Model Resiliency To Limited Partial Sum Representation

Lior David   Oz Adani

***Abstract*—Approximate computing fits well to the new environment introduced by deep neural networks (DNNs). First, DNNs are usually trained and used for error-tolerant applications; and second, DNNs exhibit inherent algorithmic resiliency to some inaccuracies in their intermediate results. For example, their activations and weights can be pruned and quantized with minor degradation in accuracy [1]. Motivated by these observations, we would like to check the model resiliency to a limited psum representation, which is usually set to 32 bits.**

I. Introduction

Machine learning has become an essential part of modern hardware and has an important role in computer engineering's latest applications. One of the main research fields of machine learning is the Deep Neural Network, which is designed to mimic the neurons of the brain. The artificial neurons are the building blocks of a neural network. They are typically separated into layers, where the outputs of neurons in one layer become the inputs of the next layer. In order to classify the input data we need to determine various aspects of the data. DNNs use algorithms, assign importance using learnable weights and biases to its layers.

Convolutional neural networks are a specialized class of deep neural networks which use convolution layers. The convolution layers aim to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. Common CNNs for classifying images are AlexNet and ResNet18. Each one of them has a different design. The convolution operation in these networks involves repeatedly accumulating the multiplication between input image data and filter weights to produce the output value stored in the partial sum component.

A common method to speed up the matrix multiplication process is using a Systolic Array which is a standard in computer architecture for parallel programs [2]. Each cell in the systolic array contains a Processing Engine, i.e., PE, which computes a single cell in the output feature map matrix. PE architecture contains a 8x8-bit multiplier, a 16-bit wide adder and a Partial Sum component, i.e., PSum. The PSum is a register that stores the result of the multiply-accumulate, i.e., MAC operation. The operation involves two numbers, A is the input image data, i.e., Activations, B is the filter weights, i.e., Weights, when both A and B are quantized to 8-bit wide values. Using the multiplier, we compute the output between A_i and B_i, then accumulate it to the PSum in the following order: PSum = $\sum_{i=1}^{n} A_i B_i$. The output of the multiplication between A_i and B_i is up to 16-bit wide due to the 8-bit quantization performed on A and B.

Conventional PSums are 32-bit wide, but the usage of these bits depends on inputs quantization, data-sets and CNNs, which affects the output of the MAC operations. Optimizing the Partial Sum component can lead to considerable benefits in reducing hardware area and price.

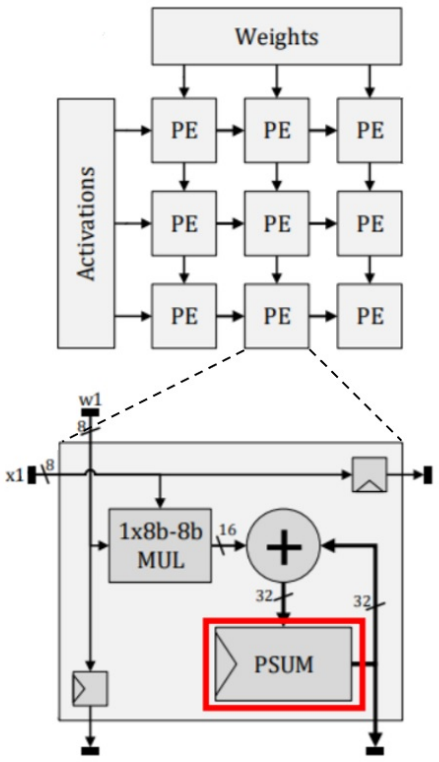

Fig. 1. Systolic Array And Its Components

II. Idea

The CNNs mentioned above have different designs: AlexNet contains eight layers. the first five are convolutional layers, and the last three are fully connected layers [3]. ResNet18 contains 17 convolutional layers and one fully connected. These changes can affect the usage of the PSum component. In addition, the diversity occurs between different layers and channels of the same network. Due to the variety in the output size, the PSum component may not be fully used in some cases, may be fully used and even get saturated in others. The integrity of the output data stored in the PSum is crucial for the image classification accuracy.

In this paper we propose some methods to reduce the size of the PSum without compromising the classification's accuracy. Changing the PSum size has a trade-off between area and accuracy:

1) Using large size registers may preserve accuracy, but not saving much hardware area.
2) Using small size registers will save hardware area, but may cause saturation and partial representation which can reduce the accuracy.

