Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations

Liangzhen Lai LIANGZHEN.LAI@ARM.COM
ARM Research, 150 Rose Orchard Way, San Jose, CA 95134 USA

Naveen Suda NAVEEN.SUDA@ARM.COM
ARM Research, 150 Rose Orchard Way, San Jose, CA 95134 USA

Vikas Chandra VIKAS.CHANDRA@ARM.COM
ARM Research, 150 Rose Orchard Way, San Jose, CA 95134 USA

Abstract

Deep convolutional neural network (CNN) inference requires significant amount of memory and computation, which limits its deployment on embedded devices. To alleviate these problems to some extent, prior research utilize low precision fixed-point numbers to represent the CNN weights and activations. However, the minimum required data precision of fixed-point weights varies across different networks and also across different layers of the same network. In this work, we propose using floating-point numbers for representing the weights and fixed-point numbers for representing the activations. We show that using floating-point representation for weights is more efficient than fixed-point representation for the same bit-width and demonstrate it on popular large-scale CNNs such as AlexNet, SqueezeNet, GoogLeNet and VGG-16. We also show that such a representation scheme enables compact hardware multiply-and-accumulate (MAC) unit design. Experimental results show that the proposed scheme reduces the weight storage by up to 36% and power consumption of the hardware multiplier by up to 50%.

1. Introduction

Deep learning spearheaded by convolutional neural networks (CNN) and recurrent neural networks (RNN) has been pushing the frontiers in many computer vision applications (LeCun et al., 2015). Deep learning algorithms have achieved or surpassed human-levels of perception in some applications (He et al., 2016; Xiong et al., 2016), enabling them to be deployed in the real-world applications. The key challenges of deep learning algorithms are the computational complexity and model size, which impede their deployment on end-user client devices, thus limiting them to cloud-based high performance servers. For example, AlexNet (Krizhevsky et al., 2012), a popular CNN model, requires 1.45 billion operations per input image with 240MB weights (Suda et al., 2016).

Many researchers have explored hardware accelerators for CNNs to enable deployment of CNNs in embedded devices and demonstrated good performance at low power consumption (Qiu et al., 2016; Shin et al., 2017). Reducing the data precision is a commonly used technique for improving the energy efficiency of CNNs. Typically, CNNs are trained on high performance CPU/GPU with 32-bit floating-point data. Fixed-point representation with shorter bit-width for CNN weights and activations has been widely explored (Judd et al., 2015; Gupta et al., 2015; Gysel et al., 2016; Lin et al., 2015), which significantly reduces the storage requirements, memory bandwidth and power consumption without sacrificing accuracy.

This work focuses on the number representation schemes for implementing CNN inference. The representation scheme has the following requirements:

- Accuracy: the representation should achieve the desired network accuracy with limited bit-width.
- Efficiency: the representation can be implemented in hardware efficiently.
- Consistency: the representation should be consistent across different CNNs.

Preliminary work. Submitted for review at the International Conference on Machine Learning (ICML) 2017.

Based on these requirements, we propose using floating-point numbers for CNN weights and fixed-point numbers for activations. We justify this choice from both algorithmic and hardware implementation perspectives. From the algorithmic perspective, using popular large-scale CNNs such as AlexNet (Krizhevsky et al., 2012), SqueezeNet (Iandola et al., 2016), GoogLeNet (Szegedy et al., 2015) and VGG-16 (Simonyan & Zisserman, 2014), we show that the representation range of the weights is the main factor that determines the inference accuracy, which can be better represented in floating-point format. From the hardware perspective, we show that multiplication can be implemented more efficiently with one floating-point operand and one fixed-point operand generating a fixed-point product.

The rest of the paper is organized as follows, Section 2 discusses related work. Section 3 gives some background about different number representation formats and typical hardware implementation of CNNs. Section 4 describes the proposed number representation scheme. Section 5 and Section 6 discuss the scheme from the algorithmic and implementation perspective. Section 7 presents the experimental results, and Section 8 concludes the paper.

2. Related Work

Precision of the neural network weights and activations plays a major role in determining the efficiency of the CNN hardware or software implementations. A lot of research focuses on replacing the standard 32-bit floating-point data with reduced precision data for CNN inference. For example, Gysel et al. (Gysel, 2016) propose representing both CNN weights and activations using minifloat, i.e., floating-point number with shorter bit-width. Since fixed-point arithmetic is more hardware efficient than floating-point arithmetic, most research focuses on fixed-point quantization. Gupta et al. (Gupta et al., 2015) present the impacts of different fixed-point rounding schemes on the accuracy. Judd et al. (Judd et al., 2015) demonstrate that the minimum required data precision not only varies across different networks, but also across different layers of the same network. Lin et al. (Lin et al., 2015) present a fixed-point quantization methodology to identify the optimal data precision for all layers of a network. Gysel et al. (Gysel et al., 2016) present a framework *Ristretto* for fixed-point quantization and re-training of CNNs based on *Caffe* (Jia et al., 2014).

