ISSN 2278-3091
Volume 9, No.3, May - June 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse202932020.pdf
https://doi.org/10.30534/ijatcse/2020/202932020

Approximation Computing Techniques to Accelerate CNN Based Image Processing Applications – A Survey in Hardware/Software Perspective

N. Manikandan[1], M. Priyanka[2], Sasikumar[3], R. Muthaiah[4]
[1]Research Assistant, School of Computing, SASTRA Deemed University, India, manikandan_phd@outlook.com
[2]PG Scholar, School of Computing, SASTRA Deemed University, India, ridhipriya3@gmail.com
[3]Research Assistant, School of Computing, SASTRA Deemed University, India,vlsisasi@gmail.com
[4]Professor, School of Computing, SASTRA Deemed University, India, muthaiah66@gmail.com

ABSTRACT

In today's technology era, Convolutional Neural Networks (CNNs) are the limelight for various cognitive tasks because of their high accuracy. With the increasing complexity in the applications, CNNs present high computation and storage demands which call for customized hardware support to boost their performance. The streaming nature of CNN's workloads makes them suitable for hardware implementations like FPGAs and ASICs. Providing sufficient resources alone cannot solve this difficulty, which makes Approximate Computing a solution. This article gives an insight into various approximate computing techniques used to accelerate the CNNs at multiple levels for the hardware implementations. The survey has been conducted by considering different metrics: approximation technique used, datasets used for evaluation, network structure (AlexNet, LeNet, Visual Geometry Group (VGG)), hardware platform for implementation (Application Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA)), training or testing phase and results (in terms of accuracy, area, power, throughput, resource utilization). The approximate computation techniques applied at the various levels of the network and layers are discussed. Necessary comparisons have also been made to know the utility of these techniques for yielding more significant performance gains with minimal losses in the accuracy. Methods are presented with recent contributions in the state-of-the-art image processing applications along with the various future outlooks based on the studies made.

Key words : Approximate Computing, Convolutional Neural Networks (CNNs), hardware accelerators, Image Processing

1. INTRODUCTION

In today's technological and applicative world, Deep Learning (DL) is being used widely for many applications in various domains ranging from speech recognition, medical image analysis, image processing, object recognition, natural language processing, healthcare and so on. Deep Learningwhich is a part of Artificial Intelligence (AI) and a class of Machine Learning (ML) has placed its mark in performing many real-time tasks because of its ability to learn the problem and give better results [1-5]. The Deep Learning techniques are composed of artificial neural networks which are inspired by the human brain. Convolutional Neural Networks (CNNs) is one of the widely used Deep Learning techniques for applications like image classification [1], digit recognition [2], image recognition [3], detection [4], and many machine vision-related tasks. The CNNs have gained immense popularity in the recent trends because of their near-human accurate results. The CNNs give high performance at the cost of requiring a massive amount of resources, memory and high computational cost. So this calls for a dedicated hardware platform to meet the performance criteria and to accelerate the networks to be power and energy-efficient for mobile applications. The acceleration of CNNs on platforms like CPU and GPU is not adequate because of low throughput and low energy-efficiency. The ASIC implementation of these networks suffers from the problem of reconfigurability, high investments and design cycle complexity. The reconfigurability, high throughput compared to CPUs and GPUs, high performance compared to CPUs, better energy efficiency features of the FPGAs make them a suitable platform for the acceleration of CNNs [6]. The development of CNNs in recent times is increasing the sparsity and use of customized data types. In this aspect, FPGAs effectively increase the utilization of the resources by supporting the customizable compact data types. However, with these attractive features, the FPGA acceleration of CNNs poses some challenges. For example, consider AlexNet, which has more than 60 million parameters and requires a storage capacity of approximately 250 MB when represented in 32-bit Floating-Point model [7]. This results in the memory overhead for the FPGA as it exceeds the on-chip memory capacity. Using the external memory for storing those parameters and moving values from/to FPGA increases

performance overheads. With the increasing size of the CNN structures like VGG, GoogleNet requires memory requirements furthermore. This calls for the necessary optimizations to be done to the CNNs at different stages like inference and training to achieve
high-performance gains in practical implementation.

1.1 Need for Approximate Computing

A general CNN architecture has an input layer and an output layer with many intermediate layers. The middle layers of CNN consist of many layers like Convolutional layers, Activation layers, Pooling layers and Full connected layers. The convolutional layers are computationally intensive, and the fully connected layers are memory expensive. The computational workload and memory requirements are based on the number of MAC (Multiply and Accumulate) operations and the number of parameters required for performing the task. So, in order to minimize the overall resource requirements, necessary optimizations are to be done to the networks to eliminate the ineffective computations at all the possible levels by using the approximation knobs [8]. The Approximate Computing (AC) uses the gap between the accuracy needed by the application and efficiency given by the computing system enabling various optimizations. The scope for AC involves the need for approximation where an inexact solution is sufficient for solving the complex problems, efficient optimization and configurable quality. The AC is mainly used for the applications which are error-tolerant like scientific computing [9], image and signal processing [10], deep learning [11] and so on. In the current scenario, Google is using this AC technique for their Tensor Processing Units (TPUs) which is a custom ASIC used as an accelerator for neural networks [12]. The TPUs use Quantization technique to reduce the neural network prediction cost, where 32-bit floating-point calculations are converted into 8-bit integers. The approximations can be made at various levels of a system used for the application. For example, approximations can be made at the circuit level where the exact adders and multipliers can be replaced with approximate adders and multipliers, which reduces the hardware overhead. The approximated multi-bit adder circumvents the carry chain in order to reduce the critical path delay, which increases the performance of the circuit and energy-efficiency but traded for accuracy [13]. The Approximate Computing is based on the relaxations provided to the exact computing in order to increase the performance efficiency of the systems in terms of area, speed and power.

In this paper, we presented the survey of approximate computing techniques applied at the various levels of CNN implementation for Image Processing applications. These CNNs are fully-connected structures which can reduce the parameters of the model without any loss in the quality of the models. This makes CNNs more suitable for many image processing applications because of the high dimensionalities in the images. The convolutional layers of the CNNs are the feature extractor blocks which extracts required features from the input image by sliding the kernels over the image, reducing the dimensionality. This makes CNNs suitable for many image processing applications like handwritten digit recognition, image classification, object detection. With the increasing complexity of the use, Approximate Computing along with hardware acceleration can fuel the performance of the CNNs. The approximations done at the structural level of the CNN includes pruning and weight sharing where the unimportant weights in the neural network structure are removed thus reducing the network density and increasing the performance in terms of energy and memory utilization. The approximate adders and multipliers used at the circuit level achieve improved efficiency in terms of power and area usage, thus reducing the hardware costs. Approximating memories also give significant reductions in the area used on the hardware. The approximations for lowering the precision of the data includes fixed-point quantization, dynamic fixed-point, power-of-2 and using binary weights which reduces the complexity in performing the complex operations of MAC units. The CNNs can be approximated at both the training phase and the inference phase of it. The pruning technique is applied to the network in the training phase, and the network is re-trained unless desired error rate is obtained [14]-[16]. The precision reduction techniques like fixed-point, dynamic fixed-point, power-of-2 and binary quantization are applied to a trained neural network for the CNN inference [17]-[20]. The approximated hardware units like approximate multipliers can also be utilized to improve the training performance of CNNs in terms of speed, area and power [21].

The paper is organized as follows, a brief discussion about the Convolutional Neural Networks and its layers with their functionality in section 2. Section 3 presents various approximate computing techniques used at different levels of CNN implementations for the acceleration of the computations by considering multiple hardware implementation platforms like FPGAs and ASICs. Section 4 summarizes the work and explores future outlook.

2. OVERVIEW OF CNNs

The Convolutional Neural Networks (CNNs) are a type of deep neural networks which have been widely applied in computer vision applications. These CNNs gained utmost popularity in computer vision application in 2012 when more significant results were obtained for the application of object detection in the ImageNet Large Scale Visual Recognition Competition (ILSVRC) [1]. The image classification error rate achieved was 15.3% which was less compared to previous years 2011 (26%). The first CNN basis was established in 1998 by an object where they considered the neuron organization of the cat's visual cortex as an artificial neural network [22]. The CNNs are deep and feed-forward neural networks. It is a sparsely connected network which has the advantage of weight sharing, which reduces the total number of parameters [23]. The CNNs are used in two phases, namely training and inference. In the training phase, the neural network is modelled by getting trained with the large volume of data set samples. Back-propagation algorithm is used to iteratively update the network parameters like weights in order to improvise the model prediction. In the inference phase, the learnt model is tested with a new set of data samples [24].

A. Convolutional layers

Convolutional layers are the building blocks of the CNNs. The main operation in this layer is *convolution*, so the name convolutional layer. In this layer, the input (also called Feature Map) is convolved with a sliding window (Kernel or Filter) of size k x k. The Filter is shifted all over the input image, and the convolutions obtained individually are summed up. The convolution operation is the element-wise multiplication between the Kernel and the kernel sized patch of the input image. The repeated convolution of the same Filter to the input image gives a map of activations which are known as Feature Maps (FMs). These FMs indicate the location of the dedicated features of the input image. The output Feature Map of a convolutional layer is mathematically given by equation (1) in [25].

$$Y_i = f(b + \sum_{j=1}^{n} X_j \otimes K_{i,j}) \quad \text{where } i = (1,2,\dots,m) \qquad (1)$$

Where b is the bias value, and f is the non-linear activation function to limit the value of a pixel to a suitable range. The $\otimes$ symbol represents the convolution operation between the input image and the Kernel. These convolution layers are computationally intensive as the number of MAC operations increase with the increasing input size.

B. Activation layers

The Activation layer follows the convolutional layer. The activation function applies a non-linear transformation to all FM values. These activation functions decide the firing of the neurons. The different activation functions are Sigmoid, tanh, ReLU, leaky ReLu, Maxout and ELU. Sigmoid and tanh are the non-linear activations which make the neural networks able to model the complicated decision boundary classification. ReLU is the widely used activation function for the neural networks as they satisfy the approximating property of neural networks. The ReLU is a piecewise linear activation function which incurs less computation cost and minimum training time and has become the default choice for deep learning networks.

C. Pooling layers

The primary use of pooling in CNNs is to reduce the size of the image obtained from the previous layers. These layers are placed in between the successive convolutional layers appropriately. With the reduced image size, the number of parameters are reduced, thereby decreasing the computations to be performed. There are two types of pooling techniques, Max pooling and Average pooling. In Max pooling, the maximum value of the neurons in the FM is considered and given as output. In Average pooling, the average value of the neurons in the neighbourhood is taken as output. Max pooling technique is being used widely in recent times. This pooling technique is the process of downsampling the adjacent pixels.

