermore, the frequency of zero weights is nonintuitively higher for VGG16 $\log_2$ 4b than linear 4b. We control the sparsity patterns again by stochastically forcing or pruning the weights less than a certain threshold to zero by $w \longleftarrow w \odot \mathbb{1}(|w| > \gamma)$, where γ is the zeroing threshold. We then retrain after pruning by freezing all weights except for `fc8` (the last layer) and subsequently quantizing `fc8`. We achieve an accuracy of 80.5% as shown in Table 2 and denoted as $\log_2$ 4b (pruned). Performing this yields an accuracy that is higher than linear but with total weight sparsity that is 67% as compared with 21% for linear and 35% for the original log (without pruning).

Logarithmic encoding on activations is also resilient at 3 and 4 bits. We apply the LogQuant and LinearQuant functions on the activations of all layers in VGG16 and AlexNet. For this setup, the weights are stored in floating-point. Our observations are as follows. With $\log_2$ 4b activations, the accuracies are 76.9% (AlexNet) and 89.8% (VGG16) as compared to linear 4b 77.1% (AlexNet), 89.4% (VGG16). The difference in performance is small at 4b but grows at 3b where $\log_2$ 3b meets 89.2% while linear 3b falls to 83.0% in VGG16. Finally, from Table 2, encoding activations at $\log_2$ 4b yields modest degradations with of $\sim$ 1% accuracy penalty compared to $\sim$ 2% for linear 4b.

Table 2. Linear versus logarithmic VGG16 after retraining using Alg. 1. We publish these models in [22].

	32b	Lin. 4b	$\log_2$ 4b	$\log_2$ 4b pruned
top-5(32b acts)	89.8%	64.6%	85.2%	80.5%
top-5(4b acts)	–	62.0%	84.8%	80.1%
sparse weights	–	21.2%	35.8%	67.0%
sparse 4b acts.	–	71.2%	58.2%	60.2%

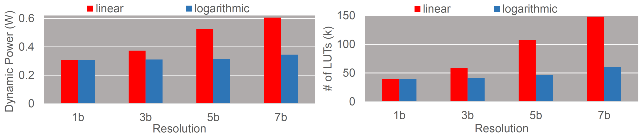


Fig. 4. Hardware costs versus resolution in matrix multiplication. The clock manager presents a ≈ 0.2W fixed cost for both linear and log.

4. LOGNET HARDWARE

In digital hardware, weights logarithmically-encoded in $\log_\beta$ interact with input activations by powers-of-β. For example, the original linear $w^T a \approx \sum_{i=1}^{n} \text{sign}(w_i) \cdot a_i \cdot \beta^{\tilde{w}_i}$, where $\tilde{w}_i = \text{Round}(\log_\beta(w_i))$, a represents the ReLU-ed activations of the previous layer, and $\text{sign}(w_i) = \mathbb{1}(w_i > 0) - \mathbb{1}(w_i < 0)$. Without loss of generality, we can cast the entire $w^T a$ operation as a scaled sum of bitshifts of integers. We express this as $w^T a \approx \Delta_{\text{LSB}}^{-1} \cdot (\sum_{i=1}^{n} \text{Bitshift}(\tilde{a}_i, \tilde{w}_i, \text{sign}(w_i))$, where $\tilde{a}_i = a_i \cdot \Delta_{\text{LSB}}$, and $\text{Bitshift}(x, y, z) = \text{sign}(z) \cdot x \cdot 2^y$ means to take the unsigned integer x and shift its bits by an integer amount $y > 0$ to the left and attach a sign bit $\text{sign}(z) = \mathbb{1}(z > 0) - \mathbb{1}(z < 0)$ to the resulting most-significant-bit (MSB) in signed representation where the summation is performed using a signed accumulator. We will simply express a_i, w_i as integer representations (*i.e.*, $a_i, w_i \in \mathbb{Z}$) for notational convenience (*e.g.* $\text{Bitshift}(a_i, w_i, \text{sign}(w_i))$).

We use bitshifts as multipliers and keep all activations and additions in the linear domain for use in the LogNet convolutional engine as shown in Fig. 1 (c). Two versions of the core were synthesized: 1) with weights stored in distributed block memories and 2) with weights that are statically-programmed into the configurable logic blocks (CLBs). All versions include activations encoded in the linear domain where the activations within the $F \times F$-size sliding window are bitshifted by an amount $\log_2(|W^{(l)}_{\star,\star,i,j}|) \in \mathbb{Z}^{F \times F}$, which are the lth-layer weights for input channel j and output i stored locally near each processing element (PE) as integers and distributed across the entire fabric. Since these static weights are encoded with ultra-low precisions in $\log_2$ 4b, they can localize data movement and therefore reduce overall energy per operation provided that FPGA congestion is well-tolerated by constraining the filter size and activation resolutions.

A matrix-vector product Wa with $W \in \mathbb{Z}^{512 \times 4096}$ is performed where W is read from on-chip block SRAM (8b port) at 100MHz. The dynamic power of the chip and hardware utilization in # of lookup-tables (LUTs) are compared for varying resolutions encoding a and W in Fig. 4. The static power is not shown as it is roughly independent of utilization. We see that both power and utilization scale gracefully with resolution under log as compared to linear. Furthermore, we observe that scaling down to 1b illustrates the convergence of linear and log while also dramatically improving hardware costs for linear. Furthermore, using $\log_2$ 4b increases hardware costs by $1.04\times$ than using 1b whereas using linear 4b requires $2.4\times$.