Researchers have also explored training neural networks directly with fixed-point weights. In (Hammerstrom, 1990), the author presents a hardware architecture for on-chip learning with fixed-point operations. More recently, in (Courbariaux et al., 2014), the authors train neural networks with floating-point, fixed-point and dynamic fixed-point formats and demonstrate that fixed-point weights are sufficient for training. Gupta et al. (Gupta et al., 2015) demonstrate network training with 16-bit fixed-point weights using stochastic rounding scheme.

Many other approaches for memory reduction of neural networks have been explored. Han et al. (Han et al., 2015) propose a combination of network pruning, weight quantization during training and compression based on Huffman coding to reduce the VGG-16 network size by 49X. In (Deng et al., 2015), the authors propose to store both 8-bit quantized floating-point weights and 32-bit full precision weights. At runtime, quantized weights or full-precision weights are randomly fetched in order to reduce memory bandwidth. The continuous research effort to reduce the data precision has led to many interesting demonstrations with 2-bit weights (Venkatesh et al., 2016) and even binary weights/activations (Courbariaux & Bengio, 2016; Rastegari et al., 2016). Zhou et al. (Zhou et al., 2016) demonstrate AlexNet training with 1-bit weights, 2-bit activations and 6-bit gradients. These techniques require additional re-training and can result in sub-optimal accuracies.

In contrast to prior works, this work proposes quantization of a pre-trained neural network weights into floating-point numbers and implementation of activations in fixed-point format both for memory reduction and hardware efficiency. It further shows that floating-point representation of weights achieves better range/accuracy trade-off compared for the fixed-point representation of same number of bits and we empirically demonstrate it on state of the art CNNs such as AlexNet (Krizhevsky et al., 2012), VGG-16 (Simonyan & Zisserman, 2014), GoogLeNet (Szegedy et al., 2015) and SqueezeNet (Iandola et al., 2016). Although this work is based on quantization only without the need for retraining the network, retraining may also be applied to reclaim part of the accuracy loss due to quantization.

3. Background

3.1. Fixed-Point Number Representation

Fixed-point representation is very similar to integer representation. The difference is that integer has a scaling factor of 1 and fixed-point can have a pre-defined scaling factor as power of 2. Some examples of fixed-point numbers are shown in Fig. 1.

Usually, all fixed-point representation is assumed to share the same scaling factor during the entire computation. In some scenarios, the computation can be classified into different sets, e.g., for different CNN layers, with fixed-point numbers of different scaling factors. This is also referred as dynamic fixed-point representation (Courbariaux et al., 2014).

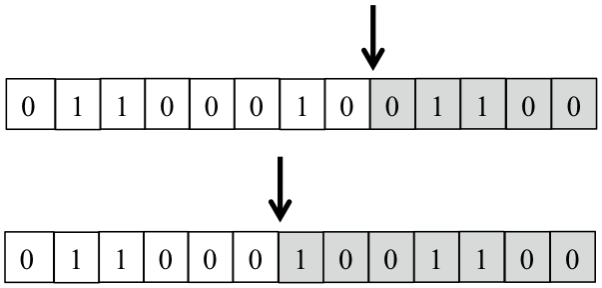

Figure 1. Examples of fixed-point representation. The arrow indicates the position of radix point. So the first number is the corresponding integer value scaled by 2^{-5} and the second one is scaled by 2^{-7}.

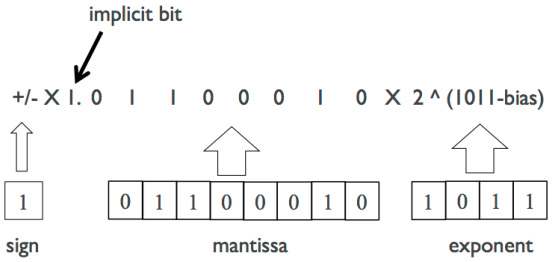

Figure 2. Example of floating-point representation. With the implicit bit, the range for the significand part is limited to be $[1, 2)$. Without the implicit bit, i.e., leading bit as 0, the range becomes $[0, 0.5)$.

3.2. Floating-point Number Representation

One example of floating-point number representation is shown in Fig. 2. For a floating-point representation, there are typically three parts: sign, mantissa and exponent. The sign bit determines whether the number is a positive or negative number. The mantissa determines the significand part and the exponent determine the scale of the value. Usually, there are some special encodings used for representing some special numbers (e.g., 0, NaN and +/- infinity),

For binary floating-point numbers, the mantissa can assume an implicit bit, which is also adopted by IEEE floating-point standard. This ensures that the value of mantissa is always between 1 and 2, so the leading bit 1 can be omitted to save storage space. However, such an implicit bit places a limit on the smallest representable number for the significand part.‘

The exponent is typically represented as an unsigned integer number with a bias. For example, for an 8-bit exponent with a bias of 127, it can represent numbers from -127 to 128, i.e, 0-127 to 255-127.

