Cross-Layer CNN Approximations for Hardware Implementation

Karim M. A. Ali[1], Ihsen Alouani[2], Abdessamad Ait El Cadi[1], Hamza Ouarnoughi[1], and Smail Niar[1]

[1] LAMIH, Université Polytechnique Hauts-de-France, France, {karim.ali, abdessamad.aitelcadi, hamza.ouarnoughi, smail.niar}@uphf.fr
[2] IEMN, Université Polytechnique Hauts-de-France, France, ihsen.alouani@uphf.fr

Abstract. Convolution Neural Networks (CNNs) are widely used for image classification and object detection applications. The deployment of these architectures in embedded applications is a great challenge. This challenge arises from CNNs' high computation complexity that is required to be implemented on platforms with limited hardware resources like FPGA. Since these applications are inherently error-resilient, approximate computing (AC) offers an interesting trade-off between resource utilization and accuracy. In this paper, we study the impact on CNN performances when several approximation techniques are applied simultaneously. We focus on two of the widely used approximation techniques, namely quantization and pruning. Our experimental results showed that for CNN networks of different parameter sizes and 3% loss in accuracy, we can obtain up to 27.9% - 47.2% reduction in computation complexity in terms of FLOPs for CIFAR-10 and MNIST datasets.

Keywords: CNNs · FPGA · Approximate Computing

1 Introduction

Artificial Intelligence (AI) has recently been used in several domains for different purposes. Computer vision is one of the domains where AI is widely applied, especially using Convolutional Neural Networks (CNN). Image classification, image segmentation, and object detection are among the functions where CNNs are able to obtain high accuracy. However, high CNNs efficiency is gleaned at the cost of high algorithm complexity.

When constrained-resource systems are used to support such algorithms, such as for edge processing and IoT, memory and processing-demands reduction techniques need to be used. In this case, FPGA-based architectures are among possible alternative to energy-hungry based GPUs systems. FPGA solutions provide, in general, a superior trade-off of performance and accuracy over GPU-based systems for CNNs. However, designers need to have tools to explore the impacts of the different optimization techniques to reduce the high memory and processing needs of CNNs.

Optimizing the implementation of CNNs on FPGA is the focus of a large amount of research projects [1].

The problem is tackled at two levels. The first level concerns the CNN hardware implementation. The aim here is to tune the hardware architecture in order to improve the CNN performances of the inference phase. To apply such optimizations a strong background on the hardware platform is required. This is not always the case of AI and CNN experts. The second optimization level concerns the algorithmic aspect of the CNN. Its purpose is to reduce the CNN complexity while keeping an acceptable level of accuracy. Such a work requires a background of the CNN internal structure in order to decide which optimization can be applied. While different approximation techniques are used in the literature, the optimization of their use to reach efficient trade-offs is still an open research direction.

Approximation techniques for convolution neural networks can be classified into: (1) At the architectural-level: They are related to the convolution network architecture like pruning. (2) At the data-level: They are related to data like quantization. (3) At the computational-level: They are related to the computations like approximated multipliers or multiplier-less convolutions. These approximations are presented in literature independently from each other. In this paper, our objective aims to study the effect of applying two or more of them to the same neural network under a certain accuracy constraint. As an example, we presented the guidelines for how to apply both quantization and pruning approximations to convolution neural networks. We can summarize our contributions in this paper as following:

1. We proposed a reconfigurable hardware architecture for CNN inference.
2. We applied approximations to different size CNNs at two different levels: (i) at the data-level (quantization). (ii) at the architectural-level (pruning).
3. We studied the effect of applying cross-layer approximations on CNN network accuracy, hardware utilization, computation complexity,....

The remainder of this paper is organized as follows. Section II presents a literature review of approximate computing techniques applied on CNNs and their hardware implementation. Section III presents our hardware architecture dedicated to CNN inference. Section IV details the different approximate computing techniques proposed for CNN optimization. Section V discusses and compares the obtained results before and after applying our approaches. Section VI concludes the paper by giving an overview and the future perspectives of the work.

2 Related Works

Motivated by the challenge of overcoming the implementation constraints of deep neural networks (DNNs), three main approximation levels have been studied in the literature.

The first is the data-oriented approximation by reducing precision of operands. Practically, the process aims at minimizing the error between the quantized and

the raw data. The precision is correlated to the number of quantization levels and consequently to the number of bits required to represent the data. The simplest quantization approach consists of a linear mapping with uniform distance between each quantization level. It usually consists of converting values from floating point to an N-bit fixed point number. Authors in [13] reduce the weights bitwidth to 8 bits and the activation to 10. In [7], both weights and activation can reach 8-bits with fine-tuning. In [6], authors manage to reduce even more aggressively the data bitwidth. By introducing the concept of binary weights (-1 and 1), the multiply operation is reduced to addition and subtraction only. The same idea is extended in [5] by using binary parameters, thereby reducing the MAC operation to an XNOR. However, these two approaches have a dramatic accuracy loss of 19% and 29.8%, respectively [16].

