Scaling Binarized Neural Networks on Reconfigurable Logic

Nicholas J. Fraser[*‡], Yaman Umuroglu[*†], Giulio Gambardella[*], Michaela Blott[*],
Philip Leong[‡], Magnus Jahre[†] and Kees Vissers[*]
[*]Xilinx Research Labs; [†]Norwegian University of Science and Technology; [‡]University of Sydney
nfraser@xilinx.com, yamanu@idi.ntnu.no

ABSTRACT

Binarized neural networks (BNNs) are gaining interest in the deep learning community due to their significantly lower computational and memory cost. They are particularly well suited to reconfigurable logic devices, which contain an abundance of fine-grained compute resources and can result in smaller, lower power implementations, or conversely in higher classification rates. Towards this end, the FINN framework was recently proposed for building fast and flexible field programmable gate array (FPGA) accelerators for BNNs. FINN utilized a novel set of optimizations that enable efficient mapping of BNNs to hardware and implemented fully connected, non-padded convolutional and pooling layers, with per-layer compute resources being tailored to user-provided throughput requirements. However, FINN was not evaluated on larger topologies due to the size of the chosen FPGA, and exhibited decreased accuracy due to lack of padding. In this paper, we improve upon FINN to show how padding can be employed on BNNs while still maintaining a 1-bit datapath and high accuracy. Based on this technique, we demonstrate numerous experiments to illustrate flexibility and scalability of the approach. In particular, we show that a large BNN requiring 1.2 billion operations per frame running on an ADM-PCIE-8K5 platform can classify images at 12 kFPS with 671 μs latency while drawing less than 41 W board power and classifying CIFAR-10 images at 88.7% accuracy. Our implementation of this network achieves 14.8 trillion operations per second. We believe this is the fastest classification rate reported to date on this benchmark at this level of accuracy.

1. INTRODUCTION

Convolutional neural networks (CNNs) provide impressive classification accuracy in a number of application domains, but at the expense of large compute and memory requirements [17]. A significant body of research is investigating compression techniques combining numerous approaches such as: weight and synapse pruning; data compression techniques such as quantization, weight sharing and Huffman coding; and reduced precision with fixed point arithmetic [9, 11, 12]. Recently, an extreme form of reduced precision networks, known as BNNs [6], have gained significant interest as they can be implemented for inference at a much reduced hardware cost. This is due to the fact that multipliers and accumulators become XNORs and popcounts respectively, and both are significantly lighter in regards to resource and power footprint. For example, a KU115 offers 483 billion floating point operations per second (GFLOPS) compared to 46 trillion operations per second (TOPS) for binary synaptic operations. This is visualized in the roofline models in Figure 4 which illustrates theoretical peak performance for numerous reduced precision compute operations.[1] Furthermore, the model size is greatly reduced and typically small enough to fit in on-chip memory (OCM), again reducing power, simplifying the implementation and providing much greater bandwidth.

FINN [25] describes a framework for mapping BNNs to reconfigurable logic. However, it focuses on BNNs for embedded applications and as such, the results reported are for smaller network sizes running on an embedded platform. In this work, we briefly summarise FINN and analyse it from the perspective of scaling to larger networks and devices, such as those targeted for data centers. Firstly, we focus on several technical issues that arise when scaling networks on FINN including: BRAM usage, throughput limitations and resource overheads. We also identify several properties of CNN layers which make them map to FINN more efficiently. Our results, measured on an ADM-PCIE-8K5 platform [2], show that indeed very high image classification rates, minimal latency with very high power efficiency can be achieved by mapping BNNs to FPGAs, even though improvements may be made. Secondly, we highlight an issue of padding, a common feature of large CNNs, which may cause significant hardware overheads. We propose an alternative form of padding, which maps more efficiently to reconfigurable logic. Specifically, the contributions of this work are: 1) measured performance results for large-scale networks on an ADM-PCIE-8K5 board; 2) an analysis of FINN for large-scale problems, highlighting some bottlenecks as well as proposing solutions; and 3) a form of padding, which achieves high accuracy while also maintaining a binary datapath.

