CNN Model Resiliency to Low Bit Floating Point Representations

Chris Shakkour, ECE Technion

Nadi Najjar, ECE Technion

Abstracts

In this paper we present a brief comparison between 8-bit integer quantized models and multiple floating-point representations quantized models, the results were conducted on the inference stage of a pretrained Resnet18 CNN trained on the ImageNet dataset. there will be two factors for this comparison, 1. Model resiliency, 2. Area of a single processing unit of a systolic array.

The conducted results point out that there are a few floating-point representations that beat the INT8 with minimal accuracy loss and significant area reduction as well.

Introduction

Convolutional Neural Networks (CNNs) have demonstrated the state-of-the-art classification performance on many Datasets over the years, these networks are usually used for image recognition, semantic segmentation and much more. The models are usually trained in single precision 32-bit floating point representation.

Once the models are trained, they are used for Inference, where a new and untrained image is feed forward thru the Network. This algorithm is characterized with many MAC (multiply and accumulate) operations Hence, the computation workload is very intense especially when done in 32-bit floating point representation.

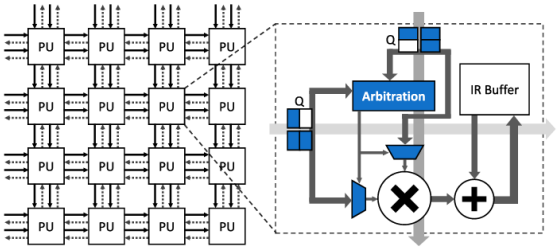

Fig 1: Systolic Array Zoom In

The feedforward workload is usually Accelerated by a Systolic array with a single Multiplier, an Adder, and a Partial Sums (PSUM) register/buffer to hold the computation value along with some minimal control and arbitration logic. all these elements are represented in 32-bit floating point.

In recent years many quantization methods have been used to reduce the number of bits representing the numbers to make this workload more efficient in power and area. Quantization have been explored both in integer and floating-point representations, The most common integer case is 8-bit integer Quantization where the weights, biases, and activations are reduced from 32-bits floating point to 8-bits integer representation. Floating point quantization includes Bfloat, Tensor Float, and FP16 formats.

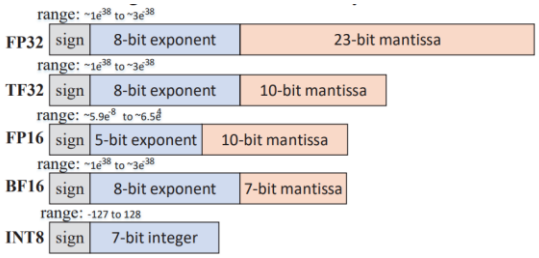

Fig 2: Widely used quantization formats.

In this paper we introduce extreme low bit floating point representations that aim to reduce the number of bits representing a floating-point number by deciding the bit width of both the mantissa(fraction) and the exponent. By doing this we aim to reduce the Power and Area of systolic arrays, and at the same time preserve the higher inference accuracy rates dominated by the 8bit integer quantization models.

Implementation

Floating point numbers are represented by 3 elements, a single sign bit, N exponent bits, and M mantissa bits, together they form the floating-point number. The Terms N, and M will be used extensively to symbol the bit-width of the exponent, and mantissa.

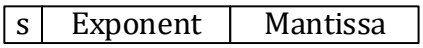

Fig 3: Generic floating-point representation

The 32-bit floating point representation consists of a single sign bit, 8 exponent bits, and 23 mantissa bits, this representation have been quantized to half the size using the 16-bit half-precision floating point number marked in blue in Fig4 that consists of 5 exponent bits, and 10 mantissa bits, and of course the sign bit that we will start ignoring from this point on. Another format that is widely used nowadays is the Bfloat16 marked in Yellow below which consists of 8 Exponent bits, and 7 mantissa bits.

To create new floating-point representation, we sweep both the exponent width N and the mantissa width M

from 1 till 14 and get the following floating-point representations, the number in each cell is the sum of Exponent bits and mantissa bits, if we add the number one (width of sign bit) to these numbers we get the bit width of the new floating-point representation.

Exponent \ Mantissa (fraction)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Fig 4: Mantissa vs Exponent variations

From these generated variations we will be testing the non-red marked variations for two reasons, (1) Very little numbers can be represented with only 2 or less bits of exponent or mantissa each with its own limitations, (2) We aim to explore variations smaller than current solutions 16-bit Bfloat or half precision representations.

Model Resiliency

To test the resiliency of these variations we will use Resnet18 network trained on ImageNet dataset, where the model is trained on 32-bit floating point representation.

Foreach variation the quantization is done from 32-bit floating point to the desired floating-point variation using Qtorch library. the weights and biases are quantized statically before inference, and the activations are quantized dynamically as the data feeds forward from layer to layer.