We would like to check the models resiliency to limited PSum representation in layer and channel granularities for AlexNet and ResNet18 models. By checking the compute values of the MAC operations, we can learn how many bits are in use in each layer and channel. In addition, by analyzing the importance of the accumulator's most significant bits and least significant bits, we can learn the influence of these values on the model's accuracy.

III. Implementations

In this section, we present our implementations of the PSum's reduced representation for quantized data, when applying it on convolution layers. As mentioned before, we would like to analyze the PSum usage with untouched representation. Using our implementations in the environment, we are able to trace the new MAC's data and statistics.

A. Total Bits Reduce

Conventional PSum register is 32-bit wide, however this space may not be fully utilized. Accordingly, we reduced the PSum's size by cutting out the most significant bits that are not in use.

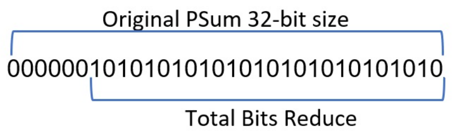

Fig. 2. Total Bits Reduce

For example, the PSum's value is ”1011” and the amount of MSB we decided to cut is 3-bit. The reduced PSum's value will be ”0001”. As a result, the representation of the compute value is smaller, according to the amount of bits cut out.

B. LSB Reduce

Following the last section, we desire to reduce the size even more. We predicted that the MSB are more crucial than the LSB for data classification. In order to implement the idea, we removed the least significant bits of our already reduced PSum from the first implementation. To restore the PSum's right representation, we need to shift left the output matrix's values by the same amount of bits we removed from the LSB.

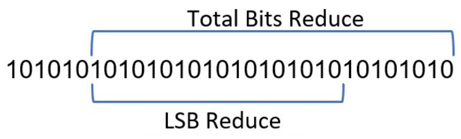

Fig. 3. LSB Reduce

For example, the reduced PSum's value is ”1011” and the amount of MSB we decided to save is 3-bit. The PSum's value will be ”101” which is the number ”1010”. The least significant bit with the value ”1” is lost. Data loss due to the omission of least significant bits, may harm the accuracy precision. In this method, we wanted to find a balance between the reduction size and accuracy.

C. LSB Dynamic Reduce

In this last method, we want to achieve a finer efficiency of the allocated PSum size while maintaining the model initial accuracy. Based on the first implementation, we selected a fixed size that defines a sliding window. At first, The window will preserve a number of fixed size as defined. While the compute's representation is growing, the window will slide to the left, allowing the representation of larger numbers but losing their lower digits precision.

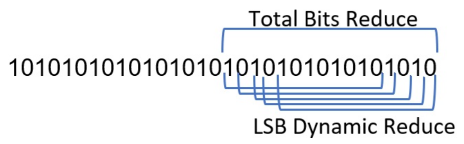

Fig. 4. LSB Dynamic Reduce

For example, total bits reduce PSum's value is ”00” and a 2 bits-wide window. The window will contain the value ”00” which are the two least significant bits the window can handle. Adding the value ”01” to the PSum, updates the window's value to ”01”. After accumulating the value ”100” to the PSum, the window's value will be ”10”, which are the 2 MSB of the number ”101”. The PSum's final value will be ”10” which is the number ”100”, while losing the value ”1” of the least significant bit.

Due to the dynamic implementation, the position of each sliding window for every cell in the output matrix may be different. To overcome this issue, we need another register for every PE which will save the amount of bits omitted. This register will be called the "Movement Register" and its size will be $\lceil log_2(TotalBitsReduce - Windowsize)\rceil$. For example, using 19 total bits for reduced PSum, with 12-bits window size, the window can move a maximum of 7 left shifts and thus the Movement Register's size will be 3-bits wide. In addition to the Movement Register, we need to add hardware logic that can determine when there is an overflow in the compute's value, choosing the right input values by using multiplexers which eventually will slide the window to the left.

IV. Evaluations

We turn to evaluate our implementations in terms of accuracy and bits capacity. Our system was created using Python, C++ and CUDA languages. The system simulates our MAC's implementations and uses tests for network models such as AlexNet and ResNet18, with ImageNet as the data-set. The two networks have numerous types of layers, but we will deal only with convolution layers. First, the inputs, images and weights, are being quantized to a 8-bit granularity. Second, we created a checkpoint for every model, which will have a fixed accuracy in order to compare the results of our methods. Lastly, we run the tests for a specific model, using our MAC's implementations.