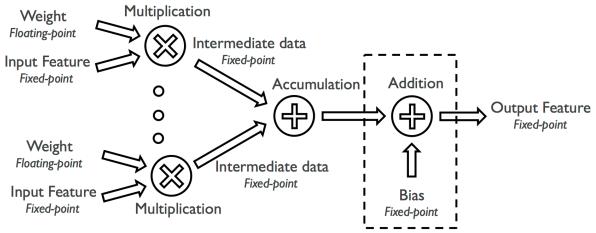

Figure 3. Overview of the data path for convolutional and fully-connected layer operations with the proposed number representation scheme. The weights are represented using floating-point numbers, while the input/output features and intermediate data are represented using fixed-point numbers.

3.3. Hardware Implementation of CNNs

CNNs typically consist of multiple convolution layers interspersed by pooling, ReLU and normalization layers followed by fully-connected layers. Convolution and fully-connected layers are the most compute and data intensive layers respectively (Qiu et al., 2016). The computation in these layers consist of multiply-and-accumulate (MAC) operations. The data path is illustrated in Fig. 3, where the input features are multiplied with the weights to get the intermediate data (i.e., partial sums). These partial sums are accumulated to generate the output features. Since fixed-point arithmetic is typically more efficient for hardware implementation, most hardware accelerators implement the MAC operations using fixed-point representation.

The power/area breakdown of the CNN hardware accelerator mostly depends on the data flow architecture. For example, in Eyeriss (Chen et al., 2016), in each processing element (PE), MAC and memory account for about 9% and 52% area respectively. For Origami (Cavigelli et al., 2015), MAC and memory account for about 32% and 34% area respectively.

4. Proposed Number Representation Scheme

The overview of our proposed number representation scheme is shown in Fig. 3. Different from most existing CNN implementations, we propose using a combination of floating-point and fixed-point representations. The network weights are represented as floating-point numbers while the input/output features are represented as fixed-point numbers. The multiplier is implemented to take one floating-point number and one fixed-point number and produces output, i.e., intermediate data, in fixed-point format. The intermediate data is in fixed-point format and can have wider bit-width than the input/output features. The accumulation is the same as fixed-point adder, which can have higher bit-width.

From hardware perspective, it is more efficient to imple-

ment multiplication using floating-point number and addition using fixed-point number. The multiplication operations have one fixed-point number input, one floating-point number input and fixed-point number output. This can be implemented with a multiplier and a shifter. The multiplier will multiply the fixed-point number with the mantissa part of the floating-point number, and the shifter will shift the results according to the exponent value. Therefore, we propose the hardware architecture illustrated in Fig. 3.

The accumulation/addition works with fixed-point numbers, which can be wider (i.e., with larger bit-width) than either of the inputs. This part is similar to most fixed-point number based implementation of CNN accelerators.

5. Algorithmic Perspective

In this section, we investigate and explain why the proposed number representation scheme is better from the algorithmic perspective. Section 5.1 demonstrates that different CNNs can have inconsistent fixed-point bit-width requirements for representing the weights. Section 5.2 investigates this inconsistency by analyzing CNN weight distribution and properties. Section 5.3 shows that the representation range is the main factor that determines the inference accuracy. Section 5.4 shows that floating-point representation is more efficient and consistent representation for CNN weights.

5.1. CNN Accuracy with Fixed-Point Weights

To evaluate different number representation schemes, we implement weight quantization based on *Caffe* (Jia et al., 2014) framework. To make a fair comparison, we assume that there is always a sign bit for representing negative values for both fixed-point and floating-point numbers. We will represent the bit-width of floating point representation as $m + e$, where m is the number of mantissa bits and e is the number of exponent bits.

We apply the weight quantization on four popular CNN networks: AlexNet (Krizhevsky et al., 2012), SqueezeNet (Iandola et al., 2016), GoogLeNet (Szegedy et al., 2015) and VGG-16 (Simonyan & Zisserman, 2014). We evaluate the network accuracy by doing quantization for all convolutional and fully-connected layer weights. The activation is quantized to 16-bit fixed-point. For each layer, we normalize the weights so that the maximum absolute value equals 1. This is similar to the dynamic fixed-point quantization (Moons & Verhelst, 2016; Gysel et al., 2016; Courbariaux et al., 2014). All accuracy results are top-1 accuracy and normalized with the top-1 accuracy using 32-bit floating-point representation. The results of top-5 accuracy correlate with that of top-1 accuracy.

The network accuracy results using fixed-point representation are shown in Fig. 4. Two observations can be made here:

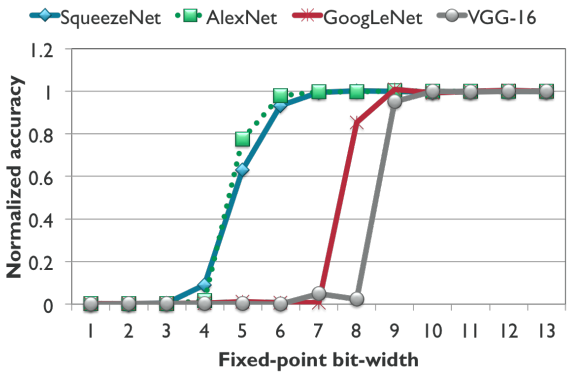

Figure 4. Normalized accuracy of the networks using fixed-point representation with different bit-width.