While these works rely on linear quantization with uniformly spaced out values, the weights and activations distributions are not uniform [8, 14]. For example, in [7, 14], weights are quantized to powers of two. Consequently, the multiply operation is substituted by a bitshift operation. In [3], authors suggest weight sharing. The approach consists of assigning a single value to different weights in order to reduce the number of unique weights by filter.

Besides the approach of tuning the data precision, a plethora of works in literature has focused on reducing the network size and the number of performed operations. The second main approximation level is the network-oriented approximation by reducing the network size and optimizing the number of operations. This includes techniques such as compression, pruning and compact network architectures.

The sparsity of the rectified linear unit (ReLU) output activation is exploited in [4] to reduce memory access, particularly to costly off-chip memory access. The proposed reconfigurable hardware skips reading the weights and performing the MAC for zero-valued activation thereby reducing energy cost by 45%. Authors in [2] go even further and instead of just gating the read and MAC operation, they suggest to skip the cycle to increase the throughput by 1.37×.

Networks are usually over-parameterized and a large amount of redundancy exists within their weights. Network pruning techniques such those proposed in [10, 18] aim at removing the redundancy. To maintain the primary accuracy level, aggressive network pruning techniques may require weights fine-tuning.

These techniques are particularly efficient in reducing CNNs size and complexity. However, since the used platforms are in a high abstraction level, they do not take into account FPGA intrinsic characteristics. To the best of our knowledge, this is the first study that explores **FPGA-dedicated** data-level and network-oriented approximations.

3 Hardware architecture

Figure 1 shows our hardware platform for accelerating CNN inference. The system is partitioned on both Processing System (PS) and Programmable Logic (PL). On PL, N processing elements (PE) are synthesized where each PE cal-

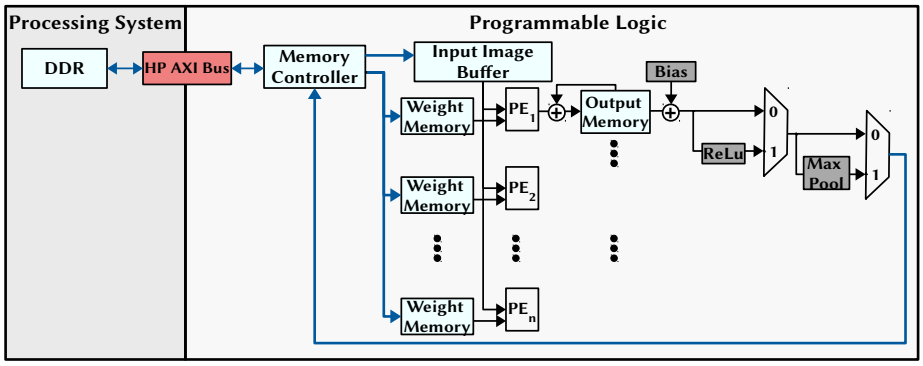


Fig. 1. Hardware architecture for CNN inference

culates one output channel. At the level of processing element, the input image is convolved in parallel with kernel of size k x k. The weights are loaded from *Weight Memory* while the input feature map is loaded from *Input Image Buffer* and distributed over the processing elements. There is no need to resynthesize the hardware architecture to run different CNN architectures of convolution layers of kernel size (k x k) or less; thus, the synthesized hardware architecture is CNN independent. The output of convolution is then accumulated in *Output Memory* until the output feature map pixel is calculated for all the input channels. The output pixel is rectified if the activation function (ReLU) is enabled while max pooling is calculated if MAX poll is enabled as well. Finally, the output pixel is written back to the DDR memory through high-performance AXI bus.

In the DDR memory, weights for all convolution layers are stored sequentially such that during runtime weights corresponding to a certain convolution layer are copied to the weight memory of the processing elements. The input/output feature maps for each layer are stored such that the output of one layer will be considered as the input for the successive layer. During CNN network configuration, three memory pointers are defined for each convolution layer. These pointers correspond to the address for weights, input and output feature maps memories.

On the processing system (PS), the architecture of CNN is defined as well as it controls the data transfer for weight memory and image buffer. For each convolution layer, we define the following parameters: (1) The size of the input feature map (*Win*, *Hin*). (2) The number of input channels (*CHin*). (3) The number of output channels (*CHout*). (4) If some features are enabled or not for that convolution layer like: padding, stride, activation and pooling functions.

