FPGA based Flexible Implementation of Light Weight Inference on Deep Convolutional Neural Networks

Shefa Dawwd
Department of Computer Engineering, University of Mosul, Iraq
shefa.dawwd@uomosul.edu.iq

Abstract: *Standard Convolution (StdConv) is the main technique used in the state of the art Deep Convolutional Neural Networks (DCNNs). Fewer computations are achieved if Depthwise Separable Convolution technique (SepConv) is used as an alternative. A crucial issue in many applications like smart cameras and autonomous vehicles where low latency is essential stems from deploying a lightweight and low cost inference models. An acceptable accuracy should be kept with tolerable computations and memory access load. A flexible architecture for different DCNN convolution types and models is proposed. The flexibility comes from the sharing of one memory access unit with different types of layers regardless of the selected kernel size, by multiplying each weight vector by local operators with variant aperture. Moreover, one depthwise computation unit can be used for both standard and pointwise layers. The learnable parameters are quantized to 8-bits fixed point representation and that gives very limited reduction of accuracy and a considerable reduction of the Field-Programmable Gate Array (FPGA) resources. To reduce processing time, inter layer parallel computations are performed. The experiment is conducted by using grey scale ORL database with shallow Convolutional Neural Network (CNN) and the colored Canadian Institute for Advanced Research 10 classes (CIFAR-10) database with DCNN, and a comparable accuracies of 93% and 85.7% are achieved respectively using very low cost of Spartan 3E and moderate cost of zynq FPGA platforms.*

Keywords: *Standard convolution, depthwise separable convolution, inference, deep convolutional neural networks, FPGA.*

Received December 31, 2023; accepted March 18, 2024
https://doi.org/10.34028/iajit/21/3/5

1. Introduction

The shallow Convolutional Neural Network (CNN) is usually used to classify image in an efficient performance. Rather than using traditional image processing techniques, CNN can process raw images or video frames in a productive way. When Deep Convolutional Neural Network (DCNN) is used, the performance in specific object recognition tasks sometime achieve the level of human [8]. The DCNN is proved to be an efficient choice to process image and video for different applications such as sign language recognition [20], character recognition [2], medical diagnosis [30], video surveillance [22], and etc. However, the better efficiency and performance comes at the expense of high computations and memory requirements. In most of embedded platforms or edge devices, where resources for computing, memory storage, and energy are constrained, DCNN is difficult to be implemented unless an efficient data and processing management is followed. To reduce the implementation cost, and preserve the parameters accuracy, the training is often offloaded using powerful computers provided with high performance multicore Central Processing Unit (CPU) and Graphics Processing Units (GPUs).While light inference model can only be embedded in the edge devices (such as mobile devices). An abstract inference model can then be used to match the application with devices of restricted size and latency.

This abstractness is usually achieved by two way: dimensionality reduction and parameters compression. Dimensionality reduction is the process of reducing the number of features in a dataset while retaining as much information as possible. This can be done to reduce the complexity of a model, improve the performance of a learning algorithm, or make it easier to visualize the data. On the other hand, parameters compression is used to compress the model parameters as long as accuracy effect is negligible. A considerable saving in cost is achieved by compressing of the pre-trained network using quantization [26] and pruning [23]. The quantization is used to reduce the word length gradually and the pruning is used to optimize the model by eliminating the weight connections or nodes that are close to zero form the model.

With DCNN of dozens or even close to a hundred of layers, using the above techniques are not sufficient to save cost effectively. Thus, the researchers searched for other mechanisms including dealing with the problem mathematically. One of the most active solutions in reducing the mathematical computations is the use of different type of convolution like SepConv [1, 19], where the Standard Convolution (StdConv) process is divided into two phases: depthwise and pointwise convolution. The intermediate results from depthwise to pointwise are stored temporarily in buffers [1]. Reducing the required number of computations can positively reflects to the time taken to perform the task, which in turn reduce the total latency. The MobileNet deep model [13] primarily uses SepConv and deals with latency and size optimization. However, if the network is already small, the SepConv may fail to properly learn during training due to the compactness of its parameters. This leads to accuracy degradation in many cases and makes the StdConv is preferable. The early versions of LeNet, AlexNet, Visual Geometry Group (VGG) and deep Residual Network (ResNet) models use StdConv type. On the other hand, some works take advantage of the favourable characteristics of both convolution types, for example, the ResNet model proposed in [27] is modified and uses full SepConv layers in its residual network.

In this paper, a low cost model that can be used in both of the mentioned convolution types and can address the reduction and compression issues is proposed. The proposed technique can be flexibly used for both convolution schemes StdConv and SepConv for small and large network sizes. The rest of paper is organized as follows: Section 2 presents a state-of-the-art design and implementation of DCNN models followed by problem statement which introduced in section 3. In section 4 a brief background of DCNN and its mathematical representation is introduced. The details of SepConv vs. StdConv are described in section 5, while the training methodology and the hardware architectural design are presented in sections 6 and 7 respectively. The experimental results are shown and discussed in section 8. Finally, most important conclusions and future works are summarized in sections 9 and 10 respectively.

2. Related Works

Two phases are involved in training a neural network: forward or inference phase and backward parameters adjustment phase. Accelerating either of these two phases leads to accelerate the training or inference separately. Or one could say: when training is accelerated, the inference will be implicitly taken into account and vice versa. In many system, the inference model is connected with larger system and used as an accelerator for forward learning phase [17]. The flexibility offered by CPU and GPU makes them preferable to perform training compared to Application-Specific Integrated Circuit (ASIC) or Field-Programmable Gate Array (FPGA). The capacity of using CPU does not offer the enough computations to implement an efficient large DCNN model. The GPU that use different development frameworks can be a good candidate to perform efficient training or inference of DCNN models [24] regardless of the network size. The technology that could outperform GPU in terms of energy and speed efficiency is the ASIC and FPGA, but with no re-programming flexibility [10]. Although, the power and speed provided by ASIC is superior to FPGA, the configurability feature makes the latter more flexible.

