Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks

Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi

***Abstract*— Convolutional neural networks (CNNs) have led to remarkable progress in a number of key pattern recognition tasks, such as visual scene understanding and speech recognition, that potentially enable numerous applications. Consequently, there is a significant need to deploy trained CNNs to resource-constrained embedded systems. Inference using pretrained modern deep CNNs, however, requires significant system resources, including computation, energy, and memory space. To enable efficient implementation of trained CNNs, a viable approach is to approximate the network with an implementation-friendly model with only negligible degradation in classification accuracy. We present Ristretto, a CNN approximation framework that enables empirical investigation of the tradeoff between various number representation and word width choices and the classification accuracy of the model. Specifically, Ristretto analyzes a given CNN with respect to numerical range required to represent weights, activations, and intermediate results of convolutional and fully connected layers, and subsequently, it simulates the impact of reduced word width or lower precision arithmetic operators on the model accuracy. Moreover, Ristretto can fine-tune a quantized network to further improve its classification accuracy under a given number representation and word width configuration. Given a maximum classification accuracy degradation tolerance of 1%, we use Ristretto to demonstrate that three ImageNet networks can be condensed to use 8-bit dynamic fixed point for network weights and activations. Ristretto is available as a popular open-source software project[1] and has already been viewed over 1 000 times on Github as of the submission of this brief.**

***Index Terms*— Arithmetic precision, convolutional neural network (CNN), efficient inference, number representation.**

I. Introduction

Convolutional neural networks (CNNs) have enabled significant advancements in a number of important pattern recognition tasks, including large-scale image classification [1]–[3]. Server-grade graphics processing units (GPUs) offer an excellent platform for both training and inference using deep CNNs in the cloud. However, utilization of such GPUs in many embedded systems is impractical due to their excessive energy dissipation. As a result, there is a genuine need to develop resource-efficient techniques, e.g., custom hardware accelerators [4], for the deployment of deep CNNs that are already trained.[2]

Weights with smaller bit-width are generally favorable for implementation due to their improved resource requirement. In custom hardware accelerators, for example, the lower bit-width directly translates to improvements in area, clock frequency, and energy dissipation of the circuitry. In software implementations, smaller permissible word width may translate to resource savings, e.g., higher memory bandwidth or less energy dissipation, if the weights can be represented in a data type that has smaller bit-width. The savings may be even more pronounced if the underlying architecture utilizes different functional units based on the operand bit-width.

In addition to bit-width, a closely related important question, particularly of value in the context of hardware accelerators, centers around the best number representation format. Model training usually assumes that weights are specified and processed in the IEEE standard floating point (FP) format. FP arithmetic is far more resource intensive compared with its fixed point or dynamic fixed point counterparts. Thus, simplification of both number representation format as well as word width can potentially lead to resource savings during inference.

The benefit of reduced precision and simpler number representation, referred to as model approximation in this brief, is intuitive from the resource-efficiency viewpoint. While it is generally understood that neural networks are robust to *reasonable* quantization of weights and biases [5], there used to exist no way for embedded system designers to easily characterize the tradeoff. Thus, they faced the problem of arriving at the best approximation parameters that do not excessively degrade the model's classification accuracy. Our goal was to address this problem for practitioners.

We present Ristretto, a popular open-source software framework for automated resource-efficient approximation of CNNs. Ristretto enables designers to investigate the impact of various number representations and arithmetic precision configurations on classification accuracy. Consequently, Ristretto helps designers to explore the tradeoff and to choose a suitable approximation of the model. Ristretto can also fine-tune the compressed network to improve its classification accuracy under specified number representation format and arithmetic precision. Given a trained CNN and tolerable degradation in classification accuracy, Ristretto yields not only the best format and word width to represent weights and activations but also the fine-tuned values of the weights in the selected form.

Iterative exploration of model approximation vis-à-vis its accuracy is key to enable useful comparison between various representations for practitioners. Hence, run time of Ristretto was a key consideration in this brief. Consequently, we made a number of design choices, such as fine-tuning network weights with default forward path that are further discussed in Section V-B. In addition, the magnitude of implementation resource savings, such as energy dissipation, computational performance, or circuit area, is heavily dependent on the choice of execution platform, its resource types, and inference implementation. As such, we restrict our discussion to the characterization of classification accuracy under model approximation. Regardless of the target platform, however, Ristretto is useful in arriving at the best implementation for the target platform subject to meeting classification accuracy requirements.