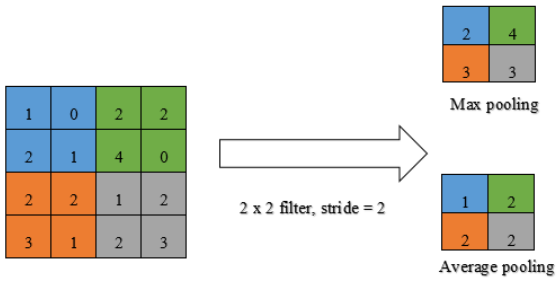

Figure 1: Example of Pooling

Figure 1 shows the different pooling schemes commonly used in the CNNs. The stride controls the convolving of the Filter with the input. The stride determines the shift of the Filter over the input. For the example shown in the above figure, the Filter is shifted by two units every time, and the corresponding pooling operation is performed. In Figure 1, we consider a 4 x 4 FM and a filter of size 2 x 2 with stride = 2. When the Kernel is slid over the first 2 x 2 patch of the FM, for Max pooling the maximum value of (1, 0, 2, 1) is given in the output FM, for Average pooling the average of (1, 0, 2, 1) is given in the output FM. This is continued until the Filter is slid all over the input FM. The output of this layer is the shrunken form of the input.

D. Fully Connected layers

The fully connected layers are memory intensive layers. The input and output layer's neurons are connected fully with each other. These layers are computed using matrix multiplications, and also the output of the fully connected layers are applied with a non-linear function similar to convolutional layers. The fully connected layers classify the features extracted by the convolutional layers.

$$Yi = \sum_{j=1}^{n} X_j * W_{i,j} \quad \text{where } i = (1,2,\dots,m) \qquad (2)$$

The equation (2) [25] represents the output vector of a fully connected layer where the input FM to this layer is multiplied with the weight matrix. In the equation X_i represents the input FM and $W_{i,j}$ represents the weight matrix of the fully connected layer.

3. CLASSIFICATION OF APPROXIMATE COMPUTING TECHNIQUES

The approximate computing techniques applied to accelerate CNN's performance are given in Figure 2. These approximations can be used at various levels of the neural network. Depending on the layers of abstraction we have two types of classifications, software-level approximation and hardware-level approximation of the CNNs. The data represented with high precision requires a large amount of memory to store the parameters and the results of the intermediate operations. The computations on such high precision data require costly hardware units which require more power and large chip area. This presents the use of hardware-software co-design approximation techniques

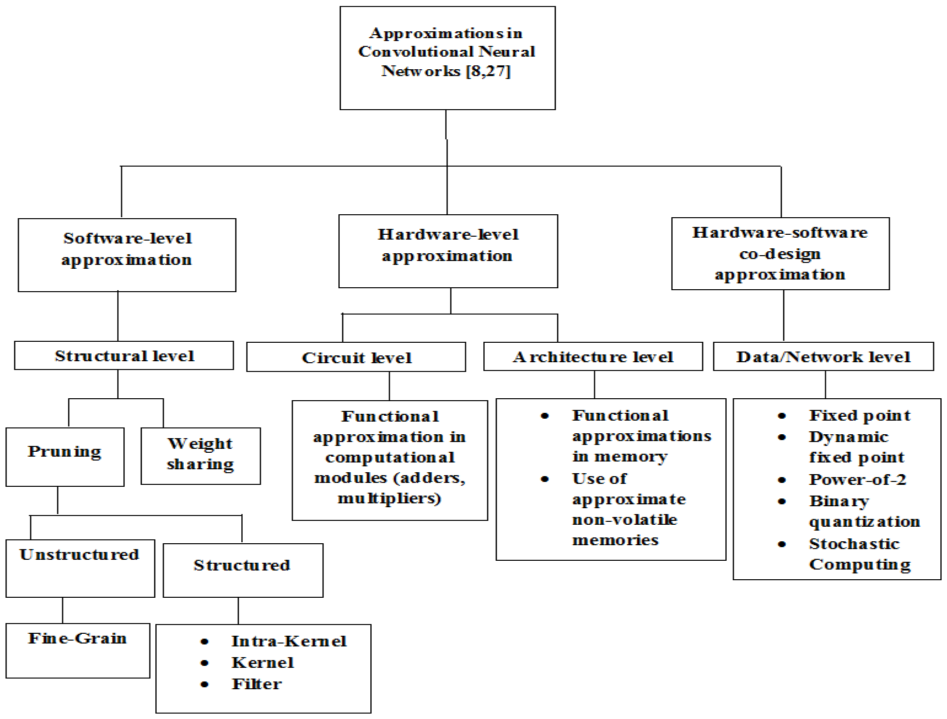
Approximations in Convolutional Neural Networks [8,27]
Software-level approximation
Hardware-level approximation
Hardware-software co-design approximation
Structural level
Circuit level
Architecture level
Data/Network level
Pruning
Weight sharing
Functional approximation in computational modules (adders, multipliers)
Functional approximations in memory
Use of approximate non-volatile memories
Fixed point
Dynamic fixed point
Power-of-2
Binary quantization
Stochastic Computing
Unstructured
Structured
Fine-Grain
Intra-Kernel
Kernel
Filter

Figure 2: Classification of Approximate Computing techniques for CNNs

which is the third classification of approximate computing techniques for CNN. The approximations to the structure of the network can be made by using Pruning and Weight Sharing techniques. These structural level approximations to the network reduce the number of parameters by removing the redundant and unimportant connections in the network layers. The CNNs can be accelerated by using approximated computational units like adders and multipliers and also approximated memories which increases the energy-efficiency and reduces hardware area and cost. The Network level approximations reduce the computational and memory workloads by decreasing the precision of the data in the network. Further sub-sections discuss the various works that use approximate computing techniques for CNN acceleration.

3.1 Software-level Approximate Computing technique

The software level approximations to the neural network try to skip the computations to improve the overall execution time. The approximation techniques at this level require the support of algorithms to tune the network with approximations, so the name software-level approximate computing. The methods under this level modify the structure of the neural network.

A. Pruning

This is the approximation technique used at the structural level of the CNN model. This is a type of model compression where the size of the network or model is reduced by making required approximations. There are two ways of compressing the network, pruning (decreasing the number of weights) and reduced precision (the bit width to represent the data is reduced). The neurons present at various layers of the network are connected to each other with the weights. These connections indicate convolution or matrix multiplication operations. Pruning is the approximate computing technique in which connections in the neural network involving in such activities are removed. This is because all the connections in a neural network are not equally important, so some of them can be ignored while performing the operations. The connections to be removed depending upon the importance of the weights adhered to those connections in the network. There are two types of pruning strategies, *structured pruning* and *unstructured pruning*. These two strategies differ regarding the connection's location information. In simple pruning technique, the non-effective weights of the network are pruned without considering the connection's location. In structured pruning technique, the pruning of non-effective weights is done by considering the connection's location. Depending on the pruning granularity, we have four different pruning techniques like Fine-grain (a), Intra-kernel (b), Filter (c) and Kernel (d) represented in Figure 3. The Fine-grain technique is an unstructured pruning technique, and the remaining three techniques are structured pruning techniques.

Unstructured Pruning

In the unstructured pruning, non-effective connections in the neural network are removed irrespective of the location of

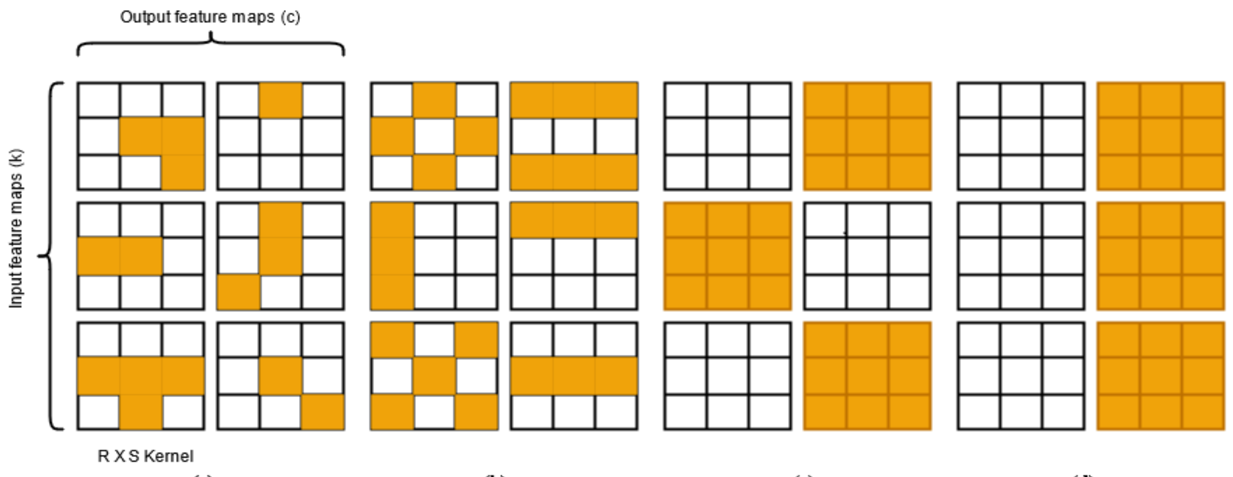

Figure 3: Pruning Granularities (a) Fine-Grain (b) Intra-Kernel (c) Kernel (d) Filter

connections, thus making a dense network into a sparse one. The number of parameters is minimized, but this technique induces irregularity in the network, which causes memory access inefficiency and low memory bandwidth utilization. Parashar et al. [26] proposed an SCNN accelerator for a sparse convolutional neural network which performs deep encoding of the sparse weights and activations in order to retrieve only non-zero values from memory. This accelerator increases the energy-efficiency by eliminating the multiplications involving zero weights and by maintaining the compressed format of weights and activations throughout the computation. The unstructured pruning achieves great compression ratios but has certain limitations like irregularity in the network structure and high training complexity [28].

Fine-Grain pruning is an unstructured pruning technique where each scalar weight can be pruned. Han et al. [29] adopted an iterative pruning methodology where the pruned network is trained again and again. When pruning was done to the AlexNet, the results showed that the accuracy did not fall even when the network was pruned 9x times. From work done in [45], high pruning ratios can be achieved, but the sparsity induced in the network is complicated. In fine-grain pruning technique, a threshold value is set for which the weights with values less than the threshold are pruned [30]. Zhu et al. [31] present a trade-off between the network model size and accuracy, and their work demonstrates that the large and sparse models give better performance compared to small and dense models.