2. BACKGROUND

A great deal of prior work on mapping neural networks

PARMA-DITAM '17 , January 25 2017, Stockholm, Sweden

ISBN 978-1-4503-4877-5/17/01…$15.00

DOI: http://dx.doi.org/10.1145/3029580.3029586

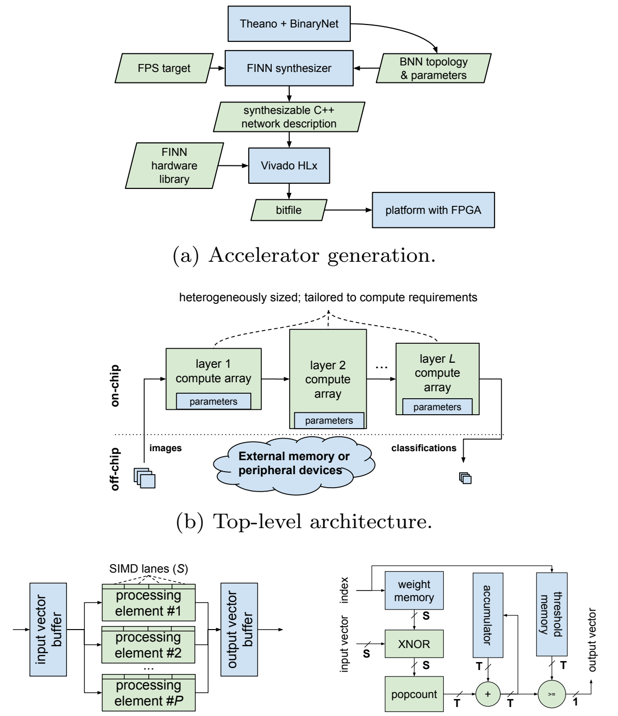
[1]Assuming 70% device utilization, 250 MHz clock frequency and 178 LUTs and 2 DSPs per average floating point operation, and 2.5 LUTs per binary XNOR-popcount operation.

to hardware exist for FPGAs, GPUs and ASICs to help increase inference rate or improve energy efficiency. We refer the reader to the work by Misra and Saha [18] for a comprehensive survey of prior works. In general we distinguish four basic architectures: 1) a *single processing engine*, usually in the form of a *systolic array*, which processes each layer sequentially [3, 5, 19, 28]; 2) a *streaming architecture* [1, 26], consisting of one processing engine per network layer; 3) a *vector processor* [8] with instructions specific to accelerating the primitives operations of convolutions; and 4) a *neurosynaptic processor* [7], which implements many digital neurons and their interconnecting weights. Significant research investigates binarization of neural networks whereby either input activations, synapse weights or output activations or a combination thereof are binarized. If all three components are binary, we refer to this as *full binarization* [15]. If not all three components are binary, we refer to this as *partial binarization*. The seminal XNOR-Net work by Rastegari et al. [20] applies convolutional BNNs on the ImageNet dataset with topologies inspired by AlexNet, ResNet and GoogLeNet, reporting top-1 accuracies of up to 51.2% for full binarization and 65.5% for partial binarization. DoReFa-Net by Zhou et al. [29] explores reduced precision with partial and full binarization on the SVHN and ImageNet datasets, including best-case ImageNet top-1 accuracies of 43% for full and 53% for partial binarization. Finally, the work by Courbariaux et al. [6] describes how to train fully-connected and convolutional networks with full binarization and batch normalization layers, reporting competitive accuracy on the MNIST, SVHN and CIFAR-10 datasets. All BNNs used in this work are trained by a methodology based on the one described by Courbariaux et al. [6], and unset bits represent a numerical -1 value while set bits represent a +1. The downside to the high performance characteristics of BNNs is a small drop in accuracy, in comparison to floating point networks. Improving the accuracy for reduced precision CNNs is an active research area in the machine learning community and first evidence shows that accuracy can be improved by increasing network sizes [22].