If the number of output channels of the convolution layer is larger than the number of synthesized PEs (N), then for the same PE, it will be responsible for executing the output channels of order m, m+N, m+2N, ... where m is the order of the output channel. If the memory size for the weight memory is smaller than the total number of weights for a certain convolution layer, then the processing

is done in folds where in each fold, the weights corresponding to a number of output channels are loaded to the weight memory.

4 CNN Approximations

4.1 Pruning

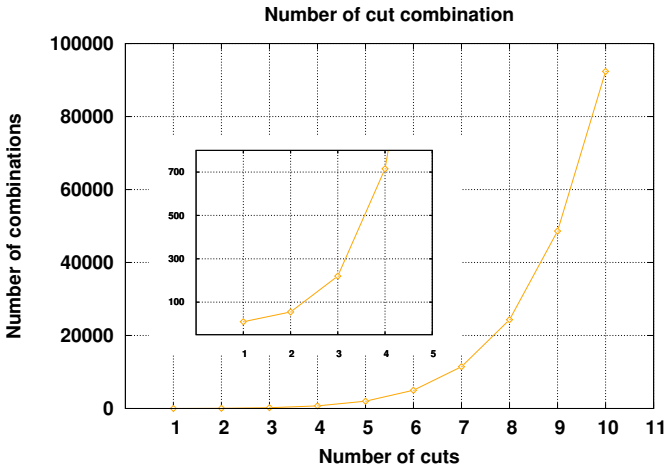


Fig. 2. Number of cut combinations increases exponentially

Pruning is used as an approximation technique to reduce the structure of the neural network. Using pruning decreases the computation complexity while having a small degradation in network accuracy. Several research works explored different granularity for pruning which could be classified into: fine-grained [11] [9], intra-kernel, kernel [12] and filter pruning [15]. In our case, we applied filter pruning because our CNNs are executed over hardware architecture of N processing elements, where each processing element processes one output channel. In general, pruning decreases the computation complexity but two conditions should be satisfied : (1) The granularity of pruning should be at the filter level. (2) The number of pruned filters should be multiples of the number of the processing elements implemented over the hardware architecture.

Our pruning criterion is cutting the output channels of the convolution layer of the smallest absolute sum of weights. Filter pruning is applied at layer-level where the number of pruned channels could be from one or different convolution layers. For example, for a CNN of m convolution layers and maximum number of cuts $= n$; then, we will have $\sum_{i=1}^{n} C\ {}_{m-1}^{i+m-1}$ possible combinations to find the pruned combination that satisfy our accuracy constraint. Figure 2 shows that

for a CNN of 10 convolution layers, the number of possible pruned combinations to be examined can grow exponentially. In practical, we are not able to test all possible pruning combinations to find the optimal one. Instead of that, we will apply greedy algorithm to converge rapidly to the solution.

The pruning algorithm is executed in four steps. In the first step, the absolute sum of weights for each convolution layer are sorted in ascending order. The second step is to prune each convolution layer independently by increasing the number of cuts till either we reach to the maximum predefined number of cuts or the accuracy loss is violated before that. After defining the maximum number of possible cuts that could be applied for each convolution layer independently, we can search in the new space for the cut combination that maximize our objective while the accuracy loss condition is still satisfied. In the third step, we formulate two matrices as depicted in Fig. 3. The first matrix is denoted as *Accuracy Loss Matrix* (A_{ij}) while the other one is denoted as *Decision Matrix* (X_{ij}) where i refers to the index of the convolution layer while j refers to the number of pruned output channels per convolution layer such that j is a multiple of the number of processing elements in the hardware architecture (N).(i.e. j = N, 2N, 3N, ...). For example, if we did 4N pruned channels to the fifth convolution layer then $X_{5,4} = 1$.

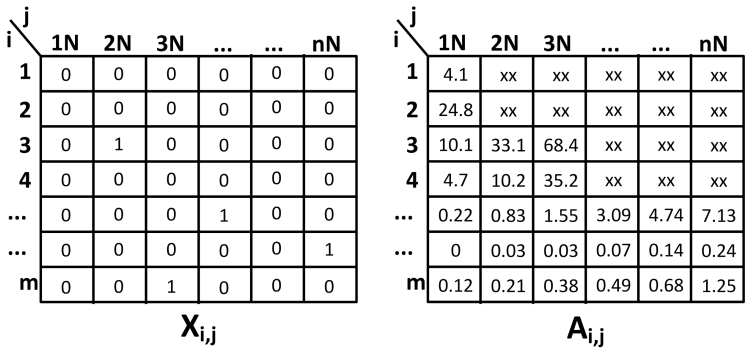


Fig. 3. Example for *Decision Matrix* (X_{ij}) and *Accuracy Loss Matrix* (A_{ij}). N denotes the number of processing elements while xx denotes that this cut is skipped because at that moment all the output channels of this layer will be removed.

