Accuracy to Throughput Trade-offs for Reduced Precision Neural Networks on Reconfigurable Logic

Jiang Su, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Gianluca Durelli, David B. Thomas, Philip Leong, and Peter Y. K. Cheung

Xilinx Research Labs, Imperial College London, University of Sydney

Abstract. Modern Convolutional Neural Networks (CNNs) are typically based on floating point linear algebra based implementations. Recently, reduced precision Neural Networks (NNs) have been gaining popularity as they require significantly less memory and computational resources compared to floating point. This is particularly important in power constrained compute environments. However, in many cases a reduction in precision comes at a small cost to the accuracy of the resultant network. In this work, we investigate the accuracy-throughput trade-off for various parameter precision applied to different types of NN models. We firstly propose a quantization training strategy that allows reduced precision NN inference with a lower memory footprint and competitive model accuracy. Then, we quantitatively formulate the relationship between data representation and hardware efficiency. Our experiments finally provide insightful observation. For example, one of our tests show 32-bit floating point is more hardware efficient than 1-bit parameters to achieve 99% MNIST accuracy. In general, 2-bit and 4-bit fixed point parameters show better hardware trade-off on small-scale datasets like MNIST and CIFAR-10 while 4-bit provide the best trade-off in large-scale tasks like AlexNet on ImageNet dataset within our tested problem domain.

Keywords: Reduced precision, neural networks, FPGA, algorithm acceleration

1 Introduction

Modern CNNs may contain millions of floating-point parameters and require billions of floating-point operations to recognize a single image. These requirements tend to increase as researchers explore deeper networks. On the other hand, the integration of computing resources on hardware platforms is hampered by the slowing down of Moores law. Therefore, it is meaningful to study efficient model designs with customized data paths and effective data representations.

Previous work showed that using reduced precision for NN parameters provide massive improvements on system performance such as throughput, computational resource usage and memory footprint [1 2 3]. For example, Figure[1] shows the roofline for Xilinx KU115 device in terms of its arithmetic intensity and peak board performance. It shows that higher performance “ceiling” can be achieved if using lower precision data in operations. However, as mentioned in [4] and [1], reduced-precision parameters
need more operations and parameters to achieve the same accuracy provided by high
precision alternatives. Additionally, the “operation” as for the y axis in Figure 1 can be
different to various data types. For example, instead of expensive Multiply Accumulate
(MAC) operations for floating point (FP) or fixed point (FIX) representations, XNOR and
popcount logic can be used for Binary Neural Networks (BNNs). Therefore, compared
to FP, binary number operations may lead to a system with higher throughput (GOps/s),
but does this higher throughput provide as good NN accuracy? Another way to ask the
question is that if a target classification accuracy is given to a particular dataset, can
binary parameter based NN still allow more efficient hardware systems than floating
point parameters? We found many questions like this remain unanswered. For example,
how does parameter precision in NNs affect the hardware throughput given a particular
system architecture? Which data type provides the best trade-off between model accuracy
and hardware performance?

In order to address above questions, we focus on an exploration space for various
data representations in NN computation in order to study their impacts to hardware
system efficiency and model accuracy. In contrast to previously published work, which
focuses either on hardware-wise efficiency [5][1][2] or model performance [6][7][8], we
consider both perspectives and tentatively provide a more comprehensive view of using
reduced-precision for NN system design. The contributions of this work is as follows:

- We report our quantization training strategy for NN inference with quantized weights
and activations in arbitrary precision types. Without any compression techniques, our
training strategy requires less memory footprint and achieves competitive accuracy
compared to several state-of-the-art compression techniques on the same task.
- We propose quantitative estimation models to show how parameter precision affects
hardware cost and system throughput for a NN hardware system.
- We publish systematic experimental results for different types of NNs with weights
and activations represented separately in 1-bit (Binary), 2-bit (INT2), 4-bit (INT4), 8-
bit(INT8), 16-bit (INT16) fixed point values and 32-bit floating point values (FP32)
and show their impacts to classification accuracy, hardware cost and inference
throughput.
- Finally, our exploration space provides useful insights and a more comprehensive
view of using reduced-precision values in NN acceleration. For example, in our
MNIST experiments, a networks with FP32 parameters is more memory efficient
than 1-bit parameters for achieving 99% accuracy due to the smaller topology

Fig. 1: Roofline Model for Xilinx KU115 FPGA
required. In general, 2-bit and 4-bit fixed point parameters show better hardware trade-off on small-scale datasets like MNIST and CIFAR-10 while 4-bit provide the best trade-off in large-scale tasks like AlexNet on ImageNet dataset within our tested problem domain.