For lightweight CNNs, the SepConv is demonstrated to be very effective technique. An FPGA based MobileNet is the most famous example that use SepConv in its processing flow. Two dedicated computing engines for pointwise and depthwise convolutions are utilized in the work presented by Wu *et al*. [28] to train the MobileNet model on ZU2 and ZU9 MPSoC FPGAs. An inference FPGA based accelerator is implemented by Kuramochi and Nakahara [17] on Xilinx Alveo U50 FPGA platform. Since no direct dependencies are included among the DCNN layers, the pipeline techniques is efficiently used in their architecture.

Also, efficient and light weight implementation of StdConv of DCNN have been a subject of intensive research. Hou and Chen [12] use OpenCL-based zynq7 FPGA to improve the training of LeNet5. A good effect is achieved on term of efficiency and accuracy. Gilan *et al*. [9] accelerates the inference of AlexNet architecture using ZC706 FPGA evalution board. Their proposed architecture is fully configurable and flexible to perform different DCNN models. While Hadnagy *et al*. [11] deals with inference acceleration of LeNet model. To make a hardware friendlier, an optimization techniques that deals with reducing the word length size and pruning for AlexNet, VGG, ResNet and MobileNet is shown in the survey presented by Dhouibi *et al*. [7]. Also, a good review can be found in the survey of Dhilleswararao *et al*. [6] that discusses the description of specialized hardware based accelerators employed in training and/or inference of different DCNN models.

According to the survey work [10], the most contribution (99%) of operations and weights in all DCNN models is found in the convolution and fully connected layers, while 1% contribution is left to the ReLU and pooling layers. So the acceleration focuses on the convolution and fully connected layers. In this paper, we only concentrate on the convolution layers acceleration and for simplicity, the kernel and feature map of fully connected layer is assumed to be of size 1×1. A flexible depthwise separable processing unit based FPGA of reduced multiplications is designed for inference DCNN using either the SepConv or StdConv.

3. Problem Statement

As demonstrated in the preceding sections, various methods and architecture have been employed to minimize the expenses associated with DCNN. Yet, none of these approaches can effectively adapt to the diverse nature of deep models, extensive layers, processing techniques, and memory access. This paper introduces a roadmap for designing and implementing DCNNs in a flexible manner, aiming for low costs and minimal or negligible degradation of accuracy, as elaborated in the subsequent sections.

4. Deep Convolutional Neural Networks

The CNN is a network of succeeded convolutional layers arranged in pyramid architecture (Figure 1). Each layer alternates between convolution and down-sampling (pooling) layer. Each layer is represented by 2D array of neurons. Simple features are extracted in the first layer(s), then the complexity (filter size) and abstractness of feature map are gradually increased from layer to layer. The layer's input resolution are decreased until a limited set of specified complex features are extracted in the final convolutional layer. Flattening is used to convert all the resultant 2D arrays of features in the final layer into a single long continuous linear vector. The flattened matrix is fed as input to the fully connected layer(s) for image/frame classification [14]. A DCNN nowadays can make use of hundreds or even thousands of layers. The number of layers does not strictly distinguish between shallow and DCNN. However, any architecture more than two or three layers can be considered as deep. Two main types of convolution are used in DCNN: StdConv and SepConv. The StdConv is used in early CNN and later DCNN. The following equation dictates the convolution operation in this type:

$$\mathrm{Conv}(W,y)_{(i,j)} = \sum_{k,l,m}^{K,L,M} W_{(k,l,m)} \cdot y_{(i+k,j+l,m)} \quad (1)$$

where W denotes to the weight of filter k of layer l in channel m convolves with y in spatial position i and j. Taking into account the transfer function, the final output of Equation (1) passes to a selected transfer function. The Rectified Linear Unit transfer function (ReLU) is used in DCNN to overcome the vanishing gradient problem, allowing models to learn faster and perform better performance. The max pooling is the transfer function of down sampling layers. However, in last fully connected layer, softmax transfer function is used for multiclass classification.

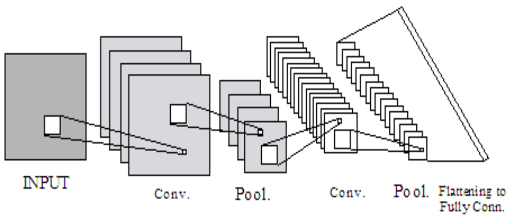


Figure 1. Two layers CNN structure.

5. Depthwise Separable Convolution

To reduce the intensive computations in deeply convolutional layers, the SepConv is performed. This technique is introduced in [25] by breaking down the Standard Convolution (StdConv) into two separated process: a depthwise convolutional followed by a pointwise 1×1 convolutional (Figure 2). The SepConv can be performed in three steps as follows in Equations (2), (3), and (4) respectively [16]:

$$\mathrm{PointConv}(Z,y)_{(i,j)} = \sum_{m}^{M} Z_m \cdot y_{(i,j)} \quad (2)$$

$$\mathrm{DepthConv}(W,y)_{(i,j)} = \sum_{k,l}^{K,L} W_{(k,l)} \circ y_{(i+k,j+l)} \quad (3)$$

$$\mathrm{SepConv}(Z,W,y)_{(i,j)} = \mathrm{PointConv}_{(i,j)}(Z, \mathrm{DepthConv}_{(i,j)}(W,y)) \quad (4)$$

where $\circ$ denotes to the element wise products. Z, W are the pointwise and depthwise weight matrices respectively.