Our objective is to minimize the number of network parameters while subjected to a constraint that the total accuracy loss is less than a certain threshold ($Acc_{threshold}$). Assume a CNN network of m convolution layers. i is the index of the convolution layer with I_i input channels, O_i output channels and kernel of size $k_i * k_i$. The total number of parameters for that convolution network can

be calculated from the following equation.

$$Num.\ of\ parameters = \sum_{i=1}^{m} O_i\ [k_i^2\ *\ I_i\ +\ 1] \tag{1}$$

Let P_i denotes the number of pruned output channels at convolution layer i. Then the total number of parameters after pruning can be calculated by the following equation.

$$Num.\ of\ parameters\ after\ pruning = \sum_{i=1}^{m} (O_i - P_i)[k_i^2(I_i - P_{i-1}) + 1] \tag{2}$$

where $P_0\ =\ 0$

We can formulate our problem as an optimization problem as following: Assume X_{ij} is the decision matrix and (A_{ij}) is the accuracy loss matrix for a network of (m) convolution layers (i.e. i = 1 to m) and maximum number of pruned channels $(n \times N)$ executed on hardware architecture of (N) processing elements (i.e. j = 1 to n).

Our objective function is to minimize the number of the network parameters:

$$\sum_{i=1}^{m} [O_i - \sum_{j=1}^{n} X_{i,j} \times N \times j]\ [k_i^2(I_i - \sum_{j=1}^{n} X_{i-1,j} \times N \times j) + 1] \tag{3}$$

and subjected to the following constraints:

$$\sum_{j=1}^{n} X_{ij} \leq 1\ \ \forall\ \ i \tag{4}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} * A_{ij} < Acc_{threshold} \tag{5}$$

$$\sum_{j=1}^{n} X_{0,j} * N * j = 0 \tag{6}$$

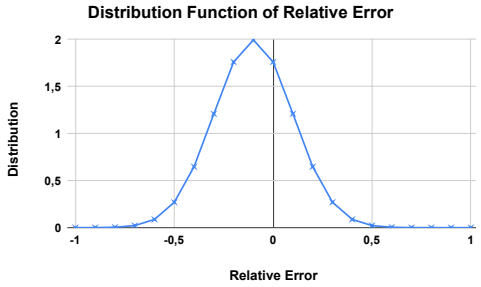
Equation (3) represents the objective function which is induced by substituting P_i in equation (2) by $\sum_{j=1}^{n} X_{i,j} \times N \times j$. While P_0 in equation (2) is formulated as a constraint as stated in equation (6). For each row in the decision matrix (X_{ij}) either we did pruning or not; therefore, the summation along the same row is either 0 or 1 which is formulated as a constraint as mentioned in equation (4). Equation (5) formulates that the estimated accuracy loss should be less than or equal to a certain threshold $Acc_{threshold}$ given by the designer.

For cross layer pruning, the metric we use for the accuracy loss is the sum of the independent accuracy losses calculated from $A_{i,j}$ matrix.

Let a_k and a_k^* be the measured accuracy and the estimated accuracy respectively for a given run k. The relative error is thereby given by the following Equation:

$$RE = \frac{a_k^* - a_k}{a_k} \tag{7}$$

Fig 4 shows that RE follows a normal distribution with a mean value of -0.1 and a standard deviation of 0.21. Moreover, we calculated the correlation between the measured and the estimated accuracy. This correlation is higher than 98%, thereby insuring the coherence of our estimated accuracy. Hence, we assume that our metric of summing independent accuracy losses can be used in the exploration phase. Nevertheless, the final results are validated based on real measured accuracy.



(a)

Fig. 4. Probability distribution function of the relative error.

In the fourth step, we run CPLEX optimizer to find the optimal pruning combination that respect the constraints and minimize the total number of parameters. The output of CPLEX is the pruning configuration and the estimated accuracy loss ($Acc_{estimated}$) due to that pruning. To validate our result, we test experimentally the pruning configuration by classifying the test dataset to obtain the real accuracy loss (Acc_{test}). If the obtained accuracy loss (Acc_{test}) is less than the defined accuracy threshold ($Acc_{threshold}$) then the pruning configuration is valid. Otherwise, the value for $Acc_{threshold}$ in equation (5) is updated with the obtained value by CPLEX ($Acc_{estimated}$) during that round as indicated in equation (8). After that, we rerun CPLEX for a second round to find the new optimal configuration. We keep iterating until the accuracy constraint is satisfied experimentally.

$$\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij} * A_{ij} \leq Acc_{estimated} \tag{8}$$

4.2 Quantization

Weights and activations for CNN are usually presented in floating-point. Applying approximations by representing them in fixed-point is an inevitable step to realize CNN over reconfigurable architectures to reduce both hardware and execution time. Fixed-point numbers are represented in Q-format where the number of integer and fractional bits are defined. For example, Q3.5 has 3 bits for integer

including the sign bit and 5 bits for the fractional number. It is important to choose the appropriate number of bits to avoid a significant degradation in the inference accuracy.