In the next section, we introduce our training strategy for reduced precision parameters. Next, Section 3 introduces the proposed estimation models for hardware cost and system throughput. The experimental results are discussed in Section 4 and Section 5 finally concludes the paper.

2 Training Strategies

In this work, weights and activation values are quantized before used in the feedforward and backward propagation. For fixed-point representations, values are represented with \( WL \) bits, in which the Most Significant Bit (MSB) indicates the sign while \( FL \) and \( (WL - FL - 1) \) bits are used for expressing the fractional and the integer parts separately.

Specifically, for binary representation, we adopted the deterministic binarization function used in [2] as our quantization method:

\[
x^Q = \text{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise}. \end{cases}
\]

For fixed point values, the quantization function converts real values to nearest pre-defined fixed point representations.

As mentioned in [2][1], for training binary parameters, batch normalization is generally conducted before the activation function while for other representations, it’s the other way around. For activation functions, we use both of the Hard Hyperbolic Tangent Function (hard-tanh) \( \sigma(x) = \min(1, \max(-1, x)) \) and the Rectified Linear Unit (ReLU) \( \sigma(x) = \max(0, x) \). Both of the activations are used in our experiments and the one that delivers a higher model accuracy is selected.

Globally, quantized low-precision weights and activations are used for feedforward and backpropagation passes. After this process, the floating point parameters are updated accordingly (line 20, Algorithm 1). High-precision values are used for updating because they can accumulate tiny value changes while lower-precision values can improve the computational efficiency during inference due to the low design complexity [9].

Our quantization training process is shown in Algorithm 1. \( \text{Quantize(*)} \) is the quantization function. \( \text{BatchNorm(*)} \) and \( \text{BackBatchNorm(*)} \) are functions that propagate neuron-generated values and gradients separately in feedforward and backpropagations. Similarly, \( \text{ActFunc(*)} \) and \( \text{BackActFunc(*)} \) are activation passes in above-mentioned bidirectional propagations. \( \text{Update(*)} \) specifies the parameter updating strategy, ADAM Updating is used in this work [10]. Network weights are initialized based on [11]. Finally, \( C \) is the cost function.

In the feedforward process, real-valued weights \( W \) are firstly quantized into low-precision weights \( W^Q \) as shown in line 2. After batch normalization and activation function, neuron activations are also quantized to low precision (line 7). Above steps form a layer-wise process until the training error \( g_{aL} \) is calculated in the last layer.
Algorithm 1 Quantization training strategy for an $L$-layer neural network

Require: At time step $t$, a batch of inputs $a_0$ and their labels $a^*$, network weights $W$, Batch Normalization parameter $\theta$, learning rate $\eta$ and its decay factor $\lambda$.

Ensure: At time step $t+1$, the updated weights $W_{t+1}$, the updated Batch Normalization parameter $\theta_{t+1}$ and the updated learning rate $\eta_{t+1}$.

1. Propagations with Limited Precision Parameters

1.a Feedforward Propagation:

1: for $i = 1$ to $L$ do
2: \[ W_i^Q \leftarrow \text{Quantize}(W_i) \]
3: \[ s_i \leftarrow a_{i-1}^Q \times W_i^Q \]
4: if $i < L$ then
5: \[ a_i \leftarrow \text{ActFunc}(\hat{s}_i) \]
6: \[ \hat{s}_i \leftarrow \text{BatchNorm}(s_i, \theta_i) \]
7: \[ a_i^Q \leftarrow \text{Quantize}(a_i) \]
8: end if
9: end for

1.b Backward Propagation:

10: Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing $a_L$ and $a^*$
11: for $i = L$ to 1 do
12: if $i < L$ then
13: \[ (g_{s_i}, g_{\theta_i}) \leftarrow \text{BackBatchNorm}(\hat{g}_{a_i}, s_i, \theta_i) \]
14: end if
15: \[ g_{a_i}^Q \leftarrow a_{i+1}^Q \]
16: \[ g_{W_i}^Q \leftarrow g_{s_i}^T \times a_{i-1}^Q \]
17: end for