The reduction of computations can be achieved by a computations comparison between the StdConv using Equation (1) with separable convolution that uses Equation (4). Suppose K filters of kernel dimension $d_k \times d_k$ of F pixels applied to input of channel M, each channel has $D_m \times D_m$ receptive fields, the dimension of each receptive field usually equals to the filter kernel dimension. Then the computations that use Equation (1) are:

$$d_k \times d_k \times M \times K \times D_m \times D_m \quad (5)$$

The total computations that use Equation (4) are the addition results of depthwise (left) and pointwise (right) convolutions when stride factor (s) equals to 1:

$$d_k \times d_k \times M \times D_m \times D_m + M \times K \times D_m \times D_m \quad (6)$$

A reduction ratio in computations can be achieved by expressing convolution as a two-step process of filtering and combining:

$$\frac{d_k \times d_k \times M \times D_m \times D_m + M \times K \times D_m \times D_m}{d_k \times d_k \times M \times K \times D_m \times D_m} = \frac{1}{K} + \frac{1}{d_k^2} \quad (7)$$

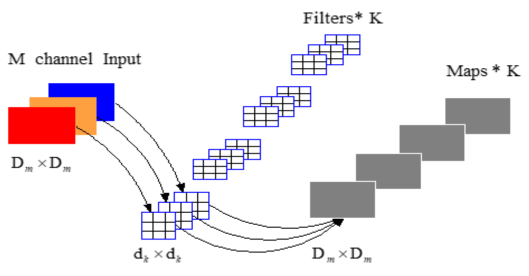
If 3×3 filter kernel is used as MobileNet, an 8 to 9 times less computations can be achieved by using Depthwise Separable Convolution (SepConv) vs. standard one. As shown in Equation (6) and Figure 2, as filter size (K) becomes larger, the pointwise computations will be dominant. According to Equation (3), the pointwise part of Equation (6) is a combination of Multiply Accumulate Operations (MACs).

6. Training the DCNN

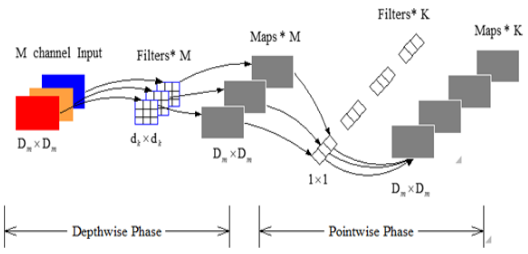
The fundamental difference in training the CNN with training DCNN is that, rather than using consequent layer-to-layer learning, an end-to-end learning approach is used to train the DCNN using supervised back propagation algorithm.

The training exploit the precision provided by floating point based calculations that implemented on high performance CPU with GPU. Besides that, in

SepConv, to keep the spatial interaction between channels and inter-channel correlations, the depthwise and pointwise convolution are used respectively to adjust the layer's parameters. The SepConv is used in inference for resource constrained devices. In this paper, leveraging the strength of both types of convolution is achieved: the compactness of SepConvs and the accuracy of ordinary convolutions.



a) StdConv.



b) SepConv.

Figure 2. Convolution types.

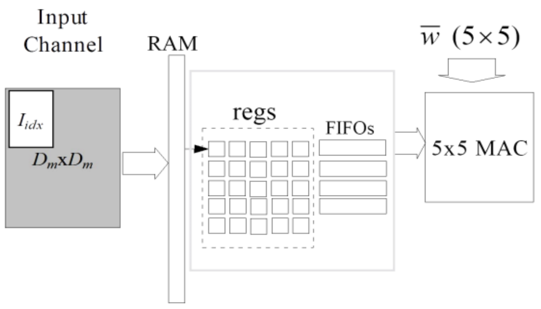
7. Hardware Inference of Deep Convolutional Neural Networks

Due to the high precision and accuracy requirements in updating parameters of DCNN, floating point calculations are used in learning. However, according to the limitations imposed of the hardware resources, a quantized fixed point calculations are used in inference. First, the hardware description and architecture of SepConv is presented, then the same hardware can be modified to be used for StdConv.

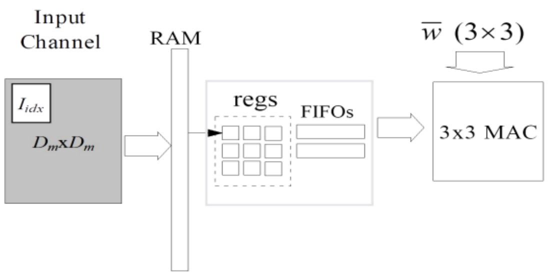
7.1. Memory Access Optimization

The input image/channel is typically stored in frame buffer in a sequential manner. However, because the two dimensional nature of the image, the adjacent pixels may not be located at adjacent addresses. Then the caching schemes may not effectively speed up or reduce the number of memory accesses. Unless an efficient addressing strategy is used, fast and direct access to a particular pixel becomes difficult. The DCNN use data in an infrequently and a non-sequential fashion. Thus, and according to work presented in [4], using window registers with First In First Out (FIFO) buffers approach can be a suitable solution to address the above mentioned issues (Figure 3). In this approach apart from the size of window (kernel size) and input channel, the chennel is fed through the window instead of feeding the window across the chennel, due to using of FIFOs.

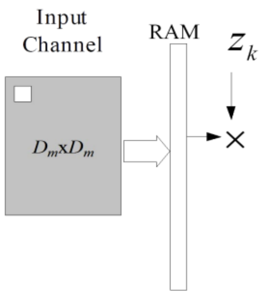
The input chennel of size $D_m \times D_m$ pixels and the window of $d_k \times d_k$ pixels require $d_k \times d_k$ registers and d_k-1 of FIFOs each of D_m-d_k locations. The only one pixel width bits is required as a bandwidth between the memory and register buffers. The up and down striding become very simple, no matter the size of the input channel or kernel. Each processing element can access each register individually. Also, the restriction of I/O memory port is released and multiple block element can access the register buffer in parallel. In other word, the benefit of using the above optimized memory access is speeding up the possible I/O bound computations that performed afterhand.



a) kernel size 5x5.



b) kernel size 3x3.



c) kernel size 1x1.