- For all networks, the accuracy starts increasing sharply after a certain threshold. For all networks, the normalized accuracy increases from close to 0 to close to 1 within 2 or 3 bits difference. This suggests that there is something dramatically different with these additional bits.
- Among different networks, the required number of bits is very inconsistent. With 7-bit fixed-point number, AlexNet and SqueezeNet can achieve close to full accuracy, while GoogLeNet and VGG-16 have very low accuracy. GoogLeNet and VGG-16 need 10 to 11 bits to achieve full accuracy.

This inconsistency in bit-width requirements across different CNNs poses challenges for hardware implementation. For the design to be general-purpose and future-proof, the designer has to use margined bit-width or use runtime adaptation (Moons & Verhelst, 2016), both of which incur significant overhead.

5.2. Weight Distribution

There can be several reasons that cause the inconsistency in Fig. 4. The network depth is one of the possible reasons. Similar to the idea in (Lin et al., 2015), the fixed-point quantization can be modeled as quantization noise for each layer. The network accuracy may drop further, i.e., accumulate more noise, as the network depth increases. The other possible reason is the number of MAC operations in each layer. Small quantization error can accumulate over a large amount of MAC operations. For example, the total number of MAC operations to calculate one output for convolutional and fully-connected layers in AlexNet are (363, 1200, 2304, 1728, 1728, 9216, 4096, 4096).

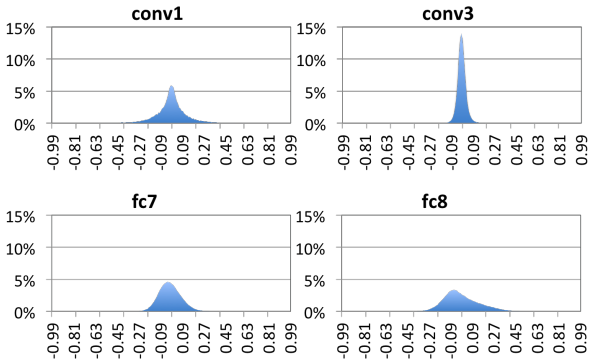

Figure 5. The weight distribution of four different layers in AlexNet. *conv1* and *conv3* are convolutional layers, and fc7 and fc8 are fully-connected layers.

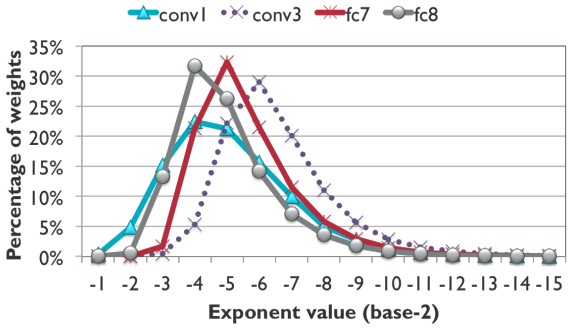

Figure 6. The weight distribution in log-scale of four different layers in AlexNet. The weights are first converted into binary floating-point format. The plot is based on the exponent value of the weights.

However, none of the above reasons can explain the first observation earlier about the sharp, instead of gradual, change in accuracy. To further investigate this, we plot the weight distribution of four different layers in AlexNet in Fig. 5. Most weights have small values even after the per-layer normalization. The distribution is concentrated at the center, which is also the motivation for Huffman encoding of the weights proposed in (Han et al., 2015).

To better visualize the difference, we plot the same weight distribution in log-scale in Fig. 6. This plot is easier to spot the weight distribution difference and explains the accuracy behavior under fixed-point quantization observed in Fig. 4. Under fixed-point quantization, the layer with the most small-valued weights, *conv3*, will be the most susceptible to quantization errors. With 4-bit fixed-point representation, more than 90% weights in *conv3* are unrepresentable, i.e., quantized to 0. This also explains why stochastic rounding works better than round-to-nearest reported in (Gupta et al., 2015).

Since the layers are cascaded, the accuracy of the entire network is limited by the weakest layer. This is why the network produces close to 0 accuracy with small bit-width as shown in Fig. 4. With higher fixed-point bit-width, a larger number of weights in *conv3* become representable, which results in the quick increase of the network accuracy.

The inconsistency in bit-width requirements observed in Fig. 4 can also be explained with the weight distribution. We pick the two layers with largest and smallest weights from AlexNet and VGG-16 and plot the weight distribution in Fig. 7. The layer *conv3_1* in VGG-16 has more weights with smaller exponent values. This explains why VGG-16 requires more bit-width when using fixed-point representation.