2. Weight Updating with High Precision Parameters

18: for $i = 1$ to $L$ do
19: \[ \theta_{t+1} \leftarrow \text{Update}(\theta_i, \eta, g_{\theta_i}) \]
20: \[ W_{t+1}^i \leftarrow \text{Clip}(\text{Update}(W_i, \gamma_i \eta, g_{W_i}^Q), -1, 1) \]
21: \[ \eta_{t+1} \leftarrow \lambda \eta \]
22: end for

according to the outputs in the output layer $a_L$ and the corresponding data label $a^*$. Then backward propagation starts with the error calculated through above feedforward pass. After going through backward passes for the activation function and batch normalization function, the quantized weights $W^Q$ are used for the calculation of gradients of both neurons and connections. Noticeably, this is the key point that the model is “aware” of the quantized parameters. This process is a layer-wise process from the output layer to input layer (line 10 - 17). Finally, parameters are updated with the gradients following the ADAM rule. Specifically, the updated values are clipped between -1 and +1 for regularization. The ADAM parameter $\theta$ and learning rate $\eta$ are also updated accordingly. If the $\text{ActFunc}(\ast)$ is hard-tanh and $\text{Quantize}(\ast)$ is Eq.1, the algorithm 1 depicts the training strategy proposed in [2] for 1 bit binary parameters.
Table 1: The Expected Cost Per Operation For Each Precision Type

| Datatype | LUTs (min) | LUTs (max) | LUTs (avg) | DSPs (min) | DSPs (max) | DSPs (avg) | $C_{avg}$ | $C_{rel}$ |
|----------|------------|------------|------------|------------|------------|------------|----------|----------|
| Binary   | 4.24       | 8.00       | 5.58       | 0          | 0          | 0          | 12.02    | 1        |
| INT2     | 10.98      | 18.74      | 13.52      | 0          | 0          | 0          | 29.12    | 2.42     |
| INT4     | 27.18      | 35.56      | 30.06      | 0          | 0          | 0          | 64.76    | 5.39     |
| INT8     | 83.28      | 91.92      | 86.38      | 0          | 0          | 0          | 186.02   | 15.48    |
| INT16    | 21.64      | 38.36      | 28.66      | 1          | 1          | 1          | 181.16   | 15.07    |
| FP32     | 356        | -          | -          | -          | -          | -          | -        | -        |
| KU115    | -663,360   | -5,520     | -          | -          | -          | -          | 766.6    | 63.79    |

The quantization training process is done offline and we only deploy the inference process online with the trained parameters. In the next section, a hardware cost model is introduced for a specific system architecture on FPGAs.

3 Hardware Cost Model For Different Precision Types

In this work, we build up our hardware impact analysis based on a hardware system architecture that is introduced in this section. Firstly, processing elements are introduced as the basic building blocks of conducting above operations. Then a hardware cost model is proposed to theoretically formulate the relationship between parameter precision and system throughput, which is then later applied to our studied trade-offs.

3.1 System Architecture

As shown in Figure 2a, the overall system architecture used in this work is based on a data-flow framework for CNN inference called FINN [1]. Network inputs are loaded from off-chip memory to layer-wise on-chip processing modules. After completing the feedforward computation, the classification outputs are finally transferred back to off-chip memory storage. As shown in Figure 2a, each layer is mapped with an array of Matrix-Vector Operation Unit (MVOU) modules as shown in Figure 2b.

Internally, the MVTU consists of an input and output buffer, and an array of Processing Elements (PEs) each with a number of SIMD lanes. The number of PEs ($P$) and SIMD lanes ($S$) are configurable to control the throughput. A PE can be thought of as a hardware neuron capable of processing $S$ synapses per clock cycle. Each PE receives exactly the same control signals and input vector data, but multiply-accumulates the input with a different part of the matrix. Figure 2c shows the PE data-path for FIX/FP numbers and Figure 2d shows its counterpart for 1 bit binary numbers, which is used in [1]. Noticeably, the Multiplier-ACcumulate (MAC) structures (Figure 2c) for FIX/FP are replaced with XNOR and popcount structure for binary numbers. Either MAC or XNOR/popcount is referred as “operation” or “fundamental operation” for its corresponding data type throughout this paper. Please note, for higher precision parameters, the dataflow model is not necessarily feasible when the chips are not sufficiently large. These situations are beyond the assumption of this work and the related analysis is only for theoretical reference.
3.2 Hardware Cost Estimation Model

