NAIST-IS-DD

Doctoral Dissertation

Primary Visual Cortex Inspired Feature Extraction Hardware Model and Applications

Tran Thi Diem

November 18, 2021

Graduate School of Science and Technology
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Tran Thi Diem

Thesis Committee:

Professor Yasuhiko Nakashima	(Supervisor)
Professor Yuichi Hayashi	(Co-supervisor)
Associate Professor Renyuan Zhang	(Co-supervisor)
Visitor Associate Professor Tran Thi Hong	(Co-supervisor)

Primary Visual Cortex Inspired Feature Extraction Hardware Model and Applications*

Tran Thi Diem

Abstract

Convolutional neural networks (CNNs) have dominated various applications, from advanced manufacturing to autonomous cars. The layers of CNNs are placed in a hierarchy to solve complications on image processing or speech recognition applications. However, the main challenges in using CNNs are latency and memory access due to tens to hundreds of megabyte parameters and operations, which require data movement between on-chip and off-chip to support the computation. Besides, with edge applications such as smart sensors, wearable, and autonomous devices, security and latency are essential considerations. There is a gap between the designers who try comprehensive CNNs with better efficiency and the hardware architects who simplify them. Many researchers have attempted to speed up the CNN performance by using graphical processing units (GPU); yet, the power consumption on GPU remains a critical issue. Moreover, the computation is subject to rigorous area and power constraints in the inference stage due to limited resources. For energy cost-efficiency, developing low-power hardware for CNNs is a research trend. In the third generation, the Spiking Neural Networks (SNNs) with biological plausibility and similarity to the functionality of the human brain are emerging. A more comprehensive study is expected to understand the inherent behavior of SNNs, especially under adversarial attacks. My research focuses on the following problems to address these challenges:

1. A primary visual cortex inspired feature extraction hardware model is created. To combine the edge and SLIT functions, the model can reduce the

*Doctoral Dissertation, Graduate School of Science and Technology, Nara Institute of Science and Technology, NAIST-IS-DD, November 18, 2021.

training time in deep neural networks. Training time is diminished by 40%, 40%, and 32%, respectively, with MNIST, CIFAR, and SVHN databases on Lenet-5 and CNN models. It also decreases by about 10% on larger paradigms such as VGG-16 and VGG-19 with the CIFAR database. Notably, the SLIT architecture efficiently merges with most popular CNNs at a slightly sacrificing accuracy of a factor of 0.27% on MNIST, ranging from 0.5% to 1.5% on CIFAR, approximately 2.2% on ImageNet, and remaining the same on SVHN databases.

2. An optimization hardware model for the inference phase is showed extremely efficiently when applying the SLIT function. Latency, power, and hardware resources of the inference step are evaluated on the chip ZC7Z020-1CLG484C FPGA with Lenet-5 and VGG schemes. On the Lenet-5 architecture, the results are reduced by 39% of latency and 70% of hardware resources with a 0.456 W power consumption compared to previous works. It is also decreased approximately 10% on hardware resources and latency with the VGG models. An advance in latency is also proved in this research, with an enhancement in the range of 2.6% to 16% when being compared with the traditional approach.

3. An efficient success in adversarial attack applications when applying SLIT function into deep spiking neural networks. In against adversarial attack for deep spiking neural networks through white-box settings with different noise budgets and variable spiking parameters, the proposal also improves the accuracy of the results when increasing noise budget. With white-box adversarial attack applications on SNNs, the accuracy of the proposal is approximately 70% higher robustness than the previous works.

Keywords:

primary visual cortex, image classification, convolutional neural network, FPGA, feature extraction, spiking neural network, vitis AI, adversarial attack

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

The human visual resolution is about 20K×20K neurons, and the primary visual cortex (V1) comprises 140 million neurons [1]. V1 has special functions such as detect object angles in a steady increase every 10°, left and right parallax, movement direction, and approach. The 1K×1K central area is more sensitive than the periphery, and 25 neurons corresponding to a 5x5 pixel block in the image are arranged in 18 separate directions for each left and right hemisphere [2]. Following the general theory, this research assumes that there are the same amount of neurons in 36 motion directions, and it is estimated that about half ($(18 \times 2 + 36) \times 1\text{K} \times 1\text{K} = 72$ million) of the neurons will be placed in the middle. The average ratio of the number of neurons to the number of pixels is 2.8:1 (36 neurons/25 pixels). There are $25 \times (36 + 36) = 1800$ neurons in the center, and (140 million - 72 million)/(20K x 20K - 1K ×1K)× $25 = 4260$ neurons in the periphery. Therefore, it is reasonable to think that the detailed structure above is based on the blueprint, not acquired by learning.

Many prototypes based on biological [3], statistics [4], or physical principles [5] are presented as the primary visual cortex model. Due to the complexity of fashioning the visual cortex, only specific functions of the visual system are usually considered. A feature extraction model is hypothesized to be made through a sequence of feed-forward and feed-backward loops. This process is often represented as an imitation of the receptor fields of neurons in the layer. A regularly feed-forward processing hierarchy of visual information is convolutional neural networks [6]. The first layers extract features from inputs, and the last layers perform classification.

