REDUCED-PRECISION STRATEGIES FOR BOUNDED MEMORY IN DEEP NEURAL NETS

Patrick Judd[1], **Jorge Albericio**[1], **Tayler Hetherington**[2], **Tor Aamodt**[2], **Natalie Enright Jerger**[1], **Raquel Urtasun,** [3] **and Andreas Moshovos**[1]

[1]Department of Electrical and Computer Engineering
University of Toronto
{juddpatr,jorge,enright,moshovos}@ece.utoronto.ca

[2]Department of Electrical and Computer Engineering
University of British Columbia
{taylerh,aamodt}@ece.ubc.ca

[3]Department of Computer Science
University of Toronto
urtasun@cs.utoronto.edu

ABSTRACT

This work investigates how using reduced precision data in Convolutional Neural Networks (CNNs) affects network accuracy during classification. More specifically, this study considers networks where each layer may use different precision data. Our key result is the observation that the tolerance of CNNs to reduced precision data not only varies *across* networks, a well established observation, but also *within* networks. Tuning precision per layer is appealing as it could enable energy and performance improvements. In this paper we study how error tolerance across layers varies and propose a method for finding a low precision configuration for a network while maintaining high accuracy. A diverse set of CNNs is analyzed showing that compared to a conventional implementation using a 32-bit floating-point representation for all layers, and with less than 1% loss in relative accuracy, the data footprint required by these networks can be reduced by an average of 74% and up to 92%.

1 INTRODUCTION

Deep learning approaches attempt to learn complex high level abstractions of the data by composing simple non-linear transformations. Convolutional Neural Networks (CNNs) have been shown to be extremely effective when solving supervised learning tasks, particularly in the context of object recognition, e.g., detection (Girshick et al., 2013; Sermanet et al., 2013), segmentation (Chen et al., 2015), image classification (Krizhevsky, 2011).

Current approaches utilize very deep architectures, which require massive amounts of memory. As a consequence it is difficult to train networks without exploiting strategies such as sample or model parallelism. More importantly, in order to make these networks applicable to real-time processing in small devices such as embedded systems or mobile devices, the memory requirements have to be addressed.

A popular approach has been to *reduce the precision of the data*, that is the number of bits used. While this introduces approximation error in the internal calculations, CNNs have proven to be very tolerant (Chippa et al., 2013). This is probably due to the fact that training is robust to noise, e.g., by using mini batches, data augmentation, and/or dropout (Srivastava et al., 2014). Software implementations of CNNs, including GPU accelerated ones, commonly use single-precision floating point values (Jia et al., 2014; AMD, 2012; Buck, 2015). However, it is recognized that the large dynamic range of floating point values in unnecessary, and that a fixed point representation might

suffice. The same is true for the 32-bit precision, as most implementations use only 16 bits (Chen et al., 2014b; Gupta et al., 2015; Courbariaux et al., 2014). Further reducing the precision has also been explored (Hwang & Sung, 2014; Anwar et al., 2015; Courbariaux et al., 2015; Lin et al., 2015).

Using a reduced precision representation has many benefits: 1) saves energy in the memory and communication channels (e.g., memory and on-chip links), 2) improves performance in memory bound systems through better memory bandwidth utilization and effective cache capacity, and 3) supports larger networks on systems with a fixed memory budget.

Previous work has primarily used a one-size-fits-all approach to choosing precision, resulting in a precision length that is short enough to work well for *all* networks. This is a *worst case* analysis approach where all networks are forced to use the longest representation needed by *any* network among those under consideration.

In contrast, this work analyzes the tolerance of CNNs to reduced precision error at a *per-layer* granularity. It first corroborates that there is significant variance in the precision needed for the weights *across* different networks and layers. This observation is the first contribution of our work. Building upon this observation, our second contribution is a method for selecting per layer precisions that maintain network accuracy within a desired range.

We study the effects of per-layer precision selection in the context of five well known CNN architectures: LeNet (Lecun et al., 1998), Convnet (Krizhevsky, 2011), AlexNet (Krizhevsky et al., 2012), Network in Network (NiN) (Lin et al., 2013), and GoogLeNet (Szegedy et al., 2014). We show that to maintain accuracy within 1% of the full precision obtained using a network with 32-bit single-precision floating-point values, layers can instead use a fixed-point representation of only 14 bits in the worst case and of just 2 bits in the best case. We demonstrate that allowing each layer to use a different precision can reduce the memory footprint of a network on average by 76% when compared to using 32-bit values, or by 51% when compared to 16-bit values. These results serve as motivation for pursuing further work in implementing such memory and computation optimizations.

The rest of this paper is organized as follows. Section 2 reports an analysis of the per layer error tolerances of five networks, follow by a method for finding the best mixed representation for a given network. Section 3 discusses related work, and Section 4 summarizes our observations and results.