Based on the architecture described in last section, we propose our hardware cost estimation model for arbitrary parameter precision type and theoretically formulate the relationship between hardware cost and parameter precision type for the given architecture. Table 1 shows the average hardware cost per fundamental operation for each precision type. In order to get this table, different levels of parallelism for the MVOU has been tried in each data representation and we report an average value for fair comparisons. Because of the sharing of control logic, the average hardware cost for the basic operations can be different depending on the level of parallelism. So we mark the minimum and maximum cost of resources as “min” and “max” in the table and eventually use the average value of them for a more precise estimation. Look-Up-Tables (LUTs) and DSP blocks are both considered as hardware cost in this work. The average cost per operation, $C_{avg}$, is calculated as follows:

$$C_{avg} = \max\left(\frac{LUTs/MAC}{LUT_{usage} * LUT_{TOTAL}}, \frac{DSPs/MAC}{DSP_{usage} * DSP_{TOTAL}}\right),$$

where $LUT_{TOTAL}$ and $DSP_{TOTAL}$ are separately the total available LUTs and DSPs on the target device and $LUT_{usage}$ and $DSP_{usage}$ are separately the proportion of LUTs and DSPs that can be used for arithmetic on the target device. We’ve estimated $LUT_{usage} = 0.7$ and $DSP_{usage} = 1.0$ in this work. $C_{avg}$ is the fraction of the target device resources that are used in average by a fundamental operation for each type and as such is a measure of scarcity of resource. Relative cost, $C_{rel}$, is used to compare the arithmetic cost of binarized networks against other precision types directly. For example, if a Binary and an INT4 network have been trained to achieve the same level
of accuracy, the INT4 network must have 5.38 less operations to have the same accuracy / computation trade-off as the binarized one.

Interestingly, modelling computational cost this way means that INT16 has a lower hardware cost than INT8, because it uses less LUTs/Op than INT8 and the proportion of DSPs that it uses per Op with respect to the total on the target device, a Xilinx Kintex UltraScale 115, is less than the proportion of LUTs/Op used by INT8. These resource usage data are calculated based on Vivado HLS 2016.3 synthesis reports. In this work, we only consider the default synthesis results from the compiler. Optimization to INT8 can be applied to recent Xilinx DSP blocks. This will improve INT8 performance but will not affect the correctness of our estimation model and hence not specially applied in our work. In essence, we assume a custom dataflow architecture generated for each specific network topology (different sizes for the compute arrays in different layers as shown in Fig.2a), meaning that the “one-size-fits-all” inefficiencies of loopback accelerators are avoided. As such, peak performance of a particular device is almost achievable in practice.

### 3.3 Throughput Estimation Model

Hardware cost is highly related to the system performance and computation efficiency. Theoretically, we formulate the relationship between inference throughput and hardware cost as follows:

\[
\text{Throughput} \approx \frac{\text{Freq.}}{\#OP} \times C_{\text{avg}} + \Delta, \tag{3}
\]

where Freq. is the working clock frequency, \#OP is the number of operations required to compute a single NN input frame, which is a fixed value once network topology is determined. \(\Delta\) stands for extra resource overhead used for control logic and \(C_{\text{avg}}\) is defined in Eq.2 as average hardware cost per operation. Because \(C_{\text{avg}}\) in our estimation model is a ratio between required resource and the overall resource budget, \(C_{\text{avg}}\) implies resource folding factor in order to get all computations done with available resource. We migrate and apply this folding effect to timing and interpret it as folding of clock cycles in unit time so that throughput can be estimated. As shown in Table 1 from binary values to 32-bit floating point values, the \(C_{\text{avg}}\) is roughly getting higher due to the increasing hardware complexity except for the case where INT16 is more efficient than INT8 due to the explanation in Section 3.2. Meanwhile, according to Eq.3 higher \(C_{\text{avg}}\) brings down the throughput for the same network implementation on a specific device. This will be demonstrated in Section 4.1.