3. BNNs ON RECONFIGURABLE LOGIC

This work builds on top of FINN [25], a framework for building scalable and fast BNN inference accelerators on FPGAs. FINN is motivated by observations on how FPGAs can achieve performance in the TOPS range using XNOR–popcount–threshold datapaths to implement the BNNs described by Courbariaux et al. [6]. Given a trained BNN and target frame rates, FINN follows the workflow in Figure 1a to compose a BNN accelerator from hardware building blocks. In more detail, a given network topology and model retrieved through Theano [24], together with design targets in form of resource availability and classifcation rate, is processed by the synthesizer which determines the scaling settings and produces a synthesizable C++ description of a heterogeneous streaming architecture.[2] The top-level architecture is exemplified in Figure 1b and has two key differentiators compared to prior work on FPGA CNN accelerators. First, all BNN parameters are kept in OCM, which greatly increases

[2]To achieve portability, we chose a commercial high level synthesis tool, Vivado High-Level Synthesis (HLS) [27], for the implementation. The tool enables faster development cycles via high-level abstractions, and provides automated pipelining to meet the clock frequency target.



(a) Accelerator generation.

(b) Top-level architecture.

(c) Building block (MVTU). (d) MVTU datapath.

Figure 1: FINN workflow and architecture, reproduced from [25].

arithmetic intensity, reduces power and simplifies the design. Furthermore, one streaming compute engine is instantiated per layer, with resources tailored to fit each layer's compute requirements and the user-defined frame rate. Compute engines communicate via on-chip data streams and each produces and consumes data in the same order with the aim of minimizing buffer requirements in between layers. Thereby each engine starts to compute as soon as the previous engine starts to produce output. In essence, we build a custom architecture for a given topology rather than scheduling operations on top of a fixed architecture, as would be the case for typical systolic array based architectures, and avoid the "one-size-fits-all" inefficiencies and reap more of the benefits of reconfigurable computing.

3.1 The Matrix–Vector–Threshold Unit

In more detail, the key processing engine in FINN is the Matrix–Vector–Threshold Unit (MVTU) as illustrated in Figure 1c, which computes binarized matrix-vector products and compares against a threshold to generate a binarized activation. Convolutions are *lowered* [4] to matrix–matrix multiplications, using a Sliding Window Unit (SWU) (described further in Section 4.2) to generate the image matrix and the MVTU to carry out the actual arithmetic. The SWU generates the same vectors as those in [4] but with the elements of the vector interleaved to reduce and simplify memory accesses and to avoid the need for data transposition between layers. Internally, the MVTU consists of an input and output buffer, and an array of P Processing Elements (PEs), shown in Figure 1d, each with a number of SIMD lanes, S. The synapse weight matrix to be used is kept in OCM distributed between PEs, and the input images stream through the MVTU as each one is multiplied with the matrix. Each PE receives exactly the same control signals and input vector data, but multiply-accumulates the input with a different part of the matrix. A PE can be thought of as a hardware neuron capable of processing S synapses per clock cycle. Finally, the MVTU architectural template can also support partial binarization for non-binarized outputs and inputs. Removing the thresholding stage provides non-binarized outputs, while using regular multiply-add instead of XNOR-popcount can handle non-binarized inputs. These features are used in the first and last layers of networks that process non-binary input images or do not output a one-hot classification vector.