The main challenges in using CNNs are latency and memory access [7, 8] due to tens to hundreds of megabyte parameters and operations, which require data movement between on-chip and off-chip to support the computation. Security and latency are important considerations in edge applications such as smart sensors, wearable, and autonomous devices [9, 10]. We have recently surveyed the performance of state-of-the-art CNNs in terms of accuracy, size, and potentiality

of various hardware platforms. The results reveal a gap between the designers who strike for comprehensive CNNs with better efficiency and the hardware architects who try to simplify them [11, 12]. Many researchers have attempted to speed up the CNN performance using graphical processing units (GPU) [13, 14]. Moreover, the computation is subject to rigorous area and power constraints in the inference stage due to limited resources. Therefore, many data scientists are focusing on increasing inference performance by designing various accelerators.

Field Programmable Gate Arrays (FPGAs) have become the best candidate for trade-off cost, flexibility, and performance in deep learning processor designs [15]. FPGAs are suitable for computationally intensive algorithms that result in a faster speed and efficient energy. A few highlights of these approaches include binary weight quantization, parameter reduction, memory bandwidth optimization, and data-flow optimization [16, 17, 18, 19]. It is essential to create a highly flexible architecture that can mold itself into the given CNNs and achieve a higher resource utilization reduction. Moreover, due to the largest input size, the first few layers that typically contribute to the most significant latency on CNN leave plenty of room for improvement.

Because of their biological plausibility and comparison to the human brain functionality, Spiking Neural Networks (SNNs) have appealed to many researchers [20, 21, 22, 23, 24]. SNNs consume lower energy when executed on neuromorphic hardware than other network topologies. The asynchronous interaction between neurons and the event-based propagation of the information through layers can achieve high energy efficiency. These features strengthen attention on neuromorphic structures, for example, IBM TrueNorth [25] and Intel Loihi [26]. Besides investment in enhancing accuracy, a recent security perspective also considers SNNs compared to conventional deep neural networks (DNNs).

1.2 Research Contribution

This dissertation aims to create the feature extraction hardware model inspired primary visual cortex principle and apply it in the current deep neural network model to improve performance. This thesis not only provides the algorithms but also describes evaluation both on software and hardware architecture. In summary, the main contributions of this dissertation are:

- The first proposal presents the primary cortex hardware structure, which includes five functions: edge detection, SLIT detection, parallax detection, moving XY detection, and approach detection. These algorithms are efficient for hardware resources when comparing with current algorithms in state-of-the-art.

- The first layer, which contributes most of the training time and latency on current CNNs, is replaced with the SLIT function. The new scheme for the deep neural network has an efficient performance. With MNIST, CIFAR, and SVHN databases on Lenet-5 and CNN models, training time is diminished by 40%, 40%, and 32%, respectively. It also decreases by approximately 10% on larger paradigms such as VGG-16 and VGG-19 with the CIFAR database. Notably, the SLIT architecture efficiently merges with most popular CNNs at a slightly sacrificing accuracy of a factor of 0.27% on MNIST, ranging from 0.5% to 1.5% on CIFAR, approximately 2.2% on ImageNet, and remaining the same on SVHN databases.

- The hardware circuit optimization for the inference step on the Lenet-5 scheme is proposed. The architecture reduces 39% of latency and 70% of hardware resources with a 0.456 W power consumption compared to previous works. The proposal has the same accuracy with higher throughput on most deep neural networks when implementing on the DPU platform.

- An efficient success in adversarial attack applications when applying SLIT layer on spiking neural networks that investigate the effect of structural parameters such as membrane threshold and time window. The input extracted from our model increases the security of SNN under adversarial attack problems. In against adversarial attack for deep spiking neural networks through inherent structural parameter method, the proposal also improves the accuracy of the results when increasing budget noise.

1.3 Dissertation Layout

The thesis is divided into six chapters which are organized as follows:

- Chapter 1 introduces the overview, contributions, and layout of this research.
- Chapter 2 gives an overview of the deep neural network. Then, the preliminary of convolution neural networks is summarized. Finally, the related works are presented.
- Chapter 3 presents the details of each function in our primary visual cortex hardware. In this context, the simple algorithm to integrate hardware design is described. All of the algorithms include just simple circuits such as AND, OR, COMPARISON, SHIFT. The optimized circuits, when implementing on the hardware platform, are showed in this chapter. The first three layers are currently optimized for the network like the Lenet-5 scheme, and the first two layers are reconfigured on the systems like VGG-16
- Chapter 4 shows the details of the results on the software platform. In this chapter, the experiments are conducted on Tensorflow and Keras to the compare accuracy and the training time.
- Chapter 5 analyses the extracted hardware resources and latency with the Vivado_HLS. This chapter also manifests the IP core of Lenet-5, which is embedded into the SoC. The DPU architecture based on the Vitis AI platform that conducts the inference on the ZCU_102 board to compare with state-of-the-art is also presented.
- Chapter 6 give a proposal on adversarial attack application for spiking neural network. How to improve the accuracy of SNN under the adversarial attack phenomena is clearly presented in this chapter
- The last chapter of this thesis concludes and emphasizes the main contributions to my work. Then, some ideas for future works are addressed.