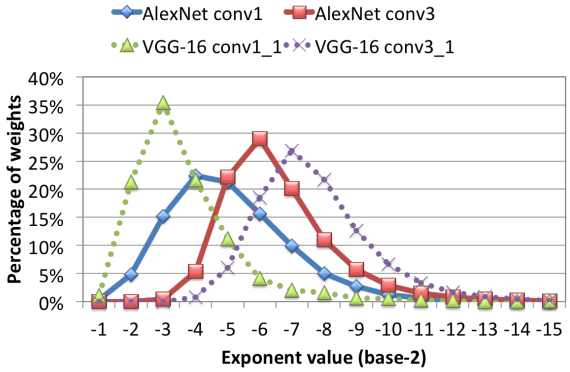

Figure 7. The weight distribution in log-scale for different layers in AlexNet and VGG-16. For each network, we pick the layers with the largest and smallest mean in the exponent value of the weights.

5.3. Range vs. Precision

The results in Fig. 4 and the weight distributions in Fig. 6 show that the network can achieve almost full accuracy, e.g., with 7-bit fixed-point for AlexNet, even when most weights are barely representable, i.e., only with 1 or 2 significant bits. This means that representation range, i.e, the ability to represent larger/smaller values, is more important than representation precision, i.e., differentiation between nearby values.

Since the representation range and representation precision is hard to decompose in fixed-point representation, we investigate this using floating-point representation. For floating-point representation, the mantissa bit-width controls the precision and the exponent bit-width controls the range.

Fig. 8 highlights some of the results of network accuracy using floating-point representation with varying exponent

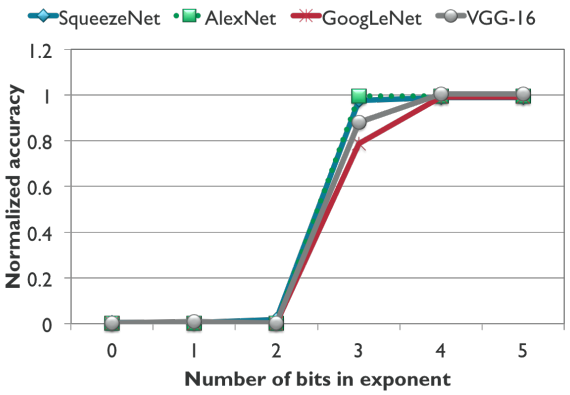

Figure 8. Normalized accuracy of the networks using floating-point representation with different exponent bit-width. The floating-point number has 3-bit mantissa with the implicit bit.

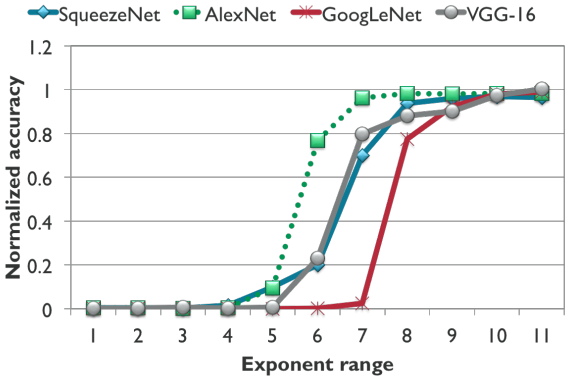

Figure 9. Normalized accuracy of the networks using floating-point representation with limited exponent range. Range of 4 is equivalent to 2-bit exponent, and range of 8 is equivalent to 3-bit exponent. The floating-point number has 2-bit mantissa with the implicit bit.

bit-width (i.e., representation range). The floating-point has 3-bit mantissa with the implicit bit. The implicit bit limits the value of the significant part, so that the representation range is controlled by the exponent part. With floating-point representation, the networks show consistent trend, with almost 0 accuracy with 2-bit exponent and quickly increase to almost full accuracy with 4-bit exponent. Unlike the behavior seen in Fig. 4, this is expected as 2 additional bits in exponent offers 4X increase in the representation range.

To get more insight into the impact of representation range, we also run experiments with floating-point like representation where we limit the exponent range rather than the exponent bits. For example, exponent range of 4 is equivalent to 2-bit exponent and exponent range of 8 is equivalent to 3-bit exponent. The results of exponent range experiments are shown in Fig. 9. The floating-point number has 2-bit mantissa with the implicit bit. The behavior of the accuracy is similar to the results with fixed-point representation, i.e., accuracy increases rapidly from almost 0 to almost 1. The relative ordering exponent range requirements also matches the bit-width requirements for fixed-point representation. This suggests that representation range is the main impacting factor for network accuracy.

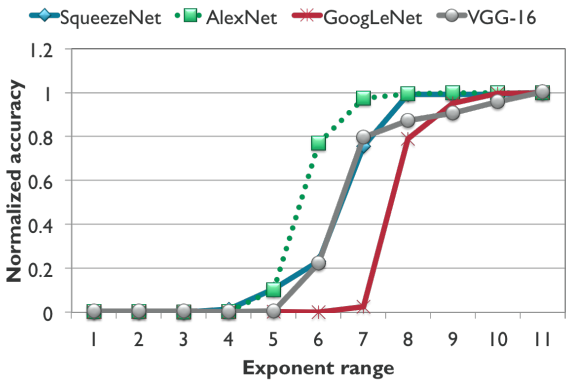

Figure 10. Normalized accuracy of the networks using floating-point representation with limited exponent range. Range of 4 is equivalent to 2-bit exponent, and range of 8 is equivalent to 3-bit exponent. The floating-point number has 6-bit mantissa with the implicit bit.