2 CNN Accuracy vs. Representation Length

This section studies the data representation length requirements for our targeted CNNs. Section 2.1 details the experimental setup and measurement methodology, defines the terminology used throughout the rest of this study, and lists the CNNs studied. Section 2.2 investigates how accuracy varies with precision across networks where all layers within each network are forced to use the same representation. This analysis corroborates that precision requirements vary across networks. Section 2.3 studies per layer precision requirements for each layer in isolation demonstrating that the precision needs vary within each network, the key result of this study. Here we study each layer in isolation varying precision one layer at a time. Finally, Sections 2.4 and 2.5 consider the effects of per layer precision selection to overall network accuracy where we may assign a different precision to each layer. This study is done in the context of memory traffic reduction. Accordingly, Section 2.4 reports the baseline memory traffic requirements whereas Section 2.5 proposes a method for selecting per layer precisions while maintaining overall network accuracy within a desired range of that of the baseline.

2.1 Measurement Methodology

CNN Library: The results of this section are collected using the popular Caffe framework (Jia et al., 2014). To measure the effects of reduced precision we used source code level modifications, a method that precluded using closed-source implementations. However, the conclusions drawn about precision variation tolerance should be applicable to other implementations of CNNs that use a 32-bit floating-point representation. The different CNNs are implemented in Caffe using a 32-bit single-precision floating-point representation for all numerical data.

How was Precision Varied per Layer: To study the effect of numerical representation on accuracy, we convert the values to the desired representation and then back to single-precision floating-point

Task	Data set	Network	Layer	Top-1 Accuracy
Digit classification	MNIST	LeNet	2 CONV and 2 FC	0.9904
Image classification	CIFAR10	Convnet	3 CONV and 2 FC	0.7173
Image classification	ImageNet	AlexNet	5 CONV and 3 FC	0.5748
		NiN	12 CONV	0.5592
		GoogLeNet	2 CONV and 9 IM	0.6846

Table 1: Networks studied: Accuracy reported is for the baseline configuration. CONV = convolution, FC = fully connected, IM = inception module. Appendix A describes the layers in more details.

prior to processing them in each layer. This is appropriate for any potential memory and communication optimizations that would not change the way computations are performed but rather how data is represented when communicated across layers either on-chip or through memory.

To perform this analysis we modified the Caffe framework to capture data read and write calls and convert the default single-precision floating-point values into a lower precision fixed-point representation. Note that precision is lost during this conversion. Prior to starting the computation for each layer, we convert the fixed-point numbers into single-precision floating point. This conversion does not restore the original floating point number, as precision was lost during the first conversion.

Target Numerical Representation: We target a fixed-point representation for all values processed by the networks and study the length required for accurate classification. Fixed-point representations are compatible with integer arithmetic units and conversion from existing numerical data types is relatively straightforward. We parameterize an N-bit fixed point value as having I integer bits and F fractional bits. We study how changing I and F across layers and networks affects the overall network accuracy.

Values Studied: We consider both the model values and the layer data outputs/inputs to each layer during classification. We will use the terms **weights** to refer to the model weights (after training) and **data** to refer to the output of each layer.

CNNs Studied: We consider the five most popular neural networks used for image classification which are listed in Table 1 along with their respective datasets. They range from the relatively simple five layer LeNet to the 22 layer GoogLeNet which was the best network in the 2014 ImageNet Competition (Russakovsky et al., 2015).

We use the models and the pre-trained weights that are available for these networks either through the Caffe distribution or the Caffe Model Zoo (Jia, 2015). For ImageNet we use the ILSVRC2012 Task 1 dataset [1]. We run each network for 100 batches of 50-100 input images from the validation set of the respective dataset. The last column of Table 1 reports the accuracy achieved on the baseline Caffe implementation, which uses single-precision floating-point values.

Accuracy Metric: We use top-1 accuracy (instead of the typical top-5) to increase the sensitivity to reduced precision error. Further reduction in precision may be possible if we were to consider the top-5 accuracy. Table 1 reports the baseline top-1 accuracy for the CNNs considered.

Assigning Precision: For most networks we consider assigning a particular precision to each layer. We could go a step further and assign a precision to each computational stage within the layer, however, we have found stages within the same layer tend to have the same precision tolerance. In support of this observation, Fig. 1 shows evidence that the precision requirements within the individual computational stages of the second convolution layer of AlexNet has very similar precision requirements. For GoogLeNet, we assign a precision to each "inception module" to simplify the analysis and refer to them as "layers" to be consistent with the other networks.

A layer typically contains a single, convolution or fully-connected stage which performs the bulk of the computation. This stage is often followed by a series of simpler stages such as ReLU, pooling and linear response normalization. Activation functions like ReLU are often considered part of the previous stage but for consistency we follow Caffe's model where activations are a separate stage.

[1] http://www.image-net.org/challenges/LSVRC/2012/

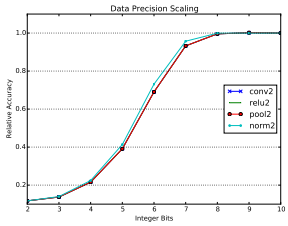
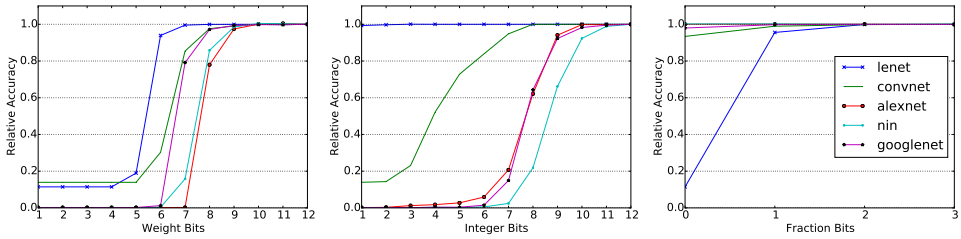


Figure 1: AlexNet accuracy variation as a function of data bits within the second convolution layer.