2 Background and Related Work

This chapter introduces the basic concepts relating to the arguments discussed in the thesis. First, the meaning of Artificial Intelligence (AI) and related topics are presented to understand the evolution of neural networks (NNs). Next, I summarize the content of CNNs. Finally, the related works that optimize the architecture of deep neural networks on software and hardware are reviewed.

2.1 Machine Learning

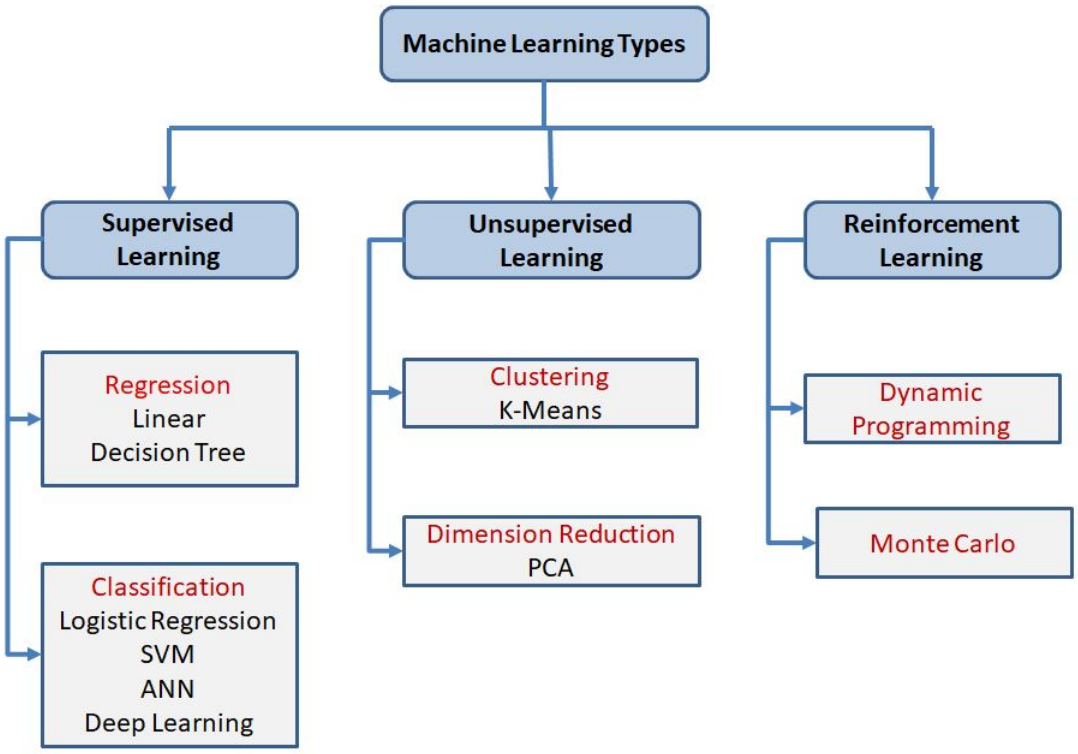


Figure 1. Machine learning category

Artificial intelligence (AI) is the psychology of imagining intelligent machines to accomplish specific goals and tasks as humans do is artificial intelligence (AI). It is a vast topic, going from video games to autonomous driving and including every

application in which a machine can learn or predict something. AI indicates the ability of a machine to learn information (training) and solve problems without being explicitly programmed every time. We temporal categorize the AI into these subgroups, as shown in Fig. 1.

2.1.1 Supervised learning

Supervised learning is a method for predicting a label of a previously unseen instance from the previous information about input and the target output [27, 28, 29, 30]. It can be seen as a machine learning task of inferring a function from training data to correctly map a class for invisible cases. Each training sample (consisting of an input vector X and its corresponding target output vector Y) may feed into the network several times so that the actual output can approach the target output. An error value is calculated from each given sample as a function of the difference between the target outputs vector, Y, and the actual output vector, Z (for example, min square error or entropy error). In neural networks, this error is utilized to update connection weights in the network. That network can generate a result closer to or exactly the desired output next time if a similar input pattern appears. The two most common approaches to minimize this error in deep neural networks are the gradient descent rule and the learning windows rule [31, 32]. First, to reduce error, a gradient descent-based learning algorithm finds a local minimum of linear systems. The second type changes the synaptic weights as a function of the relative timing of pre-and post-synaptic action potentials.

2.1.2 Unsupervised learning

Unsupervised learning is a technique for searching data to find some natural structures in the input under an unknown probability distribution. The convergence examination of unsupervised learning is much more difficult than other learning as the input datasets are unlabelled. For traditional artificial neural networks, an n-dimensional input is processed by the same number of computing units or by minimizing a cost function for feature extraction, dimension reduction, clustering, etc. Self-organizing map (SOM), adaptive resonance theory (ART), independent component analysis (ICA), Hebbian learning, principal component analysis

(PCA) [33, 34], and BCM rule are generally employed unsupervised learning algorithms. In spiking neural networks, Hebbian learning, BCM rule, and STDP rules are famously used as unsupervised learning methods in real-world applications. STDP learning is an asymmetric form of Hebbian learning in tightening temporal correlations between weakening and strengthening connections. Since unsupervised learning takes into account competition and sidelong interference, the weights of the winner neurons are increased. In contrast, other neurons sustain a small weight reduction. Furthermore, the STDP learning rule regards the lateral inhibition between pairs of spikes: a pre-post pairing causes potentiation, and a post-pre pairing causes depression. The most recent presynaptic and postsynaptic spike pair is adapted to detect the correlations of the next attempting fire.