Structured Pruning

This pruning technique approximates a large part of the neural network, like channel or layer. The less important connections are removed by considering their location, so that irregularity is not introduced in the network structure. Depending on the pruning granularities, we have Intra-kernel pruning, Kernel pruning and Filter pruning. The largest granularity exploited by the pruning is deleting a single feature map or many feature maps. Pruning an entire FM implies pruning all the incoming and outgoing kernels of the FM. The next granularity in pruning is removing kernels completely where a kernel implies one complete convolution; this is Kernel pruning. The lowest granularity is exploited by Intra-kernel pruning.

Intra-kernel pruning is a structured pruning technique where the set of weights in a kernel with a regular structure are pruned. Anwar et al. [32] used a particle filtering approach to determine the important connections and paths in the network, followed by optimization for the data representation. Their approach reduced the convolution complexity by utilizing both the intra-kernel striding and convolutional lowering.

Kernel pruning is structured pruning technique where any kernel from the output feature maps can be pruned. Anwar et al.[33] combined both feature map pruning and kernel-level pruning for the CIFAR-10 dataset, and the results showed that kernel pruning gives better results by reducing the number of parameters for the convolutional layers. Their work does not pose constraints regarding the pruning mask for the outgoing kernels of the FM as that of in [32] for reducing the size of FM and kernels. This type of coarse pruning technique gives a sparse representation of the network, which is beneficial for VLSI based implementations. Kernel level pruning is coarser than intra-kernel pruning and finer than FM pruning which makes it to achieve better pruning ratios.

Filter pruning is a structured pruning technique where a group of kernels belonging to same output feature map are pruned. Molchanov et al.[34] proposed a Taylor-expansion based pruning criteria to reduce the cost of hardware and also exploited filter pruning technique.

In the 1990s, the optimal brain damage study was performed where the diagonal Hessian approximation was used for computing saliency of the parameters and the less salient

Table 1: FPGA based CNN acceleration with Pruning technique

Contributors	**Pruning type**	**Network layer**	**Network structure**	**Device**	**Frequency (MHz)**	**Bit width**	**Hardware resource utilization**	**Accuracy (%)**	**Achievements**
Sun et al., 2017 [41]	Unstructur-ed	FC layer	AlexNet	Virtex-7 VX485T FPGA	100	16- bit fixed- point	2378 (D), 287396 (L), 498.6 GOP/s (T),23.4 (P)	-	The power efficiency of 21.3 GOP/s/w and number of weights in FC layer reduced by 10x
Zhang et al., 2019 [14]	Unstructur-ed	Convolutional and FC layers	AlexNet	Xilinx Zynq ZCU104	300	8- bit fixed -point	696 (D), 101953 (L), 198.50 (B),290.40 GOP/s (T) for convolutional layers,14.11 GOP/s (T) overall, 17.6 (P)	57.14 reverse accuracy and 57.18 peak accuracy	AlexNet size reduces from 240 MB to 8.73 MB, 182.3x and 1.1x improvements for latency and throughput respectively
Caiwen et al., 2017 [37]	Structured	FC layer	AlexNet	Altera Cyclone V 5CEA9 FPGA	250	16- bit fixed- point	700 GOPs (T)	56	6 – 102x energy efficiency improvements
Seungsik et al., 2019 [15]	Structured	Whole network	VGG-16	Stratix V FPGA	100	7b linear quantization	324 (D)	53.46	Upt o 30% reduction in memory footprint
Liqiang et al., 2019 [28]	Unstructur-ed and structured	Convolutional layer	VGG, AlexNet, ResNet, GoogLeNet	Xilix ZCU102	200	16- bit fixed- point	1144 (D), 522k (L), 912 (B), 23.6 (P),309 GOP/s (T), 223 GOP/s (T), 291.4 GOP/s (P), 257.4 GOP/s (P) respectively	-	2.4x – 12.9x speed up and 89.2%, 88.3%, 76.5%, 65.8% sparsities for the mentioned networks without any accuracy loss
Niu et al., 2019 [16]	Structured (Kernel)	Convolutional layer	VGG-16	Xilinx Virtex-7 XC7VX690T	200	16- bit fixed- point	3200 (D), 237k (L), 1200 (B)	90.8	24x higher throughput with this implementation
Kang et al., 2019 [31]	Structured	Whole network	VGG-16	Xilix Virtex 7	210	-	3074 (D), 181k (L), 1470 (B), 10.14 (P),8975 GOPS (T)	78.3	87.5% pruning is achieved and 42 FPS frame rate
Rastislav et al., 2019 [40]	Unstructur-ed (fine-grain)	Convolutional, pooling and FC layers	AlexNet VGG-16	Xilinx ZCU102	214	16- bit fixed- point	237 (D), 244464 (L), 516 (B),68.71 (T) GOP/s, ,213.26 GOP/s (T) respectively	-	14.10 times faster than Eyeriss accelerator
Li et al., 2019 [42]	Stuctured pruning	Convolutional and FC layers	VGG-16 ResNet-18, ResNet-152	Xilinx Zynq ZC706	140	16- bit fixed-point	592 (D), 218600 (L), 545 (B) ,130.1 GOP/s (T) for convolutional layers and 109.18 GOP/s (T) for FC layers in case of VGG-16,94..28 GOP/s (T), 115.56 GOP/s (T) respectively	-	Offers high flexibility and 88% reduction in weights for FC layers

D : DSPs, L : LUTs, B : BRAM, P : Power(W), T: Throughput(GOP/s)

parameters of the network were pruned using second-order Taylor's approximation [35]. Hassibi et al. do even the work [36] used pruning to remove the less salient weights where the inverse Hessian matrix was used to get the saliency of the parameters in the network. Sicheng et al. [38] proposed a hardware-software co-design framework to speed up the acceleration of sparse CNNs by using sparsification schemes and the network was implemented on Xilinx Zynq ZC706 and has taken less processing power compared to a dense CNN. Seungisik et al. [15] proposed a network stacking strategy where multiple networks with different pruning ratios are compressed by stacking the networks. In their proposed work, the network requiring considerable memory resources were trained using highest pruning ratio and then using structured pruning strategy, an accurate model was developed for edge level applications. While performing the pruning, a sparse network is obtained with irregularity and also random connections causing limitation to the CNN inference. Liqiang et al. [28] proposed a hardware accelerator on FPGA for the inference of sparse CNNs. Their work proposed weight-based dataflow technique which performs element-wise matrix multiplication instead of the spatial convolution. Heavy pruning of the spatial neural network may reduce the computation and overhead memory problems but may not have faster inference. The work done by Niu et al. [16] was the first work to prune the spectral CNNs and accelerate them. Their work proposed SPEC^2, a new way to speed up the inference of the spectral CNNs. Their work reduces the computations by using the spectral transformation and uses kernel pruning for reducing the memory resources to store the parameters. The buffer multipliers to store weights are not used efficiently when a pruned CNN is accelerated on the hardware. This is because of the irregular sparsity and miss-alignment caused by pruning. In order to overcome this problem Kang et al.[39] used an accelerator-aware pruning technique and accelerated on XC7VX690T FPGA so that the weight storage buffers and multipliers are used effectively.

Table 1 presents the various works done in recent years for the CNN acceleration using pruning technique. Various metrics of the CNN implementation on the FPGA platform are compared, and the achievements of the works are discussed. Many works presented, have used the Imagenet dataset for their evaluation and achieved more significant results for image recognition and detection applications using the structured pruning technique. From the table, it can be inferred that structured pruning has been widely used for getting better compressibility of the network and performance throughput. Kernel level granularity of the structured pruning has achieved better accuracy results compared to other structured pruning granularities, and this gives 100% pruning. For significant computations savings on hardware, Intra-kernel sparsity has a greater advantage. Larger performance gains are observed when pruning and quantization were used together to approximate CNN. The structured pruning is well suited for the CNNs but may not be extended for other neural networks like LSTM, which lack structural properties. So efficient optimizations can be made to structured pruning technique to make it extensible for other neural networks. The unstructured pruning obtained high pruning ratios but induces irregularity into the networks, which causes over-fitting problem and memory access burden. This gives a scope to find optimizations in the unstructured pruning to remove the irregular sparsity problem.

B. Weight Sharing

Weight sharing is another type of approximate computing technique applied at the structural level of the CNN model. Weight sharing technique does not reduce the computational workload, but the memory required to store these weights is decreased, thus decreasing the memory overhead. Chen et al. [43] used HashNet, which applies a hashing trick to the network to find shared weights for a layer. Their technique determines weight sharing before the training, but the work proposed in [44] finds the weight sharing for a fully trained network. Han et al.[44] introduced an in-depth compression approach which follows three pipeline stages, pruning the network, quantization for the trained network and Huffman coding. In their work, the number of essential weights is limited by making the multiple connections share the same weights and later the shared weights are fine-tuned. Their work used k-means clustering technique to find the shared weights for every layer of the neural network and has achieved 35x to 45x reduction in the storage without degrading the accuracy. Parallel accumulate share MAC (PMAC) is used in a weight-shared CNN in order to accelerate the CNN in terms of power and area [45]. Weight sharing can be the area of interest mainly for the applications requiring less memory bandwidths and is also applicable for the deep learning networks like RNNs, and LSTM. This technique supports the fast inference of the CNNs in an embedded platform requiring small storage areas. The pruned network has been standard on different hardware platforms, but the weight sharing for the quantized network has not been improvised. The advantage of weight sharing technique, to fit the network model in hardware is still uncovered. This advantage can be exploited by a hardware solution, where a customized ASIC architecture can be built to deploy the quantized sparse neural network by extending the advantage of having customized bit width for quantization. The HashNets showed greater results when implemented on GPU platforms, so for better hardware achievements, they can be implemented in FPGA and ASIC platforms. In further works, HashNets along with pruning can be used for even better performance.

3.2 Hardware-level Approximate Computing technique

The hardware layer is approximated in order to have simplified hardware structures for implementing the CNNs and accelerating their performance. The MAC operations in the CNNs require lots of computations and memory resources, so approximations can be made at the hardware level to ease the calculations and reduce hardware costs. So approximate adders and multipliers can be designed in order to execute MAC operations by increasing performance and energy gains quickly. Approximations can be made at a memory level for improving the energy and power requirements in the circuit.