We analyzed the convolution layers by checking the data size stored in the PSum, at the end of a run test of each layer. The following histogram describes this analyzation:

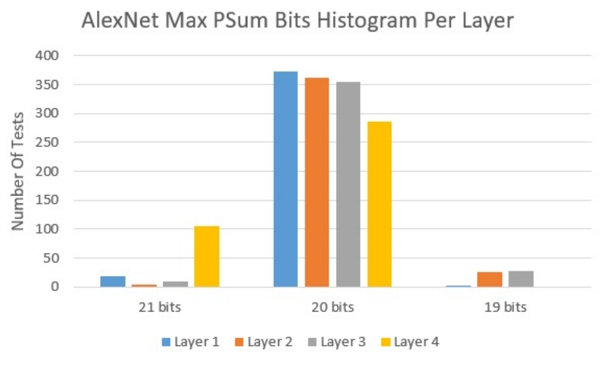

Fig. 5. AlexNet Histogram Max PSum Size Per Layer

When looking at the histogram, we noticed that AlexNet uses at most 21 bits for each layer, when running the network through all the model's tests. In a similar way, we found out that ResNet18 uses at most 21 bits for the 18th layer, and 19-20 bits for all the other layers. Thus, we discovered that there is no significant difference between the layers' bits usage, so we looked into channels perspective. Both networks, in channel level inspection, used no more than 20 bits and a small minority of them 21 bits. In addition, we looked into the average usage in both layer and channel perspective, and found out that it is around 16-18 bits. Following these results, we deduced that there is no need for such a large PSum component, and adjusting a special implementation for each layer by its bits usage is not practical. From now on, we will focus only on ResNet18 due to the similar results.

Using our first implementation, we want to achieve smaller PSum considering the results we got. We examined ResNet18 model's accuracy as a function of the component's size:

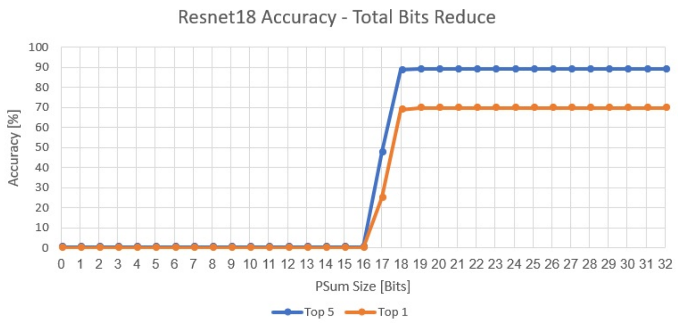

Fig. 6. Reduce Total Bits ResNet18 Graph

As we anticipated, there is no change in the model's accuracy above 21-bits due to the maximum size we found. It can be noticed that 20-bits PSum can still maintain original model's accuracy [4]. In addition, lower than 19-bits PSum, leads to a significant reduction in accuracy, therefore the MSB has an important role in accuracy preservation. As well, very small representation size loses all the important features the convolution layers produce hence the poor accuracy. Better results can be achieved when cutting out the LSB of the PSum register.

Using our second implementation, we would like to take the initial saving (cutting off the unused MSB), and on top of that, removing the register's LSB. This method is based on the assumption that the lower digits have a smaller affect on the model's accuracy. The bigger values in the accumulation phase are these that influence the most on the PSum's final value, hence determine the characteristic of the classification. As we saw in Fig. 6, reducing the PSum's size to 19-bits has the same accuracy as the original model. Thus we will not check bigger representations. In addition, we want to analyze the model reaction in other cases such as 16-18-bits PSum size. We examined ResNet18 model's accuracy while changing the amount of LSB reduce. For example, the line "19 Total Bits Reduced PSum" in the graph, at x=12, saves the 12 MSB (from 19 to 8) and drops the 7 LSB.

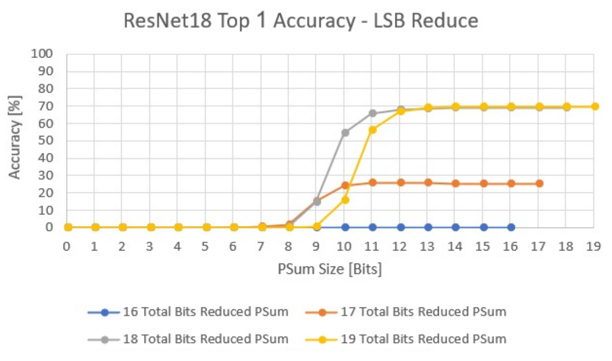

Fig. 7. LSB Reduce ResNet18 Top1 Graph

It can be observed that implementations using less then 18 total bits reduced will not preserve the high accuracy of the model. Using 19 total bits reduced PSum, we are able to maintain the original accuracy level while keeping only the 15 MSB, and saving a lot more space. Furthermore, keeping only the 12-14 MSB can save even more space but with a slight impact on accuracy.