Figure 3. 2D memory access for input channel of $D_m \times D_m$ pixel and different kernel sizes.

From the above figure, one can see that different hardware is required when kernel size changes. To use the same architecture and avoid changing the hardware with different types of layers. The idea of using central apertere to deal with image reciptive field that is used in our previos work [3] can be employed here but with hardware implementation. The weights can be manipulated and multiplied with local operators for border of zeros and a central aperture of one(s) (Figure 4). Now the same depthwise processing unit of 5×5 basic convolutional kernel size can be used for another convolutional layer such as for 3×3 kernel size or in pointwise layers and in fully connected layer for kernel size of 1×1.

$$\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}\begin{bmatrix}0&0&0&0&0\\0&1&1&1&0\\0&1&1&1&0\\0&1&1&1&0\\0&0&0&0&0\end{bmatrix}\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&1&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}$$

a) Kernel size 5x5. b) Kenel size 3x3. c) Kernel size 1x1.

Figure 4. Local operators used in 5×5 registers buffers and employed in different kernel size.

For example, if the hardware is fixed to 3×3 register buffers for depthwise phase, the same hardware can be used in pointwise phase. If the local operator matrix is assumed to be $w_{(i,j)}$, where i and j are the row and column indexes respectively, then we can represent the center pixel ($w_{(2,2)}$) as Z_k weight and all border pixels ($w_{(1,1)}$ to $w_{(1,3)}$, $w_{(2,1)}$,$w_{(2,3)}$, and $w_{(3,1)}$ to $w_{(3,3)}$) are zeros.

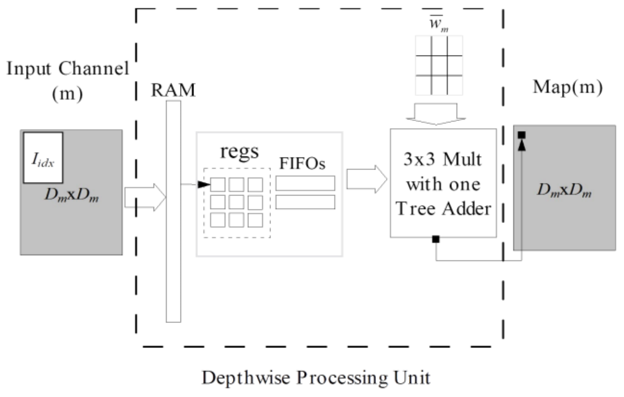
Since the outer borders of the 3×3 MAC is multiplied by zeros, then no accumulation is performed. The central pixel of I_{idx} is only multiplied by Z_k. That is equivalent of pointwise multiplication of 1×1 window. Thus, no need to change the available hardware used for depthwise convolution that shown in the middle part of Figure 3 to implement the pointwise convolution (no need to use the specific hardware that shown in the Figure 3-c).

7.2. Hardware Inference of Depthwise Separable Convolution

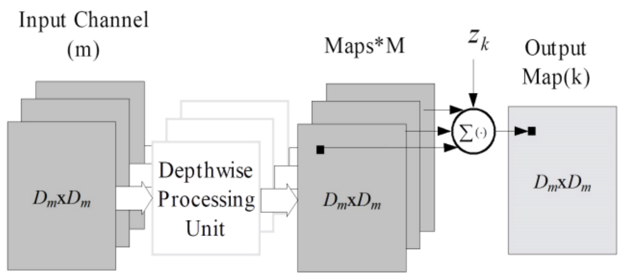
According to the available hardware resources, 2D array of multipliers is used to multiply each depthwise weights vector with channel receptive field (Figure 5-a). To avoid redundant memory access for every overlapping receptive field in input channel, and to make data reusing, the memory optimization technique is employed. Now, in every clock cycle, one receptive field of 3×3 kernel size is available and all of its pixels can be accessed in parallel. The output of 2D array is connected to adder tree to achieve MAC operations. The adder tree output is multiplied with pointwise weight associated with this channel. These components are grouped together to form the depthwise processing unit. Now each channel only requires one multiplication to process the pointwise part of the separable convolution. By scanning all the channel receptive fields, one depthwise channel map is produced. If enough hardware resource are available, multiple depthwise channel operations can be achieved in parallel by using multiple depthwise processing units (Figure 5-b). Each resulted map is stored in temporal buffer. Then a pointwise calculation for kernel size of 1×1 is performed by sequentially multiplying each map's pixel by a 1×1 pointwise kernel (Z_k) to produce each of the k output map. Algorithm (1) show the hardware procedure to implement the SepConv.

Algorithm 1: Depthwise Separable Convolution

```
Input stride s; $W_m$ , $\forall m \in \{1,\ldots,M\}$; $Z_k$ , $\forall k \in \{1,\ldots,K\}$; $I_{idx}$ ,
        $\forall idx \in \{1,\ldots, D_m^2\}$
Output reduction(+:$y_{k(i,j)}$), $\forall i \in \{1,\ldots,D_m\}$, $\forall j \in \{1,\ldots,D_m\}$
Initialization
parallel for m=1....M do
    for idx=1,.....,$D_m^2$ step s do
        $y_{m(i,j)} \leftarrow (\sum W_m . I_{m,idx})$
    end
end
for k =1....K do
    for m =1....M do
        for idx=1,.....,$D_m^2$ do
            $y_{k(i,j)} + \leftarrow y_{m(i,j)} . Z_k$
        end
    end
end
```



a) Architecture of depthwise processing unit.



b) Channel wise parallelism using multiple processing units.