The remainder of this brief is organized as follows. Section II summarizes the existing work in CNN approximation. In Section III, we describe the neural network forward path using low-precision

Manuscript received August 12, 2016; revised February 28, 2017 and September 15, 2017; accepted February 10, 2018. This work was supported by the U.S. National Science Foundation under Grant CCF-1346812. *(Corresponding author: Philipp Gysel.)*

P. Gysel, M. Motamedi, and S. Ghiasi are with the Laboratory for Embedded and Programmable Systems, Electrical and Computer Engineering Department, University of California at Davis, Davis, CA 95616 USA (e-mail: pmgysel@ucdavis.edu).

J. Pimentel is with the VLSI Computation Laboratory, Electrical and Computer Engineering Department, University of California at Davis, Davis, CA 95616 USA (e-mail: jjpimentel@ucdavis.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2018.2808319

[1] https://github.com/pmgysel/caffe

[2] We interchangeably use the terms deployment or inference using a trained CNN to imply forward evaluation of the model with known input data and parameters. The implementation resource requirement, depending on the target platform and technology, may refer to energy dissipation, computational performance, and/or circuit area.

arithmetic. In Section IV, we detail our three approximation strategies. Section V introduces our open-source approximation framework for neural networks. We use this framework in Section VI to measure the impact of our proposed approximations on neural network accuracy. Finally, in Section VII, we compare our different approximation strategies in terms of theoretical and empirical quantization errors.

II. Related Work

Courbariaux *et al*. [6] represent weights and activations of maxout networks with dynamic fixed point format. They show that 8-bit numbers are sufficient to train networks on 10-class data sets. The other work reduces the word width even further to just one bit for weights and activations [7]. While this number format works well on 10-class data sets, training larger networks with binary weights and activations has turned out to be challenging. A recent work termed quantized neural networks (QNN) [8] trains networks in low precision for the 1000-class ImageNet data set. Their AlexNet [2] approximation uses 1-bit weights and 2-bit activations and achieves a top-5 accuracy of 73.67%. For the more optimized GoogleNet network [3], they propose using 4 bits for weights and activations, achieving an impressive 83.4% top-5 accuracy. Another work termed trained ternary quantization (TTQ) [9] uses ternary weights to remove multiplications from the forward pass. Their approach starts with a model trained in full precision, then converts the weights to 2 bits, and finally fine-tunes the resulting network.

The aforementioned approaches use a regular data access pattern. In contrast, the deep compression pipeline [10] exploits sparsity in the network weights. The authors observe that a large number of the trained weights are close to zero, and remove them from the network. To further increase the compression factor, weight values are shared, and a lossless data compression scheme is applied. As a result, the VGG-16 network is compressed by 49 times with no loss in accuracy.

When compressing a CNN, the question arises whether to train a model in compressed format from scratch or apply the compression to a trained network. The deep compression pipeline [10] uses the first approach and starts with a trained network. In contrast, other works for fixed point networks [6] and binary networks [7] show good results for training CNNs in compressed format from scratch. As trained models are already available, the first approach is more attractive to many practitioners and embedded system developers, and thus, it is the focus of this brief.

III. CNNs With Reduced Word Width

The complexity of deep CNNs can be split into two parts. Convolutional layers impose the majority of arithmetic operations, while fully connected layers typically contain a large part of the network weights. For AlexNet, over 90% of arithmetic operations are in convolutional layers, and over 90% of network weights are in fully connected layers. Newer CNN architectures [11] avoid large fully connected layers to decrease networks size. In this case, the convolutional layers become the bottleneck both in terms of computation and memory. Our approximation framework, therefore, concentrates on these two layer types.

Our framework simulates reduced word width arithmetic. For this purpose, we compress the number format in convolutional and fully connected layers. These two layer types require the same arithmetic operations, namely a series of multiply–accumulate (MAC) operations. The simulated data flow graph is shown in Fig. 1. The layer activations are multiplied with the network weights, and the results are accumulated to form the output. Notice that the word width of layer inputs, layer outputs, and network weights is reduced. We use a 32-bit number format for accumulation. In order to simulate this data path, our framework first quantizes the layer inputs and weights, and then dequantizes the numbers back to FP format. Next, we use a matrix multiplication routine to perform the layer's forward path. Since our reduced word width data path uses 32-bit accumulators, it is impossible for the accumulator to overflow, even for very large layers. This allows us to use existing FP matrix multiplication routines for the forward path, which speeds up the simulation of approximated networks drastically.