A. Functional Approximations in computation modules

The computational module for a CNN is the MAC unit, which comprises of adders and multipliers to perform the convolution operations. The multiplication operations generally are power consuming and require sophisticated hardware units which call for approximations to achieve required performance gains. The approximations made to adders and multipliers increases the throughput and power-efficiencies. Approximate computing is exploited for adders, multipliers and dividers in order to achieve high performance for error-tolerant applications [46]. Adders are the basic arithmetic circuits in any computing systems. A multiplier comprises of mainly three parts partial product generation by AND gate, partial product reduction by an adder and adding the final result by a Carry Propagation Adder

Table 2: Approximate adders and multipliers for CNN acceleration in VLSI implementations

Contributors	Approximate adder / multiplier	Network structure	Bit width (bits)	Implementation	Performance metrics ASIC/FPGA	Frequency (MHz)	Accuracy (%)	Achievements
Kim et al., 2015 [51]	Approximate Mitchell's log multiplier	AlexNet	16,32	32 nm digital standard cell library, synopsis design compiler	0.61 mW (P), 1.08 mW (P) respectively,113.5 um^2 (A), 209um^2 (A) respectively	190	84.87	Proposed design saves 80% of energy compared to 32 bit fixed point multiplier
Kim et al., 2018 [52]	Approximate Mitchell's log multiplier	LeNet CudaConvNet	8, 16, 32	32 nm digital standard cell library, synopsis design compiler	0.197 mW (P), 0.549 mW (P), 1.41 mW (P) respectively,312 um^2 (A) , 909 um^2 (A), 2161 um^2 (A) respectively	250	99.02,81.43 respectively	76.6 % power reductions compared to fixed point multiplier
Faraone et al., 2019 [53]	Reconfigurable constant coefficient multiplier (RCCM),2 ADD, 3-ADD,4-ADD	AlexNet	4,6,8	Xilinx KU115 FPGA	7.6W (P), 7.6W (P), 7.8W (P) respectively,187 k (L) , 205.6 k (L), 255.8 k (L) respectively	250	79.8, 79.8, 80	50% resource savings is obtained
Wang et al., 2019 [54]	Approximate Multiply Accumulate Array (AMMA)	Random input data samples	8, 12, 15	Xilinx XC7Z020	432 (L), 922 (L), 1186 (L)	236.74	-	10.7x faster than an exact multiplier
Ansari et al., 2019 [59]	Cartesian-genetic programing (CGP) based approximate multipliers	LeNet	-	28 nm CMOS technology, Synosis design compiler	4330 mW (P), 92.86 um^2 (A), 87 (L)	-	87	71.45% reduction in energy and 61.55% reductions in area
Kowsalya et al., 2019 [58]	Pipelined hybrid merged adders (PHMAC)	-	13, 32, 64	Xilinx Virtex 7 FPGA	2 mW (P), 11.70 mW (P), 45.56 mw (P),16 (L), 25 (L), 233 (L)	-	-	Area and power consumptions are reduced by 50%
Luo et al., 2019 [57]	Single Clock Cycle Adder (SCCA)	LeNet	-	65 nm CMOS technology	0.417 mW (P), 799.9 um^2 (A)	-	98.7	Speed up increased by 2.8x and 59.9% reduction in PDP
Chuliang et al., 2020 [55]	Reconfigurable approximate multiplier	VGG-16	-	Xilinx ZCU102 FPGA	0.21 mW (P), 129 (L)	200	-	17% and 15% reductions in latency and power savings

P : Power(W), A : Area(um^2), L : LUTs

(CPA). The approximations can be performed to the multipliers at these three parts [47]. Using logarithmic multiplier for approximating the computations in neural networks is a promising research area. Many works have been developed having improvements over original work done by Mitchell et al. [48]. This multiplier has given significant performance in terms of power and area by maintaining low error rates [49],[50]. To reduce the critical path delay, Yizhi et al. [21] proposed approximate binary multipliers for CNNs with binary weights which use 2's complement to represent the data. Based on the error tolerance property of the CNNs, they also proposed approximate adders for the data path of their
architecture, which reduces the delay and area consumption. Reconfigurable constant coefficient multiplier (RCCM) is proposed in [53] for area-efficient implementation of CNN on FPGA. In their proposed accelerator, the coefficients are made constant, and input is multiplied to the fixed coefficient using multiplexers or bit shifts so that the multiplier is highly optimized for FPGA implementation. For the quantized CNNs, the data is represented with different bit precision. Implementation of such CNNs may require different multipliers for different precisions at various layers of the network, thereby causing area overhead and increasing the hardware cost. For such quantized CNN applications, Chuliang et al. [55] proposed a reconfigurable approximate multiplier which can implement multiplications involving various bit width representation of the data. In this approximate multiplier, the long bit width multiplications are made into short ones, and later the results are combined using a merging module. Kamel et al. [56] investigated two strategies to reduce the hardware cost of the adders, which are used to map the CNN directly on to the hardware. Their

strategy, serialization of adders and approximating computing of the adders when synthesized on FPGA did not yield greater results. Thus they put forth an idea of having individual DSP blocks in the FPGA. Their work explored how the multi-operand adders are challenging for CNNs on the FPGA.

Table 2 presents various works using approximate adders and multipliers for the CNN acceleration with hardware implementations. From the table, it can be observed that the approximated log multipliers require less power compared to other approximate multipliers for the ASIC implementations. The Approximate binary multiplier has less area requirements compared to other binary multipliers. It is also observed that more significant achievements in terms of area and power have been obtained by using these approximated adders and multipliers for quantized CNNs. The approximate multipliers can still optimize the CNNs in further works when combined with pruning and matrix decomposition approaches. This gives customized arithmetic, a new dimension to be explored. The studied approximate multipliers and adders can further enhance their performance by using Stochastic Computing approximation.

B. Functional Approximations in memories

At the hardware level, the memories face limitations like high memory bandwidth, latency for memory access, memory congestion and power consumptions due to leakages for CNN accelerations. The approximate computing is a paradigm which allows the error tolerance for many applications in order to have power and energy-efficient implementations. So with this inexactness in computing, the memories can also be approximated to have required savings in energy. The approximations in memory include reduction of refresh rate reduction for DRAM, threshold voltage scaling for SRAM and approximation in SRAM registers. Many modern accelerators have widely used embedded DRAMs (eDRAMs) because of their high density. Shafiee et al.[60] proposed a memristor crossbar array-based CNN accelerator with pipeline architecture and used eDRAM buffers for fast data transfers between the network layers.

Chowdhury et al. [61] proposed a memristive accelerator for binary CNNs called MB-CNN, which performs the XNOR based convolutions and bit-count operations in the 2R crosspoint arrays within the memory. Fengbin et al.[62] proposed Retention-aware Neural Acceleration (RANA) framework which is based on the principle that the DRAM refreshment is not required if the lifetime of eDRAM's data is less than the retention time of the eDRAM. Their work used a refresh-optimized eDRAM controller at the architecture level of CNN and implemented RANA in TSMC 65nm GP technology. Imani et al.[63] proposed RAPIDNN, a framework to accelerate CNN by employing neuron to memory transformation. Their structure is modelled to have non-volatile memory blocks to perform the operations like additions, pooling, activations and additions.

C. Use of Non-Volatile memories

The non-volatile memories (NVMs) like PCM, STT-RAM when used in a system, they enhance the power and energy considerations. Even though hardware platforms like FPGA and ASIC are used to accelerate the CNNs, they face a challenge of data movement cost. All the weights and input values from the networks are to be stored, which asks for a dedicated memory. One solution to reduce the data movement in memories is to use Processing in-memory technique, as the logic is implemented inside the memory blocks. This PIM performs some computations like bitwise operations within the memory, thereby increasing the acceleration performance and reducing memory workload. But these NVMs face challenges regarding the lifetime of memory and the performance which limits their use for practical systems. Approximations are introduced in the systems to overcome those challenges [63].

The main aim of approximate computing in NVMs is to improve the energy efficiency while maintaining the reliability of the system. The use of eDRAM and SRAM based crossbar memories can minimize the weights transfer between the processing units and the off-chip memory but still does not give the required performance for the input and output data movement. Chi et al. [64] proposed PRIME, a new PIM (Processing-in Memory) architecture for CNN acceleration which has ReRAM as main memory. In their architecture, with the PIM logic, a part of computations can be performed within the memory, thus obtaining high memory bandwidth through in-memory communication of the data. Deliang et al. [65] used in-memory computations for CNN acceleration. They proposed an architecture which works both as NVM and reconfigurable in-memory logic to perform the convolution layer operations. SRAMs can be substituted by many NVM technologies like phase-change memory, RRAM and STT-RAM for mainly minimizing the standby power. The STT-RAM technique is highly adopted because of its high density and power leakage, which is near to zero. Pan et al. [66] proposed a computing in-memory architecture, MLC-SST based on multilevel cells and STT-MRAM memories. This architecture was used for performing the convolution operations of BCNN to achieve less power consumptions. Sun et al. [67] proposed a co-processor for CNN with MRAM and in-memory design architecture and implemented in 22 nm CMOS technology. The weights were stored in the MRAM memory and reused for performing the operations with the input. For the faster computations, Angizi et al. [68] proposed IMCE, a convolutional accelerator for the less bit-width CNN. The SOT-MRAM proposed in their work does two works, performs AND/OR logic operations and memory read/write operations.

Table 3: Approximate memories used for the CNN acceleration in ASIC implementations

Contributors	**Approximate memory technique**	**Dataset**	**Implementation**	**Area (mm^2)**	**Energy (J)**	**Achievements**
Chi et al., 2016 [64]	PRIME, ReRAM based main memory	MNIST, ImageNet	65 nm GP technology	-	18.2pJ	~2360x performance improvements and ~895x energy consumption improvements
Shafiee et al., 2016 [60]	ISSAC, eDRAM to store weights and in-situ MAC operations	ImageNet	CACTI 6.5 at 32 nm technology	0.215	-	14.8x, 5.5x, 7.5x improvements in throughput, energy and computational density respectively
Diliang et al., 2017 [65]	SOT-MRAM array for BCNN	ImageNet	NCSU 45 nm CMOS PDK	5.28	310.42 uJ	~7x and ~1.7x reduced energy and area
Angizi et al., 2018 [68]	IMCE, SOT-MRAM based design	ImageNet, SVHN, MNIST	45 nm technology node	2.12, 0.01, 0.009	785.25 mJ, 135.26 mJ, 0.92 mJ	~3x reductions in energy to process low bit-width AlexNet
Angizi et al., 2018 [69]	CMP-PIM, SOT-MRAM based design	SVHN, MNIST	45 nm CMOS PDK	1.7	87.54 mJ, 0.14 mJ	~94x and 3x increased energy efficiency when compared with CNN and LBCNN

Angizi et al. [69] also proposed a comparator based PIM (CMP-PIM) accelerator, which uses parallel memory sub-array based on SOT-MRAM as a fundamental unit for processing CNNs. Their work used CMPNET, a modified CNN to replace the high computation multiplications with additions and comparisons. Ikegami et al. [70] proposed a CNN accelerator using Voltage control spintronics memory (VoCSM), where the binary and ternary computations are done using this NVM which gives high throughput. Roohi et al. [71] also proposed an in-memory CNN accelerator with SOT-MRAM non-volatile device. Their proposed accelerator can execute AlexNet with 3.8x and 4.5x reduction in energy and area compared to ReRAM based designs when implemented on NCSU 45nm CMOS PDK in Cadence Spectre. Joshi et al. [72] proposed a methodology to train the CNN (ResNet) with no significant accuracy loss when weights were stored in phase-change memory (PCM) devices which were implemented in 90 nm CMOS technology.