(a) Weights (b) Data: Integer (c) Data: Fraction

Figure 2: Accuracy relative to the baseline when the bit width is uniform for all layers.

Table 1 reports the number and type of layers per network. There are three layer types according to the main layer that affects accuracy: 1) CONV where the main stage is a convolution, 2) FC where the main stage is a fully-connected layer, and 3) IM, for GoogLeNet, where we treat inception modules as layers. Appendix A lists the stages in each layer of each network.

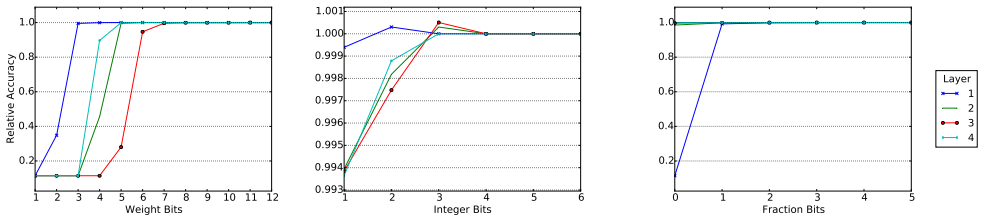
2.2 Uniform Representation Across All Layers

The results of this section confirm that precision requirements vary *across* networks. Specifically, this section studies the per network, minimum *uniform* representation length. For this analysis, we require that the same representation be used by all layers in the network. Since existing implementations choose a presentation that is sufficient for *any* network, they use a *worst case* analysis approach. The results of this section demonstrate, that this current *worst case* analysis is suboptimal.

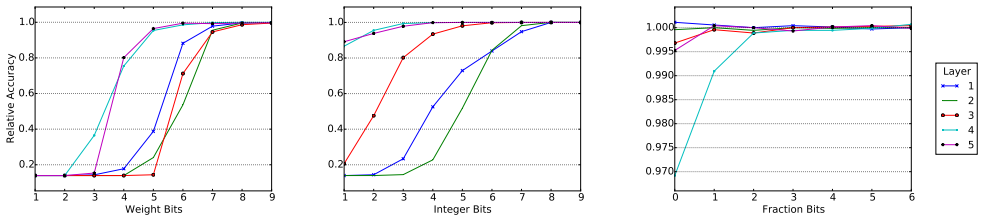
Weights: Fig. 2(a) shows accuracy variation with the number of bits used for the weights. Weights real numbers, typically between -1 and 1, and hence we fix the integer part to 1 bit and only report results when varying the fractional part of the fixed-point representation. Using 10-bit weights is sufficient to maintain accuracy for all networks studied. Note that LeNet can use 8-bit fixed-point weights as well with no loss in accuracy.

Data: Figs 2(b) and (c) report accuracy as we vary the number of bits used for the integer (b) and fractional (c) portions of the fixed-point representation for the data. Accuracy in Convnet and LeNet persists when the integer portion is at least 8 bits, whereas the other networks require at least 11 bits. Fig. 2(c) shows that most networks need just one fractional bit and some require at most two. These results suggest that a uniform fixed-point representation for the intermediate data values flowing through the network will require a 14-bit fixed-point representation, with 12 integer and 2 fractional bits.

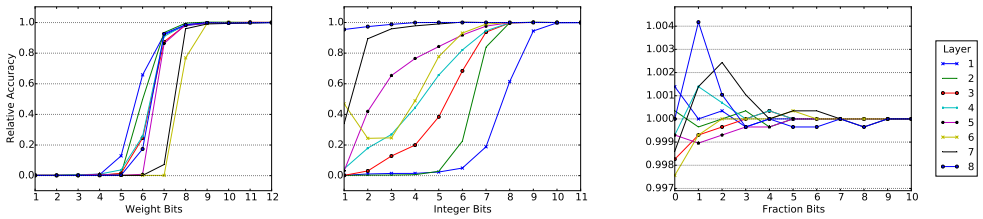
If we had to chose a uniform fixed-point representation for the weights and the data, it would have to be at least 12 bits for the integer portion to accommodate the needs of the intermediate data values, whereas the fractional portion would have to be at least 9 bits to accommodate the fractional portion of the weights. In total we would need 21 bits. This result corroborates the findings of past work that suggested using fixed-point representations (Holt & Baker, 1991; Presley & Haggard, 1994). The results of this section also support the design choices of recent accelerators that use a 16-bit fixed-point representation with little loss in accuracy (Chen et al., 2014a;b).