2.1.3 Reinforcement Learning

Reinforcement learning is a control optimization technique that is used to recognize the best action in every state visited by the system [35, 36, 37, 38]. In reinforcement learning, a general error signal back ("reward") is determined in every state that describes how well the system performs. The typical framing of reinforcement learning is the following scenario. An agent takes action in an environment. Based on the action, the agent changes state, and the learning algorithm also receives a reward signal a short time later. The current state and reward are both then fed back into the agent. The algorithm modifies its strategy to achieve the highest reward.

2.2 Artificial Neural Networks

Humans aim to create machines that work like the brain, so it is reasonable to talk about brain-inspired computation. The artificial structures that model the real biological neural networks are called artificial neural networks (ANNs) [39] or just neural networks (NNs). They are computational models composed of many layers of artificial neurons, which is the main computational unit of the brain. The structure of ANN is shown in Fig. 2. The neurons are connected in a NN, so the output of one neuron, called the axon, represents one of the inputs of another

one, called dendrites. The outcome of a neuron corresponds to the weighted sum of the inputs. A synapse is a connection between an axon and a dendrite. This link between the output and input of two neurons is called presynaptic and postsynaptic neurons, respectively. Scales the axon output signal by a quantity called weight. A neural network learns information by updating the values of the weights in response to input stimuli. This process is called learning or training. Once the NN is trained using the training data, the values of the weights are determined. Performances of the NN are evaluated on the test data. Hence considering a complete dataset, we can recognize the training data used for the training process and the test data used for the inference process.

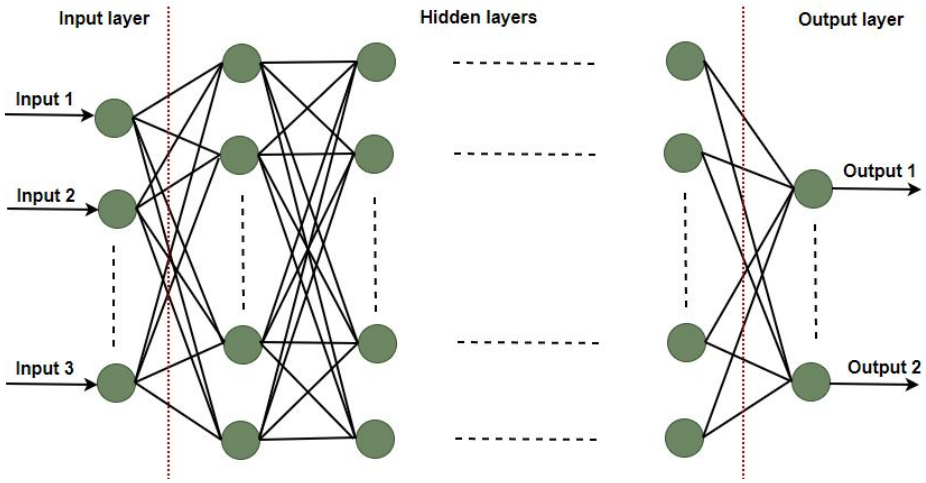


Figure 2. General ANN architecture

2.3 Preliminary Convolutional Neural Networks

Through development over 20 years, the network initially inspired by neuroscience has attracted spacious attention in image processing and computer science [40, 41, 42]. Today, some object recognition systems based on CNN can recognize objects with super-human accuracy. As we can observe in Fig. 3, a convolution neural network (CNN) is a subset of NN with more than three layers. In general, the first layer is called the input layer, while the last one output layer. The layers between these two ones are called hidden layers. The network is deeper, increasing the number of hidden layers. CNN perceives an object using the feature

extraction step and the classification phase. The feature extraction step included the convolutional and sub-sampling layers to find variances of an input image such as lines and edges. Combining the fully connected (FC) layers, the classification phase decides the most likely class object based on the extracted features. CNN can achieve a highly accurate classification performance using the convolutional (CONV), sub-sampling, and FC layers.

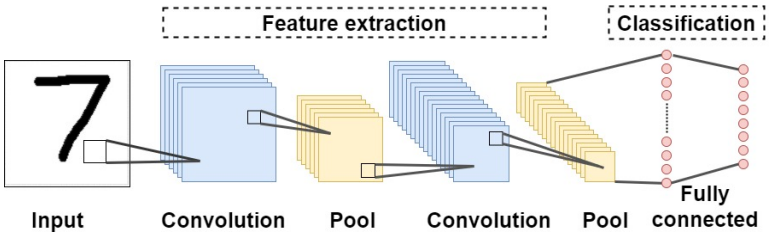


Figure 3. General CNN architecture

The CONV layer receives features as input and executes convolution operation with a filter kernel window to generate one pixel in one output feature map. The output channels are filtered through an activation function such as Relu, Sigmoid, and Tanh. Total output feature maps form a set of the input channels for the next CONV layer. Summary of the process which calculates one output channel is formulated in Eq. 1.

$$O_j^k = f(\Sigma_{i \in M} I_i^{k-1} * W_{ij}^k + b_j^k) \tag{1}$$

where O_j^k is the current output of the j^{th} channel at k^{th} layer, I_i^{k-1} is the previous feature map of the i^{th} channel in M channels, W is the ij^{th} kernel filter, b_j^k is corresponding the bias of the j^{th} channel, and f is the activation function, the symbol "$*$" is the element-wise multiplication operation.