In our approach weight quantization is applied in 2 steps. Firstly, the integer number of bits is determined by scanning the weights for the maximum absolute value in addition to one bit for signed numbers as indicated in the following equation: Num of I bits = $Max(log_2\lceil \mid w \mid \rceil + 1)$ where w $\in$ weights.

Secondly, for the fractional number of bits, we examined the inference accuracy for a range of bits that extends from 0 to N-bits. Algorithm 1 explains how the number of fractional bits is chosen. For each weight, we quantize it at different number of bits dedicated for the fractional part (F). We calculated the ceiling and floor quantized values then the value of the minimum error difference is selected.

Input: Weights, maximum number of fractional bits (N)
Result: Number of fractional bits (F)
for $F \leftarrow 0$ **to** N **do**
  for $w \in weights$ **do**
    $w_1 = w * 2^F$
    $w_{ceil} = \lceil w_1 \rceil / 2^F$
    $w_{floor} = \lfloor w_1 \rfloor / 2^F$
    if $\mid w_{ceil} - w_1 \mid \leq \mid w_{floor} - w_1 \mid$ **then**
      $w_{quantized} = w_{ceil}$
    else
      $w_{quantized} = w_{floor}$
    end
  end
  for $i \leftarrow 0$ **to** *testimages* **do**
    if *detected class == truth class* **then**
      correct++
    else
      false++
    end
  end
end

Algorithm 1: Choosing the number of fractional bits for weight representation

Order of applying approximations. Sorting the weights of the output channels for each convolution layer is considered as a step for applying pruning. The output channels are sorted according to the absolute sum of their weights. Consequently, quantization should be firstly applied so that the quantized weights are sorted during pruning to allow correct results.

	CNN 1	**CNN 2**	**CNN 3**
Total num. of parameters	4,389,418	14,488,650	27,187,210
Num. of conv. layers	9	14	17
Accuracy for CIFAR-10	85.49	85.92	86.84
Accuracy for MNIST	99.02	99.09	99.1
Conv. (# CHin, # CHout, output image)	L1 (3, 32, 32x32)	L1 (3, 64, 32x32)	L1(3, 128, 32x32)
	L2 (32, 32, 30x30)	L2 (64, 64, 32x32)	L2, L3, L4 (128, 128, 32x32)
	L3 (32, 64, 30x30)	L3 (64, 128, 32x32)	L5, L6, L7 (128, 128, 30x30)
	L4 (64, 64, 28x28)	L4 (128, 128, 32x32)	L8 (128, 256, 30x30)
	L5 (64, 256, 14x14)	L5 (128, 128, 30x30)	L9 (256, 256, 30x30)
	L6 (256, 256, 12x12)	L6 (128, 128, 30x30)	L10 (256, 256, 28x28)
	L7 (256, 512, 6x6)	L7 (128, 256, 30x30)	L11(256, 512, 14x14)
	L8 (512, 512, 6x6)	L8 (256, 256, 30x30)	L12 (512, 512, 14x14)
	L9(512, 10, 1x1)	L9 (256, 256, 28x28)	L13(512, 1024, 14x14)
		L10 (256, 512, 14x14)	L14(1024, 1024, 12x12)
		L11 (512, 512, 12x12)	L15(1024, 512, 6x6)
		L12 (512, 768, 6x6)	L16(512, 512, 6x6)
		L13 (768, 768, 6x6)	L17(512, 10, 1x1)
		L14 (768, 10, 1x1)	

Table 1. CNN properties

5 Experimental Results

By comparing the distribution of weights and operations for different CNN models in the literature; we can easily conclude the following: (1) Fully connected layers has the large portion of the network's weights. (2) The convolution layers contribute to the large portion of operations. For example in VGG-16, the weights for the fully connected layers represent 90% of the total weight while the convolution layers contribute to 92% of the computation complexity.

During our experiments, we focused on studying the effect of approximations (quantization and pruning) on convolution layers. For that reason, we designed three convolution neural networks which are similar to the conventional networks like Alexnet or VGG but without having fully connected layers.