Table 3 presents the area and energy requirements obtained by using various approximated memories for the CNN acceleration in hardware implementation. It is observed that the NVM techniques used for CNN acceleration can be utilized both as a memory to store weights and to perform the convolution layer operations. It is also observed that the use of non-volatile memories has shown more significant performance improvements compared with DRAM and SRAM memories. The MultilevelMultilevel Cell strategy can be used by SOT-MRAM based designs for BCNNs and extended network architectures to have greater in-memory computing and less power consumptions. The MB-CNN architecture uses a memristive crossbar array to perform the computations using XNOR gates, which makes this technique to be used for more complex CNN structures in future. The Stochastic Computing strategy can be further used with these approximated memories even further to increase the convolution computations.

3.3 Hardware-software co-design approximate computing

In the state-of-art of the neural network applications, the networks are not much suitable for edge level computations because of the increased number of computations and requirement of large memory bandwidth. Memory congestion and computation complexity occur due to high precision representation of the network parameters (bit width) and the data paths. For the CNN inference, approximate arithmetic can be a better option to design simplified network accelerators by reducing the complexity of the design. Many hardware-software co-design techniques are being used to reduce the memory overload and reduce energy consumption for the implementation of neural networks.

The arithmetic operations performed by using a floating-point representation of the data makes the computations complex and increases the complexity of hardware units like adders and multipliers. Many approximations have been made even for the floating-point representation units in recent years. Lai et al, [2017][73] proposed a numbering scheme which uses floating-point representation for the weights and fixed-point representation for the activations. Their work made a study on range Vs precision and concluded that floating-point representation could provide variable range and precisions compared to fixed-point description. Block floating point (BFP) arithmetic with quantization and rounding schemes have been used in recent trends to improve the hardware and energy efficiencies [74],[75]. In order to have reduced computational complexity and hardware costs, the floating-point arithmetic is replaced with reduced precision formats discussed below.

A. Fixed point

For faster performance and less memory or resource consumption, the floating-point representation can be replaced with fixed-point arithmetic by not prioritizing the precision. For fixed-point arithmetic the data representation is an integer, so the operations on such data can be easily implemented in FPGAs. Hamerstrom et al. [76] proposed an on-chip learning framework to have high performance and low-cost architecture which used 8-16 bit fixed-point representation. When designing complex CNN, using DSP blocks to store the network causes issues like chip area wastage, and also a large number of unimportant details of the DSP block confuses. Ahmed et al. [79] proposed an 8-bit fixed point parallel MAC unit for full customization of the FPGA accelerator of CNN instead of using DSP blocks, achieving high computational speeds. Solovyev et al. [24] proposed an FPGA implementation for CNN with the fixed-point representation of the data in convolutional blocks for handwritten digit recognition application. The bit width for data representation in convolution block was chosen depending on whether rounding the values after each addition and multiplication or rounding the values at the end of convolution. It is observed that turning the value after each elementary operation results in higher performance but also increases memory overhead. So it is suggestible to perform rounding of the value after convolution. Xiao et al. [17] proposed an FPGA implementation of the CNN for handwritten digit recognition where all the weights and parameters were represented in 18 bit fixed point format where the accuracy obtained was 97.5% for recognizing the MNIST data set digits. It is seen that handwritten digit recognition by CNN with fixed-point arithmetic has higher accuracy compared to that of floating-point arithmetic in [80]. Hashemi et al. [81] made a study on different representations of data, and they evaluated for three different datasets, and also they proposed a methodology to maintain accuracy for increasing network size at low precision.

B. Dynamic Fixed point

For all the network layers, the values associated with weights and parameters may have a different range at different layers of the network. The fixed point representation should be capable of providing a wide range of values for different network layers, or else there will be accuracy degradation. This dynamic fixed-point representation is used for the intermediate values of the network layers. The wide dynamic range coverage limits the fixed-point representation. So the dynamic fixed point is used when large range for the activations of the network layers is allowed. Ristretto, an approximation framework is introduced to analyze the CNNs regarding the numerical representation of the weights and outputs of both convolutional and fully connected layers [19]. This framework uses dynamic fixed-point representation and results when compared, says that the fixed point maintains the accuracy for bit width up to 18, below which there is accuracy degradation. With this type of representation of the network, the advantage is that the computation complexity and the resources required are reduced.

C. Power-of 2 Quantization

It is known that for any CNN, the multiplication operation requires a lot of computation, and it requires a complex design to be implemented in hardware. So the multipliers are to be replaced in order to have power and area-efficient hardware units. This power-of-2 is one such approximate computing technique where the weights in the CNN are quantized to 2^i format. This type of quantization represents the data with low bit width and gives less accuracy degradation. This approximation allows using barrel shifters instead of multipliers for performing the operations, thus reducing the complexity. In many of the neural networks, the weights and activations may not be distributed uniformly. For such non-uniform distribution of data, non–linear quantization can be used. Vogel et al. [83] presented a power-of-arbitrary-log based quantization for the pre-trained CNN, and their performance in terms of power and energy efficiency were compared with 8-bit fixed point multiplier. Using different precision and different arithmetic for the same CNN structure can give better results with less accuracy degradation. Zhao et al. [84] was the first work to use the multi-precision and multi arithmetic representations for the CNN acceleration on the FPGAs. They proposed a *Tomato* framework for CNN acceleration, where they used both power-of-2 and fixed-point representation for the weights. Large bit reduction for the data representation in the neural network suffers from the accuracy loss. This challenge is addressed by Fong et al. [86], where they proposed Incremental Network Quantization (INQ) strategy. The weights are represented in low bit format as power-of-2 such that computation difficulty is minimized as general shift operations replace the multiplications.

D. Binary Quantization

This type of quantization reduces the accessing of memory as the weights and parameters are represented with a single bit, thus reducing the memory bandwidth. For this quantization, the complex multipliers are replaced with multiplexers. In this type of representation, the weights are represented in binary. If the weights and activations of the CNNs are described in the binary format, then those neural networks are called Binarized Neural Networks (BNNs). There are two types of binarization:

1. Binarized weights and activations with full precision
2. Binarized weights and activation

The first type uses conditional negation for multiplications and reduces the memory requirements for the storing of weights. In the second type, the MAC operations are replaced with XNOR based activities and signed bit count. It is known as *full-binarization* when all the input, output activations and the weights are represented in binary. If either of those is binary, then it is known as *partial-binarization*. The BNNs

Table 4 : FPGA based CNN acceleration using approximate arithmetic

Technique	**Contributors**	**Network layer**	**Bit width (bits)**	**Accuracy**	**Dataset**	**Device**	**Frequency (MHz)**	**Hardware resource utilization**	**Remarks**
Fixed point	Gupta et al., 2015 [77]	Convoluti-onal and pooling layer	16	-	CIFAR-10	Xilinx Kintex325 T FPGA	166	812(D), 62922(L), 334(B), 7W(P)	Energy efficiency and computation al throughput are increased
	Zhou et al., 2015 [90]	Convoluti-onal and pooling layers	11	8 (bad/250fr ames)	MNIST	Virtex-7 FPGA	150	83(D), 80175(L), 0(B)	16.42x faster implementat ion compared to PC platform
	Zhisheng et al., 2017 [91]	Convoluti-onal and FC layers	8	98.16%	MNIST	Xilinx 485T	-	574(D), 7204(L), 343.5(B), 0.47W(P)	31.43% and 47.95% reductions in latency and power compared to 32 bit floating point engine
	Wijeratene et al., 2018 [78]	Convoluti-onal layer	32 Q point	-	ImageNe-t	Xilinx Virtex 7	200	576(D), 117k(L), 226.2 GOPs(T)	Reduced resource utilization with a throughput of 226.2 GOPs
	Solovyev et al., 2018 [24]	Convoluti-on and FC layers	12	96%	MNIST	DE0-Nano developme nt board	143	5947(L)	Overall processing speed of 150frames/s ec is achieved
	Edwin et al., 2019 [92]	Whole network	12	97.59%	MNIST	Xilinx Zynq 7000, SoC	100	158(D), 4254(L), 45(B)	17% higher throughput compared to software implementat ion
	Xiao et al., 2020 [17]	Whole network	18	97.57%	MNIST	Cyclone 10 FPGA	150	274(D), 12588(L)	Highest recognition rate for LeNet has been achieved
	Cho et al., 2020 [18]	Whole network	11	98.64%	MNIST	Xczu9eg-f fvb1156-2 -I FPGA	-	143(D), 32589(L), 95(B)	40% and 90% reductions in memory usage and latency
Dynamic Fixed Point	Qiu et al., 2016 [93]	Whole network	16	86.66%	ImageNe-t	Xilinx Zynq ZC706	150	780(D), 182616(L), 486(B)	Only 0.4% loss in accuracy and a frame rate