The sub-sampling layer or the pooling layer is generally sandwiched between the two CONV layers. The pooling layer reduces the size of feature maps from the previous layer. Besides, this layer is employed to avoid the over-fitting problem and redundancy in the channels. There are two main pooling methods: mean-pooling and max-pooling. The output of the max-pooling (MP) layer is determined as shown in Eq. 2.

$$u_{i,j}^m = \max_{0<=i,j \in P} u_{(i,P+i),(j,P+j)}^n \tag{2}$$

where u^m is the max output value in the kernel size P of the m^{th} channel, the u^n is input value in the kernel size P.

The FC layers that control object classification into various categories in CNNs are conjoined after multiple convolutional and sub-sampling layers. The term "fully connected" means that all neurons in the previous layer are connected to all neurons in the next layer. For example, the last layer of the Lenet-5 has ten possible outputs, and each output corresponds to a number from "0" to "9". A neuron output V_k^{out} in the FC layer is obtained by using Eq. 3. It is a typical matrix multiplication and addition with a bias.

$$V_k^{out} = \Sigma_{i=0}^{N} W_{ki} \times V_i^{in} + bias_k \tag{3}$$

where W_{ki} is weights corresponding with N input neurons at k^{th} position, V_i^{in} is the total neurons of the previous layer, and $bias_k$ is the bias of k^{th} output neuron.

2.4 Related Work

2.4.1 Researches on primary visual cortex

The purpose of the visual system is to predict neural responses to arbitrary stimuli, including those seen in nature. To achieve this goad, researchers create models based on one or more linear receptive fields. Basic models of neurons at the earliest stages of visual processing (retina, LGN, and V1 simple cells) typically include a single linear filter. On the other hand, models of neurons at later processing stages (V1 complex cells and beyond) require multiple filters [43].

The primary visual cortex model following the feedforward pathways begins with the description by Hubel and Wiesel (1962) [44]. Two functional classes of cortical cells: simple cells and complex cells, are famously described in the research of Hubel and Wiesel. The simple cells respond to oriented stimuli (e.g., bars, edges, gratings). The complex cells tuned to oriented stimuli tend to have larger receptive fields and exhibit the location of the motive within their receptive fields. A V1-like circuit that connects a complex cell to an array of simple cells is shown in Fig. 4. A simple cell in Fig. 4(a) (lower green cell) was gained by pooling over upper green cells aligned along a preferred axis of orientation. At the next stage, a complex cell in Fig. 4(b) can be obtained by selectively pooling over simple afferent cells with the same preferred orientation.

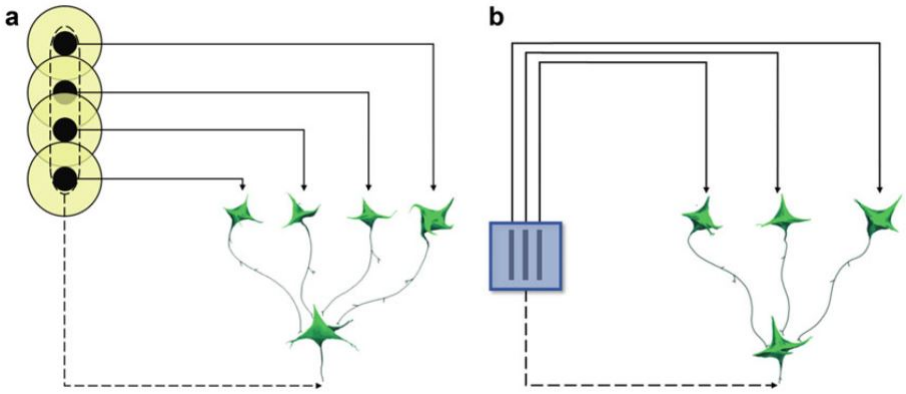


Figure 4. V1_like or Hubel and Wiesel model. Source: Fig.1 of the reference [44]