3.2 Folding

Depending on the use case, a neural network inference accelerator may have different throughput requirements in terms of the images classified per second (FPS). In FINN, FPS is controlled by the per-layer parameters P (number of PEs in an MVTU) and S (number of SIMD lanes in each PE). If the number of synapses, Y, connected to a neuron is greater than S, then the computation is *folded* across the PE, with the resulting PE producing an activation every $F^s = Y/S$ clock cycles. Similarly, if the number of neurons, X, in a layer exceeds P, then each PE is responsible for calculating activations for $F^n = X/P$ neurons. In total, it would take the MVTU $F^s \cdot F^n$ clock cycles to compute all its neuron activations. The MVTUs are then rate balanced by adjusting their P and S values to match the number of clock cycles it takes to calculate all required activations for each layer. As this is a balanced streaming system, the classification throughput FPS will be approximately F_{clk}/II, where F_{clk} is the clock frequency, and the II (Initiation Interval) is equal to the total folding factor $F^{\text{tot}} = F^s \cdot F^n$ cycles for a fully-connected layer. Note that convolutional layers have an extra folding factor, F^m, which is the number of matrix–vector products which need to be computed, i.e., the number of pixels in a single output feature map (OFM). Therefore, for convolutional layers the total folding factor is: $F^{\text{tot}} = F^s \cdot F^n \cdot F^m$.

3.3 BNN-specific Operator Optimizations

The methodology described in [6] forms the basis for training all BNNs in this paper. Firstly, in regards to arithmetic, we are using 1-bit values for all input activations, weights and output activations (full binarization), where an unset bit represents -1 and a set bit represents +1. Binary dot products result in XNORs with popcounts (which count the number of set bits instead of accumulation with signed arithmetic). Secondly, all BNN layers use batch normalization [13] on convolutional or fully connected layer outputs, then apply the sign function to determine the output activation. In [25] it is shown how the same output can be computed via thresholding, which combines the bias term, batch normalization and activation into a single function. Finally, the networks described in [6] perform pooling prior to activations, i.e. pooling is performed on non-binarized numbers, which are then batch normalized and fed into the activation function. However, as shown in [25], pooling can be equally performed after activation, once binarized, in which case it can be effectively implemented with the Boolean OR-operator.

4. PADDING FOR BNN CONVOLUTIONS

This section describes the improvements made to FINN in this work.

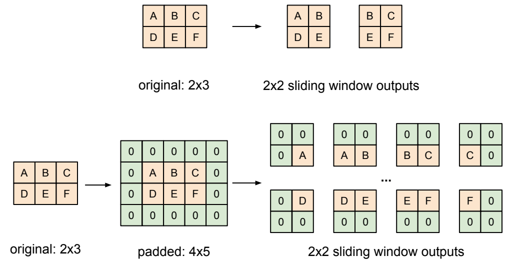


Figure 2: Convolution without (top) and with (bottom) padding.

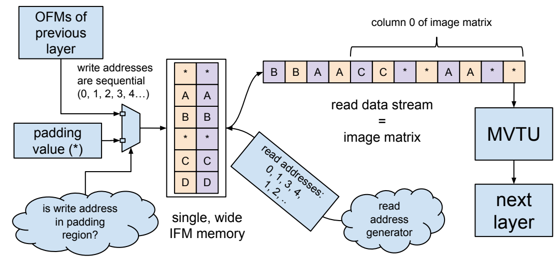


Figure 3: FINN SWU enhanced with streaming padding.

4.1 Padding using nonzero values

Zero-padding is commonly applied for convolutional layers in deep neural networks, in order to prevent the pixel information on the image borders from being "washed away" too quickly [14]. Figure 2 illustrates the sliding window outputs on the same image with and without padding. Observe that the pixels on the border (such as A and F) occur more frequently in the sliding window outputs when padding is used, thus preventing them from being "washed away" too quickly in the next layer.

A challenge arises for zero-padding in the context of BNNs with only $\{-1, +1\}$ arithmetic: there is no zero value defined. In fact, the original BinaryNet [6] paper uses ternary values $\{-1, 0, +1\}$ for the forward pass, with zeros used for padding. However, ternary values require two bits of storage, essentially doubling the OCM required to store values and the bitwidth of the datapath. Since FINN focuses on BNNs that fit entirely into on-chip memory of a single FPGA, minimizing the resource footprint is essential. Thus, a padding solution that avoids ternary values is preferable. A straightforward solution would be to use e.g. -1 as the padding value, and expect that the BNN learns weights which compensate for these values. Surprisingly, -1-padding works just as well as 0-padding according to our results, which are presented in Section 5.2.