According to our observation in real systems, as resource usage of the target device increase, the models become more accurate. For concrete examples, we compare results from our estimation model to real implementation from Fraser et al [12]. The measured GOPs/s for their cnn(1/2) and cnn(1) models are 1856 and 7407. According to our estimation model, the estimated minimum performance for the corresponding models are 2051 and 8596, which are 35% and 16% difference. The discrepancy between estimated and measured performance could be due to the following factors: 1. Difference in clock frequency between estimated and measured models. 2. An underestimation of the control

---

1 This assumes that both networks have the same memory footprint for their parameters.
logic overhead when a small portion of the target device is used. 3. the model doesn’t take into account that the first layer has 8-bit pixel images as inputs.

4 Experimental Evaluation

We tested on 6 precision types: 1-bit binary values (Binary), fixed point representations with 2-bit (INT2), 4-bit (INT4), 8-bit (INT8), 16-bit (INT16) and single-precision floating point values (FP32). 2 bits are reserved for the integer part and rest for fractional part ($F/L$). The fully-connected NNs are tested on the MNIST dataset. CNNs are tested on CIFAR-10 [13] and ImageNet [14] datasets. All input images are expressed in 8-bit fixed point numbers.

We used Fully-Connected (FC) and CNN models in our experiments. FC is a reference network topology with 3 hidden layers with each containing 4096 hidden neurons fully connected to its proceeding layer. For CNN, the reference topology is the VGG-16 inspired model [15], which contains a succession of $(3 \times 3$ convolution, $3 \times 3$ convolution, $2 \times 2$ maxpool) layers repeated three times with 128-256-512 channels, followed by two fully-connected layers with 1024 neurons in each. For ImageNet tasks, we use AlexNet [14] as baseline model. In terms of activation function, all the precision options are trained with ReLU and hard-tanh and the best accuracy results are used to report the performance. Additionally, 5 values for scaling factor $s$ are applied to the reference networks in order to expand or shrink the reference topology in a specific ratio. The values are 0.03125, 0.0625, 0.125, 0.25, 0.5, 1. For example, all tested FC networks have the same number of hidden layers, but with $1024 \times s$ neurons correspondingly. Similarly to CNNs, scaling factors are multiplied with the number of filters in each conv layer, but they do not change the depth of the topology. For ImageNet tasks, smaller models provide unacceptably low accuracy, so we only report the results of 0.25, 0.5 and 1.

In this work, we use Xilinx Kintex UltraScale 115 as the target FPGA device. The working clock frequency is 250 MHz. In terms of metrics, throughput is measured in this work as frames per second and a frame is an image fed to a neural network. Hardware Cost is studied through computational resources and block ram (BRAM) usage. Since we are not competing for the best model accuracy, better classification results can be achieved if using other optimization techniques, which can be orthogonal to the training strategy used in this work. To make fair comparison, we train all experiments for each dataset/topology with the same hyper-parameters including number of epochs, learning rate decay strategy etc.

4.1 Experimental Results

The results shown in this section are based on our estimation models. Figure 3 and 4 show the trade-off curves for different dataset and network combinations. Each curve indicates a data representation for both weight and activation. Each marker on the curve shows the result for a network with a specific scaling factor. In Figure 3, the areas highlighted in red colour are emphasized in an attached zoom-in view in order to show more information about regions for high classification accuracy regions, which may deliver more insights that global trends cannot display.
MNIST on FC Layers From Figure 3, we can see that FP32 delivers the highest accuracy in 3 of its topology options and Binary provides best options in terms of hardware efficiency with a much higher accuracy drop (6.24%) compared to the best FP32 results. In general, INT2, FP32 and INT4 dominate the Pareto Frontier. From the

MNIST Classification on FC Layers

CIFAR-10 Classification on VGG Net

Fig. 3: Experimental Results for CIFAR10 and MNIST Classification

zoom-in views, a noticeable observation is that among solutions that give no higher than 1.2% classification error, which is the best achievable result for binary, INT2, INT4 and even FP32 can all provide more efficient solutions than Binary in terms of memory usage. The reason for this is that Binary requires a larger topology and more computation to achieve the same model accuracy. For example, only 4096 × 0.125 = 512 neurons are needed in each hidden layer for FP32 to achieve 1.02% error while 4096 neurons per hidden layer are needed for Binary to achieve a similar error of 1.2%. Required memory for Binary is 37.0 Mb and 29.8 Mb for FP32.