									of 4.45 fps
	Yao et al., 2018 [20]	Convoluti-onal, pooling and FC layers	16	-	ImageNe-t	Zynq-XC7 Z100	-	364(D), 310688(L), 543.6(B), 9.63W(P)	0.054GOPS/ DSP and 5.24GOPS/ w performance density and power efficiency are obtained respectively
	Ding et al., 2019 [82]	Whole network	16	-	Serial network structure	Cyclone V FPGA	125	80% DSP utilized, 3.78W(P), 6.63 GOPS(T)	The precision error obtained is 1% for 16 bit fixed point representati on
Power-of-2	Vogel et al., 2018 [83]	Covolutio-nal layer	5b	80.59	ImageNe-t	Xilinx Virtex7	--	9(D), 29.04k(L), 740(B), 4.329W(P)	22.3% reduction in power consumption s and reduced resource utilization
	Zhao et al., 2019 [84]	Convoluti-onal layers	mixed	68.02	ImageNe-t	Intel Stratix V	156	256(D), 362.7k(L), 828(B), 3536 GOPs(T)	Achieved a frame rate of 3000fps and very little latency
	Piyasena et al., 2019 [85]	Convoluti-onal layers	2,3	99.9,97.53	MNIST CIFAR-10	Xilinx Virtex ultrascale^{+} XCVU9P	100	680k(L), 30.5(B), 1.926W(P)	Achieved greater power savings and run time computation s
	Fong et al., 2019 [86]	Convoluti-onal and FC layers	32b	78.17	Imagene-t	Stratix V GXA7	155	0(D), 155.5k(L), 2061(B), 8.694W(P), 195.350 GOPs(T)	1.87x throughput improvemen ts and 20% improvemen ts in latency
Binary quantization	Yaman et al., 2016 [94]	Convoluti-on, pooling and FC layers	1	95.8	MNIST	Xilix Zynq ZC706	200	91131(L), 4.5(B), <22W(P)	Achieved greater throughput
	Liang et al., 2017 [88]	Convoluti-onal and FC layers	1	98.24, 86.31, 66.80	MNIST, CIFAR-10, AlexNet	Altera stratix V fpga	150	384(D), 2210(L), 26.2(B)	Greater seed up is achieved compare to CPU platforms
	Zhao et al., 2017 [95]	Convoluti-onal and FC layers	1	-	CIFAR-10	Xilinx Zynq 7000 SoC	143	3(D), 46.9k(L), 4.7W(P), 207.8 GOPs(T)	15.1x better performance and 11.6x improved throughput

	Guo et al., 2018 [96]	All layers	1	88.61, 96.9	CIFAR-10, SVHN	Xilinx Zynq ZC702	-	29.6k(L), 103(B), 3.2W(P), 2236 GOPs (T), 722 GOPs(T)	3.1x, 5.4x, 4.9x improved performance, resource efficiency and power efficiency
	Guan et al., 2019 [97]	-	1	97.53	MNIST	Intel Altera 5CEBA7F	125	19(D), 38794(L), 615(B)	Weight reduction by 4-6x and 14x reduced latency

D : DSPs, L : LUTs, B : BRAM, P : Power, T: Throughput

have the advantage of efficiently mapping the network to the hardware without degrading the accuracy of the network. In a CNN, the input to the first layer will be from average pooling layer where the data is in either fixed format or floating format. In order to perform the multiplications, the binarized weights use multiplexers instead of XNOR gates. But for a fully connected layer, the input comes from batch-normalization layer, so XNOR gates replace multiplication in the dot product [87]. Liang et al. [88] used bit-level XNOR gates and shifting operations to reduce the bottleneck caused by using multipliers for MAC operations and also used data quantization techniques to reduce the memory footprint. The inference of this BNN on Stratix-v FPGA achieved tera operations per second (TOP/s) performance with less accuracy loss. Two approximations, CNNs with binary weights and CNNs with XNOR gates, are introduced for faster and less memory required convolution operations [89]. In binary weight CNNs, the weights are represented in binary, so the convolutions involving binary data are fast and require less memory to store the results. For the XNOR networks, both the weights and inputs to the network layers are in binary format. Wang et al. [30] proposed the LUTNet, a neural network accelerator which uses K-LUTs as operators for the inference of a neural network. The XNOR based dense BNN architecture is modified to K-LUTs based sparse network where K-inputs are directly mapped to K-LUTs. This approach used in [30] has achieved higher efficiencies in area and energy compared to regular BNN accelerators.

Table 4 gives comparisons between different network level approximations made to CNNs. Resource utilization and performance in terms of power and throughput are also compared. It is observed that the highest accuracy for MNIST dataset is found by using power-of-2 quantization, for ImageNet by using dynamic fixed-point representations. The fixed point representations have achieved higher accuracies but less throughput compared to dynamic fixed-point representation. The advantage of this fixed point representation can be further increased at the hardware level by using parallel and pipelined convolutional units to reduce the computation complexity and increase the inference speed. The multipliers to perform the convolution operations with these fixed-point numbers can be built using LUTs, thus increasing the utilization of hardware resources. The heterogeneity property can be exploited in FPGAs to make efficient use of DSPs by allowing different bit widths along with binarized data. High throughputs can be obtained by utilising the data-level parallelism in the PEs along with the fixed point and dynamic fixed-point representations. Simple rounding schemes for fixed-point representations can be identified in order to have reduced hardware complexity. The use of binary quantization for CNNs has dramatically reduced the memory footprint and obtained higher speeds compared to the other approximate arithmetic circuits. These approximate arithmetic can be further extended to lightweight neural networks for further performance increments.

E. Stochastic Computing

The stochastic computing (SC) differs from conventional computing concerning the representation of the numbers in the network. In conventional computing, base two notation is used for representing the numbers. In stochastic computing, the numbers are represented with probability p, which has a stochastic value v. The stochastic value is single bit representation and may vary with the system clock. The stochastic value is determined in two ways, non-polarized and polarized. In non-polarized mapping, the probability (p) is directly mapped to the logical value (v). In polarized mapping, the probability (p) of range [0,1] is mapped to its logical value (v) of range [-1,1]. The negative numbers cannot be represented using non-polarized mapping. This is a low-cost approximation technique which uses small circuits to perform the operations. This SC, when applied to the neural network, has the advantage of enhanced power efficiency and reduced hardware cost. In SC simple logic gates like AND, XOR is used to perform the multiplications and multiplexers are used to perform the scaled additions. This SC is a substitute for binary computing at a low-cost implementation. The major advantages of using this SC to accelerate the CNNs is the reduced hardware cost and less memory and power consumptions. The convolutional block in a CNN is the feature extractor, and Convolutional block requires a large number of operations. This computational complexity can be reduced using the SC for the feature extractor operations. One such method is proposed by Ren et al. [98], by implementing four designs for the feature extractor block and weight storage methods are proposed to reduce the area consumption. The SC-based feature extractor design is presented and optimized form the view of precision calculation and their hardware design achieved significant reductions in area, power and energy when compared to CPU, GPU and binary ASIC based implementations [99]. SC for CNN basis functions like inner product calculation, pooling and activation function is presented by Hamdan et al. [100] to exploit the correlation and the network is synthesized targeting the Xilinx Zynq Z706 FPGA. Their work proposed a MUX tree to calculate the inner product through SC for the convolutional layers, and it has more accuracy and requires less hardware compared to a conventional method. For having reduced memory resources, Xiaolong et al. [101] proposed Domain Wall Memory (DWM) technique for SC-DCNNs. This DWM is a high-density memory and non-volatile which is used to replace SRAMs. An effective resource sharing scheme is proposed for storing the weights of convolutional and fully connected layers based on DWM strategy. HEIF, an SC based framework, is proposed by Li et al. [102]for the applications including LeNet and AlexNet. In their work, the required optimizations were done on the functional block connections in the CNN, reducing the bitstream length and achieved 6.5x area efficiency and 5.6x energy enhancements.

The weights of CNN represented in binary reduces the complexity to perform the MAC operations and also reduces the memory footprint. The binarization of the weights of CNN can be either *deterministic* or *stochastic*. The FPGA implementation of stochastic BNN is proposed to enhance the learning ability of the BNNs [103]. With this stochastic binarization, the inference speed for both the MNIST and CIFAR-10 datasets have been increased significantly compared to a non-regularized BNN. Two SC-based designs are compared in terms of area, power consumption and accuracy [104]. Their analysis for MNIST handwritten digit recognition dataset says that Binary Interface Stochastic Computing (BISC) has outperformed the Extended Stochastic Logic (ESL) which is 50x faster and consumes less area and power. The limitation with the SC-based accelerators is that they can implement shallow neural networks having limited depth. This gives the insight to optimize the SC-based designs further and extend its advantages to more significant and complex networks. The SC approach can also be used along with other approximation techniques like pruning and weight sharing to reduce hardware complexity and increase the inference speeds. These SC-DCNN implementations can further be extended to use many other NVMs like Re-RAM, SST-MRAM and SOT-MRAM for storing weights instead of SRAMS.

4. CONCLUSION

This survey presented the various approximate computing techniques applied to CNNs for acceleration. The dependencies of performance metrics like accuracy, power, area, throughput with approximation techniques for image processing applications are enumerated. Various works for each approximation techniques are compared for various image processing applications like classification, detection and recognition without a loss in the accuracy. At the outset, the approximation techniques such as pruning, weight sharing and reduced precision allow re-training of the network by which a part of efficiency lost during the use of these approximations can be regained. The hardware approximations like approximating computational models, approximating memories do not support re-training since it is difficult for keeping track of all errors occurred due to approximations. This limitation makes the accuracy loss non-recoverable, and thus its scope is limited in hardware for the deep learning applications. These approximations can further achieve more magnificent performances when it is applied to lightweight neural networks. This survey widens the window in the field of deep learning acceleration to further extend the research scope with the insights provided.

REFERENCES

1. O. Russakovsky et al., “**ImageNet Large Scale Visual Recognition Challenge**,” Int. J. Comput. Vis., vol. 115, no. 3, pp. 211–252, 2015. https://doi.org/10.1007/s11263-015-0816-y
2. A. G. Scanlan, “**Low power & mobile hardware accelerators for deep convolutional neural networks**,” Integration. VLSI Journal, October, 2018.
3. S. Chokkadi, “**A Study on various state of the art of the Art Face Recognition System using Deep Learning Techniques**,” International Journal of Advanced Trends in Computer Science and Engineering, pp. 1590–1600, 2019. https://doi.org/10.30534/ijatcse/2019/84842019
4. R. Girshick, “**Fast R-CNN**.” In Proceedings of the IEEE international conference on computer vision, pp. 1440-1448, 2015.
5. M. Syamala, “**A Deep Analysis on Aspect based Sentiment Text Classification Approaches**,” International Journal of Advanced Trends in Computer Science and Engineering, pp. 1795–1801, 2019. https://doi.org/10.30534/ijatcse/2019/01852019
6. K. Ovtcharov, O. Ruwase, J. Kim, J. Fowers, K. Strauss, and E. S. Chung, “**Accelerating Deep Convolutional Neural Networks Using Specialized Hardware**,” Microsoft Research Whitepaper pp. 1–4, 2015.