Figure 5. Hardware Implementation of SepConv.

7.3. Hardware Inference of Standard Convolution

Simple modification to the depthwise processing unit shown in Figure 5-a) can be performed to implement the StdConv (Figure 6). However, the same channel wise parallelism can be used for both types of convolution. The only difference is in the access of different weight vectors each for a specified kernel (k).

In Figure 6, one can see that the first modification is the use of another index (k) that attached with the weight matric. The attached index refers to the different weight vector used for each iteration (see Figure 2-a) and Equation (1)). The StdConv operation flow is described in Algorithm (2). The second modification is the delete of pointwise weight matrix. Hence that the channel wise and point wise convolution are fused and be used to combine the output of each channel.

Algorithm 2: Procedure Standard Convolution

```
Input stride s; Wmk , $\forall m \in \{1,\dots,M\}, \forall k \in \{1,\dots,K\}$; Iidx , $\forall idx \in \{1,\dots, D_m^2\}$
Output reduction(+:yk(i,j)), $\forall i \in \{1,\dots,D_m\}, \forall j \in \{1,\dots,D_m\}$
Initialization
for k =1.....K do
    parallel for m=1....M do{
        for idx=1,.....,Dm2 step s do
            $y_{m(i,j)} \leftarrow (\sum W_{mk} \cdot I_{m,idx})$
            yk(i,j)+ ← ym(i,j)
        end
    end }
end
```

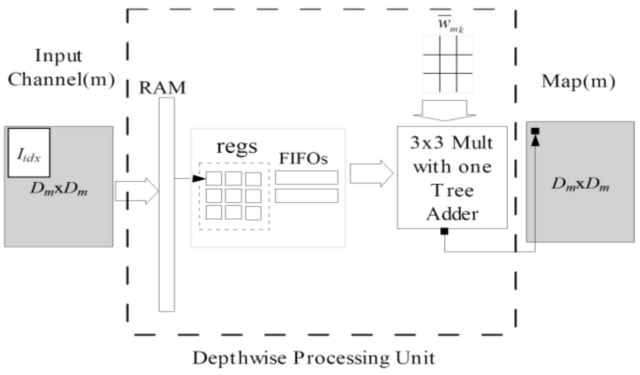


Figure 6. Architecture of depthwise processing unit used to implement the StdConv.

8. Results and Discussions

The performance of Most DCNNs is benchmarked for image classification. Thus, this application is considered in this paper. The ORL database for faces classification (32×32 pixel's image) are used in training a shallow DCNN of three convolutional layers and one fully connected layer. The number of filter taps (K) are 4, 8, and 16 for 1st, 2nd, and 3rd layers respectively, and the filters kernel size is 3×3. Multiple networks with different specifications and parameters are proposed. These networks are completely built and trained from scratch using software models running in Matlab R2020a. Table 1 shows the specifications analysis of the designed DCNNs using StdConv.

Table 1. Specification analysis of DCNN using StdConv(d_k×d_k: 3×3), s=1.

Layers	Feature maps	Learnable	# param	#MAC
Input	32×32×1			
SC1[*]	34×34×4	weights: 3×3×1×4, bias: 1×1×4	40	36864
Pool1	17×17×4			
SC2	19×19×8	weights: 3×3×4×8, bias: 1×1×8	296	83232
Pool2	9×9×8			
SC3	11×11×16	weights: 3×3×8×16,bias: 1×1×16	1168	93312
Pool3	5×5×16			
FC	1×1×40	weights: 40×400,bias: 40×1	16040	16040
Total			17544	229448
*SC denotes to StdConv.				

While the SepConv analysis is shown in Table 2. The weight sharing property is not used in fully connected layers. Therefor most of storage requirements are conducted to the parameters of fully connected layers rather than convolution layers.

Table 2. Specification analysis of DCNN using SepConv (dk×dk: 3×3), s=1.

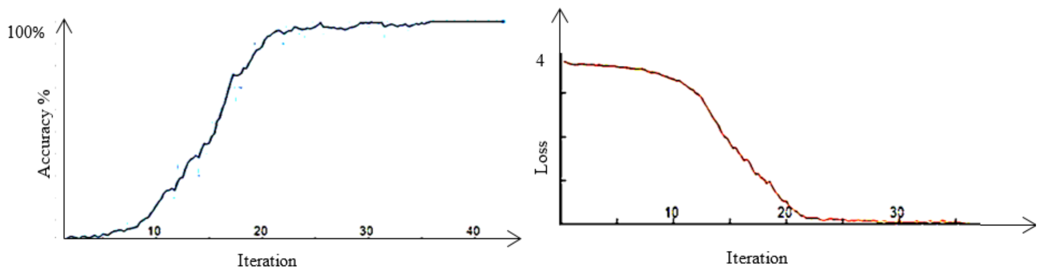
Layers	Feature maps	Learnable	# param	#MAC
Input	32×32×1			
DWC1[**]	32×32×1	weights: 3×3,bias: 1×1	10	9216
SC1[**]	32×32×4	weights: 1×1×1×4,bias: 1×1×4	8	4096
Pool1	16×16×4			
DWC2	16×16×1	weights: 3×3×4,bias: 1×1	37	9216
SC2	16×16×8	weights: 1×1×1×8,bias: 1×1×8	16	2048
Pool2	8×8×8			
DWC3	8×8×1	weights: 3×3×8,bias: 1×1	73	4608
SC3	8×8×18	weights: 1×1×1×18,bias: 1×1	36	1152
Pool3	4×4×18			
FC	1×1×40	weights: 40×288,bias: 40×1	11560	11560
Total			11740	41896
**DWC denotes to depthwise convolution while SC denotes to pointwise convolution (1×1 StdConv).				