With 2-bit mantissa, some networks saturate with normalized accuracy less than 1, as shown in Fig. 9. To compare with the effect of precision, the experiments are repeated with 6-bit mantissa as shown in Fig. 10. Comparing Fig. 10 with Fig. 9, the additional 4 bits in mantissa does not have significant impact on the network accuracy. This also validates the initial hypothesis that representation range is more important than representation accuracy.

5.4. CNN Accuracy with Floating-Point Weights

Comparing to fixed-point representation, floating-point is better for representation range, which increases exponentially with the exponent bit-width. The results shown in Fig. 8 show that 4-bit exponent is adequate and consistent across different networks.

The next question for floating-point representation is how much precision, i.e., how many mantissa bits are needed. Fig. 11 highlights some of the results of network accuracy using floating-point representation with varying mantissa bit-width. The floating-point number has 4-bit exponent, and is with the implicit bit. Most networks have very high accuracy even with 1-bit mantissa and achieve full accuracy with 3-bit mantissa. This is also consistent across different networks, which further proves that the inconsistency with

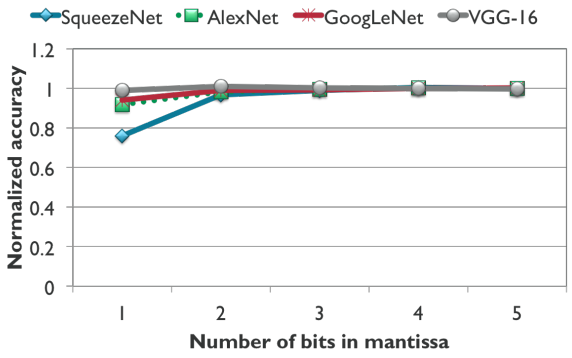

Figure 11. Normalized accuracy of the networks using floating-point representation with different mantissa bit-width. The floating-point number has 4-bit exponent and is with the implicit bit.

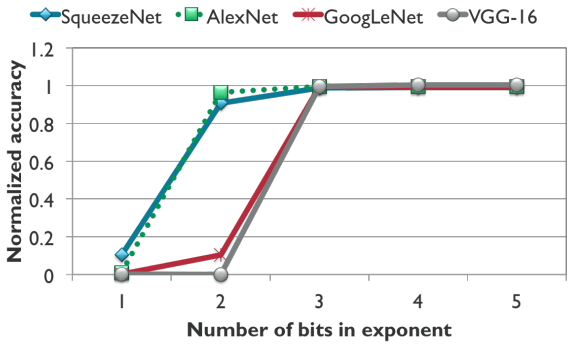

Figure 12. Normalized accuracy of the networks using floating-point representation with different exponent bit-width. The floating-point number has 4-bit mantissa without the implicit bit.

fixed-point representation seen in Fig. 4 is mainly from the inconsistent requirements for representation range rather than from the representation precision.

We also repeat the experiments with floating-point representation without the implicit bit in mantissa. The results are highlighted in Fig. 12 and Fig. 13. The implicit bit in mantissa limits the range of the significand part to $[0.5, 2)$. Removing it helps further extend the range of the representation, especially for representing small numbers. That is why the network accuracy saturates with 3-bit exponent instead of 4-bit. The implicit bit also improves the precision of the significand part. Hence, we need 4-bit mantissa instead of 3-bit to achieve full accuracy.

6. Implementation Perspective

This section motivates the proposed number representation scheme from the hardware implementation perspective. The implementation considerations are discussed in

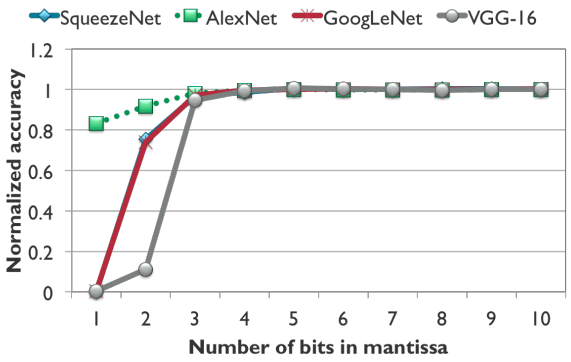

Figure 13. Normalized accuracy of the networks using floating-point representation with different mantissa bit-width. The floating-point number has 3-bit exponent. The mantissa is without the implicit bit.