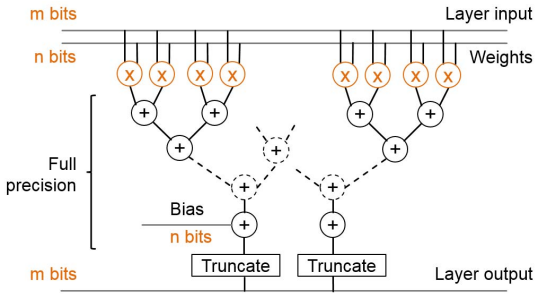

Fig. 1. Simulated data path for convolutional and fully connected layers. The layer inputs, outputs, and weights are discrete values ($m, n < 32$).

IV. Approximation Strategies

In this section, we propose three different strategies for approximating trained 32-bit FP CNNs. Each strategy uses a low-precision number format to represent layer activations and network weights.

A. Dynamic Fixed Point

In this section, we discuss the quantization of a CNN to dynamic fixed point [6]. We extend the standard fixed point format to dynamic fixed point and show why it performs better at low word widths compared with the traditional fixed point format.

In dynamic fixed point format, each number is represented as follows:

$$(-1)^s \cdot 2^{-FL} \sum_{i=0}^{B-2} 2^i \cdot x_i \tag{1}$$

where B denotes the word width, s denotes the sign bit, FL denotes the fractional length, and x denotes the mantissa value. Numbers in a similar dynamic range are grouped together and share a common fractional length FL.

The challenge of traditional fixed point approximation is the wide dynamic range of values in CNNs. As a case in point, for the AlexNet [2] network, 97% of the network weights are in the range $[2^{-11}$–$2^{-3}]$, while 99% of the layer outputs are in the range $[2^{-2}$–$2^{8}]$. On average, activations are 2^9 times larger than network weights. Fixed point has only limited capability to cover this large dynamic range. Dynamic fixed point overcomes this problem by adapting the number format according to the range of different network segments. We split each network layer into three groups: one for the layer inputs, one for the layer weights, and one for layer outputs. FL is chosen based upon the dynamic range of each number group.

B. Minifloat

In this section, we detail our approximation strategy using minifloat, i.e., FP numbers with less than 32 bits. We approximate layer inputs, weights, and layer outputs with one common minifloat format. While it is well known that CNNs perform well in half precision mode [12], we are—to the best of our knowledge—the first to approximate both network weights and activations by 8-bit minifloat. Our minifloat format follows the IEEE-754 standard, with the

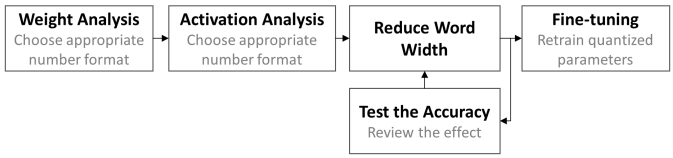

Fig. 2. Network approximation flow with Ristretto.

following modifications. First, any exponent and mantissa bit-width combination is allowed. Second, to reduce overhead, the following are not supported: exception handling, not a number, ±infinity, denormalized values, and alternative rounding modes.

C. Multiplier-Free Arithmetic

In this section, we explain an approximation, which replaces all multiplications with bit shifts. We approximate convolutional and fully connected layers according to the following equation:

$$z_i = \sum_j [x_j \cdot w_j] + b_i \approx \sum_j [x_j << \text{round}(\log_2(w_j))] + b_i. \tag{2}$$

The first part of (2) shows the necessary operations in a 32-bit FP network. The layer inputs x_j are multiplied with layer weights w_j and the accumulation yields z_i. To simplify this discussion, we assume that the input data have been rearranged such that output $z_i = \mathbf{w}^T \cdot \mathbf{x}$. In order to switch to multiplier-free arithmetic, we first approximate weights by their closest integer-power-of-two number. Now, output z_i can be approximated through a series of shift-and-accumulates. Notice that the second part of (2) relies on the integer-power-of-two exponents $e_j = \text{round}(\log_2(w_j)) \in \mathbb{Z}$, not the original weights. To simplify the second part of (2), we assumed strictly positive weights.

We choose to represent activations in 8-bit dynamic fixed point format. The activations are shifted according to the exponents e_j and summed up in 32-bit fixed point format.

When we first submitted this brief for review, we were—to the best of our knowledge—the first to consider converting deep FP networks to use integer-power-of-two weights. In the meantime, another work [13] has been published, which uses exactly this approach, combined with an improved retraining procedure.

V. Ristretto: Approximation Framework

In this section, we present Ristretto, our approximation framework for CNNs. Our compressor reduces the word width of weights and activations while preserving the CNN's ability to extract image features. Ristretto augments Caffe [14], a widely used deep learning framework, by introducing quantized layer definitions.