Noticeably, with a relatively small budget of BRAM smaller than 1Mb, only Binary, INT2 and INT4 are feasible options for hardware implementations while INT2 can achieve the highest accuracy (98.1%). If comparing the representability of Binary and FP32 by looking at the solutions with best accuracy for each, Binary requires only 27 Mb memory for an accuracy of 98.8% while FP32 needs 1.2 GB for only a 0.25% higher accuracy. Additionally, if setting the accuracy goal as 98% on MNIST task (red dotted line on the global figure), INT2 provides the most efficient option in terms of computational resource and memory usage. Meanwhile, Binary at least requires 6.2× more computational resources and 7.8× more memory compared to the optimal INT2 option. Besides, INT4 also provides more resource/memory efficient options than Binary.

Moreover, our low-precision training strategy allows memory saving when conducting inference, which achieves a very similar effect of network compression. We compare our results with several state-of-the-art compression works on the same dataset (MNIST)
Table 2: Memory Saving and Model Accuracy Comparison Between Our Work and RER [16], LRD [6], DK [17], HashNet [18], Q-CNN [8] and Q-CNN(EC) [8] on MNIST Classification Task with 784-1000-1000-1000-10 FC Networks

|         | Ours | RER  | LRD  | DK   | HashNet | Q-CNN | Q-CNN(EC) |
|---------|------|------|------|------|---------|-------|-----------|
| Mem. Saving |     |      |      |      |         |       |           |
| Binary  | 32x  |     |      |      |         |       |           |
| INT2    | 16x  |     |      |      |         |       |           |
| INT4    | 8x   | 1.25 | 1.16 | 1.13 | 1.13    | 1.24  | 1.16      |
| INT8    | 4x   | 1.13 | 1.13 | 1.13 | 1.13    | 1.26  | 1.16      |
| INT16   | 2x   | 1.13 | 1.13 | 1.13 | 1.13    | 1.22  | 1.16      |
| Error   | 1.5  | 1.25 | 1.16 | 1.13 | 1.13    | 1.34  | 1.19      |

Table 2 shows that without using any compression techniques, our INT4, INT8, INT16 results can achieve higher model accuracy than the other methods [8]. Meanwhile, INT2 and Binary achieve higher memory saving rate and still keep competitive accuracy. As highlighted in red colour, our results either achieve best compression rate or highest accuracy on the exactly same network topology compared to the other state-of-art results.

CIFAR10 on VGGNet Second row in Figure 3 shows trade-offs for CIFAR10 classification with VGGNets. Noticeably, INT4, INT8 and INT16 provide very close best accuracy and all higher than FP32 alternative. The rounding noise introduced in parameter quantization may help to improve the classification accuracy in this particular case. Similarly, Binary provides the most efficient solution among all precision types but with much higher error (54%). INT2 and INT4 provide high accuracy options with relatively higher throughput, lower resource and memory usage, as shown in the zoom-in views. They are considered as optimal parameter data type as they contribute most of the Pareto-efficient options. FP32 options are not advantageous on either model accuracy or hardware cost because of the high complexity. As shown by the red dotted lines in the zoom-in views, for the range where accuracy is higher than 90%, it is INT2, rather than Binary, that provides the most efficient options in terms of computational resource and memory usage. Specifically, for 91% accuracy, INT2 provide 13.9K FPS which is 26x higher than FP32, 6x higher than INT8 and INT16 alternatives. On the other hand, Binary requires a larger topology to achieve the same level of accuracy compared to the other alternatives by 1 scaling factor. But it presents only 1.7% accuracy degradation with 32x less memory and 63.8x less computational resource requirement if we compare the most accurate options provided by Binary and FP32.