4.2 Streaming padding for FINN

FINN lowers [4] convolutions to matrix-matrix multiplication of the filter weight matrix with the image matrix. The image matrix is generated on-the-fly by the SWU. Figure 3 illustrates how the FINN SWU is enhanced to support streaming padding for convolution layers. The key operational principle is the same as in FINN. Namely, a single, wide input feature map (IFM) memory is used to store the feature maps into OCM in the order they arrive, and the addresses that correspond to the sliding window pixels are read out. Padding is achieved by a multiplexer that chooses the data source for writing into the IFM memory. If the current write address falls into the padding region, the padding value (e.g. -1) is written into the memory; otherwise, an element from the output stream of the previous layer is written instead.

Table 1: Accuracy with different padding modes for CIFAR-10.

		Padding Mode		
		no-padding	0-padding	-1-padding
Scale	$\sigma = {}^1/_4$	75.6%	78.2%	79.1%
	$\sigma = {}^1/_2$	80.1%	85.2%	85.2%
	$\sigma = 1$	84.2%	88.6%	88.3%

5. EVALUATION

5.1 Experimental Setup

5.1.1 *BNN Topologies*

The network topologies used for our experiments are all based on the CNN topology described in [6], which we denote as cnn. This topology is inspired by the VGG16 network [21], which consists of three groups of (3x3 convolution – 3x3 convolution – 2x2 maxpooling) layers, and two fully-connected layers at the end. To explore how FINN performs on a range of network sizes, we introduce a scaling factor, σ, to scale the width of each layer, and denote the resulting topology as cnn(σ). Note that σ does not influence the number of layers in a network, it merely affects: 1) the number of neurons in each fully connected layer; and 2) the number of filters in each convolutional layer. Specifically, cnn(0.5) has half as many filters in each convolutional layer and half as many neurons in each fully connected layer, compared to the CNN described in [6]. In terms of convolutional networks, [25] only evaluated a single non-padded BNN topology ($\text{cnn}_{\text{NoPad}}({}^1/_2)$). In this work, we consider cnn(${}^1/_2$) as well as smaller (cnn(${}^1/_4$)) and bigger (cnn(1)) padded convolutional topologies to investigate how FINN scales.

In order to simulate a realistic use case, we consider an application with a fixed FPS requirement, i.e., real-time object recognition of a video stream. If one considers an 800 × 600 video stream at 25 FPS, which partitioned into tiles of 32 × 32 for classification. In order to classify the tiles in real-time, a classification rate of approximately 12 kFPS would be required. We use this image rate as our target for all experiments and adjust the number of PEs and SIMD accordingly in each layer of each design.

5.1.2 *The Platform*

The target board is an Alpha Data ADM-PCIE-8K5 which features a Xilinx Kintex UltraScale XCKU115-2-FLVA1517E FPGA (KU115). The KU115 offers 663k LUTs, 2160 BRAMs (36k) and 5520 DSPs and is running at 125 MHz for our experiments. The host machine is a IBM Power8 8247-21L with 80 cores at 3.69 GHz and 64 GB of RAM and it is running Ubuntu 15.04. In all experiments, all parameters are stored in OCM while the test images and the predicted labels are read from and written to the host memory directly. The provided resource counts include the PCI Express infrastructure used for moving data streams as well as the BNN accelerator. Although we are not able to provide per-experiment power measurements, the maximum power consumption observed for this board was 41 W on a board power dissipation benchmark test, and we expect that the real power dissipation values for BNN accelerators will be significantly lower than this.

5.2 Effects of Padding

To investigate how different padding modes affect accuracy, we trained a set of convolutional BNNs on the CIFAR-10

Table 2: Operations per image with different padding modes for CIFAR-10.