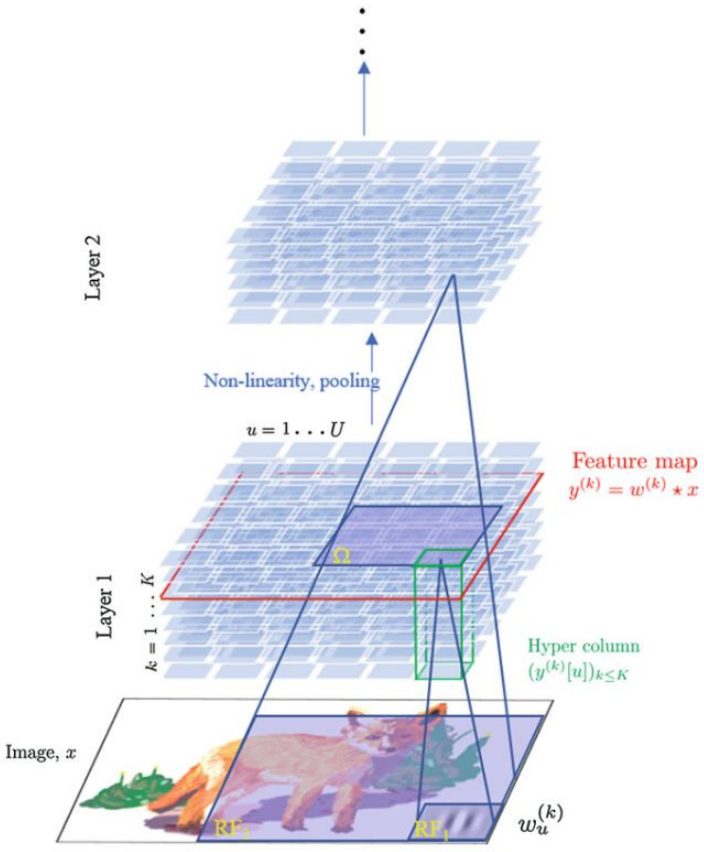


Figure 5. Models of the visual system based feed-forward wiring diagram. Source: Fig. 4 of reference [44]

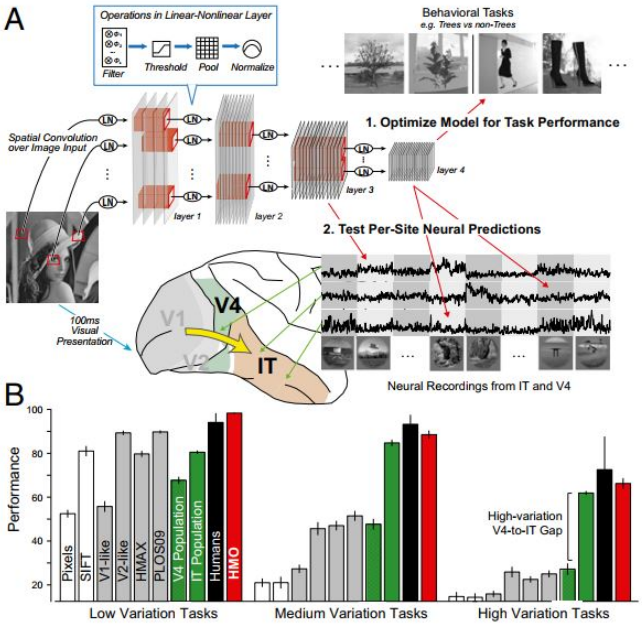


Figure 6. Neural-like model via performance on image classification task. Source Fig. 2 of reference [45]

Up-to-date feedforward models that determine the part of the operation in the visual cortex by introducing further, "deeper" processing stages, each with numerous learned filters. These models extend the Hubel and Wiesel's circuit from V1 to higher areas of the ventral stream. A general wiring diagram of a feedforward model of the visual system is shown in Fig. 5. Using the same classifier training protocol as with the neural data and control models, the hierarchical modular optimization (HMO) got better results on low variation tasks. However, for more variation tasks, HMO is not equal to the human object recognition ability like in Fig. 6. So, there is a need for more study to visualize exactly what happened in V1 and V4 regions [45].

2.4.2 A review feature extraction methods for image classification

Feature extraction creates new features by merging original bands. The new features store most of the important information. The feature extraction methods can be subdivided into four classes: knowledge-based such as non-parametric weighted feature extraction (2004), maximum noise fragment (1998), local binary

pattern (1990), statistical, for example, principal component analysis (2002), independent component analysis (2011), wavelet-based such as discrete wavelet transform (2002), and deep learning-based: convolutional neural network (2012), deep belief network (2016). Knowledge-based methods improve specific characteristics of the relevant bands to separate the objects or surface features of interest. The statistical feature extraction transforms the high-dimensional data into some lower-dimensional feature space reducing redundancy and enhancing class separability. The progress of these methods depends on their capacity to feature transformation without sacrificing the loss of information. The wavelet-based feature extraction relies on wavelet transform decomposing the signal into constituent wavelets of different scales and positions.

Table 4. Classwise and global classification accuracies obtained using major conventional spectral feature extraction techniques for Pavia University. The number of features is given within brackets.

Class	Raw	PCA (6)	MNF (5)	DWT (7)	ICA (15)	NWFE (40)	CNFE (11)	DBFE (27)
Class 1	0.8050	0.8394	0.8255	0.8134	0.8357	0.7593	0.7579	0.9155
Class 2	0.7185	0.6692	0.6673	0.6619	0.6525	0.5606	0.7691	0.8683
Class 3	0.7612	0.7254	0.7226	0.7374	0.6762	0.6986	0.7774	0.8144
Class 4	0.9627	0.9429	0.9287	0.9469	0.9504	0.8301	0.8619	0.9461
Class 5	0.9861	0.9776	0.9981	0.9907	0.9981	0.9870	0.9946	1.0000
Class 6	0.8263	0.8235	0.7887	0.7555	0.7927	0.5093	0.6721	0.9232
Class 7	0.8849	0.8751	0.8663	0.8784	0.8470	0.8394	0.9046	0.8950
Class 8	0.8428	0.8176	0.7746	0.8069	0.8156	0.7433	0.8937	0.8710
Class 9	1.0000	0.9917	0.9986	1.0000	0.9945	0.9807	0.9798	1.0000
Global κ	0.8041	0.7838	0.7717	0.7691	0.7702	0.6710	0.7269	0.8968
OA (%)	85.16	83.49	82.65	82.46	82.81	74.92	79.21	92.34

Table 5. Classwise and global classification accuracies obtained using major deep learning spectral feature extraction techniques for Pavia University. The number of features is given within brackets.