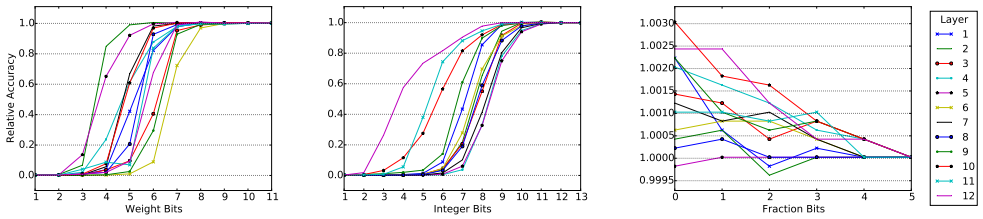
(a) LeNet: Weights (b) LeNet: Data: Integer (c) LeNet: Data: Fraction



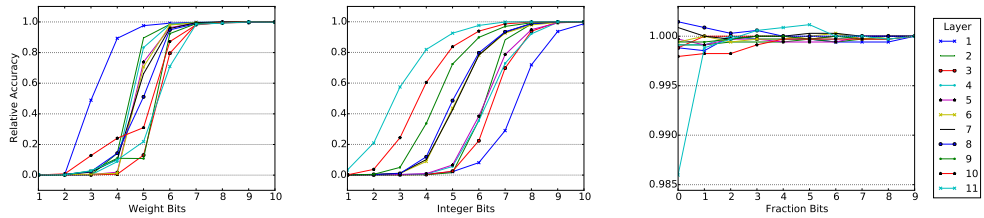
(d) Convnet: Weights (e) Convnet: Data: Integer (f) Convnet: Data: Fraction



(g) AlexNet: Weights (h) AlexNet: Data: Integer (i) AlexNet: Data: Fraction



(j) NiN: Weights (k) NiN: Data: Integer (l) Nin: Data: Fraction



(m) GoogLeNet: Weights (n) GoogLeNet: Data: Integer (o) GoogLeNet: Data: Fraction

Figure 3: Accuracy vs. Representation Length Per Layer: Weights (left), Intermediate Results: Integer (center), and Fraction (right) portions.

2.3 Per Layer Representation Requirements

This section demonstrates that precision requirements vary even *within* each network. This suggest that we should allow each layer to use a different representation, reducing memory footprint and communication needs further. For the purpose of these experiments we maintain the baseline numerical representation for all layers, and vary the representation for one layer at a time. Section 2.5 considers the combined effect of choosing different representations per layer for all layers simultaneously.

Weights: The first column of Fig. 3 shows how CNN accuracy varies when we change the fixed-point representation used for the weights of one layer at time. As weights are typically between -1 and 1, we use a single integer bit (sign bit) and vary the number of fractional bits used by the fixed point representation. The results show that the minimum number of bits needed varies per layer and per network. For example, in LeNet, three bits are sufficient for layer 2, whereas seven bits are needed for layer 3, and in NiN, five bits are needed for layer 2 while nine are needed for layer 3.

Data: The last two columns of Fig. 3 show how CNN accuracy varies when we change respectively the integer and the fractional portions of the fixed-point representation used for the intermediate data values one layer at time. Recall, that the *data* are the inputs and the values produced and communicated between layers. Focusing on middle column of Fig. 3, the integer portion requirements vary greatly across layers and across networks. For example, for Convnet, layer 4 requires just three bits, whereas layer 1 needs eight. The right column of Fig. 3 shows that variation exists in the per layer and per network needs for the fractional portion as well but to a lesser extent for each network.

2.4 Data Traffic Measurements

This section reports the number of data access performed by the networks including the input data, the intermediate data read and written by the layers, and the weights. We will use these measurements in the next section where we present a method for selecting different precisions per layer with the goal of minimizing overall data traffic.

The reported measurements underestimate the amount of traffic generated by the CNNs and thus the benefits that may be possible from reducing off-chip traffic. Specifically, these experiments assume that once a layer touches a piece of data this data is transferred from or to memory only once for the duration of the layer's execution. In practice, layers read values multiple times. These measurements assume that there is enough buffering on chip to capture any data reuse by a layer no matter the reuse distance. In practice this may not be possible as the amount of buffering needed may be prohibitive. For example, a convolution layer will need to buffer multiple full lines or tiles of its input to avoid reading data twice. This may be impractical depending on the image size, e.g., for images or video captured by modern mobile devices.