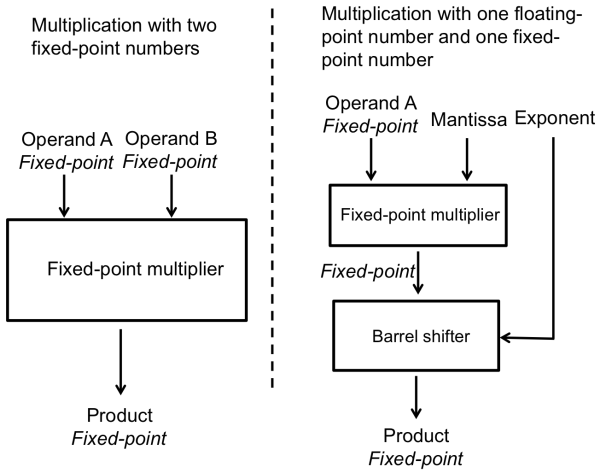

Figure 14. Illustration of hardware implementation of multiplication with two fixed-point numbers and mixed fixed-point and floating-point numbers.

Section 6.1. Hardware trade-off results are presented in Section 6.2.

6.1. Hardware Implementation Considerations

As discussed in Section 3.3, computations in CNNs are typically implemented as MAC operations. For the same 32-bit wide operations, hardware implementation of fixed-point arithmetic can be more efficient than floating-point arithmetic. This is one of the reasons why most of previous work focuses on the optimization for fixed-point representation.

The comparison becomes less obvious when the bit-width is smaller, especially when the number of exponent bits in floating-point representation is small. For example, as shown in Fig. 14, multiplier with a floating-point number

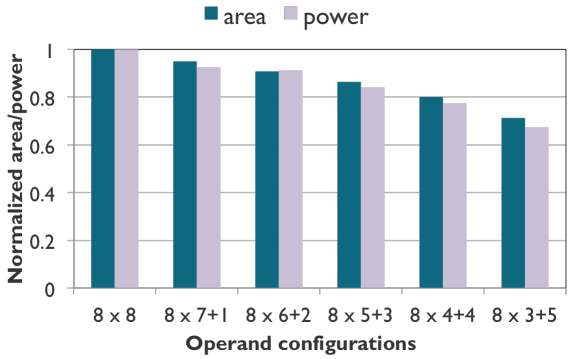

Figure 15. Hardware implementation trade-off results for different multiplier configurations. The area and power are normalized with respect to the case with two 8-bit fixed-point operands. The floating-point operands are represented as mantissa bit-width + exponent bit-width.

can be implemented with a multiplier (of the bit-width of mantissa) and a barrel shifter (of the bit-width of exponent). This can be more efficient than multiplying two fixed-point numbers, as the multiplier becomes smaller and the shifter is simpler.

6.2. Hardware Trade-off Results

To validate the claim in Section 6.1, we implement the multiplier with different operand configurations using a commercial 16nm process technology and libraries. The results are highlighted in Fig. 15. The floating-point configuration is represented as m+e, i.e., mantissa bit-width + exponent bit-width. Here we assume the baseline is a multiplier with two 8-bit fixed-point operands, i.e., 8×8. The area and power numbers are all normalized with respect to the baseline. With the same bit-width, the proposed scheme of combining fixed-point and floating-operands can reduce both area and power. The reduction increases with less mantissa and more exponent bits, as a shifter is more efficient than a multiplier. As an example, the floating-point operand with 4-bit mantissa and 4-bit exponent, i.e. 8x4+4, can reduce the power and area by more than 20%, compared to 8x8 fixed-point multiplication.

7. Experimental Results

As discussed in Section 5.3, we perform the weight quantization based on *Caffe* (Jia et al., 2014). We denote the representation as (m, e), where m is the mantissa bit-width and e is the exponent bit-width. e=0 means fixed-point representation.

We evaluate the network accuracy with different bit-width setting. Table 1 highlights some results for AlexNet. The network is non-functioning (i.e., with close to 0 accuracy) when e is small, i.e., with limited representation range. To achieve full accuracy, AlexNet requires 8 bits for fixed-point representation, i.e., (7, 0) with 1 sign bit, or 7 bits for floating-point representation with (3, 3) configuration. If the implementation only targets AlexNet, the proposed number representation can achieve 12.5% weight storage reduction and 8% power reduction in multiplication. The benefit will increase for CNNs that require more fixed-point bit-width.

Table 1. AlexNet accuracy with different representations. e=0 means fixed-point representation. The floating-point number is with the implicit bit.

	e=0	e=1	e=2	e=3	e=4	e=5
m=1	0.002	0.002	0.003	0.920	0.918	0.918
m=2	0.003	0.003	0.003	0.983	0.982	0.982
m=3	0.002	0.003	0.003	0.995	0.995	0.995
m=4	0.016	0.003	0.003	0.997	1.001	1.001
m=5	0.775	0.002	0.003	0.994	0.999	0.998
m=6	0.979	0.002	0.003	0.995	0.999	0.999
m=7	0.996	0.002	0.003	0.995	0.999	0.999
m=8	0.998	0.002	0.003	0.995	1.000	1.000
m=9	1.001	0.002	0.003	0.995	1.000	1.000
m=10	0.999	0.002	0.003	0.995	1.001	1.001

Table 2. Normalized Accuracy for different networks.