In the pipelined architectures (Fig. 1), the convolutions of 64 filters are performed in parallel for each $F \times F$ and the window raster-scanned across all activation maps $a \in \mathbb{Z}^{W \times W \times C^{(l)}}$ simultaneously across W, where W is the width (and length) and $C^{(l)}$ the number of input channels. The weights are statically-programmed into CLBs. The engine is demonstrated on ILSVRC-2012 in Fig. 5. Three layers of convolutions in $\log_2$ 4b with pruned weights are performed on the FPGA and are part of a larger network containing fully-connected layers that are performed off-chip. The same network is profiled using the Nvidia Titan X (Maxwell) with CuDNN(R5) [23]. We measure the FPGA chip (VX485T) power using the TI Power Designer [24] and the frame rates by the number of frames per second in a real-time environment. We assume only 50% (similar to EIE's 70% [17]) of the GPU board powers computed from `nvidia-smi` are used to power the chips. The efficiency advantage of the pipelined LogNet ($\log_2$ 4b) over GPUs is apparent when pooling is not used because layers with time-decimated inputs are still clocked at the full clock rate set by the first layer's sampling rate.

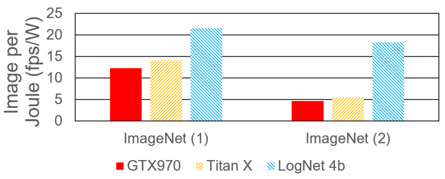


Fig. 5. Efficiencies of GPU and proposed pipelined LogNet. ImageNet (1) contains $\{$`conv-ReLU-maxpool`$\} \times 3$, and (2) contains $\{$`conv-ReLU`$\} \times 3$ with input size 224×224, 64 filters per layer, batch size of 1, and pruned 4b log weights.

5. CONCLUSION

This paper presents the motivations for log computation to enable efficient inference for convolutional networks. We show that log computation can enable more accurate encoding of weights and activations than can linear at resolutions $\leq$ 4b. We take advantage of log encoding of weights to enable convolutions and matrix-multiplications on the FPGA.

6. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, Eds., 2012, pp. 1097–1105.

[2] Karen Simonyan and Andrew Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:11409.1556*, 2014.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep residual learning for image recognition,” *arXiv preprint arXiv:1512.03385*, 2015.

[4] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao, “Improving the speed of neural networks on cpus,” in *Proceedings of Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[5] Martín Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, Software available from tensorflow.org.

[6] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan, “Deep learning with limited numerical precision,” in *Proceedings of The 32nd International Conference on Machine Learning (ICML2015)*, 2015, pp. 1737–1746.

[7] Sungho Shin, Kyuyeon Hwang, and Wonyong Sung, “Fixed point performance analysis of recurrent neural networks,” in *Proceedings of The 41st IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP2016)*. 2016, IEEE.

[8] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang, “Resiliency of deep neural networks under quantization,” *arXiv preprint arXiv:1511.06488*, 2015.

[9] Song Han, Huizi Mao, and William J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[10] Song Han, Jeff Pool, John Tran, and William Dally, “Learning both weights and connections for efficient neural network,” in *Proceedings of Advances in Neural Information Processing Systems 28 (NIPS2015)*, 2015, pp. 1135–1143.

[11] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio, “Neural networks with few multiplications,” *arXiv preprint arXiv:1510.03009*, 2015.

[12] Daisuke Miyashita, Edward H. Lee, and Boris Murmann, “Convolutional neural networks using logarithmic data representation,” *CoRR*, vol. abs/1603.01025, 2016.

[13] Clément Farabet et al., “Hardware accelerated convolutional neural networks for synthetic vision systems,” in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS),*. 2010, pp. 257–260, IEEE.

[14] Jiantao Qiu et al., “Going deeper with embedded fpga platform for convolutional neural network,” in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA, 2016, FPGA ’16, pp. 26–35, ACM.

[15] Chen Zhang et al., “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in *Proceedings of 23rd International Symposium on Field-Programmable Gate Arrays (FPGA2015)*, 2015.

[16] Yu-Hsin Chen, Joel Emer, and Vivienne Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” 2016.

[17] Song Han et al., “EIE: efficient inference engine on compressed deep neural network,” *CoRR*, vol. abs/1602.01528, 2016.

[18] Mark Horowitz, “Computing’s energy problem (and what we can do about it),” *IEEE International Solid State Circuit Conference*, 2014.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[20] Yangqing Jia et al., “Caffe: Convolutional architecture for fast feature embedding,” in *Proceedings of the 22nd ACM International Conference on Multimedia*. 2014, pp. 675–678, ACM.

[21] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton, “Chainer: a next-generation open source framework for deep learning,” in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

[22] Edward H. Lee, “https://github.com/edhlee/,” 2017.

[23] Nvidia, “https://developer.nvidia.com/cudnn,” 2016.

[24] Texas Instruments Power Designer, “http://www.ti.com/tool/fusiondigitalpowerdesigner,” 2016.