Ristretto has the following strengths.

1) *Automation:* Automatic quantization of any given CNN.
2) *Flexibility:* Various quantization schemes are supported.
3) *Accuracy:* Ristretto fine-tunes trimmed networks.
4) *Speed:* Ristretto leverages optimized CUDA routines and runs on compatible NVIDIA GPUs.

A. Quantization Flow

Ristretto can condense any 32-bit FP CNN to either dynamic fixed point, minifloat, or multiplier-free arithmetic. Ristretto's quantization flow has four steps (see Fig. 2). In the first step, the trained model is analyzed to determine a suitable number format for network weights. For dynamic fixed point and minifloat approximation, we use enough integer bits and exponent bits, respectively, to avoid saturation of large values. In the second step, we run several image batches through the network's forward path. In each layer, we choose an appropriate number format for the input and output values, using the same strategy as in the previous step. For the third step, Ristretto performs a search to find the smallest possible word width, which still satisfies the user-defined error margin. For this purpose, the quantized network is benchmarked on the validation data set. In a last step, the quantized network is fine-tuned using the training data set.

B. Fine-Tuning

During the fine-tuning procedure, the network learns how to classify images with discrete-valued weights $\mathbf{w}'$. During retraining, we will calculate small weight updates $\Delta\mathbf{w}$. Since these small weight updates may be below the quantization step size of the discrete weights, we also keep a set of full-precision weights $\mathbf{w}$. During both the forward and backward propagation, we use the quantized weights $\mathbf{w}'$ and the full-precision activations $\mathbf{z_l}$, as described in our previous work [15]. This enables Ristretto to analytically compute the error gradient with respect to each weight. Note that scoring of the network is always done with quantized activations $\mathbf{z}'_\mathbf{l}$. That is, whenever we report a network accuracy, no matter if it is before or after fine-tuning, we quantize both the weights and the activations.

The fact that we do not quantize activations during fine-tuning puts a limitation on our framework. We do not teach the network how to deal with quantized activations. As a result, we have to avoid reducing the bit-width for activations too aggressively. We argue that this is a reasonable approach, given the scope of this brief. We carefully lower the bit-width and make sure that a short fine-tuning stage is enough to get back close to the original network accuracy.

As mentioned in Section III, Ristretto uses FP math to simulate reduced precision arithmetic. Our framework can, therefore, leverage highly optimized FP matrix multiplication routines for both the forward and backward propagation. This enabled fast fine-tuning of quantized networks.

VI. Approximation Results

In this section, we empirically evaluate the impact of reduced word width and the choice of number format on network accuracy. We apply our network approximation to two tasks: image classification and semantic segmentation. In all experiments, we use CNNs from the Caffe Model Zoo, which were trained in full-precision format. We use Ristretto to quantize the networks to smaller word width according to the strategies explained in Section IV and fine-tune the condensed models.

A. Image Classification

1) Baseline Networks: Initially, we tested our approximation strategies on the 10-class LeNet network. For all approximation strategies, the FP network can be compressed to 8-bit activations and parameters, without any drop in accuracy. We, therefore, consider a more challenging task to see how the approximation strategies affect deeper networks. We consider the ImageNet data set, which has 1000 classes ranging from animal and plant species to man-made objects. First, we quantize AlexNet [2], the winner of ILSVRC 2012 [1]. The second network is GoogleNet [3], which won the same competition in 2014 in image classification. This network has 144 layers and seven million parameters. Compared with AlexNet, this network requires fewer arithmetic operations while achieving a higher prediction accuracy. It is, therefore, more optimized and potentially harder to approximate by low-precision arithmetic. Third, we approximate SqueezeNet [11], a CNN with the accuracy of AlexNet but 50 times fewer parameters. This network has less than 1.5 million parameters, which renders it suitable for deployment in memory constraint mobile devices. We evaluate each approximated network in center crop mode on the 50-K ImageNet validation data set.

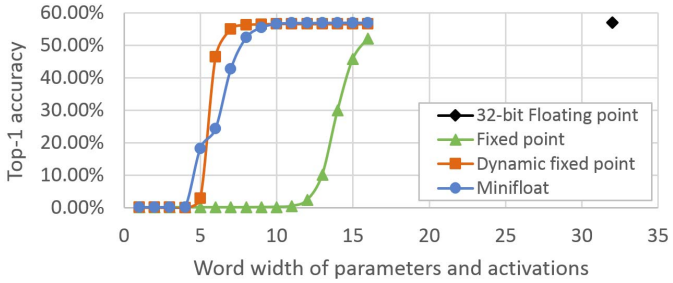

Fig. 3. Approximation of AlexNet without fine-tuning.