ImageNet on AlexNet ImageNet tasks show clearer relative positions among curves in Fig. 4. This can be caused by the higher complexity in the classification tasks compared to MNIST and CIFAR10. Noticeably, Binary and INT2 solutions cannot achieve comparable model accuracy to other data types as they are in MNIST and CIFAR10 tasks. As shown in all figures, there is an accuracy gap between these two types and the others. Some very recent works like [19] try to target on this accuracy gap by optimizing the quantization function for parameters with extremely low bitwidth. This topic is very interesting and definitely deserves more efforts, but it is beyond the scope of this paper. In particular, INT4, compared to FP32, provide solution with 8x memory saving and 11.8x higher throughput with only less than 1% accuracy drop. Similarly, INT8 can provide a 4.3x higher throughput solution with no accuracy loss compared to FP32. Therefore, FP32 again loses its advantage on either model accuracy or hardware.

---

[2] The reason that the particular 784-1000×3-10 structure is selected in Table 2 is that it is the only structure that is reported in all mentioned works. We compare different methods on the same structure in the same classification task for fair comparisons on memory and accuracy.
efficiency. In general, INT4, INT8 and INT16 and FP32 present accuracy with negligible difference. However, INT4 has the best trade-off due its less memory and computational resource requirements as well as higher system throughput possibly provided.

Trade-offs to Top-1 Error on ImageNet

Trade-offs to Top-5 Error on ImageNet

Fig. 4: Experimental Results for ImageNet Classification on AlexNet

5 Summary and Conclusion

In this work, we firstly introduce our quantization training strategy that allows training NNs with arbitrary parameter precision. Then, we propose our hardware cost and throughput estimation models. Finally, we conduct our experiments in the exploration space consist of 6 different data types, 3 different NN models and 3 different benchmarks. We found that Binary does not necessarily provide hardware solutions with highest efficiency due to larger amount of parameters required for Binary to achieve the same level of model accuracy with its high precision alternatives. Within our studied cases, INT2 and INT4 generally provide better trade-offs in small image classification tasks, MNIST and CIFAR10, while INT4 provide the best trade-offs among all other types in ImageNet tasks. More insightful observations have been pointed out in Section 4 which hopefully can be helpful to reduced-precision NN system design on reconfigurable hardware.

Acknowledgments

The authors from Imperial College London would like to acknowledge the support of UKs research council (RCUK) with the following grants: EP/K034448, P010040 and N031768. The authors from The University of Sydney acknowledge support from the Australian Research Council Linkage Project LP130101034.
Bibliography

[1] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. H. W. Leong, M. Jahre, and K. A. Vissers, “FINN: A framework for fast, scalable binarized neural network inference,” CoRR, vol. abs/1612.07119, 2016.
[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1,” CoRR, vol. abs/1602.02830, 2016.
[3] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks,” in ECCV, 2016.
[4] W. Sung, S. Shin, and K. Hwang, “Resiliency of deep neural networks under quantization,” CoRR, vol. abs/1511.06488, 2015.
[5] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, “DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” CoRR, vol. abs/1606.06160, 2016.
[6] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, “Predicting parameters in deep learning,” CoRR, vol. abs/1306.0543, 2013.
[7] K. Hwang and W. Sung, “Fixed-point feedforward deep neural network design using weights +1, 0, and -1,” in Proc. IEEE ICASSP, pp. 1–6, IEEE, 2014.
[8] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, “Quantized convolutional neural networks for mobile devices,” CoRR, vol. abs/1512.06473, 2015.
[9] M. Courbariaux, Y. Bengio, and J. David, “Low precision arithmetic for deep learning,” CoRR, vol. abs/1412.7024, 2014.
[10] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” CoRR, vol. abs/1412.6980, 2014.
[11] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in In Proceedings of AISTATS10, 2010.
[12] N. J. F. et. al., “Scaling binarized neural networks on reconfigurable logic,” CoRR, vol. abs/1701.03400, 2017.
[13] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Technical Report, 2009.
[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Proc. NIPS, pp. 1097–1105, 2012.
[15] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” CoRR, vol. abs/1409.1556, 2014.
[16] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, “High-performance neural networks for visual object classification,” CoRR, vol. abs/1102.0183, 2011.
[17] J. D. G. Hinton and O. Vinyals, “Distilling the knowledge in a neural network,” CoRR, vol. abs/1503.02531, 2015.
[18] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, “Compressing neural networks with the hashing trick,” CoRR, vol. abs/1504.04788, 2015.
[19] Z. Cai, X. He, J. Sun, and N. Vasconcelos, “Deep learning with low precision by half-wave gaussian quantization,” CoRR, vol. abs/1702.00953, 2017.