We trained three convolution networks CNN1, CNN2 and CNN3 on two datasets CIFAR-10 and MNIST. The training was held on a machine equipped by NVIDIA Quadro P5000 GPU card. Table 1 listed the properties of the three networks. For each CNN, we listed the total number of parameters, the number of convolution layers, the structure of the network and its accuracy to classify CIFAR-10 and MNIST datasets. Without using fully connected layers, we relied

Number of PE	Precision	SLICE	FF	LUT	BRAM (18K)	DSP
16	Floating point	22154	68912	67070	218	768
16	Fixed point	4071	9456	12327	186	173
32	Fixed point	5604	8465	16021	326	317
64	Fixed point	8699	10725	24829	614	605

Table 2. Resource utilization for CNN hardware architecture at different cores number

on max-pooling and non-padded convolutions to reduce the size of the input image.

The hardware architecture presented in Section 3 was synthesized by Vivado Design Suite 2015.2 on Xilinx Zynq 706 evaluation board [17]. As listed in Table 2, when the hardware architecture was synthesized for 16 processing elements at floating point precision, 85% of DSP was consumed (Max. DSP = 900) with 15% FF and 30% LUT utilization.

Firstly, we applied quantization to reduce the DSP utilization. For the integer part of the weights, we examined the weight parameters for the maximum and minimum values. After that, we could deduce the required number of bits to represent the integer part. For example, the minimum and maximum weight values while training CNN1 on CIFAR-10 were -1.22545 and 1.11254 respectively. Therefore, two bits were enough to represent the integer part including the sign bit.

For the fractional part of the weights, we could choose the correct number of bits by running Algorithm 1. Figure 5 depicted the accuracy while classifying CIFAR-10 and MNIST datasets. It showed the classification accuracy of the three networks CNN1, CNN2 and CNN3 by quantizing the weights at different fractional bit width extending from 0 to 16 bits. From the figure, we could deduce that 6 bits for the weight fractional part were enough to keep the same classification accuracy as it was while representing the weights in floating-point.

By synthesizing the hardware architecture for 16 processing elements with fixed-point representation, we were able to reduce the DSP utilization by 78%, FF by 86%, LUT by 81% and BRAM by 15%. This reduction in hardware resources gave us the possibility to duplicate the number of realized processing elements as listed in Table 2.

Secondly, we applied pruning to reduce the computation complexity. Channel pruning is applied at multiples of the number of processing elements otherwise some processors will be idle while the other will be utilized. To apply convolution layer pruning for an architecture containing 16 PEs means to prune the output channels in multiples of 16. For example, a pruning setup for CNN1 like following (0, 5, 10, 1, 2, 0, 0, 0, 0) means to prune 80 (5×16), 160 (10×16), 16 (1×16), 32 (2 ×16) output channels from convolution layers L8, L7, L6, and L5 respectively.

During our experiments, we searched for the pruning setup that had an accuracy loss less than 3%. Choosing that value was arbitrary and could vary from one application to another. As a first step, the accuracy loss matrix (A_{ij}) was

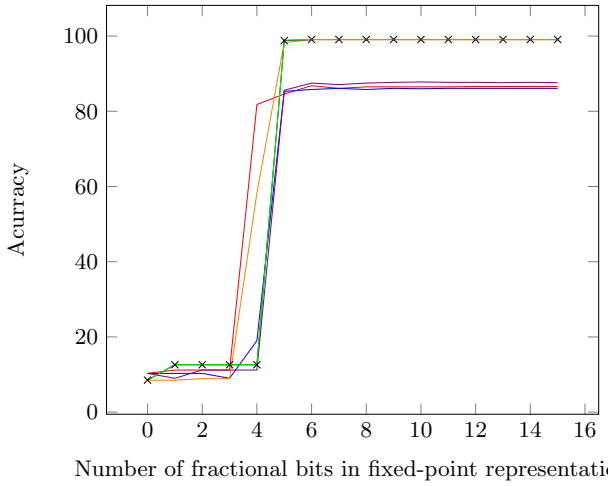


Fig. 5. Fixed-point inference accuracy for CNN1-Cifar10 ——, CNN2-Cifar10 ——, CNN3-Cifar10 ——, CNN1-MNIST ——, CNN1-MNIST —×— and CNN1-MNIST ——.

calculated for each convolution network. To achieve that step, we tested the maximum possible pruning that could be done for each convolution layer separately without exceeding 3% in accuracy loss. After that, by using CPLEX tool, the three CNN networks were modelled by using equations 3, 4, 5 and 6 to find the optimal pruning setup.