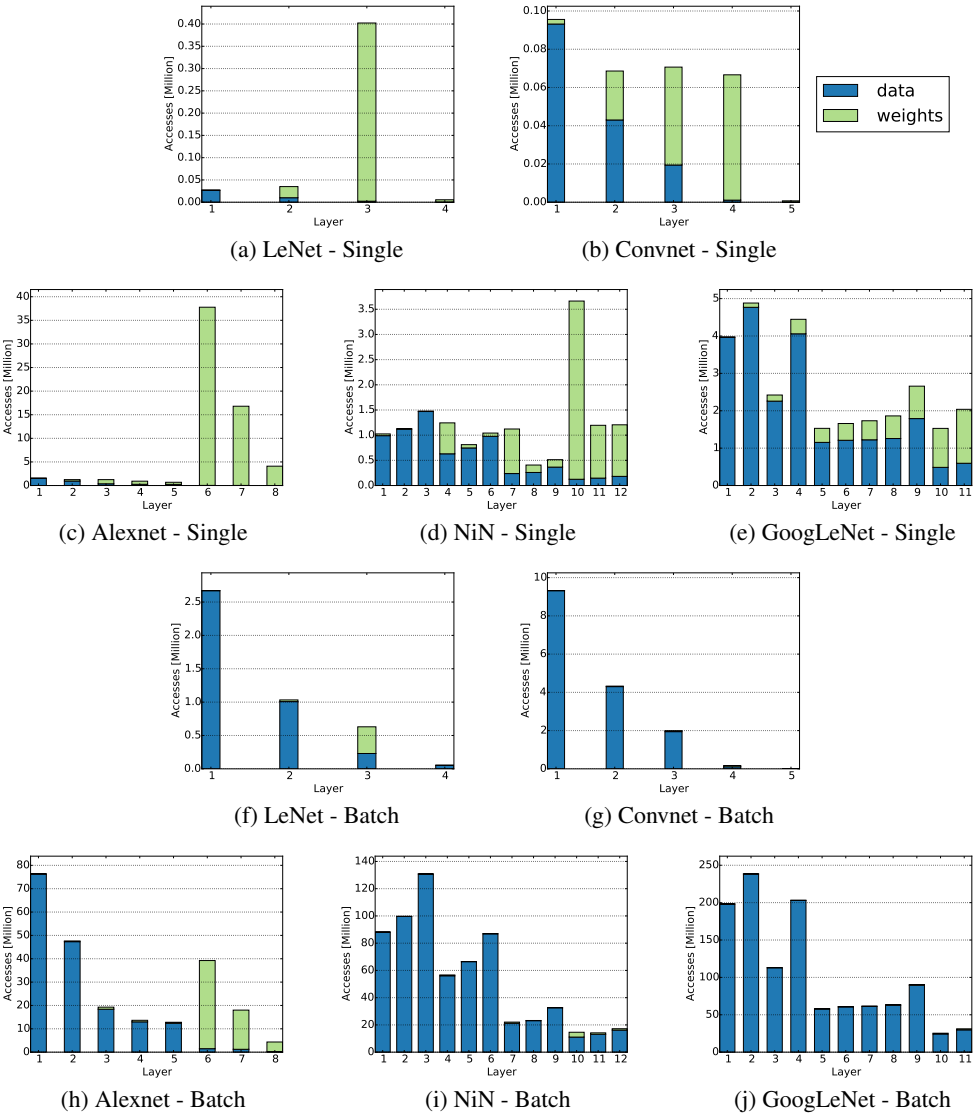
Fig. 4 shows the traffic in millions of accesses, where each access is a single data element. The figure shows two use cases for each network: performing classification on a *single* image or in *batches* of multiple images. When processing a single image the weights make up a significant portion of the traffic, and dominate traffic in three of the networks. GoogLeNet is the exception where data dominate traffic even in the single image use case. In batch processing, the intermediate data dominate traffic. Batch processing feeds multiple images through the same layer before proceeding with the next layer. In turn this requires reading the weights from memory only once per layer and not once per image. In the single image use case, data dominate at the beginning of the network whereas weights dominate towards the end. This is expected as networks tend to use an initial series of convolution layers followed by multiple fully connected layers.

In the interest of space, and given that when possible batching is used in practice since it reduces how often weights ought to be read, the rest of this work focuses on the batch use case.

The amount of data that is needed when processing CNNs in practice may potentially be higher depending on how the intermediate computations are performed. Also, as image resolution as well as network fidelity and complexity increases, memory traffic is bound to increase.

2.5 Choosing the Per Layer Data Representation

So far we studied the effect of changing the representation for one layer at a time. While some loss in fidelity in one layer can be acceptable, this does not suggest that we can simultaneously change the representation for multiple layers and still maintain overall network accuracy. This section studies how accuracy varies as we adjust the representation used by all layers at the same time. The goal is to find the *minimum* length representation possible per layer while maintaining overall network accuracy within acceptable limits. Such a configuration minimizes data traffic while maintaining accuracy.



