Low Complexity Multiply Accumulate Unit for Weight-Sharing Convolutional Neural Networks

James Garland and David Gregg, Trinity College Dublin

Abstract—Convolutional Neural Networks (CNNs) are one of the most successful deep machine learning technologies for processing image, voice and video data. CNNs require large amounts of processing capacity and memory, which can exceed the resources of low power mobile and embedded systems. Several designs for hardware accelerators have been proposed for CNNs which typically contain large numbers of Multiply Accumulate (MAC) units. One approach to reducing data sizes and memory traffic in CNN accelerators is “weight sharing”, where the full range of values in a trained CNN are put in bins and the bin index is stored instead of the original weight value. In this paper we propose a novel MAC circuit that exploits binning in weight-sharing CNNs. Rather than computing the MAC directly we instead count the frequency of each weight and place it in a bin. We then compute the accumulated value in a subsequent multiply phase. This allows hardware multipliers in the MAC circuit to be replaced with adders and selection logic. Experiments show that for the same clock speed our approach results in fewer gates, smaller logic, and reduced power.

Index Terms—Convolutional neural network, power efficiency, multiply accumulate, arithmetic hardware circuits.

1 INTRODUCTION

Convolutional neural networks require large amounts of computation and weight data that stretch the limited battery, computation and memory in mobile systems. Researchers have proposed hardware accelerators for CNNs that include many parallel hardware multiply-accumulate units.

CNN training involves incrementally modifying the numeric “weight” values associated with connections in the neural network. Once training is complete, there are no further updates of the weight values, and the trained network can be deployed to large numbers of embedded devices. Han et al. [4] found that similar weight values can occur multiple times in a trained CNNs. By binning the weights and retraining the network with the binned values, they found that just 16 weights were sufficient in many cases. They encode the weights by replacing the original numeric values with a four-bit index that specifies which of the 16 shared weights should be used. This greatly reduces the size of the weight matrices.

We propose a novel Multiply Accumulate (MAC) circuit design aimed specifically at hardware accelerators for CNNs that use weight sharing. Rather than performing the MAC operation directly, we instead count the frequency of each weight and accumulate the corresponding image value in a bin. This allows the hardware multiplier in the MAC to be replaced with counting and selection logic. After accumulation, the accumulated image values in the bins are then multiplied with the corresponding weight value of that bin. Where the number of bins is small, the counting and selection logic can be significantly smaller than the corresponding multiplier.

2 BACKGROUND

The most computationally-intensive part of a CNN is the multi-channel convolution. Fig. 1 shows pseudo-code for the main loop of the 2D multi-channel convolution operator. The input of the operator consists of an image with many (often 256 or 512) channels, which is combined with a large number of kernels to create an output image, again with many channels. The innermost loops can be implemented with a hardware MAC and accelerators for CNNs commonly include a large number of parallel MACs (9).

A simple MAC unit is a sequential circuit that accepts a pair of numeric values, computes their product and accumulates the result. The accumulator register is typically located within the MAC to reduce routing complexity and delays. When one of the inputs to the MAC is encoded using weight sharing, an extra level of indirection is required. Fig. 2 shows a simplified version of weight sharing decode with a single MAC. The weights in the CNN are represented by, for example, a 4-bit number which is an index to a table of 16 shared weight values.

The most common computation in a CNN is multiply-accumulate. Hardware accelerators for CNNs may contain dozens, hundreds or even thousands of MAC units. However, each MAC unit is a relatively expensive piece of hardware consuming large floor area (i.e. large numbers of gates) and power in an Application Specific Integrated Circuit (ASIC). Hardware accelerators for CNNs typically use 8-, 16-, 24- or 32-bit fixed point representations. The area and power overhead of a hardware MAC circuit is reduced when it is directly connected to high-speed memory. However, this reduces the efficiency of the CNN by requiring the MAC circuits to be replaced with more complex hardware. A more efficient approach is to use the MAC to reduce the complexity and power of the hardware.

Fig. 1. Simplified code for 2D multi-channel convolution with a single multi-channel input and multiple multi-channel convolution kernels. Note that special treatment of edge boundaries is not shown in this code.

\begin{verbatim}
input[width][height][input_channels];
kernels[output_channels][K][K][input_channels];
output[width][height][output_channels];
for w in 1 to width
    for h in 1 to height
        for o in 1 to output_channels {
            sum = 0;
            for x in 1 to K
                for y in 1 to K
                    for i in 1 to input_channels
                        sum += input[w+x][h+y][i] * kernels[o][x][y][i];
            output[w][h][o] = sum;
        }
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{convolution.png}
\caption{Simplified code for 2D multi-channel convolution with a single multi-channel input and multiple multi-channel convolution kernels. Note that special treatment of edge boundaries is not shown in this code.}
\end{figure}
bit fixed point arithmetic. A combinatorial $w$-bit multiplier requires $O(w^2)$ logic gates to implement which makes up a large part of the MAC unit. Note that sub-quadratic multipliers are possible, but are inefficient for practical values of $w$.