The base accuracy of this network on ImageNet is Acc1=69.818%, and Acc5=89.106 when quantizing to INT8 the accuracy is almost still the same Acc1=69.804, and Acc5=89.06.

Exponent \ Mantissa (fraction)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	31.4	31.4	31.4	31.4	33.2	33.3	33.1	31.4	31.4	31.4	0	0
4	0	0	31.4	31.4	31.4	31.4	33.1	33.1	33.3	31.4	31.4	0	0	0
5	0	0	69.5	69.5	69.8	69.9	69.8	69.9	69.8	69.8	0	0	0	0
6	0	0	69.5	69.5	69.8	69.9	69.9	69.7	69.8	0	0	0	0	0
7	0	0	69.5	69.5	69.8	69.9	69.7	69.8	0	0	0	0	0	0
8	0	0	69.5	69.7	69.8	69.7	69.8	0	0	0	0	0	0	0
9	0	0	69.5	69.9	69.9	69.9	0	0	0	0	0	0	0	0
10	0	0	69.5	69.9	69.8	0	0	0	0	0	0	0	0	0
11	0	0	69.5	69.9	0	0	0	0	0	0	0	0	0	0
12	0	0	69.5	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig 5: Acc@1 of floating-point representations

In Fig5 we can clearly see that the exponent has more effect on the model accuracy, all the variations when the exponent bit width is 4 or 3 have very low accuracy measurements regardless of the mantissa bit width. But when the exponent is expanded to 5 or more bits, we can clearly see that the accuracy is improved drastically.

Area analysis

Area analysis is usually complicated due to many factors, Technology node, Frequency, Cell vendor, Synthesis tools, and Design constraints. In this paper we choose a unique and fair metric to provide an idea of the area in a fair way given all the conducted tests were Synthesized with the same Parameters and constraints

The area metric that we will be using is called "Logic Kilo Gates" in this metric we synthesize a block of logic and divide the area of the block with the area of a single NAND gate cell, this gives us the number of NAND gates needed to build this block, this way we eliminate the effect of technology node and cell vendor weather it is 5nm, 16nm, or 28nm we expect to have the same results in this metric. To get rid of the effects of Frequency and timing of logic cells we synthesize with a very low frequency, herein we choose 1Mhz.

With that said, to get started we designed in System Verilog two modules. (1) a parametric integer PE, (2) a parametric floating-point PE, where any variation from fig 2 is chosen by 2 parameters the N, and M (the widths of the exponent and mantissa).

Foreach variation in Fig 1 the parametric floating point PE block is synthesized, and the results are presented below in a Bar plot where X-axis represents N, the width of the exponent, and the color of the bar represents the width of mantissa, Y-axis is the Logic K-gate measurement.

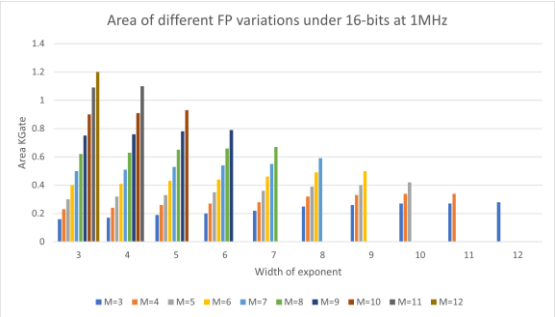

Fig 6: Area bars of chosen variations

Exponent \ Mantissa (fraction)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.16	0.23	0.3	0.4	0.5	0.62	0.75	0.9	1.09	1.2	0	0
4	0	0	0.17	0.24	0.32	0.41	0.51	0.63	0.76	0.91	1.1	0	0	0
5	0	0	0.19	0.26	0.33	0.43	0.53	0.65	0.78	0.93	0	0	0	0
6	0	0	0.2	0.27	0.35	0.44	0.54	0.66	0.79	0	0	0	0	0
7	0	0	0.22	0.28	0.36	0.46	0.55	0.67	0	0	0	0	0	0
8	0	0	0.25	0.32	0.39	0.49	0.59	0	0	0	0	0	0	0
9	0	0	0.26	0.33	0.4	0.5	0	0	0	0	0	0	0	0
10	0	0	0.27	0.34	0.42	0	0	0	0	0	0	0	0	0
11	0	0	0.27	0.34	0	0	0	0	0	0	0	0	0	0
12	0	0	0.28	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig 7: floating-point PE Area results table

If we look at one group of bars in the X-axis each group has a fixed exponent width and each bar represents a different mantissa width (M), as can be seen the area of a PE is affected exponentially by the number of mantissa bits since the mantissa is calculated mainly by an integer

multiplier, however, if we look at a single-color bar but with different exponent bit widths the exponent bits affect the area in a linear manner since the exponent is calculated mainly by an integer adder.