(a) LeNet - Single (b) Convnet - Single

(c) Alexnet - Single (d) NiN - Single (e) GoogLeNet - Single

(f) LeNet - Batch (g) Convnet - Batch

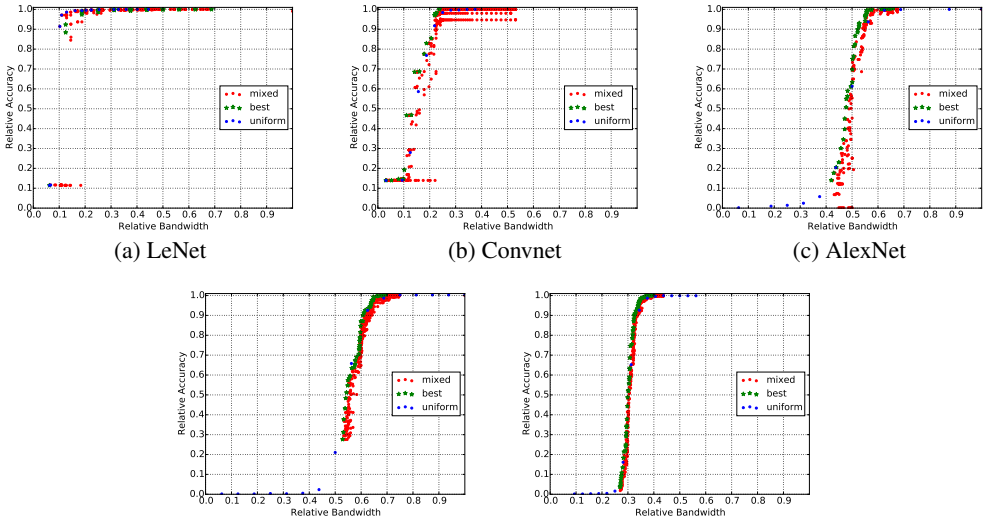
(h) Alexnet - Batch (i) NiN - Batch (j) GoogLeNet - Batch

Figure 4: Data traffic

To explore the design space in a reasonable amount of time we use gradient descent targeting the output accuracy of the network while adjust the representation length used at each layer. Since the reduced precision error tolerance does not strictly decrease as data propagates through the network we cannot easily prune the search space. For example in Figure 3(h) AlexNet layer 5 is more error tolerant than layer 6. As such it is necessary to explore an exponential space of configurations to consider. To make the search tractable we use slowest gradient descent to approximate the Pareto frontier in the accuracy vs. data traffic space. The algorithm is as follows:

1. Initialize all layers to a uniform precision with less that 0.1% error, found in Figure 2(a)
2. Create a set of delta configurations by reducing each parameter, integer bits and fractional bits, in each layer by one.
3. Use the delta configuration with the best accuracy to initialize the next iteration.

For the more complex networks, AlexNet, NiN and GoogLeNet, running this iterative algorithm is time consuming. To reduce the parameter space we fix the fractional bits to 0, 0 and 2 respectively. These bit values achieve less than 0.1% error in Figure 3 (right column). LeNet and Convnet are



(a) LeNet (b) Convnet (c) AlexNet

(d) NiN (e) GoogLeNet

Figure 5: Design Space Exploration: Change in traffic and accuracy when using a different fixed-point representation per layer. X-axis: lower is better, Y-axis: higher is better.

simpler networks and are less tolerant to fractional error so we also vary the fractional bits in our exploration for these networks.

Figure 5 shows the results of this exploration reporting the resulting traffic (x-axis) and accuracy (y-axis) for several configurations studied. Traffic is calculated as the number of accesses times the number of bits per element (weight or data).

Traffic is normalized to the baseline of 32 bits per element. Configurations are assigned to three categories: (1) *uniform* where all layers use the same numerical representation, (2) *mixed* where layers use different numerical representations, and (3) *best* which highlight the Pareto frontier of the mixed configurations.

Generally, the best mixed configurations achieve lower bandwidth than uniform configurations for the same accuracy. However, there is one uniform configuration in Figure 5(d) that lies outside the Pareto frontier of the mixed configurations. This shows that our iterative algorithm is not optimal, otherwise it would have found this better uniform configuration. Thus, there are potentially better mixed configurations that were not explored.

Table 2 reports the configurations that offer the minimum bandwidth for mixed networks given a limit on the error relative to the baseline accuracy of the network. We expect than when accuracy can be traded for traffic savings there will be a hard constraint on the error that is acceptable. We use 1%-10% as a reasonable range of tolerances that we expect for most use cases. Beyond 10% error the plots in Figure 5 tend to drop off sharply, so there is little traffic reduction for a large increase in error. On average the optimal mixed configurations reduce the traffic by 74% with 1% error tolerance and 76% over the 1%-10% tolerance range. Compared to a 16-bit fixed-point baseline, the traffic reductions would be half of those reported here and thus still significant.

3 Related Work

Reduced precision neural networks has been an active topic of research for many years (Xie & Jabri, 1991; Presley & Haggard, 1994; Holt & Baker, 1991; Strey & Avellana, 1996; Larkin & Kinane; Asanovic & Morgan, 1993; Holt & neng Hwang, 1993). Gupta et al. (2015) trains neural networks with 16-bit fixed-point numbers and stochastic rounding. They also propose how to add hardware support for stochastic rounding.