For CIFAR-10 dataset in Table 3, points #1, #3 and #7 represent CNN1, CNN2 and CNN3 before pruning at floating point precision. While points #2, #4 and #8 represent the pruning setup for them with accuracy loss of 2.89%, 2.6% and 2.9% and percentage decrease in computation complexity of 23.4%, 19.5% and 27.9% respectively. For the same CNN, the optimal pruning setup found by CPLEX can change by varying the number of synthesized processing elements (PEs) in the hardware architecture. For example for CNN3, points #8, #10, and #11 represent the pruning setup when it is executed on hardware architecture of 16, 32 and 64 processing elements respectively. We can notice that CNN3 achieves 27.9%, 25.6% and 21.8% decrease in computation complexity respectively. The reason behind is that each hardware architecture has a different multiple of pruned channels (i.e. multiples of 16 for 16-PEs, multiples of 32 for 32-PEs and multiples of 64 for 64-PEs). Therefore, fine pruning can be achieved on architectures of fewer PEs. For example, with PE=16, one cut will remove only 16 channels, while with PE=64, one cut will remove 64 channels at once which could violate our accuracy loss constraint.

The pruning step changes when floating-point weights are quantized. For instance, the pruning setup for CNN3 at point #8 did not achieve the same accuracy loss when its weights were quantized to Q2.5 at point #16. Therefore, we should take into consideration the impact of quantization. To achieve accuracy loss less than a certain objective; in our case, we fixed accuracy loss to $\leq 3\%$;

Image classification CIFAR-10 dataset						
Point	**Architecture**	**Pruning setup (L17, L16, L15,.... L3, L2, L1)**	**Num of param.**	**Acc. loss (%)**	**FLOP**	**Dec. ↓ (%)**
1	CNN1-16PE-FP	No Pruning	4389418	0	2.97E+08	0
2	CNN1-16PE-FP	(0,5,10,1,2,0,0,0,0)	2847466	2.89	2.27E+08	23.4
3	CNN2-16PE-FP	No Pruning	14488650	0	2.68E+09	0
4	CNN2-16PE-FP	(0,3,22,15,9,0,1,0,0,0,0,0,0,0)	7486282	2.6	2.158E+09	19.5
5	CNN2-16PE-6Bit	(0,3,22,15,7,0,0,0,0,0,0,0,0,0)	7712122	3	2.246E+09	16.2
6	CNN2-16PE-5Bit	(0,3,10,10,7,0,0,0,0,0,0,0,0,0)	10152330	2.7	2.364E+09	11.8
7	CNN3-16PE-FP	No Pruning	27187210	0	5.346E+09	0
8	CNN3-16PE-FP	(0,3,7,19,20,4,1,1,3,0,0,0,0,0,0,0,1)	16873418	2.9	3.853E+09	27.9
9	CNN3-16PE-FP	(0,3,5,20,20,1,7,0,1,3,0,0,0,0,0,0,0)	17028202	4.2	3.861E+09	27.8
10	CNN3-32PE-FP	(0,1,3,9,10,2,0,0,1,0,0,0,0,0,0,0,1)	17524234	1.7	3.978E+09	25.6
11	CNN3-64PE-FP	(0,1,1,4,5,1,0,0,0,0,0,0,0,0,0,0,0)	18259594	2.4	4.183E+09	21.8
12	CNN3-16PE-6Bit	No Pruning	27187210	0.22	5.346E+09	0
13	CNN3-16PE-6Bit	(0,3,3,20,21,1,7,0,1,2,0,0,0,0,0,0,1)	17225930	3	3.872E+09	27.6
14	CNN3-16PE-5Bit	No Pruning	27187210	2.17	5.346E+09	0
15	CNN3-16PE-5Bit	(0,1,3,19,22,1,7,0,1,2,0,0,0,0,0,0,1)	17353258	2.9	3.865E+09	27.7
16	CNN3-16PE-5Bit	(0,3,7,19,20,4,1,1,3,0,0,0,0,0,0,0,1)	16873418	7.5	3.853E+09	27.9
17	CNN3-32PE-5Bit	(0,0,1,9,11,0,3,0,0,1,0,0,0,0,0,0,1)	18080106	1.9	3.981E+09	25.5
18	CNN3-64PE-5Bit	(0,1,0,4,5,0,1,0,0,0,0,0,0,0,0,0,0)	19218122	2.7	4.256E+09	20.3
Digit recognition MNIST dataset						
19	CNN1-16PE-FP	No Pruning	4388842	0	2.97E+08	0
20	CNN1-16PE-6Bit	(0,19,8,5,0,0,3,0,0)	1923290	2.85	1.566E+08	47.23
21	CNN1-16PE-5Bit	(0,15,5,5,0,0,3,0,0)	2343882	2.85	1.71E+08	42.2
22	CNN2-16PE-FP	No Pruning	14487498	0	2.67E+09	0
23	CNN2-16PE-6-bit	(0,26,36,15,15,0,1,1,0,0,0,0,0,0)	4307498	3	1.94E+09	27.7
24	CNN2-16PE-5-bit	(0,25,10,10,15,0,0,0,0,0,0,1,1,1)	7432362	2.7	2.116E+09	21
25	CNN3-16PE-FP	No Pruning	27184906	0	5.343E+09	0
26	CNN3-16PE-6-bit	(0,3,20,30,35,0,0,0,2,1,1,0,0,0,0,0,5)	11753034	2.9	3.302E+09	38.2
27	CNN3-16PE-5-bit	(0,4,23,36,35,0,5,0,0,0,1,1,0,1,0,0,5)	10341322	3	3.216E+09	39.8