By excluding the parameters of the fully connected layer, one can see from the above tables that a considerable reduction in parameters from 1476 to 147 of the convolution layers are achieved. About 6× reduction of MAC operations are achieved to implement the SepConv (5th columns in Tables 2 and 3). Another comparison between StdConv and SepConv is shown in Table 3. The positive effects of memory size when reducing word length representation of data from floating point to fixed point of 8 bits and 6 bits respectively is shown in this table. Also, an enormous reduction of required memory when using SepConv vs. StdConv with only 2% scarification in accuracy can also be seen in the last two columns of Table 3.

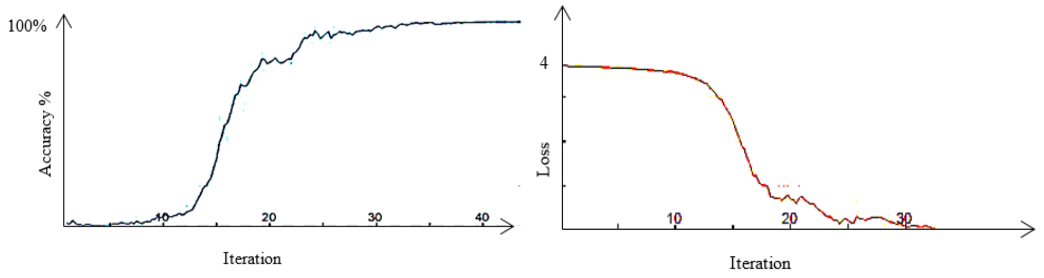
Table 3. Effect of quantization on StdConv vs SepConv on memory and accuracy.

Type of Conv.	Prec	Memory (KB)	Reduction in memory % (Compared to fp32)	Accur.%	Reduction in accur % (Compared to fp32)
StdConv	fp32	5.76	---	95	---
	fix8	1.44	75	95	0%
	fix6	1.1	80	92	3%
SepConv	fp32	0.57	---	93	---
	fix8	0.14	75	93	0%
	fix6	0.11	80	91	2%

Due to random initialization of weights that imposed by software training model, the presented accuracy is measured as the average of three simulation runs for 70% of the total dataset that used in training and the rest for testing. In contrast, for a network of reduced capacity as the proposed in SepConv based model, slower convergence (160 iterations vs. 145 iterations) may arise as the model tries to generalize better (see Figure 7).



a) Accuracy and loss convergence of StdConv.



b) Accuracy and loss convergence of SepConv.

Figure 7. Accuracy and loss convergence.

Another test for the proposed hardware is done for large DCNN and dataset. The CIFAR-10 dataset which contains 6000 colored images of 10 classes of 32×32 pixels is used (5000 images for training and the remaining for testing). The MobileNet v1 model that based on SepConv is investigated in the experiment. The specification analysis is shown in Table 4.

Table 4. Specification analysis of MobileNet v1.

Operator	Input×M	s	k	n^{**}	#MAC	MOP
SC 3×3	32×32×3	1	32	-	884736	0.9
DWC 3×3	32×32×32	1	32	1	29491	0.3
SC 1 ×1		1	64		2097152	2
DWC 3×3	32×32×64	2	64	1	147456	0.1
SC 1 ×1		1	128		2097152	2
DWC 3×3	16×16×128	1	128	2	294912*2	0.3
SC 1 ×1		1	128		4194304*2	8
DWC 3×3	16×16×128	2	128	1	294912	0.3
SC 1 ×1		1	256		8388608	8
DWC 3×3	8×8×256	1	256	3	147456*3	0.45
SC 1 ×1		1	256		4194304*3	4
Pool	8×8×256	-	256	3	221184	0.2
FC	1×1×256	-	10	-	2560	0.0025
Softmax	1×1×10	-	-	-	-	-
** n denotes to repeated identical layer.						

A reduction of word length leads to an enormous reduction on the utilized FPGA area. Relatively two low cost Xilinx FPGA models: Spartan3E (XC3S500) and zynq(XC7Z020) platforms are used to test the performance of the proposed model. Table 5 shows the utilization and performance comparisons among them. The parallelism factor and the frequency depend on the type and size of the selected FPGA platform. A maximum of 8 depthwise processing unit is used in very low cost FPGA (Spartan3E) vs. 18 (and more) is used in zynq. As shown in Table 6, a low throughput is achieved using Spartan3E even if it operates on maximum frequency. While well performance can be achieved when moderate FPGA cost (zynq) is used. This performance can further be increased when increasing the parallelism factor due to the availability of redundant FPGA area (14% utilization is only consumed). Exploiting the Digital Signal Processing (DSP48E1S) embedded resources can improve the performance and make the rest of the FPGA resources available for further operations. Also, a comparison with different models and platforms in terms of accuracy, resource utilization, and performance, is depicted in Table 6. One can see that an acceptable accuracy and performance is achieved with lower utilization area when ultra-low cost FPGA platform (Spartan 3E: XC3S500) and moderate platform (zynq: XC7Z020) are used respectively. Also, although the resulted frame rate does not exceed what is achieved in some of these works, but the reduction in the utilized resources outperforms all the presented state of the art works.

Table 5. Utilization summary and performance for Xilinx FPGA.

Design	Parall.	DSP	LUT	FF	Slice	freq (MHz)
Spartan 3E	8	0	7384 (80%)	4552 (48%)	4112 (88%)	80
Zynq	18	0	14364 (27%)	9720 (9%)	14364 (27%)	185
Zynq	18	180 (81%)	7380 (14%)	7200 (6%)	7380 (14%)	196

Table 6. Overall accuracy, resource utilization, and performance comparison for CIFAR-10 classification for different platforms.