		Padding Mode		
		no-padding	0-padding	-1-padding
Scale	$\sigma = {}^1/_4$	30.4 M	78.5 M	78.5 M
	$\sigma = {}^1/_2$	118.9 M	310.3 M	310.3 M
	$\sigma = 1$	530.1 M	1234.1 M	1234.1 M

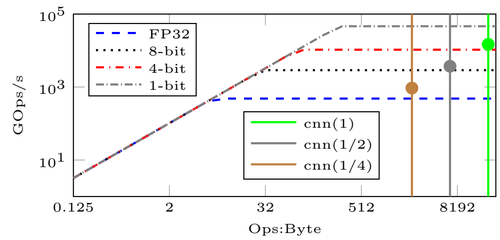


Figure 4: KU115 roofline with different datatypes.

dataset with different scaling factors (σ). The convolutions used are 3×3, so one pixel of padding is added on each border. The results are summarized in Table 1. As expected, using 0-padding improves accuracy by 4-5% compared to no-padding, indicating that the conventional wisdom on padding increasing accuracy also applies to BNNs. Furthermore, we can see that the accuracy of -1-padded networks are on par with the 0-padded ones of same scale. This suggests that BNNs are able to learn to compensate for the -1 values used for padding by adjusting the weight values and thresholds, and the accuracy benefits can be still obtained with a binary (as opposed to ternary) datapath.

It should also be noted that no-padding results in a significant reduction in the amount of operations per frame and the number of parameters. Thus, it is worthwhile to examine the computation versus accuracy tradeoffs in the context of padding. Table 2 lists the total number of XNOR-popcount operations necessary to classify one image using different padding modes and scaling factors. We can observe that the no-padding topology variant for the same scale factor requires $2 - 3\times$ less computation. However, this comes at a cost of higher error rate, and a smaller-but-padded network may be advantageous over a larger-but-not-padded network. For instance, cnn(${}^1/_4$) classifies at 79% accuracy using 78.5 M operations, whereas the $\text{cnn}_{\text{NoPad}}({}^1/_2)$ classifies at 80.1% accuracy using 118.9 M operations. Thus, cnn(${}^1/_4$) may be preferable due to its lower computational cost if a 1% drop in accuracy is acceptable for the use case at hand.

5.3 Scaling to Larger Networks

A results summary is shown in Table 3 which also shows the accuracy achieved by the implemented networks on a number of benchmark datasets. The new padded CNN results are provided in the top portion of Table 3, while key results from [25] are shown in the lower portion. Note that for

Table 3: Key performance and resource utilization results achieved by this work (top) and FINN (bottom) on a number of BNN topologies.

	Network	Device	LUT	BRAM	kFPS	GOps/s
Padded	cnn(${}^1/_4$)	KU115	35818	144	12.0	938
	cnn(${}^1/_2$)	KU115	93755	386	12.0	3,711
	cnn(1)	KU115	392947	1814	12.0	14,814
FINN [25]	$\text{cnn}_{\text{NoPad}}({}^1/_2)$	Z7045	54538	192	21.9	2,466
	mlp(${}^1/_{16}$)	Z7045	86110	130.5	12,361	8,265
	mlp(${}^1/_8$)	Z7045	104807	516.5	6,238	11,613
	mlp(${}^1/_4$)	Z7045	79097	398	1,561	9,086

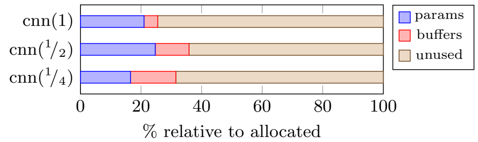


Figure 5: Utilization of allocated BRAM storage space.

comparison, scaled versions of the multilayer perceptrons (MLPs) consisting only of fully-connected layers described in [6] are also shown and denoted as mlp(σ).