TABLE I
CLASSIFICATION ACCURACY OF 8-BIT CNNS ON THE IMAGENET 1000 DATA SET

Network (32-bit float)	Dynamic fixed point	Minifloat	Multiplier-free arithmetic
Top-5 accuracy			
AlexNet (80.09%)	79.50%	78.23%	78.25%
GoogleNet (89.16%)	88.63%	87.69%	81.38%
SqueezeNet (80.37%)	79.99%	78.28%	67.37%
Top-1 accuracy			
AlexNet (56.90%)	56.14%	52.26%	53.57%
GoogleNet (68.93%)	68.37%	64.02%	57.63%
SqueezeNet (57.68%)	57.21%	54.80%	41.60%

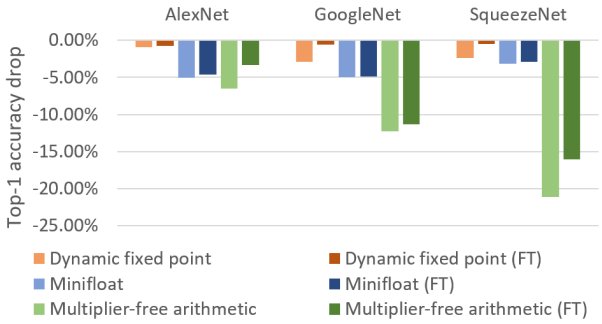

Fig. 4. Top-1 accuracy drop of 8-bit networks. We show the accuracy difference between the approximated and the 32-bit FP network. Fine-tuned networks are denoted with *FT*. Our dynamic fixed point networks achieve the best performance and are within 1% of the original network.

2) Quantization to Different Word Widths: In a first step, we use our framework to see how far we can reduce the word width while still maintaining reasonably high network accuracy. Fig. 3 shows the effect of quantizing AlexNet to different word widths. No fine-tuning was used for this experiment. The classification accuracy of a 16-bit static fixed point network is comparable to the 32-bit FP baseline, but the accuracy drops sharply for lower word widths. Dynamic fixed point performs better for small word widths; this approximation compares well to the baseline for as low as 8 bits. Minifloat approximation performs slightly worse than the dynamic fixed point approximation.

3) Comparison of 8-Bit Approximations: In a next step, we fix the word width to 8 bits and compare our proposed approximations for the three baseline networks. We report our experimental results in two forms. First, we show the achieved network accuracies after fine-tuning in Table I. Second, we show the accuracy difference between the approximated networks and the original FP network in Fig. 4.

Dynamic fixed point shows the best accuracy among the different approximation strategies. We can approximate the three ImageNet networks using 8-bit dynamic fixed point, with an absolute top-1 accuracy drop below 1%. This finding could be used to create a dedicated hardware accelerator, which only requires 8-bit fixed point multipliers for convolutional and fully connected lasers. Alternatively, this knowledge could be leveraged to write a CNN compiler, which only uses 8-bit variables for weights and activations.

TABLE II
SEMANTIC SEGMENTATION WITH REDUCED WORD WIDTH

Word width	Mean IU
32-bit floating point	65.51%
8-bit dynamic fixed point	64.21%

Minifloat approximation proves to be less effective than the dynamic fixed point approximation. When switching from 32-bit float to 8-bit minifloat, the top-1 accuracy drops by an absolute value between 2.88% and 4.91%. The reason for the inferior performance of minifloat lies in the fact that both mantissa and exponent need to share the available word width. For all ImageNet networks, we need five exponent bits to represent layer activations, and thus, only 2 bits can be used for the mantissa.

Multiplier-free arithmetic imposes the most sever quantization errors. AlexNet still performs decently when replacing multiplications by bit shifts, but GoogleNet and SqueezeNet are affected drastically. AlexNet shows better results than GoogleNet and SqueezeNet, since it has more redundancy in its architecture. As multiplier-free arithmetic incurs a significantly higher accuracy drop than the other approximation strategies, we performed additional experiments with different word widths. Interestingly, enlarging the bit-width for weights does not increase the accuracy. On the contrary, the bit-width can actually be reduced further. For AlexNet, a 4-bit weight network has a top-1 accuracy of 53.15%, while an 8-bit weight network has an accuracy of 53.57%.