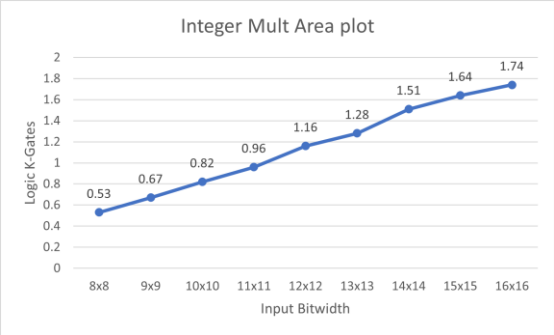

Fig 8: Area plot of integer PE variations

As expected, the more bits there are the area is increased, mainly due to the Multiplication unit, this behavior is expected.

Now if we take Fig 6 and mark in green the potential floating point PE representations that take less area than the Int8 PE we can see that there are many potential representations.

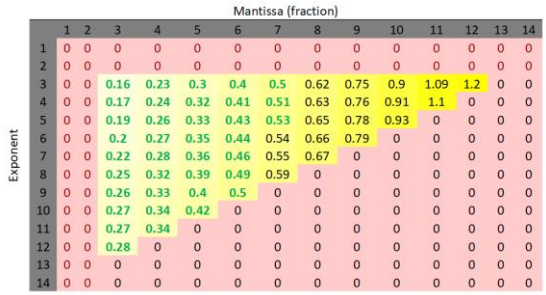

Fig 9: Area bars of chosen variations

Conclusions

In one hand integer quantization has been a great solution to reduce the power and area of Systolic arrays. To improve the accuracy, the number of bits needs to be higher than 8 requiring more silicon area and power.

In the other hand floating point models are highly accurate since more resolution is added to the game but very costly in Power and Area.

In this research we attempted to get the best of both worlds, a floating-point representation that outperforms the 8-bit integer quantized accuracy, and at the same time enjoy a similar low power low area design.

Our conclusion is that you can beat the 8-bit Integer quantization method using a few floating-point representations that are presented in Fig 9, since low bit representations provide quite good accuracy measurements due to their added resolution, but also enjoy low power and low area implementations.

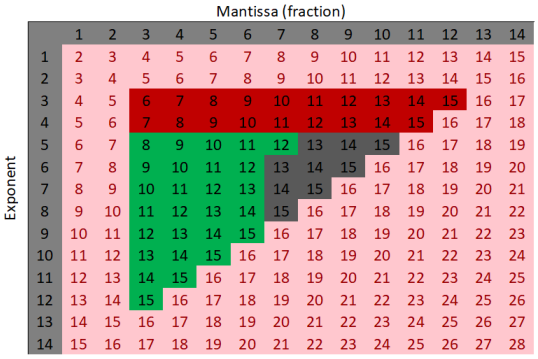

Fig 10: floating point representations

In Fig 9 the red variations are variations that lack accuracy measurements as can be seen in Fig 5. The grey are the variations that exceed the area measurements of INT8. The green variations however are the interesting representations since they offer high accuracy measurements and lower area than the INT8.

If we take the concluded floating-point representations marked in green that aim to potential replace the INT8, we can see that they have higher bit count than INT8 which directly explains the accuracy measurement since more resolution is added to the computations, But, in terms of area the reason why they are lower is because the floating-point multiplication is roughly calculated by multiplying the mantissas and adding the exponents hence the mantissa width determines the size of the Multiplier, while the exponent determines the size of the adder logic hence, a representation of M=5 and N=5 we get a logic circuit with a multiplier of size 5 bits and an adder of size 5 bits which seem to be lower in area than a 8-bit multiplier for INT8 calculation. This observation is backed by our own synthesis results.

References

[1] *Shomron, Gil, Tal Horowitz and Uri C. Weiser. "SMT-SA: Simultaneous Multithreading in Systolic Arrays."* IEEE Computer Architecture Letters *18 (2019): 99-102.*

[2] *Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778*

[3] *J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database,"* 2009 IEEE Conference on Computer Vision and Pattern Recognition, *Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.*

[4] *"IEEE Standard for Floating-Point Arithmetic - Redline," in IEEE Std 754-2008 (Revision of IEEE Std 754-1985) - Redline , vol., no., pp.1-82, 29 Aug. 2008, doi: 10.1109/IEEESTD.2008.5976968.*

[5] *H. Saadat, H. Bokhari and S. Parameswaran, "Minimally Biased Multipliers for Approximate Integer and Floating-Point Multiplication," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no.*

11, pp. 2623-2635, Nov. 2018, doi: 10.1109/TCAD.2018.2857262.

[6] D. Choi and H. Kim, "Hardware-friendly Log-scale Quantization for CNNs with Activation Functions Containing Negative Values," 2021 18th International SoC Design Conference (ISOCC), Jeju Island, Korea, Republic of, 2021, pp. 415-416, doi: 10.1109/ISOCC53507.2021.9613921.