Table 3. Pruning setup for CNN1, CNN2 and CNN3 trained for CIFAR-10 and MNIST dataset

while combining quantization and pruning. Firstly, the convolution network is quantized then the pruning search process is run to find the optimal pruning setup. For example, the weights of CNN3 were quantized to Q2.5 (5-bit) and Q2.6 (6-bit) with accuracy loss of 0.22% and 2.17% at points #12 and #14 respectively. Then pruning is applied to the quantized networks to achieve total loss of 3% and 2.9% with decrease in FLOP by 27.6% and 27.7% at points #13 and #15.

Regarding computation complexity, pruned floating-point networks (point #8) are less in FLOP than pruned quantized one (points #13 and #15) by 0.5% and 0.31% respectively. In contrast, regarding hardware utilization, quantized implementations are smaller than floating-point one in average by 80% for LUT,

FF and DSP. Therefore, both quantization and pruning can be combined for better results.

For digit recognition MNIST dataset, pruned networks with weight quantization at Q2.5 and Q2.6 achieved reduction in computation complexity up to 42.2% and 47.23% for CNN1, up to 21% and 27.7% for CNN2 and up to 39.8% and 38.2% for CNN3 respectively.

6 Conclusion

Using approximation techniques to reduce CNN complexity is inevitable to implement these networks on embedded platforms such as FPGAs. While different approximation techniques are presented in the literature independently, we aimed in this paper to study the effect of applying more than one approximation to the same neural network. We presented a through FPGA-oriented exploration of both quantization and pruning to reduce the hardware cost and computation complexity with controlled loss in CNN accuracy. We succeeded to reduce hardware resources by 80%, computation complexity by 30% with accuracy loss less than 3%. As a future work, we will develop a tool for estimating the effect of different approximation techniques when applied to a certain CNN in terms of frame rate, accuracy and hardware cost. The tool will generate automatically the corresponding CNN structure with the hardware architecture required to achieve those objectives.

References

1. Abdelouahab, K., Pelcat, M., Sérot, J., Berry, F.: Accelerating CNN inference on fpgas: A survey. CoRR **abs/1806.01683** (2018), http://arxiv.org/abs/1806.01683
2. Albericio, J., Judd, P., Hetherington, T., Aamodt, T., Jerger, N.E., Moshovos, A.: Cnvlutin: Ineffectual-neuron-free deep neural network computing. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). pp. 1–13 (June 2016)
3. Chen, W., Wilson, J.T., Tyree, S., Weinberger, K.Q., Chen, Y.: Compressing neural networks with the hashing trick. In: Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37. pp. 2285–2294. ICML'15, JMLR.org (2015)
4. Chen, Y., Krishna, T., Emer, J.S., Sze, V.: Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits **52**(1), 127–138 (Jan 2017)
5. Courbariaux, M., Bengio, Y.: Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. CoRR **abs/1602.02830** (2016)
6. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 3123–3131. Curran Associates, Inc. (2015)
7. Gysel, P., Motamedi, M., Ghiasi, S.: Hardware-oriented approximation of convolutional neural networks. CoRR **abs/1604.03168** (2016), http://arxiv.org/abs/1604.03168

8. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. ICLR (2016)
9. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In: 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016)
10. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1. pp. 1135–1143. NIPS'15, MIT Press, Cambridge, MA, USA (2015)
11. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1. pp. 1135–1143. NIPS'15, MIT Press, Cambridge, MA, USA (2015)
12. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. arXiv (2017), http://arxiv.org/abs/1707.06168
13. Ma, Y., Suda, N., Cao, Y., Seo, J., Vrudhula, S.: Scalable and modularized rtl compilation of convolutional neural networks onto fpga. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL). pp. 1–8 (Aug 2016)
14. Miyashita, D., Lee, E.H., Murmann, B.: Convolutional neural networks using logarithmic data representation. CoRR **abs/1603.01025** (2016)
15. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient transfer learning. arXiv (2016)
16. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. CoRR **abs/1603.05279** (2016)
17. Xilinx: ZC706 Evaluation Board for the Zynq-7000 XC7Z045 All Programmable SoC User Guide
18. Yang, T., Chen, Y., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 6071–6079 (July 2017)