Class	SAE (60)	SSAE (60)	DBN (30)	CNN (9)
Class 1	0.9146	0.9188	0.9276	0.9346
Class 2	0.9524	0.9465	0.9432	0.9518
Class 3	0.8812	0.8937	0.9128	0.9124
Class 4	0.9627	0.9598	0.9584	0.9457
Class 5	0.9894	0.9914	0.9926	0.9842
Class 6	0.9063	0.9126	0.9077	0.9014
Class 7	0.9149	0.9189	0.9209	0.9165
Class 8	0.8928	0.8876	0.9143	0.9248
Class 9	0.9924	0.9925	0.9915	0.9965
Global κ	0.9236	0.9284	0.9316	0.9368
OA (%)	93.16	93.48	94.31	94.62

Figure 7. Classwise and global classification accuracies using various feature extraction techniques for Pavia University database. Source Table 4 and Table 5 of reference [46]

Comparison performance on image classification application based on Pavia university dataset show on Fig. 7. The results show that feature extraction can reduce dimensions without significantly compromising classification accuracy. Supervised techniques provide better accuracy than their unsupervised counterparts. In the vacancy of training data, unsupervised feature extraction can provide acceptable solutions. It is also observed from the results that spatial features produce complementary information that can help to improve classification accuracy. Recently emerged deep learning techniques have shown promising performance. Deep learning methods hierarchically learn features with the help of complex, layered architecture [46].

2.4.3 Approaches to accelerate CNN inference on FPGAs

In response to the hurdles of CNNs, many researchers have tried to optimize memory access or convolution operations. Recent works showed that sparsity optimization involving pruning and exploiting activation sparsity could reduce 89% of memory access and 67% of computation operations [47]. Activation sparsity, which can cut memory accesses and multiply-accumulate (MAC) operations by a half [48] based on rectified linear unit non-linearity to produce many zero outputs. The pruning and compression were also investigated with the Bayesian network. These reductions have nevertheless required hardware that was customized with the data movement and control. The reducing parameter approaches [49, 50] with a factor of 50× were studied due to the expense of many MAC operations. The low-rank approximation (LRA) that obtained a sparse convolution 2 - 4.5× faster than the corresponding value at the absence of sparsity with a 1% accuracy loss was reported by Denton et al. [51] . Due to a large number of hyper-parameters, the LRA has remained to be a big problem in training.

Another approach is bit-width optimization, which aims to decrease parameter bit-width from a floating point to a fixed point. This investigation reduced the precision to get higher efficiency in exchange for memory access and computation operations. Ternary weight network and BinaryConnect [52, 53] diminished the bit-width of weights to 2-bits or 1-bit. Some studies quantized the activation function of the neural network, which achieved a significant reduction in memory or computation cost [8, 54]. These proposals, nevertheless, were a trade-off for

a considerable accuracy loss. Researchers have also discovered the computation of CNNs in other domains to reduce complexity. Fast Fourier transform (FFT) applied a single filter in the CONV layer is an example of this attempt. As a result, the gain of compact network architecture is the loss of accuracy.

When accelerating a CNN to an FPGA device, the challenge is to find an efficient mapping model [55]. Current FPGA-based accelerators for CNNs rely on three main optimizations to efficiently infer CNNs. Algorithmic Optimizations for FPGA-Based CNN Acceleration accelerate the execution of convolutional and fully connected layers [56, 57, 58, 59, 60, 61, 62]. Computational transforms are employed on the feature maps and kernels to accelerate the execution of conv and FC layers. This method focuses on vectorizing the implementations and reducing the number of arithmetic operations occurring during inference. Various software libraries, such as OpenBlas CPUs and cuBLAS for GPUs, are a platform to deploy these computational transforms. Besides this, multiple implementations make use of such transforms to map CNNs on FPGAs.

The FPGA datapath optimizations for FPGA accelerators aim to solve the resource limitation of FPGA devices. [57, 63, 64, 65, 66, 67, 17, 68, 69, 70, 71]. Early FPGA-based accelerators for CNNs implemented systolic arrays to accelerate the 2D filtering in convolutions layers [72, 73, 74, 75, 76]. Finding the optimal PE configuration can be seen as a loop optimization problem [77, 78, 79, 18]. It is impossible to fully exploit all the parallelism patterns, especially with the sheer volume of operations involved in deep topologies. Applying dataflow process networks (DPNs), static data-flow (SDF) to accelerate CNN implementations on FPGAs is investigated in [80, 81, 82].