(m, e)	AlexNet	SqueezeNet	GoogLeNet	VGG-16
(7, 0)	1.00	1.00	0.85	0.02
(10, 0)	1.00	0.99	0.99	1.00
(3, 4)	0.99	0.99	0.99	1.00

As discussed earlier, one of the requirement for number representation scheme is the consistency across different networks. This is especially important for the hardware implementation to be future-proof and viable for different CNN models. Some results of the normalized accuracy of different network are highlighted in Table 2. The 7-bit fixed-point configuration used for AlexNet also works for SqueezeNet, but is not adequate for GoogLeNet and VGG-16. 10-bit fixed-point representation is required to get consistent accuracy across all networks used in this study.

By using proposed number representation scheme, we only need 7-bit floating-point, i.e, (3, 4) configuration. Therefore, we can replace 11-bit weights (10-bit fixed-point number plus sign bit) with 8-bit weights (3-bit mantissa, 4-bit exponent and 1 sign bit). This results in 36% storage reduction for weights and 50% power reduction in multiplication.

8. Conclusion

In this work, we propose CNN inference implementation with floating-point weights and fixed-point activations. We

give the motivation for the proposed number representation scheme from both algorithmic and hardware implementation perspectives. The proposed scheme can reduce the weight storage by up to 36% and the multiplier power by up to 50%. Future work will investigate the impacts of network topology and training on the number representation requirements.

References

Cavigelli, Lukas, Gschwend, David, Mayer, Christoph, Willi, Samuel, Muheim, Beat, and Benini, Luca. Origami: A convolutional network accelerator. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 199–204. ACM, 2015.

Chen, Yu-Hsin, Krishna, Tushar, Emer, Joel S, and Sze, Vivienne. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 2016.

Courbariaux, Matthieu and Bengio, Yoshua. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv:1602.02830*, 2016.

Courbariaux, Matthieu, David, Jean-Pierre, and Bengio, Yoshua. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

Deng, Zhaoxia, Xu, Cong, Cai, Qiong, and Faraboschi, Paolo. Reduced-precision memory value approximation for deep learning. 2015.

Gupta, Suyog, Agrawal, Ankur, Gopalakrishnan, Kailash, and Narayanan, Pritish. Deep learning with limited numerical precision. *CoRR, abs/1502.02551*, 392, 2015.

Gysel, Philipp. Ristretto: Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1605.06402*, 2016.

Gysel, Philipp, Motamedi, Mohammad, and Ghiasi, Soheil. Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1604.03168*, 2016.

Hammerstrom, Dan. A vlsi architecture for high-performance, low-cost, on-chip learning. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pp. 537–544. IEEE, 1990.

Han, Song, Mao, Huizi, and Dally, William J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Iandola, Forrest N, Han, Song, Moskewicz, Matthew W, Ashraf, Khalid, Dally, William J, and Keutzer, Kurt. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

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.

Judd, Patrick, Albericio, Jorge, Hetherington, Tayler, Aamodt, Tor, Jerger, Natalie Enright, Urtasun, Raquel, and Moshovos, Andreas. Reduced-precision strategies for bounded memory in deep neural nets. *arXiv preprint arXiv:1511.05236*, 2015.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521(7553):436–444, 2015.

Lin, Darryl D, Talathi, Sachin S, and Annapureddy, V Sreekanth. Fixed point quantization of deep convolutional networks. *arXiv preprint arXiv:1511.06393*, 2015.

Moons, Bert and Verhelst, Marian. An energy-efficient precision-scalable convnet processor in a 40-nm cmos. *IEEE Journal of Solid-State Circuits*, 2016.

Qiu, Jiantao, Wang, Jie, Yao, Song, Guo, Kaiyuan, Li, Boxun, Zhou, Erjin, Yu, Jincheng, Tang, Tianqi, Xu, Ningyi, Song, Sen, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 26–35. ACM, 2016.

Rastegari, Mohammad, Ordonez, Vicente, Redmon, Joseph, and Farhadi, Ali. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv:1603.05279*, 2016.

Shin, Dongjoo, Lee, Jinmook, Lee, Jinsu, and Yoo, Hoi-Jun. Dnpu: An 8.1tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pp. 240–241. IEEE, 2017.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Suda, Naveen, Chandra, Vikas, Dasika, Ganesh, Mohanty, Abinash, Ma, Yufei, Vrudhula, Sarma, Seo, Jae-sun, and Cao, Yu. Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 16–25. ACM, 2016.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Venkatesh, Ganesh, Nurvitadhi, Eriko, and Marr, Debbie. Accelerating deep convolutional networks using low-precision and sparsity. *arXiv:1610.00324*, 2016.

Xiong, W, Droppo, J, Huang, X, Seide, F, Seltzer, M, Stolcke, A, Yu, D, and Zweig, G. The microsoft 2016 conversational speech recognition system. *arXiv preprint arXiv:1609.03528*, 2016.

Zhou, Shuchang, Ni, Zekun, Zhou, Xinyu, Wen, He, Wu, Yuxin, and Zou, Yuheng. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv:1606.06160*, 2016.