The gates column of Table 1 shows the circuit complexity in gates of each sub-component, assuming fixed-point arithmetic. The bit-width of the data is $w$ and the number of bins is $b$ in the weight-shared designs. For example, a simple MAC unit contains an adder ($O(w)$ gates) a multiplier ($O(w^2)$ gates) and a register ($O(w)$ gates). A weight shared MAC also needs a small register file with $b$ entries to allow fast mapping of encoded weight indices to shared weights. The Parallel Accumulate and Store (PAS) is our novel weight-shared MAC unit which is described in Section 3.

### Table 1

| Sub Component | Gates | Simple MAC | Weight Shared MAC | PAS |
|---------------|-------|------------|-------------------|-----|
| Adder         | $O(w)$ | 1          | 1                 | 1   |
| Multiplier    | $O(w^2)$ | 1         | 1                 | $b$ |
| Register      | $O(w)$ | 1          | 1                 | 1   |
| File Port     | $O(wb)$ | 1          | 1                 | 2   |

### 3 Approach

We propose to reduce the area and power consumption of the MACs by re-architecting the MAC to do the accumulation first followed by a shared post-pass multiplication. Rather than computing the Sum Of Products (SOP) in the MAC directly, we instead count how many times each of the $b$ weight indexes appears and store the corresponding image value in a register bin.

For example, if the shared weight with index 2 had the value 19 and were multiplied and accumulated with the image value 25, then a weight-sharing MAC would compute $19 \times 25 = 475$ and add this value to the accumulator. Instead we keep $b$ separate accumulators, one for each weight value. If we encounter the shared weight with index 2, value 19 and image value 25, then rather than performing any multiplication, we instead add 25 to accumulator number 2 in the local $b$-entry register file. Storing this result in a register file that is local the MAC unit reduces unnecessary data movement.

Our Parallel Accumulate and Store (PAS) unit represents the sum as an unevaluated sum of a set of coefficients of each of the $b$ weights (see Fig. 3). The system computes the dot product by computing the total of how many of each of the weights appear in the sum. This turns the multiply-accumulate step into an array-index-and-add operation.

To compute the final dot product value the final SOP of weights and their respective frequencies is calculated using a standard MAC unit. We refer to this combination of a PAS and MAC as a Parallel Accumulate Shared MAC (PASM). The post-pass MAC requires just $b$ multiplications whereas the standard multiply system were used in a standard CNN then the number of multipliers (one for each MAC) would be prohibitive.

The latter MAC stage can be implemented on a MAC unit if the shared weight is less than or equal to 256 bins.

The PASM proposed in Fig. 3 stores the $w$ bit image values in corresponding $b$ weight-bins indexed by the binIndex signal. The PASM has two phases: (1) to multiply the corresponding weights with the weights and (2) to multiply the binned values with the weights (completing the PASM).

Fig. 4 shows how image value 26.7 is accumulated into bin 0 indexed by binIndex. Next 3.4 is accumulated into bin 1. This continues until finally accumulating 6.1 into bin 0 to give $26.7 + 6.1 = 32.8$.

Fig. 5 demonstrates the multiply phase, multiplying and accumulating bin 0 pretrained weight with bin 0 accumulated image value, giving $32.8 \times 17.7 = 574.6$. The contents of pretrained weight bin 1 in multiplied and accumulated with image value in bin 1 and so on until all the corresponding bins are multiplied and accumulated into the result register, giving $98.8$.

The PASM can be replaced by PAS units and a number of PAS units share a single MAC. Due to its parallel nature, the MAC bin count and parallelism can be increased until it becomes the bottleneck. However, the parallel PASM can be implemented to share 16 MACs.

With far fewer multiplications needed, one post-pass MAC unit can be shared between several PAS units.

Of course, the PASM is only efficient in a weight shared CNN. If this weight-shared accumulate-shared-multiply system were used in a standard CNN then the number of the weight-bin registers would be prohibitive.

Table 1 shows each sequence of MAC operations will consist of $k \times k \times c$ inputs (see lines 10 - 13 of Fig. 1). For example, if $b = 16$ bins is used for $k = 5 \times 5$ kernels and $c = 32$ channels, then 800 accumulations and 16 multipliers (one for each MAC) would be prohibitive.
4 Evaluation

We designed a multi-channel, multi-kernel convolution accelerator unit to perform a simplified version of the accumulations in Fig. 1. Our accelerator accepts 4 image inputs and 4 shared-weight inputs each cycle, and uses them to compute 16 separate MAC operations each cycle. The standard version performs these operations on 16 weight-shared MAC units (16-MAC).

Our proposed PASM unit (16-PAS-4-MAC) has 16 PAS units and uses 4 MAC units for post-pass multiplication. Both are coded in Verilog 2001 and synthesized to a flat netlist at 100MHz with a short 0.1ns clock transition time targeted at a 45nm process ASIC. We measure and compare the timing, power and gate count in both designs for the same corresponding bit widths and same numbers of weight bins.

The standard 16-MAC and the proposed 16-PAS-4-MAC each have \( w \) bit image and weight inputs and the 16-PAS-4-MAC has a \( w \)-bit binIndex input to index into the \( b = 2^{w-1} \) weight bins. The designs are coded using integer/ fixed point precision numbers. Both versions are synthesized to produce a gate level netlist and timing constraints designed using Synopsys Design Constraints (SDC) so that both designs meet timing at 100MHz.