B. Semantic Segmentation

We apply our CNN approximation framework to the task of semantic segmentation. In this challenge, the neural network has to assign a label to each pixel of the input image. We consider the fully convolutional network architecture by Long *et al*. [16]. At the time of the publication by Long *et al*., their network achieved the state-of-the-art accuracy on the PASCAL VOC 2011 data set [17]. We approximate their best performing model, the FCN-8s architecture, with 8-bit dynamic fixed point. For fine-tuning of the dynamic fixed point model, we use the PASCAL training data set only. Furthermore, we use the same 736 validation images as Long *et al*. We measure the network's accuracy in mean intersection over union. After fine-tuning, our dynamic fixed point network is within 1.3% of the full-precision model (see Table II). Note that our resulting dynamic fixed point model requires no 32-bit FP multiplications, as we approximate both convolutional and deconvolutional layers.

C. Impact on Hardware Implementations

In this section, we investigate the impact of our proposed number formats on a hardware implementation. Convolutional and fully connected layers require a series of MAC operations. Here, we assess the impact of replacing 32-bit FP MAC with our proposed low-precision number formats. We implemented the multiplication and accumulation units for the three different approximation strategies in Verilog RTL and synthesized them using Synopsys dc Compiler in 65-nm CMOS at 1.3 V and a clock frequency of 1.2 GHz. The power was gathered by scaling the results of synthesis by the data from AsAP2 [18].

The synthesis results are shown in Table III. Replacing 32-bit FP by 8-bit fixed point leads to a speedup of over 3 times in performing the MAC operation and simultaneously decreases its power consumption by over 30 times. These results underscore the importance of using

TABLE III
QUALITY METRICS FOR MAC UNITS

Number format	Timing in ns	Power in mW
32-bit float	2.49	54.73
8-bit minifloat	2.25	7.85
8-bit fixed point	0.75	1.78
8/4-bit barrel shifter	0.75	1.02

Numbers show the timing and power of one MAC unit (smaller means better). All of the MAC units use 8-bit multipliers and 32-bit accumulators. The barrel shifter uses an 8-bit input and 15-bit output.

TABLE IV
COMPARISON TO OTHER WORK: COMPRESSION OF IMAGENET 1000 NETWORKS

Author and CNN	Bit-width weights / activations	Top-5 accuracy	Re-training epochs
QNN [8]: GoogleNet	4 / 4	83.4%	N/A
TTQ [9]: ResNet-18B	2 / 32	87.2%	+ 64
Ours: GoogleNet	8 / 8	88.63%	+ 1.8

a suitable low-precision number format to represent weights and activations.

Note that both 32-bit float and 8-bit minifloat need three clock cycles to carry out one MAC operation, while dynamic fixed point and multiplier-free arithmetic only require one cycle. Consistent with our approximation assumption, the implemented MAC units for float and minifloat use a 32-bit FP adder, which takes significant amount of time. Finally, note that the barrel shifter uses 8-bit activations and 4-bit weights. Representing weights in either 4 bits or 8 bits, under multiplier-free representation, does not change network accuracy in a significant way.

D. Comparison to Other Works

In this section, we compare our approximation results on the ImageNet data set with previous works. We compare our dynamic fixed point GoogleNet network with the 4-bit QNN [8] GoogleNet and the ternary weight TTQ [9] ResNet-18B (see Table IV). Both QNN and TTQ achieved the state-of-the-art results at the time of their respective publication. Among these three networks, our 8-bit GoogleNet achieves the highest top-5 accuracy of 88.63%. QNN gets a top-5 accuracy of 83.4% and TTQ gets 87.2%. Both the other works achieve impressive network compression; however, this comes at the cost of a considerably longer training or fine-tuning phase. Our Ristretto framework cannot lower the bit-width as drastically as the other works, but allows to quantize and fine-tune a network in a brief time frame. In contrast, QNN requires training from scratch, which typically takes 50 epochs or more on ImageNet, and TTQ requires a fine-tuning stage of 64 epochs. Our open-source Ristretto, therefore, offers the capability of comparing different approximation strategies in a reasonably short time frame while achieving impressive network accuracy.

VII. ANALYSIS OF DIFFERENT APPROXIMATION STRATEGIES

A. Theoretical Analysis

During the quantization of a full-precision network, quantization errors are introduced into the network's forward path, which can lead to a loss in classification accuracy. The magnitude of the error depends on the approximation strategy chosen. In this section, we use a parametric analysis to compute the theoretical maximum quantization error when quantizing one full-precision number. We will compute this error for each of the proposed low-precision formats.

Note that we use the same number format properties that Ristretto uses for approximation of neural networks. In particular, Ristretto avoids saturation of a given number distribution, as explained in Section V-A.