Tolerance	Bits per layer in I.F	TR	Tolerance	Bits per layer (I+F)	TR
LeNet			**AlexNet** (F=0)		
1%	1.1-3.1-3.0-3.0	0.08	1%	10-8-8-8-8-8-6-4	0.28
2%	1.1-2.0-3.0-2.0	0.06	2%	10-8-8-8-8-8-5-4	0.28
5%	1.1-1.0-2.0-2.0	0.05	5%	10-8-8-8-7-7-5-3	0.28
10%	1.0-2.1-3.0-2.0	0.05	10%	9-8-8-8-7-7-5-3	0.26
Convnet			**NiN** (F=0)		
1%	8.0-7.0-7.0-5.0-5.0	0.24	1%	10-10-9-12-12-11-11-11-10-10-10-9	0.32
2%	7.0-8.0-6.0-4.0-4.0	0.22	2%	10-10-9-12-12-11-11-11-10-10-10-9	0.32
5%	7.0-8.0-5.0-3.0-3.0	0.22	5%	10-10-10-11-10-11-11-11-10-9-9-8	0.32
10%	7.0-8.0-5.0-3.0-3.0	0.22	10%	9-10-9-11-11-10-10-10-9-9-9-8	0.30
			GoogLeNet (F=2)		
			1%	14-10-12-12-12-12-11-11-11-10-9	0.36
			2%	13-11-11-10-12-11-11-11-11-10-9	0.35
			5%	12-11-11-11-11-11-10-10-10-9-9	0.34
			10%	12-9-11-11-11-10-10-10-10-9-9	0.32

Table 2: Minimum bandwidth for mixed precision for error tolerance between 1% and 10%. TR reports the traffic ratio over the 32-bit baseline. LeNet and Convnet report the integer bits and fractional bits as $I.F$. Fractional bits are fixed for AlexNet, NiN and GoogLeNet and the total bit width is reported.

Courbariaux et al. (2014) used three different data formats for intermediate data: floating point, fixed point, and dynamic fixed point. They demonstrate how networks can be trained with a low-precision data format. However, they used a uniform representation for across the whole network. Lin et al. (2015); Courbariaux et al. (2015) shows that networks can be trained with binary weights without loss of accuracy. For MNIST, they use a fully connected network with more weights than LeNet. Interestingly, the total number of weight bits is comparable: 2.9 million for their network vs 3 million for LeNet with 7 bit weights. Seide et al. (2014) were able to reduce the precision of gradients to one bit in the training of a neural network using Stochastic Gradient Descent with almost no accuracy loss.

Hwang & Sung (2014) and Anwar et al. (2015) quantize the signal (data) and weights in a fully connected network and CNNs and consider different quantization steps per layer. In the latter work, they also analyze the per-layer sensitivity to the number of quantization levels in LeNet but select the number of bits per layer manually.

4 Conclusion

Classification applications and quality in deep neural networks is currently limited by compute performance and the ability to communicate and store numerical data. An effective technique for improving performance and data traffic is via the use of reduced length numerical representations.

This work provides a detailed characterization of the per-layer reduced precision tolerance of a wide range of neural networks. We highlight a trend of higher precision needs for newer, more complex networks.

We proposed a method for determining an assignment of representations to layers that offers a good trade off between accuracy and data traffic. We estimate that on average we can reduce the storage requirements for the intermediate data in our set of Convolutional Neural Networks by 74% while maintaining classification accuracy to within 1%.

We did not consider the effects of reduced precision during training. The results of this work serve as motivation for studying these further. Promisingly, training with reduced precision has been shown to increase the network's tolerance to the error from reduced precision (Chippa et al., 2013), however, care must be taken to ensure convergence. The results of this work also motivate further work in exploiting the reduced precision for reducing memory bandwidth and footprint, communication bandwidth, and potentially computation bandwidth. The potential benefits include energy reduction, higher performance and the possibility of supporting larger networks.

REFERENCES

AMD. AMD GRAPHICS CORES NEXT (GCN). Whitepaper. "https://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf", 2012.

Anwar, S., Hwang, Kyuyeon, and Sung, Wonyong. Fixed point optimization of deep convolutional neural networks for object recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1131–1135, 2015. doi: 10.1109/ICASSP.2015.7178146.

Asanovic, Krste and Morgan, Nelson. Using simulations of reduced precision arithmetic to design a neuro-microprocessor. *Journal of VLSI Signal Processing*, pp. 33–44, 1993.

Buck, Ian. NVIDIA's Next-Gen Pascal GPU Architecture to Provide 10X Speedup for Deep Learning Apps. "http://blogs.nvidia.com/blog/2015/03/17/pascal/", 2015.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. Semantic ImageSegmentation with Deep Convolutional Nets and Fully Connected CRFs. http://arxiv.org/abs/1412.7062, 2015.

Chen, T, Du, Z, Sun, N, Wang, J, Wu, C, Chen, Y, and Temam, O. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, 2014a.

Chen, Yunji, Luo, Tao, Liu, Shaoli, Zhang, Shijin, He, Liqiang, Wang, Jia, Li, Ling, Chen, Tianshi, Xu, Zhiwei, Sun, Ninghui, and Temam, O. Dadiannao: A machine-learning supercomputer. In *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pp. 609–622, Dec 2014b. doi: 10.1109/MICRO.2014.58.

Chippa, Vinay K., Chakradhar, Srimat T., Roy, Kaushik, and Raghunathan, Anand. Analysis and Characterization of Inherent Application Resilience for Approximate Computing. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pp. 113:1–113:9, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2071-9. doi: 10.1145/2463209.2488873. URL http://doi.acm.org/10.1145/2463209.2488873.