7. N. Suda et al., "**Throughput-Optimized Open-CL based FPGA accelerator for Large-Scale Convolutional Neural Networks**," In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* pp. 16–25, 2016.
8. M. A. Hanif and M. U. Javed, "**Hardware-Software Approximations for Deep Neural Networks**," Approximate Circuits, pp. 269–288,2019.
9. M. Lass, D. K. Thomas, C. Plessl, and S. Member, "**Using Approximate Computing for the Calculation of Inverse Matrix p -th Roots**," IEEE Embedded Systems Letters, vol. 10, no. 2, pp. 33–36, 2017. https://doi.org/10.1109/LES.2017.2760923
10. T. Ayhan and M. Altun, "**Circuit Aware Approximate System Design With Case Studies in Image Processing and Neural Networks**," IEEE Access, vol. 7, pp. 4726–4734, 2020.
11. C. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkataramani, "**Exploiting Approximate Computing for Deep Learning Acceleration**," In 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 821–826, 2018.
12. N. P. Jouppi et al., "**In - Datacenter Performance Analysis of a Tensor Processing Unit**," In Proceedings of the 44th Annual International Symposium on Computer Architecture, pp. 1–12, 2017..
13. A. B. Kahng and S. Kang, "**Accuracy-configurable adder for approximate arithmetic designs**,"In Proceedings of the 49th Annual Design Automation Conference, pp. 820-825. 2012. https://doi.org/10.1145/2228360.2228509
14. M. Zhang, L. Li, H. Wang, Y. Liu, H. Qin, and W. Zhao, "**Optimized Compression for Implementing Convolutional Neural Networks on FPGA**,"Electronics, vol.8, no.3,pp. 295, 2019.
15. S. Moon, S. Member, Y. Byun, and S. Member, "**Memory-Reduced Network Stacking for Edge-Level CNN Architecture with Structured Weight Pruning**," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 4, pp. 735-746, 2019.
16. Y. Niu et al., "**SPEC 2 : SPECtral SParsE CNN Accelerator on FPGAs**,"In 2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC), pp. 195–204, 2019. https://doi.org/10.1109/HiPC.2019.00033
17. R. Xiao, J. Shi, and C. Zhang, "**FPGA Implementation of CNN for Handwritten Digit Recognition**," In 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) pp. 1128–1133, 2020.
18. Y. Kim, "**Implementation of Data-optimized FPGA-based Accelerator for Convolutional Neural Network**." In 2020 International Conference on Electronics, Information, and Communication (ICEIC), pp. 1-2. IEEE, 2020.
19. Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi, "**HARDWARE-ORIENTED APPROXIMATION OF CONVOLUTIONAL NEURAL NETWORKS**," arXiv preprint arXiv, 2016.
20. Y. Yao et al., "**A FPGA-based Hardware Accelerator for Multiple Convolutional Neural Networks**," 2018 14th IEEE Int. Conf. Solid-State Integr. Circuit Technol., pp. 1–3, 2018.
21. Y. Wang, J. Lin, and Z. Wang, "**An Energy-Efficient Architecture for Binary Weight Convolutional Neural Networks**," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.26, no.2, pp. 280-293, 2017.
22. Y. Lecun, P. Haffner, L. Bottou, and Y. Bengio, "**Object Recognition with Gradient-Based Learning**," Shape, Contour and Grouping in Computer Vision Lecture Notes in Computer Science, pp. 319–345, 1999. https://doi.org/10.1007/3-540-46805-6_19
23. J.-K. Kim, M.-Y. Lee, J.-Y. Kim, B.-J. Kim, and J.-H. Lee, "**An efficient pruning and weight sharing method for neural network**," 2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), 2016.
24. R. A. Solovyev, A. A. Kalinin, A. G. Kustov, D. V. Telpukhov, and V. S. Ruhlov, "**FPGA Implementation of Convolutional Neural Networks with Fixed-Point Calculations**," arXiv, vol. 1808.09945, pp. 1–9, 2018.
25. Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, and Y. Zhou, "**Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Network**," ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 10, no. 3, pp. 1-23, 2017.
26. A. Parashar et al., "**SCNN : An Accelerator for Compressed-sparse Convolutional Neural Networks**." ACM SIGARCH Computer Architecture News, vol.45, no. 2, pp. 27-40, 2019. https://doi.org/10.1145/3140659.3080254
27. K. Abdelouahab et al., "**Accelerating CNN inference on FPGAs : A Survey,**" arXiv preprint arXiv, 2018.
28. L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "**An Efficient Hardware Accelerator for Sparse Convolutional Neural Networks on FPGAs**," IEEE 27th Annu. Int. Symp. Field-Programmable Cust. Comput. Mach., pp. 17–25, 2019.
29. Song Han, Jeff Pool, John Tran, and William J. Dally, "**Learning bothWeights and Connections for Efficient Neural Networks**," Advances in neural information processing systems, pp. 1135–1145, 2015.
30. E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides, "**LUTNet : Rethinking Inference in FPGA Soft Logic**," 2019 IEEE 27th Annu. Int. Symp. Field-Programmable Cust. Comput. Mach., pp. 26–34, 2019, doi: 10.1109/FCCM.2019.00014.
31. M. H. Zhu, "**To prune , or not to prune : exploring the efficacy of pruning for model compression**," arXiv : 1710 . 01878, vol.2, Nov 2017.

32. S. Anwar, K. Hwang, and W. Sung, "**Structured Pruning of Deep Convolutional Neural Networks**." ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 13, no. 3, pp. 1-18, 2017. https://doi.org/10.1145/3005348
33. S. Anwar, W. Sung, and C. Science, "**COMPACT DEEP CONVOLUTIONAL N EURAL NETWORK**," vol. 1, no. 2015, pp. 1–10, 2017.
34. Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz, "**PRUNING CONVOLUTIONAL NEURAL NETWORKS FOR RESOURCE EFFICIENT INFERENCE**," arXiv preprint arXiv, 2016.
35. Yann Le Cun, John S. Denker, and Sara A. Solla, "**Optimal Brain Damage**," Advances in neural information processing systems, 1990.
36. B. Hassibi, D. G. Stork, G. J. Ivolff, S. Hill, and R. Suite, "**Optimal Brain Surgeon and General Xetlwork Pruning**." IEEE international conference on neural networks,1993.
37. C. Ding et al., "**CirCNN : Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices**." Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. 2017.
38. S. Li, W. Wen, and Y. Wang, "**An FPGA Design Framework for CNN Sparsification and Acceleration**," In 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 28-28, 2017.
39. H. Kang, "**Real-Time Object Detection on 640x480 Image With VGG16 + SSD**," In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pp. 419–422, 2019. https://doi.org/10.1109/ICFPT47387.2019.00082
40. R. J. Struharik, B. Z. Vukobratović, A. M. Erdeljan, and D. M. Rakanović, "**CoNNa–Hardware accelerator for compressed convolutional neural networks**," Microprocessors and Microsystems, vol. 73, p. 102991, 2020.
41. F. Sun et al., "**A High-Performance Accelerator for Large-Scale Convolutional Neural Networks**," In 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), pp. 622-629,2017.
42. Y. Li and Y. Du, "**A Novel Software-Defined Convolutional Neural Networks Accelerator**," IEEE Access, vol. 7, pp. 177922–177931, 2019.
43. W. Chen, J. W. W. Edu, C. Cse, and W. Edu, "**Compressing Neural Networks with the Hashing Trick"** In International conference on machine learning, pp. 2285-2294, 2015**.**
44. Song Han, Huizi Mao, and William J. Dally, "**Deep Compression: Compressing Deep Neural Networks With Pruning, Trained Quantization And Huffman Coding**," arXiv preprint arXiv, 2015.
45. J. Garland and D. Gregg, "**Low Complexity Multiply-Accumulate Units for Convolutional Neural Networks with Weight-Sharing**," ACM Transactions on Architecture and Code Optimization (TACO), vol. 15, no. 3, pp. 1–24, 2018. https://doi.org/10.1145/3233300
46. H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. I. E. Han, "**A Review , Classification , and Comparative Evaluation of Approximate arithmetic circuits**," ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 13, no. 4, pp. 1–34, 2017.
47. H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, "**A Comparative Evaluation of Approximate Multipliers**,"In 2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), pp. 191–196, 2016.
48. J. N. Mitchell and J. R. Associate, "**Computer Multiplication and Division Using Binary Logarithms** *," IRE Transactions on Electronic Computers,pp. 512–517, 1962. https://doi.org/10.1109/TEC.1962.5219391
49. A. Avramovic and P. Bulic, "**An iterative logarithmic multiplier**," Microprocessors and Microsystems vol. 35, pp. 23–33, 2011.
50. W. Liu and D. Wang, "**Design of Approximate Logarithmic Multipliers**," In Proceedings of the on Great Lakes Symposium on VLSI 2017, pp. 47–52,2017. https://doi.org/10.1145/3060403.3060409
51. S. Member, M. S. Kim, A. A. Del Barrio, and L. T. Oliveira, "**Efficient Mitchell ' s Approximate Log Multipliers for Convolutional Neural Networks**," IEEE Transactions on Computers, vol. 68, no. 5, pp. 660-675, 2018
52. M. S. Kim, A. A. D. Barrio, R. Hermida, and N. Bagherzadeh, "**Low-power implementation of Mitchells approximate logarithmic multiplication for convolutional neural networks**," 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), 2018.
53. J. Faraone et al., "**AddNet : Deep Neural Networks Using FPGA-Optimized Multipliers**," IEEE Trans. Very Large Scale Integr. Syst., vol. PP, pp. 1–14, 2019.
54. Z. Wang, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "**Approximate Multiply-Accumulate Array for Convolutional Neural Networks on FPGA**," 2019 14th Int. Symp. Reconfigurable Commun. Syst., pp. 35–42, 2019.
55. C. Guo, L. Zhang, X. Zhou, W. Qian, and C. Zhuo, "**A Reconfigurable Approximate Multiplier for Quantized CNN Applications**," In 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 235-240, 2020
56. F. Berry, I. Pascal, and U. M. R. Cnrs, "**The Challenge of Multi-Operand Adders in CNNs on FPGAs**," In Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, pp. 157-160. 2018.