TABLE V
MAXIMUM QUANTIZATION ERROR

Dynamic Fixed Point	Minifloat	Multiplier-free arithmetic
$\frac{x}{2^B}$	$\frac{x}{2^{B-E+1}}$	$\frac{x}{4}$

Let us assume that the number to be quantized lies in a fixed range, and we quantize the number using B bits. We now compute the upper bound of the quantization error for different low-precision number formats. To keep our analysis simple, we assume that the number distribution in question is in the symmetric interval $[-x, +x]$, where x is some integer-power-of-two.

1) Dynamic Fixed Point: We choose the format, which can cover the whole dynamic range. We have 2^B symbols to represent a range of $2x$; thus, the maximum quantization error is $x/2^B$.

2) Minifloat: Each minifloat number is represented by E exponent bits, M mantissa bits, and one sign bit, so the total number of bits is $B = E + M + 1$. In minifloat number format, small values can be represented with the finest granularity, whereas for large values, the quantization step size increases. Therefore, the largest possible quantization error occurs for large values. The value of $n_1 = x$ can be represented precisely in minifloat, as it is an integer power of 2. To get the largest quantization step size of minifloat, we have to find the second largest value in the range $[-x, +x]$, which is as close to x as possible. This second largest value n_2 has a mantissa of all ones and an exponent of $e_2 = \log_2(x) - 1$. The IEEE standard specifies that the first mantissa bit before the radix point is stored implicitly, so all normalized mantissa values are between 1 and 2. In the case of n_2, the mantissa value is $m_2 = 2 - 2^{-M}$, and therefore, the second largest minifloat number is $n_2 = x/2 \cdot (2 - 2^{-M})$. Thus, the largest possible quantization error is $\Delta_{\max} = (n_1 - n_2)/2 = x/2^{M+2} = x/2^{B-E+1}$.

3) Multiplier-Free Arithmetic: We can represent x, and the next smaller symbol is $x/2$. Thus, the maximum quantization error is $x/4$. Note that this analysis pertains to only the weights. The activations are represented with dynamic fixed point.

4) Summary: We summarize the maximum quantization error in Table V. Under the reasonable assumption that $B \geq 2$, $M \geq 1$, and $E \geq 1$, the maximum quantization error of dynamic fixed point is not more than minifloat, which is not more than multiplier-free arithmetic. Dynamic fixed point is a similar number format as minifloat, with the significant difference that the exponent is stored implicitly in dynamic fixed point, but explicitly for minifloat. Thus, minifloat has fewer bits for the mantisssa and larger quantization error for large values. The maximum quantization error of multiplier-free arithmetic does not depend on the bit-width, which implies that this approximation will always have a large maximum quantization error, even for large bit-widths. To sum up, dynamic fixed point has the smallest maximum error among all number formats, which aligns with our experimental results from Section VI-A.

B. Experimental Analysis

In this section, we empirically compare the different approximation strategies. We analyze the average quantization error in weights of AlexNet as well as the quantization error in the activations of GoogleNet. A neural network's ability to extract features from images depends on its learned filters. As we quantize the network parameters, those filters incur a quantization error. To quantify this quantization error, we calculated the signal-to-quantization noise ratio (SQNR) of the quantized parameters $\mathbf{w}'$, relative to the original FP parameters $\mathbf{w}$. In Fig. 5, we show the SQNR of the quantized 8-bit parameters of AlexNet (higher SQNR is better). Averaged

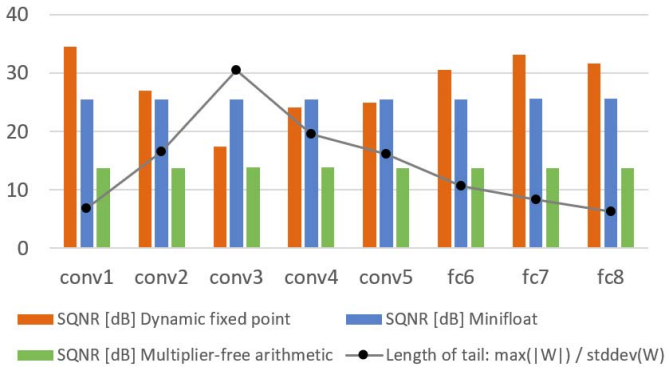

Fig. 5. Signal-to-quantization noise of 8-bit weights in AlexNet. In each layer, we show the SQNR for the three different approximation strategies (higher is better). Moreover, we show the tail length of the weights $\mathbf{w}$ from the original FP network.