Courbariaux, M., Bengio, Y., and David, J.-P. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *ArXiv e-prints*, November 2015.

Courbariaux, Matthieu, Bengio, Yoshua, and David, Jean-Pierre. Low precision arithmetic for deep learning. *CoRR*, abs/1412.7024, 2014. URL http://arxiv.org/abs/1412.7024.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv preprint arXiv:1311.2524*, 2013.

Gupta, Suyog, Agrawal, Ankur, Gopalakrishnan, Kailash, and Narayanan, Pritish. Deep learning with limited numerical precision. *CoRR*, abs/1502.02551, 2015. URL http://arxiv.org/abs/1502.02551.

Holt, J.L. and Baker, T.E. Back propagation simulations using limited precision calculations. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume ii, pp. 121–126 vol.2, Jul 1991. doi: 10.1109/IJCNN.1991.155324.

Holt, Jordan L. and neng Hwang, Jenq. Finite precision error analysis of neural network hardware implementations. *IEEE Trans. on Computers*, 42:281–290, 1993.

Hwang, Kyuyeon and Sung, Wonyong. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pp. 1–6, Oct 2014. doi: 10.1109/SiPS.2014.6986082.

Jia, Yangqing. Caffe model zoo. *https://github.com/BVLC/caffe/wiki/Model-Zoo*, 2015.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Krizhevsky, Alex. cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks. https://code.google.com/p/cuda-convnet/, 2011.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C.J.C., Bottou, L., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Larkin, Daniel and Kinane, Andrew. Towards hardware acceleration of neuroevolution for multimedia processing applications on mobile devices.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. 2013. URL http://arxiv.org/abs/1312.4400.

Lin, Zhouhan, Courbariaux, Matthieu, Memisevic, Roland, and Bengio, Yoshua. Neural Networks with Few Multiplications. *arXiv:1510.03009 [cs]*, October 2015. URL http://arxiv.org/abs/1510.03009. arXiv: 1510.03009.

Presley, R.K. and Haggard, R.L. A fixed point implementation of the backpropagation learning algorithm. In *Southeastcon '94. Creative Technology Transfer - A Global Affair., Proceedings of the 1994 IEEE*, pp. 136–138, Apr 1994. doi: 10.1109/SECON.1994.324283.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015. doi: 10.1007/s11263-015-0816-y.

Seide, Frank, Fu, Hao, Droppo, Jasha, Li, Gang, and Yu, Dong. 1-bit stochastic gradient descent and application to data-parallel distributed training of speech dnns. In *Interspeech 2014*, September 2014. URL http://research.microsoft.com/apps/pubs/default.aspx?id=230137.

Sermanet, Pierre, Eigen, David, Zhang, Xiang, Mathieu, Michaël, Fergus, Rob, and LeCun, Yann. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15 (1):1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.

Strey, Alfred and Avellana, Narcís. A new concept for parallel neurocomputer architectures, 1996.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

Xie, Yun and Jabri, Marwan A. Training algorithms for limited precision feedforward neural networks. Technical report, 1991.

A SUPPLEMENTARY MATERIAL

Table 3 shows the Caffe models used for each network and the Caffe layers (computaional stages) assigned to each layer in our analysis.

Network	**Source**	**Layer**	**Caffe Layers**
alexnet	http://git.io/v480W	Layer 1	conv1,relu1,pool1,norm1
		Layer 2	conv2,relu2,pool2,norm2
		Layer 3	conv3,relu3
		Layer 4	conv4,relu4
		Layer 5	conv5,relu5,pool5
		Layer 6	fc6,relu6,drop6
		Layer 7	fc7,relu7,drop7
		Layer 8	fc8
convnet	http://git.io/v48RM	Layer 1	conv1,pool1,relu1
		Layer 2	conv2,relu2,pool2
		Layer 3	conv3,relu3,pool3
		Layer 4	ip1
		Layer 5	ip2
googlenet	http://git.io/v480Q	Layer 1	conv1/*
		Layer 2	conv2/*
		Layer 3	inception_3a/*
		Layer 4	inception_3b/*
		Layer 5	inception_4a/*
		Layer 6	inception_4b/*
		Layer 7	inception_4c/*
		Layer 8	inception_4d/*
		Layer 9	inception_4e/*
		Layer 10	inception_5a/*
		Layer 11	inception_5b/*
lenet	http://git.io/v48Eu	Layer 1	conv1,pool1
		Layer 2	conv2,pool2
		Layer 3	ip1,relu1
		Layer 4	ip2
nin	http://git.io/v48EA	Layer 1	conv1,relu0
		Layer 2	cccp1,relu1
		Layer 3	cccp2,relu2,pool0
		Layer 4	conv2,relu3
		Layer 5	cccp3,relu5
		Layer 6	cccp4,relu6,pool2
		Layer 7	conv3,relu7
		Layer 8	cccp5,relu8
		Layer 9	cccp6,relu9,pool3,drop
		Layer 10	conv4-1024,relu10
		Layer 11	cccp7-1024,relu11
		Layer 12	cccp8-1024,relu12,pool4

Table 3: Networks: links to sources and layer definitions.