	Model	Accuracy %	DSP	FF	LUT	GOPS	FPS	platform
[5]	MobileNetv1	79.92	-	-	-	-	125	GPU Jetson Xavier
[15]	ResNet20	91	1054	-	181440	21.12	344	Xilinx ZCU104
[18]	BCNN	87.8	1096	70769	342126	7663	1000	Vertix 7
[21]	BCNN-based ResNet18	92.14	465	112347	161306	-	4938	FPGA Alevo U280
[29]	HyBNN-U5D3	89.8	205	99074	51927	-	4302	AMD-Xilinx Ultra96-V2
Ours	MobileNetv1	85.7	0	4552 ↓	7384 ↓	11	170	Spartan3E
Ours	MobileNetv1	85.7	0	9720 ↓	14364 ↓	60	909	zynq
Ours	MobileNetv1	85.7	81 ↓	7200 ↓	7380 ↓	64	1000	zynq with DSP

9. Conclusions

A road map to implement a DCNN of complex computations and memory demands in low cost FPGA is presented. Efficient compact hardware techniques are used to achieve this goal: One depthwise unit is alternatively used either in StdConv or SepConv. To minimize the access to the memory where the input channel data are stored, receptive fields' pixels are caches to local registers. Multiplying weight with local operators of variant aperture add more flexibility to the proposed architecture and makes it available to be used with different types of layers. The designed processor can process multiple convolutional channels in parallel. A considerable reduction of memory storage and FPGA resources are achieved due to quantizing the weights to 8-bits length of word. All these techniques led to achieve a very low consumption of hardware resources. The experiments that conducted on face recognition application show that an acceptable performance whether in the resulted accuracy or in the FPGA cost is achieved. The architecture can be flexibly reconfigured to be used for different DCNN models that used either StdConv or SepConv techniques.

10. Future Works

Further reduction on cost can be achieved using the pruning optimization technique. The weight, node, channel and filter pruning can reduce the DCNN parameters and memory demand and consequently reduce the overall inference time. This technique can be analyzed, then another tradeoff between cost and accuracy can be achieved. Also further investigation of the proposed methodology and architecture for either StdConv or SepConv can be applied to more datasets and deep models, and the results can then be compared and discussed.

References

[1] Bai L., Zhao Y., and Huang X., "A CNN Accelerator on FPGA Using Depthwise Separable Convolution," *IEEE Transactions on Circuits and Systems 2: Express Briefs*, vol. 65, no. 10, pp. 1415-1419, 2018. DOI:10.1109/TCSII.2018.2865896

[2] Belbachir K. and Tlemsani R., "Temporal Neural System Applied to Arabic Online Characters Recognition," *The International Arab Journal of Information Technology*, vol. 16, no. 3A, pp. 514-424, 2019. https://www.iajit.org/portal/PDF/Special%20Issue%202019,%20No.%203A/18597.pdf

[3] Dawwd S., "GLCM Based Parallel Texture Segmentation Using A Multicore Processor," *The International Arab Journal of Information Technology*, vol. 16, no. 1, pp. 8-16, 2019. https://www.iajit.org/portal/PDF/January%202019,%20No.%201/9828.pdf

[4] Dawwd S., "The Multi 2D Systolic Design and Implementation of Convolutional Neural Networks," *in Proceedings of the IEEE 20th International Conference on Electronics, Circuits, and Systems*, Abu Dhabi, pp. 221-224, 2013. DOI:10.1109/ICECS.2013.6815394

[5] Dbouk H. and Shanbhag N., *Advances in Neural Information Processing Systems*, Curran Associates, 2021. https://www.proceedings.com/content/063/063069webtoc.pdf

[6] Dhilleswararao P., Boppu S., Manikandan M., and Cenkeramaddi L., "Efficient Hardware Architectures for Accelerating Deep Neural Networks: Survey," *IEEE Access*, vol. 10, pp. 131788-131828, 2022. DOI:10.1109/ACCESS.2022.3229767

[7] Dhouibi M., Salem A., Saidi A., and Saoud S., "Accelerating Deep Neural Networks Implementation: A Survey," *IET Computers and Digital Techniques*, vol. 15, no. 2, pp. 79-96, 2021. https://doi.org/10.1049/cdt2.12016

[8] Geirhos R., Janssen D., Schütt H., Rauber J., Bethge M., and Wichmann F., "Comparing Deep Neural Networks Against Humans: Object Recognition When the Signal Gets Weaker," *arXiv Preprint*, arXiv:1706.06969v2, pp. 1-31, 2018. https://arxiv.org/pdf/1706.06969.pdf

[9] Gilan A., Emad M., and Alizadeh B., "FPGA-Based Implementation of a Real-Time Object Recognition System Using Convolutional Neural Network," *IEEE Transactions on Circuits and Systems 2: Express Briefs*, vol. 67, no. 4, pp. 755-759, 2020. DOI:10.1109/TCSII.2019.2922372

[10] Guo K., Zeng S., Yu J., Wang Y., and Yang H., "[DL] A Survey of FPGA-Based Neural Network Inference Accelerators," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, no. 1, pp. 1-26, 2018. https://doi.org/10.1145/3289185

[11] Hadnagy A., Feher B., and Kovacshazy T., "Efficient Implementation of Convolutional Neural Networks on FPGA," *in Proceedings of the 19th International Carpathian Control Conference,* Szilvasvarad, pp. 359-364, 2018. https://ieeexplore.ieee.org/document/8399656

[12] Hou Y. and Chen Z., "LeNet-5 Improvement Based on FPGA Acceleration," *The Journal of Engineering*, vol. 2020, no. 13, pp. 526-528, 2020. https://doi.org/10.1049/joe.2019.1190

[13] Howard A., Zhu M., Chen B., Kalenichenko D., Wang W., and Weyand T., Andreetto M., Adam H., "MobileNet: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv Preprint*, arXiv:1704.04861, pp. 1-9, 2017. https://doi.org/10.48550/arXiv.1704.04861