Additional MAC improvements may be implemented by minimizing the accumulator and multiplier components' size. Due to LSB cut off, the lower digits will always be zero for every value in the output. Therefore, calculating the LSB of the inputs, both in the multiplier and the accumulator, will not change the output value. This way, we can work with partial representation of the inputs, use smaller components and thus reduce the overall computations.

Moreover, we simulated the second implementation with Batch Normalization method, in order to achieve better results with data manipulation. Batch Normalization is a method that processes the input layer by re-centering and re-scaling using unit standard deviation. The following graph compares the last method with and without Batch Normalization:

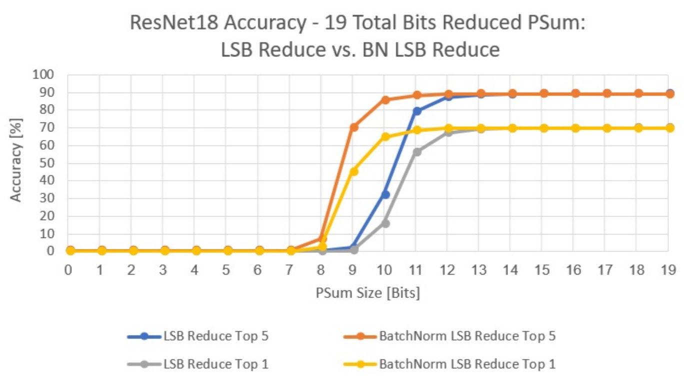

Fig. 8. LSB Dynamic Reduce With And Without Batch Normalization Graph

As shown in the graph, using BN method on top of the second implementation will improve the saving by 2 extra bits. Now, using 19 total bits reduced PSum, we can keep only the 13 MSB without decreasing the original accuracy. Furthermore, it can be noticed that the slope of the BN graph is much smoother, resulting in better accuracy for slightly smaller representation.

Our third implementation takes the idea of the last method, which is reducing the LSB, but only when needed. The MSB of the growing number are always being kept throughout all the accumulations, while the LSB that cannot be represented anymore by the sliding window are lost. We examined ResNet18 model's accuracy while changing the sliding window's size. In the following graph we present a comparison between the second and third methods while choosing a fixed size window which is also the PSum's size. For example, the line "LSB Reduce Top 5", in the graph, at x=12, saves the 12 MSB (from 19 to 8) and drops the 7 LSB. The line "Dynamic LSB Reduce Top 5", at x=12, starts with a 12-bit size window, represents bits 12 to 1, and depending on the growing sum, can shift the window left, up to 19 to 8-bits representation.

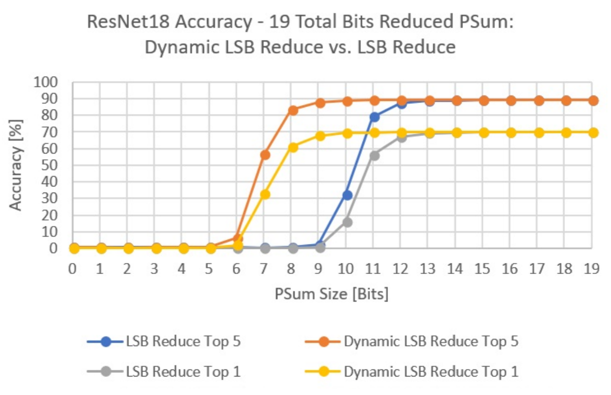

Fig. 9. Dynamic LSB Reduce Vs. LSB Reduce

As indicated in fig. 9, there is no change in accuracy between the second and third method for 15-bits and more. With the new method we can get the same original accuracy while using only 12-bits. This is a 3-bits reduction from the LSB Reduce method. Another outcome of the graph is when using the Dynamic LSB Reduce method, in the range of 11 to 8-bits, the model is still keeping high accuracy, while the other method is not practical as the accuracy collapses.

V. Results

Using the three methods, we gathered all the statistics into one table and one graph. Table I describes the proportional saved space compared to the original 32-bits PSum. In addition, we compared the proportional accuracy between our methods and the original ResNet18 Top 1 accuracy which is 69.78%. Fig. 10 presents a comparison between our three methods.