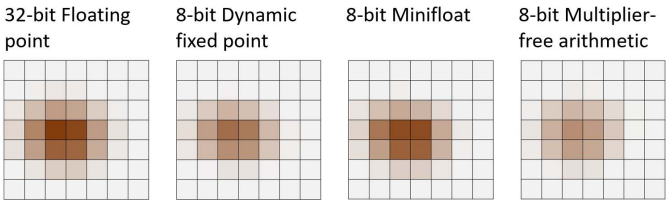

Fig. 6. Feature map visualization for the last concatenation layer of GoogleNet. We show the activations for the original network and for the quantized networks. High activation values are shown in dark red, and activations which do not fire are white.

over all layers, the dynamic fixed point network has the highest SQNR, while the multiplier-free network has the lowest SQNR. These findings support our results from Section VI-A, where dynamic fixed point achieved the highest network accuracy and multiplier-free arithmetic achieved the lowest accuracy. For the fixed point network, the quantization error varies according to a given layer's number distribution (see Fig. 5). Since we try to cover the whole dynamic range with our fixed point format, the number distributions with long tails are harder to approximate. In contrast, minifloat approximation and multiplier-free arithmetic introduce a similar quantization error in all network layers.

Given the quantization error introduced in each network layer, the question arises whether the approximated networks still have the same ability to extract features from the input image. As a case in point, we consider the GoogleNet architecture and visualize the activations in the last concatenation layer. We ran a random image through the FP network and searched for the feature map with the highest activations. In Fig. 6, we plot this 7×7 feature map for both the full-precision network and the quantized networks. We can see that this feature map has a similar pattern for all networks. This proves that despite the quantization error introduced in the previous 139 network layers, the approximated networks still have the same ability to extract features and classify images. We can also see that the activations for multiplier-free arithmetic differ the most from the original network. Multiplier-free arithmetic uses a logarithmic quantization for parameters, which introduces relatively large quantization errors for high numbers. As shown in the previous work [10], high parameter values are significantly more important than low values. It is, therefore, understandable that multiplier-free arithmetic introduces the largest error in network activations. As a side note, we also performed experiments where we calculated the SQNR of the activations in the last concatenation layer. The results revealed that minifloat approximation yields the lowest SQNR for activations. Interestingly, this does not directly translate into network accuracy, as dynamic fixed point produces better predictions.

VIII. Conclusion

We present Ristretto, a CNN approximation framework that enables an empirical investigation of the tradeoff between distinct number representations and word width choices and the classification accuracy of the model. By applying our approximation to three ImageNet networks, we demonstrate that 8-bit dynamic fixed point is well suited for large-scale image classification. Ristretto is available as a popular open-source software project.

Acknowledgment

The authors would like to thank NVIDIA for GPU donation. They would also like to thank T. O'Neill for his advice in editing this brief.

References

[1] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.

[3] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[4] M. Motamedi, P. Gysel, and S. Ghiasi, "PLACID: A platform for FPGA-based accelerator creation for DCNNs," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 13, no. 4, p. 62, 2017.

[5] Z. Du, A. Lingamneni, Y. Chen, K. V. Palem, O. Temam, and C. Wu, "Leveraging the error resilience of neural networks for designing highly energy efficient accelerators," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 8, pp. 1223–1235, Aug. 2015.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. (2014). "Training deep neural networks with low precision multiplications." [Online]. Available: https://arxiv.org/abs/1412.7024

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. (2016). "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1." [Online]. Available: https://arxiv.org/abs/1602.02830

[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. (2016). "Quantized neural networks: Training neural networks with low precision weights and activations." [Online]. Available: https://arxiv.org/abs/1609.07061

[9] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[11] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. (2016). "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size." [Online]. Available: https://arxiv.org/abs/1602.07360

[12] Y. Gao, Z. Liu, and D. Wang, "Error models of finite word length arithmetic in CNN accelerator design," in *Proc. Vis. Commun. Image Process. (VCIP)*, 2016, pp. 1–4.

[13] T. Hokchhay, S. Hashemi, R. I. Bahar, and S. Reda, "Hardware-software codesign of accurate, multiplier-free deep neural networks," in *Proc. 54th Annu. Design Autom. Conf. (DAC)*, 2017, pp. 1–6.

[14] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[15] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," in *Proc. Workshop Contribution to Int. Conf. Learn. Represent. (ICLR)*, 2016.

[16] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[17] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results*. Accessed: Mar. 3, 2018. [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2011/results/index.html

[18] D. N. Truong *et al.*, "A 167-processor computational platform in 65 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 4, pp. 1130–1144, Apr. 2009.