[14] Iba H. and Noman N., *Deep Neural Evolution:*

Deep Learning with Evolutionary Computation, Springer, 2020. https://doi.org/10.1007/978-981-15-3685-4

[15] Isik M., Inadagobo K., and Aktas H., "Design Optimization for High-Performance Computing Using FPGA," *arXiv Preprint*, arXiv: 2304.12474, pp. 1-19, 2023. https://arxiv.org/pdf/2304.12474.pdf

[16] Kaiser L., Gomez A., and Chollet F., "Depthwise Separable Convolutions for Neural Machine Translation," *arXiv Preprint*, arXiv:1706.03059, pp. 1-10, 2017. https://arxiv.org/pdf/1706.03059.pdf

[17] Kuramochi R. and Nakahara H., "A Low-Latency Inference of RandomlyWired Convolutional Neural Networks on an FPGA," *IEICE Transactions on Information and System*, vol. E104.D, no. 12, pp. 2068-2077, 2021. https://doi.org/10.1587/transinf.2021PAP0010

[18] Li Y., Liu Z., Xu H., Yu H., and Ren F., "A GPU-Outperforming FPGA Accelerator Architecture for Binary Convolutional Neural Networks," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 14, no. 2, pp. 1-16, 2018. https://doi.org/10.1145/3154839

[19] Liang F., Tian Z., Dong M., Cheng S., Sun L., Li H., Chen Y., and Zhang G., "Efficient Neural Network Using Pointwise Convolution Kernels with Linear Phase Constraint," *Neurocomputing*, vol. 423, no. 3, pp. 572-579, 2021. https://doi.org/10.1016/j.neucom.2020.10.067

[20] Oguntimilehin A. and Balogun K., "Real-Time Sign Language Fingerspelling Recognition Using Convolutional Neural Network," *The International Arab Journal of Information Technology*, vol. 21, no. 1, pp. 158-165, 2024. https://doi.org/10.34028/iajit/21/1/14

[21] Peng H., Zhou S., Weitze S., Li J., and Islam S., Geng T., Li A., Zhang W., Song M., Xie M., Liu H., and Ding C., "Binary Complex Neural Network Acceleration on FPGA," *in Proceedings of the IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors*, New Jersey, pp. 85-92, 2021. DOI:10.1109/ASAP52443.2021.00021

[22] Pushparaj S. and Arumugam S., "Using 3D Convolutional Neural Network in Surveillance Videos for Recognizing Human," *The International Arab Journal of Information Technology*, vol. 15, no. 4, pp. 693-700, 2018. https://www.iajit.org/portal/PDF/July%202018,%20No.%204/8768.pdf

[23] Salehinejad H. and Valaee S., "Edropout: Energy-Based Dropout and Pruning of Deep Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no.10, pp. 5279-5292, 2022. DOI:10.1109/TNNLS.2021.3069970

[24] Salim U., Dawwd S., and Ali F., "U-Net Cost Analysis Using Roofline Model," *Al-Rafidain Engineering Journal*, vol. 27, no. 2, pp. 198-205, 2022. DOI: 10.33899/rengj.2022.133825.1172

[25] Sifre L. and Mallat S., Rigid-Motion Scattering for Image Classification, Ph.D. Thesis, Ecole Polytechnique, CMAP, 2014. https://www.di.ens.fr/data/publications/papers/phd_sifre.pdf

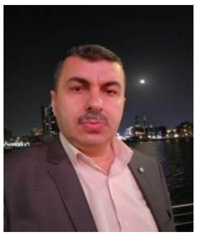
[26] Wang P., He X., Chen Q., Cheng A., Liu Q., and Cheng J., "Unsupervised Network Quantization Via Fixed-Point Factorization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 6, pp. 2706-2720, 2021. DOI:10.1109/TNNLS.2020.3007749

[27] Wang Y., Li K., Xu L., Wei Q., Wang F., and Chen Y., "A Depthwise Separable Fully Convolutional ResNet With ConvCRF for Semisupervised Hyperspectral Image Classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 4621-4632, 2021. DOI:10.1109/JSTARS.2021.3073661

[28] Wu D., Zhang Y., Jia X., Tian L., Li T., Sui L., Xie D., and Shan Y., "A High-Performance CNN Processor Based on FPGA for MobileNets," *in Proceedings of the 29th International Conference on Field Programmable Logic and Applications*, Barcelona, pp. 136-143, 2019. DOI:10.1109/FPL.2019.00030

[29] Yang G., Lei J., Fang Z., Li Y., Zhang J., and Xie W., "HyBNN: Quantifying and Optimizing Hardware Efficiency of Binary Neural Networks," *ACM Transaction on Reconfigurable Technology and Systems*, vol. 1, no. 1, 2023. https://www.sfu.ca/~zhenman/files/J16-FPT-TRETS2023_HyBNN.pdf

[30] Zavalsız M., Alhajj S., Sailunaz K., Ozyer T., and Alhajj R., "A Comparative Study of Different Pre-Trained Deep Learning Models and Custom CNN for Pancreatic Tumor Detection," *The International Arab Journal of Information Technology*, vol. 20, no. 3A, pp. 515-526, 2023. https://doi.org/10.34028/iajit/20/3A/9



Shefa Dawwd is currently a professor at the computer engineering department-university of Mosul. He received the B.Sc in Electronic and Communication Engineering, M.Sc and Ph.D in Computer Engineering. He has authored more than 55 research papers and two book chapters. He has completed the supervision of 12 M.Sc and Ph.D. students. His research interest is in the Processing Acceleration of 1D, 2D, and 3D signals, Parallel Architecture, Real Time Applications, Deep Neural Networks, and Heterogeneous Computing. He selected as a chair, co-chair, and member in many scientific committees of national and international conferences and symposiums.