57. L. Luo, Z. Chen, X. Yang, F. Qiao, Q. Wei, and H. Yang, "**A single clock cycle approximate adder with hybrid prediction and error compensation methods**," Microelectronics J., vol. 87, no. September 2018, pp. 45–50, 2019. https://doi.org/10.1016/j.mejo.2019.03.007
58. T. Kowsalya, "**Area and Power Efficient Pipelined Hybrid Merged Adders for Customized Deep Learning Framework for FPGA Implementation**," Microprocess. Microsyst., p. 102906, 2019.
59. M. S. Ansari, S. Member, V. Mrazek, B. F. Cockburn, L. Sekanina, and S. Member, "**Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers**," IEEE Trans. Very Large Scale Integr. Syst., vol. PP, pp. 1–12, 2019.
60. A. Shafiee, A. Nag, N. Muralimanohar, and R. Balasubramonian, "**ISAAC : A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars**," ACM SIGARCH Computer Architecture News, vol.44, no. 3, pp. 14–26, 2016.
61. A. P. Chowdhury, P. Kulkarni, and V. G. G. V. Alex, "**MB-CNN : Memristive Binary Convolutional Neural Networks for Embedded Mobile Devices**," Journal of Low Power Electronics and Applications, pp. 1–27, 2018. https://doi.org/10.3390/jlpea8040038
62. F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "**RANA : Towards Efficient Neural Acceleration with Refresh-Optimized Embedded DRAM**," 2018 ACM/IEEE 45th Annu. Int. Symp. Comput. Archit., pp. 340–352, 2018.
63. M. Imani, M. Samragh, Y. Kim, and S. Gupta, "**RAPIDNN : In-Memory Deep Neural Network Acceleration Framework**."arxiv preprint arxiv, 2018.
64. P. Chi, S. Li, and C. Xu, "**PRIME : A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory**," ACM SIGARCH Computer Architecture News , vol. 44, no. 3, pp. 27-39 2016.
65. D. Fan and S. Angizi, "**Energy Efficient In-Memory Binary Deep Neural Network Accelerator with Dual-Mode SOT-MRAM**," In 2017 IEEE International Conference on Computer Design (ICCD), pp. 609-612, 2017.
66. Y. Pan, P. Ouyang, and A. M. Cell, "**A MultilevelMultilevel Cell STT-MRAM-Based Computing In-Memory Accelerator for Binary Convolutional Neural Network**," IEEE Trans. Magn., vol. PP, pp. 1–5, 2018
67. Baohua Sun, Daniel Liu, and Leo Yu, "**MRAM Co-designed Processing-in-Memory CNN Accelerator for Mobile and IoT Applications**," arXiv preprint arXiv, 2018.
68. S. Angizi, Z. He, F. Parveen, D. Fan, and C. Florida, "**IMCE : Energy-Efficient Bit-Wise In-Memory Convolution Engine for Deep Neural Network**." In 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 111-116, 2018..
69. S. Angizi, Z. He, A. S. Rakin, and D. Fan, "**CMP-PIM : An Energy-Efficient Comparator-based Processing-In-Memory Neural Network Accelerator**," Proceedings of the 55th Annual Design Automation Conference, pp. 1-6, 2018
70. K. Ikegami et al., "**Binary and ternary convolutional neural network acceleration by in-nonvolatile memory computing with Voltage Control Spintronics Memory (VoCSM)**," 2019 Electron Devices Technol. Manuf. Conf., pp. 225–227, 2019.
71. A. Roohi, S. Angizi, D. Fan, and R. F. Demara, "**Processing-In-Memory Acceleration of Convolutional Neural Networks for Energy-Efficiency, and Power-Intermittency Resilience**," 20th Int. Symp. Qual. Electron. Des., pp. 8–13, 2019.
72. V. Joshi et al., "**Accurate deep neural network inference using computational phase-change memory**," Nat. Commun., no. 2020, pp. 1–13, 2020.
73. l. Lai et al., "**Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations**," arxiv preprint arxiv, 2017.
74. T. J. Webb et al., "**Flexpoint : An Adaptive Numerical Format for Efficient Training of Deep Neural Networks**," In Advances in neural information processing systems, pp. 1742-1752. 2017.
75. X. Lian, Z. Liu, Z. Song, and J. Dai, "**High-Performance FPGA-Based CNN Accelerator With Block-Floating-Point Arithmetic**," IEEE Trans. Very Large Scale Integr. Syst., vol. 27, no. 8, pp. 1874–1885, 2019.
76. D. Hammerstrom, "**A VLSI architecture for high-performance, low-cost, on-chip learning**," 1990 IJCNN International Joint Conference on Neural Networks, 1990.
77. S. Gupta, A. Agrawal, K. Gopalakrishnan, Y. Heights, P. Narayanan, and S. Jose, "**Deep Learning with Limited Numerical Precision**." In International Conference on Machine Learning, pp. 1737-1746. 2015.
78. S. Wijeratne, S. Jayaweera, M. Dananjaya, and A. Pasqual, "**Reconfigurable Co-Processor Architecture with Limited Numerical Precision to Accelerate Deep Convolutional Neural Networks**," 2018 IEEE 29th Int. Conf. Appl. Syst. Archit. Process., pp. 1–7, 2018. https://doi.org/10.1109/ASAP.2018.8445087
79. H. O. Ahmed, M. Ghoneima, and M. Dessouky, "**Concurrent MAC Unit Design using VHDL for Deep Learning Networks on FPGA**," 2018 IEEE Symp. Comput. Appl. Ind. Electron., pp. 31–36, 2018.
80. M. Gallus and A. Nannarelli, "**Handwritten Digit Classification using 8-bit Floating Point based Convolutional Neural Networks**," 2018.
81. S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "**Understanding the Impact of Precision Quantization on the Accuracy and Energy of Neural**

Networks." In Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 1474-1479, 2017.

82. R. Ding, G. Su, G. Bai, W. Xu, N. Su, and X. Wu, "**A FPGA-based Accelerator of Convolutional Neural Network for Face Feature Extraction**," 2019 IEEE Int. Conf. Electron Devices Solid-State Circuits, pp. 1–3, 2019.
83. S. Vogel and A. Guntoro, "**Efficient Hardware Acceleration of CNNs using Logarithmic Data Representation with Arbitrary log-base**.", In Proceedings of the International Conference on Computer-Aided Design, pp. 1-8. 2018.
84. Y. Zhao et al., "**Automatic Generation of Multi-precision Multi-arithmetic CNN Accelerators for FPGAs**," pp. 45–53, 2019.
85. D. Piyasena, R. Wickramasinghe, D. Paul, S. Lam, and M. Wu, "**Reducing Dynamic Power in Streaming CNN Hardware Accelerators by Exploiting Computational Redundancies**.",I n 2019 29th International Conference on Field Programmable Logic and Applications (FPL), pp. 354-359. IEEE, 2019.
86. C. Fung, B. Fong, J. Mu, and W. Zhang, "**A Cost-Effective CNN Accelerator Design with Configurable PU on FPGA**," 2019 IEEE Comput. Soc. Annu. Symp. VLSI, pp. 31–36, 2019
87. S. Fpgas, J. Lin, A. Lotfi, V. Akhlaghi, Z. Tu, and R. K. Gupta, "**Accelerating Local Binary Pattern Networks with Refresh-Optimized Embedded DRAM**," 2019 Des. Autom. Test Eur. Conf. Exhib., pp. 1112–1117, 2019.
88. S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "**FP-BNN : Binarized neural network on FPGA**," Neurocomputing, vol. 0, pp. 1–15, 2017.
89. M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "**XNOR-Net : ImageNet Classification Using Binary Convolutional Neural Network**," In European conference on computer vision, pp. 525-542, 2016. https://doi.org/10.1007/978-3-319-46493-0_32
90. Y. Zhou and A. C. N. N. Basics, "**An FPGA-based Accelerator Implementation for Deep Convolutional Neural Networks**," no. Iccsnt, pp. 829–832, 2015.
91. Z. Li et al., "**Laius : An 8-bit Fixed-point CNN Hardware Inference Engine**," In 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), pp. 143-150, 2017.
92. V. Luna and F. Ariza, "**El siguiente artículo ha sido aceptado para ser publicado en el,**" vol . 30 , no . 1 de la revista Ciencia e Ingenieria Neogranadina . Esta versión es preliminar y puede contener algunos errores .," vol. 30, no. 1, 2019.
93. J. Qiu et al., "**Going Deeper with Embedded FPGA Platform for Convolutional Neural Network**," In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays pp. 26–35, 2016.
94. Y. Umuroglu, N. J. Fraser, G. Gambardella, and M. Blott, "**FINN : A Framework for Fast , Scalable Binarized Neural Network Inference**," In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 65-74, 2017.
95. R. Zhao, W. Song, W. Zhang, T. Xing, and J. Lin, "**Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs**," In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 15–24, 2017.
96. P. Guo et al., "**FBNA : A Fully Binarized Neural Network Accelerator**," In 2018 28th International Conference on Field Programmable Logic and Applications (FPL), pp. 51-513, 2018.
97. T. Guan, P. Liu, X. Zeng, M. Kim, and M. Seok, "**Recursive Binary Neural Network Training model for Efficient Usage of On-Chip Memory**," IEEE Transactions on Circuits and Systems I: Regular Papers, pp. 1–13, 2019.
98. A. Ren, J. Li, Z. Li, and C. Ding, "**SC-DCNN : Highly-Scalable Deep Convolutional Neural Network using Stochastic Computing**." ACM SIGPLAN Notices, pp.405-418, 2017.
99. Z. Li et al., "**Structural Design Optimization for Deep Convolutional Neural Networks using Stochastic Computing**," In Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 250-25, 2017
100. H. Abdellatef, M. Khalil-hani, N. Shaikh-husin, and S. O. Ayat, "**Stochastic Computing Correlation Utilization in Convolutional Neural Network Basic Functions**," *Telkomnika*, vol. 16, no. 6, 2018. https://doi.org/10.12928/telkomnika.v16i6.8955
101. X. Ma et al., "**An Area and Energy Efficient Design of Domain-Wall Memory-Based Deep Convolutional Neural Networks using Stochastic Computing**," In 2018 19th International Symposium on Quality Electronic Design (ISQED), pp. 314-321, 2018.
102. J. Draper, B. Yuan, J. Tang, and Q. Qiu, "**HEIF : Highly Efficient Stochastic Computing based Inference framework for Deep Neural Networks**," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 8, pp.: 1543-1556, 2018.
103. C. Lammie, W. Xiang, and M. R. Azghadi, "**Accelerating Deterministic and Stochastic Binarized Neural Networks on FPGAs Using OpenC**L," In 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 626-629,2019.
104. A. Khadem, "**Design Challenges of Neural Network Acceleration Using Stochastic Computing**," arXiv preprint arXiv ,pp. 1–14, 2020.