We can see that larger networks scale well to larger FPGAs, with our best designs achieving 14.8 TOPS and 671 μs image classification latency. Furthermore, even with the largest network tested, all model parameters fit within OCM of the KU115 and thus avoids potential bottlenecks on external memory access. However, if we were to attempt a larger network (such as cnn(2)) the design would no longer fit in OCM without also reducing the frame rate. This is discussed further in Section 5.3.1.

While the results described in Table 3 represent state-of-the-art in terms of image classification rates and energy efficiency, it is still work in progress. Our best raw performance number (14.8 TOPS) outperforms that of the smaller FPGA device used in FINN [25] (11.6 TOPS), which is no surprise. However, the MLPs shown in [25] do achieve performance figures closer to the theoretical peak of the device. This is mostly due to the simplicity of MLPs versus CNNs. Figure 4 shows the estimated peak performance of the KU115 with vertical lines indicating the arithmetic intensity of the 3 CNN networks and coloured markers indicating actual performance of FINN. We can see that our implementations still fall below the KU115's theoretical peak. We expect that with planned improvements, including those in Section 5.3.1, significant performance gains can still be achieved. However it should be noted, that the largest design cnn(1) shown in Table 3 requires 1.2 billion operations (GOP) per frame, which is similar in computational requirements to the popular AlexNet [16] which requires 1.45 GOP per frame. In comparison the GPUs, the NVidia Titan X can achieve 3.2 kFPS at 227 W for AlexNet inference, compared to 12 kFPS at less than 41 W on the KU115 FPGA.[3] It should be noted that these figures are in terms of 32-bit floating point operations, as opposed to the binarized ones discussed in this work. However, high accuracy has been achieved by fully binarized [10] and partially binarized [29] versions of AlexNet and we expect to be able to achieve high performance on such networks.

5.3.1 *BRAM Efficiency*

Since FINN currently focuses on BNNs that fit entirely onto the on-chip memory of a single FPGA, making the most out of the available on-chip memory is essential. Figure 5 illustrates how much of the allocated BRAM space (as reported by Vivado) is actually utilized by the accelerator. The two largest contributors to BRAM usage in FINN are the network parameters (BNN weights and thresholds), and stream buffers (such as FIFOs and input-output buffers), which are shown with different colors in the bar chart. As can be expected, the majority of the utilized storage is for weights, although the streaming buffers occupy roughly equal storage for cnn($^1/_4$) since there are not as many parameters.

[3] https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf

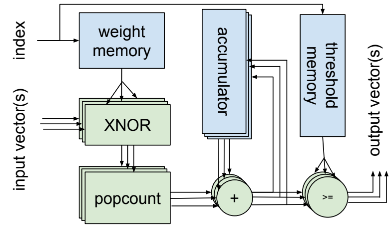


Figure 6: Datapath for matrix–multiple vector product.

A bigger concern is that on average only ~22% of the storage space in the allocated BRAMs is actually used. For scaling to even larger networks, this under–utilization could constitute a problem as synthesis will fail trying to allocate more BRAMs than is available in the FPGA. Further analysis into this issue revealed that this is a consequence of how convolutions are currently handled in FINN. Recall that the total folding factor is $F^{\text{tot}} = F^s \cdot F^n \cdot F^m$ for a convolution layer. The F^m folding factor here arises due to implementing matrix–matrix products as a sequence of matrix–vector products Unlike F^s and F^n, F^m is currently not controllable, since only one matrix–vector product is computed at a time in each MVTU. When high FPS is desired, the initiation interval must be minimized, which can only be achieved by small values F^n and F^s since F^m is constant. This requires creating many PEs and SIMD lanes operating in parallel, each of which have their own weight and threshold memories operating independently. However, this causes the weight matrix to be split and distributed into many small pieces, thus causing the observed storage under–utilization.