TABLE I
CONVENTIONAL 32-BITS PSUM VS. IMPLEMENTATIONS
RESNET18 TOP 1

PSum Size	PSum Bits Saving	Preserved Accuracy		
		Method#1	Method#2	Method#3
8	75.00%	0.14%	0.12%	87.36%
9	71.87%	0.14%	0.76%	97.01%
10	68.75%	0.14%	22.55%	99.28%
11	65.62%	0.14%	80.26%	99.86%
12	62.50%	0.14%	95.97%	99.96%
13	59.37%	0.14%	99.06%	99.97%
14	56.25%	0.14%	99.71%	99.95%
15	53.12%	0.14%	99.94%	99.94%
16	50.00%	0.18%	99.93%	99.99%
17	46.87%	36.10%	99.97%	99.95%
18	43.75%	98.66%	100%	100%
19	40.62%	100%	100%	100%

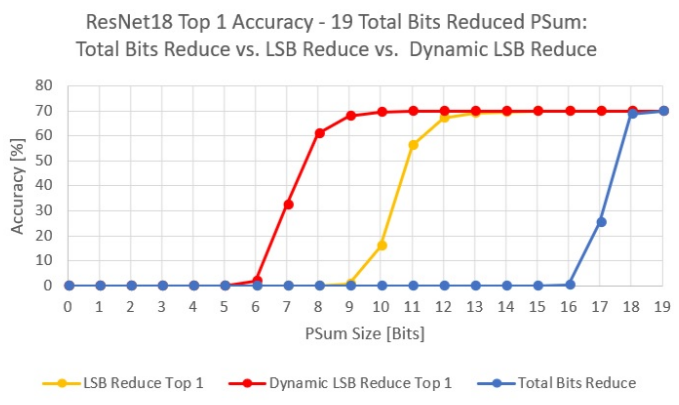

Fig. 10. Total Bits Reduce Vs. LSB Reduce Vs. Dynamic LSB Reduce

The visualisation of this table and graph helps summarize each method's benefits and drawbacks:

1. Total Bits Reduce: The main benefit is demanding no extra computations as we preform standard MAC operations, using a smaller register. Still, we mange to reduce the size to 19-bits and save 40.62% area without impacting the original accuracy. On the other hand, using a register smaller than 18-bits, will cause the model to be impractical.

2. LSB Reduce: Using this method we can even use 14-bits registers with a minor loss in accuracy. In addition, the MAC's operations are still standard and we may reduce the hardware logic of the multiplier and the accumulator components due to smaller values being processed in the MAC. However, this is still not the best in terms of PSum register size.

3. LSB Dynamic Reduce: With our last method, we managed to achieve best results to limited PSum representation. There is a 99.96% accuracy preservation using only 12-bits, which is 62.50% PSum area save. Even registers as small as 10-bits preform well. In resemblance to the second method, the MAC's components can be reduced. Despite all the great benefits, every PE needs to have a Movement Register as we mentioned before, which consumes extra area. This extra area is not included in the PSum's size from Table I and needs to be taken into consideration when analyzing the overall system area. The method also demands a special logic which needs to determine the position of the window according to the compute's value and logic for computing the relevant digits in the MAC's components for the current window.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we examined model resiliency to limited partial sum representation. Convolutional neural networks involve many multiply-accumulate operations of which a significant part are not fully utilized. As we noticed through our implementations, the PSum component can be dramatically reduced while maintaining high accuracy on both AlexNet and ResNet18. We presented three methods that are suitable for our simulation's assumptions, by exploiting the quantized values and error tolerant applications. In addition to our main focus on the PSum component, we detected that both the multiplier and accumulator components can be reduced, resulting in even better power and area efficient hardware. An interesting discussion that came up while implementing the methods is how much power consumption we can save due to the improvements in the MAC's components and smaller PSum registers. As power consumption is the main challenge of modern architecture, further research into our implementations is needed and may lead to a possible decrease in power.

REFERENCES

[1] Gil Shomron and Uri Weiser. Non-blocking simultaneous multithreading: Embracing the resiliency of deep neural networks. In International Symposium on Microarchitecture (MICRO), 2020.
[2] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey", 2017.
[3] Krizhevsky Alex, Sutskever Ilya, Hinton Geoffrey E. "ImageNet classification with deep convolutional neural networks", 2017.
[4] Yu-Hsin Chen, Student Member, IEEE, Tien-Ju Yang, Student Member, IEEE, Joel Emer, Fellow, IEEE, and Vivienne Sze, Senior Member, IEEE. "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices", 2019.