Cadence Genus (version 15.20 - 15.20-p004_1) is used for synthesizing the Register Transfer Level (RTL) into the OSU FreePDK 45nm process ASIC and applying the constraints in order to meet timing. Genus supplies commands for reporting approximate timing, gate count and power consumption of the designs at the post synthesis stage. The “report timing”, “report gates” and “report power” of Cadence Genus are used to obtain the results for both MAC and PASM designs. Graphs of the gate count and power consumption results are produced for the two different designs at different bit widths and different numbers of weight bins, showing that the PASM is consistently smaller and more efficient than the weight-sharing MAC.

![Fig. 6. Logic gate count comparisons (in NAND2X1 gates) for \( \text{Bit Width (w)} = 4, 8, 16, 32 \)](image)

![Fig. 7. Power consumption (in W) comparisons for \( \text{Bit Width (w)} = 4, 8, 16, 32 \) ](image)
We also experimented with implementing the designs on a Xilinx Kintex Ultrascale Field Programmable Gate Array (FPGA) however, for larger values of \( b \) bins, such as \( b = 64 \) and \( b = 256 \), our approach quickly becomes inefficient. The weight-shared PASM introduces a delay in the processing of the output of the PAS units. The PAS unit has a throughput of one pair of inputs per cycle, and so computes the initial accumulated values in about \( n \) cycles. The post pass MAC unit also has a throughput of one pair of inputs per cycle, so requires a cycle for each of the \( b \) accumulator bins, for a total of \( n + b \) cycles. In contrast a simple MAC unit requires just \( n \) cycles.

Whilst more parallelism of standard MACs could be applied to accelerate a CNN, these parallel MACs would consume more power and area that that of our parallel PASs and shared MACs.

Future improvements should include placing and routing the MAC and PASM designs to a fully routed test chip. Back annotated gate level simulations should be executed on the final netlist to obtain design toggle rates and applied to the power reporting tool. The placement and routing impacts and simulation toggle rates shall increase the power analysis accuracy comparisons of the designs.

5 Related Work

Other research groups have proposed numerous hardware accelerators for CNNs in both FPGA \([3, 9]\) and ASIC \([1]\). Gupta et al. \([5]\) show increased efficiency in a hardware accelerator of a 16-bit fixed-point representation using stochastic rounding without loss of accuracy. Zhang et al. \([2]\) deduce the best CNN accelerator taking FPGA requirements into consideration and then implement the best on an FPGA to demonstrate high performance and throughput. Chen et al. \([1]\) design an ASIC accelerator for large scale CNNs focusing on the impact of memory on the accelerator performance.

Chen et al. \([2]\) address the problem of data movement which consumes large amounts of time and energy. They focus on data flow in the CNN to minimize data movement by reusing weights within the hardware accelerator to improve locality. This was implemented in ASIC and power and implementation results compared showing the effectiveness of weight reuse in saving power and increasing locality.

Han et al. \([9]\) have proposed an Efficient Inference Engine which builds on their ‘Deep compression’ \([7]\) work to perform inferences on the deeply compressed network to accelerate the weight-shared matrix-vector multiplication. This accelerates the classification task whilst saving energy when compared to Central Processing Unit (CPU) or Graphics Processor Unit (GPU) implementations.

Both weight sharing \([6, 7]\) and weight reuse \([2]\) reduce redundant data movement through different but complementary approaches. This paper builds on these ideas to reduce the circuit complexity of the underlying MAC units.

6 Conclusion

In this paper we propose a novel low-complexity MAC unit to exploit weight-sharing in CNN accelerators. The gate count and area for the 16-PAS-4-MAC is significantly lower when compared with the 16-MAC. We also found better power efficiency with the 16-PAS-4-MAC. CNN accelerators can contain hundreds or thousands of MAC units. We believe that our reduced-complexity design offers the possibility of adding more parallel MAC units to ASIC hardware CNN accelerators within a given clock speed, logic area and power budget.

References

[1] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine Learning: ASPLOS, 2014.
[2] Y. Chen, J. Emer, V. Sze, Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks: IEEE International Solid-State Circuits Conference (ISSCC), 2016.
[3] M. Füster, Faster Integer Multiplication: ACM Symposium on Theory of Computing, June 1113, 2007, San Diego, California, USA
[4] S. Gangadharan, S. Churiwal, Constraining Designs for Synthesis and Timing Analysis - A Practical Guide to Synopsis Design Constraints (SDC): Springer, 2013.
[5] S. Gupta, A. Agrawal, K. Gopalakrishnan, N. Narayanan, Deep Learning with Limited Numerical Precision: CoRR, vol. abs/1502.02551, 2015.
[6] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, W. Dally, EIE: Efficient Inference Engine on Compressed Deep Neural Network.
[7] S. Han, H. Mao, W. Dally, Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding: ICLR, 2016.
[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going Deeper with Convolutions: CVF, 2015.
[9] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks: ACM, 2015.