One way of addressing this problem would be enabling control over the F^m parameter by enhancing the MVTU to enable multiplying the same matrix by multiple vectors in parallel. In this manner, fewer PEs and SIMD lanes could be instantiated, each working on a larger portion of the weight matrix and utilizing BRAM storage better. Figure 6 shows how the MVTU datapath could be enhanced to support multiple vectors, broadcasting the same data from the weight memory to multiple XNOR-popcount-accumulate datapaths. Note that only the datapath is duplicated; the weight and threshold memories have a single copy. We leave further investigation of the matrix–multiple vectors for future work.

6. CONCLUSION

In this work, we explored the scaling of BNNs on large FPGAs using the FINN framework. We highlight an issue with padding in convolutional layers in BNNs described in [6] which would cause them to require a 2-bit datapath. We show that a small modification to padding (padding with -1 values) improves accuracy over no-padding and is comparable to 0-padding, while still allowing networks to maintain a binary datapath. We found that high performance for large networks can be attained, with our highest demonstrated performance achieving 12 kFPS at less than 41 W of board power and 14.8 TOPS of raw computational performance. When scaling to large networks, we also show that the efficiency of BRAM usage in FINN is low, and propose an architectural modification which would allow for better BRAM utilization. Alternatively, if a higher number of smaller BRAMs were available on FPGAs devices, this would allow FINN to better exploit the available resources.

For future work, we will further enhance the FINN framework to support partial binarization, and different kinds of convolutional layers, such as inception layers [23] and fire-modules [12]. The architectural improvements, described in Section 5.3.1 will be implemented to further improve the BRAM usage efficiency of architectures produced by FINN. Further networks which have been trained on larger datasets, i.e., ImageNet, will also be implemented. Finally, better power measurements will be attained rather than using "worst-case" power dissipation values.

References

[1] H. Alemdar, N. Caldwell, V. Leroy, A. Prost-Boucle, and F. Pétrot. Ternary Neural Networks for Resource-Efficient AI Applications. *CoRR*, abs/1609.00222, 2016.

[2] Alpha Data. *ADM-PCIE-8K5 Datasheet*, 1.3 edition, 9 2016.

[3] R. Andri, L. Cavigelli, D. Rossi, and L. Benini. YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights. *CoRR*, abs/1606.05487, 2016.

[4] K. Chellapilla, S. Puri, and P. Simard. High performance convolutional neural networks for document processing. In *Proc. ICFHR*. Suvisoft, 2006.

[5] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proc. ACM/IEEE ISCA*. IEEE, 2016.

[6] M. Courbariaux and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[7] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, et al. Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing. *CoRR*, abs/1603.08270, 2016.

[8] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun. CNP: An FPGA-based processor for convolutional networks. In *Proc. IEEE FPL*, pages 32–37. IEEE, 2009.

[9] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman coding. *CoRR*, abs/1510.00149, 2015.

[10] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016.

[11] F. N. Iandola, K. Ashraf, M. W. Moskewicz, and K. Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. *CoRR*, abs/1511.00175, 2015.

[12] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 1MB model size. *CoRR*, abs/1602.07630, 2016.

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, pages 448–456, 2015.

[14] A. Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition.

[15] M. Kim and P. Smaragdis. Bitwise neural networks. *CoRR*, abs/1601.06071, 2016.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1097–1105, 2012.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.

[18] J. Misra and I. Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 74(1–3):239–255, 2010.

[19] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. Chung. Accelerating deep convolutional neural networks using specialized hardware, February 2015.

[20] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*, 2016.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[22] W. Sung, S. Shin, and K. Hwang. Resiliency of deep neural networks under quantization. *CoRR*, abs/1511.06488, 2015.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. IEEE CVPR*, pages 1–9, 2015.

[24] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688, May 2016.

[25] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proc. ACM/SIGDA ISFPGA*, 2017.

[26] S. I. Venieris and C.-S. Bouganis. fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs. In *Proc. IEEE FCCM*, pages 40–47. IEEE, 2016.

[27] Xilinx Inc. Vivado design suite user guide: High-level synthesis. *White Paper*, 2016.

[28] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proc. ACM/SIGDA ISFPGA*, pages 161–170. ACM, 2015.

[29] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.