Approximate computing of CNN models the computational transforms, pruning, and quantization technologies are other approaches for accelerating FPGA CNNs [16, 83, 84, 85, 86, 16, 87, 88, 89, 54, 52, 90, 91, 92, 93, 94, 95, 96]. Several studies in [97, 98, 99] demonstrate that inference of CNNs can be achieved with reduced precision of operands. Also, works in [100, 101, 102] demonstrate fixed-point arithmetic applicability to train CNNs. As highlighted in [103], CNNs as over-parametrized networks and many weights can be removed or pruned without critically affecting the classification accuracy. In its simplest form, pruning is performed according to the magnitude, such as the lowest values of the weights

are truncated to zero [104]. The summary of optimization for FPGA accelerators is showed in Fig. 8.

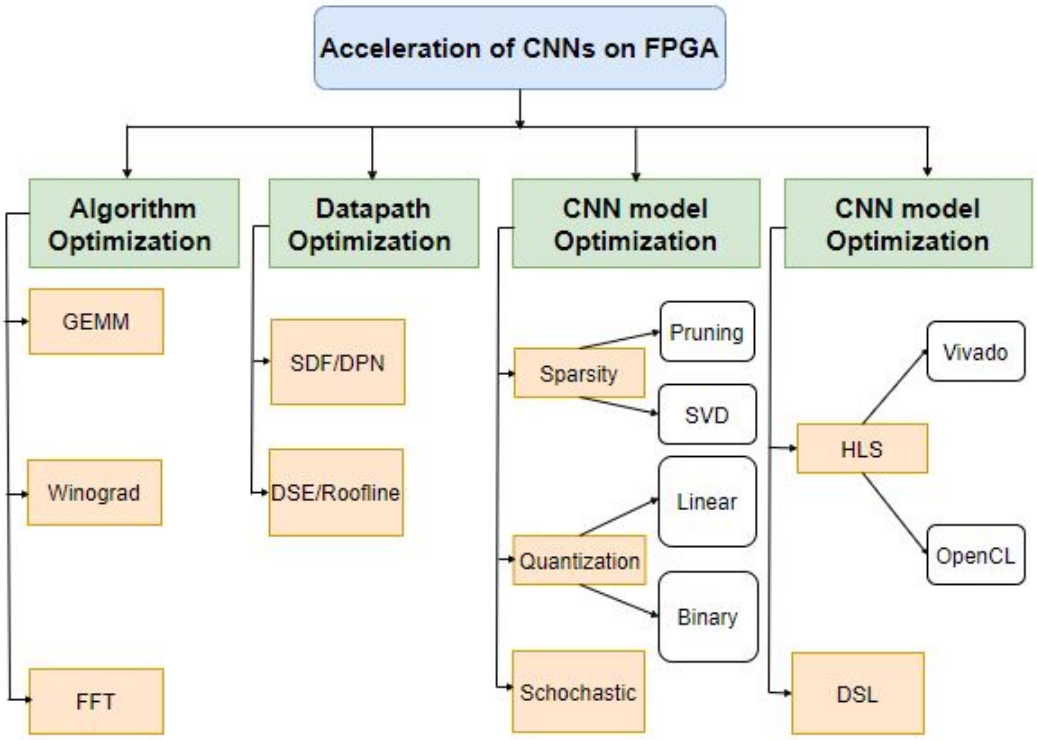


Figure 8. Approaches to accelerate CNN inference on FPGAs

For hardware generation, the using tools supporting circuit implementation on FPGA are investigated. These tools support estimating the new idea algorithm on hardware without deep expertise on FPGA/ASIC. The high-level synthesis (HLS) tools [105, 106, 107, 108] that emerge and use frequently on research is Vivado of Xilinx and OpenCl of Intel [109, 110]. Domain-specific languages (DSLs) [111, 112] are another approach.

3 Feature Extraction Primary Visual Cortex Hardware Model

The proposed prototype is a partial implementation of the standard model, which focuses on information-processing mechanisms in V1 [1, 113]. Generally, realizing the large-scale computations needed for the simulation is the biggest challenge in making a biologically accurate model. The complex calculations are simplified with uncomplicated hardware circuits. The results indicate that the proposal is able to keep the information necessary for the feature description

3.1 The Functions of the Primary Visual Cortex Hardware Model

3.1.1 Edge Detection

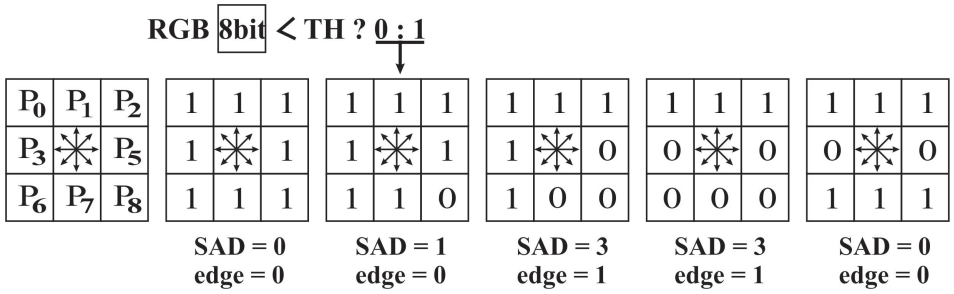


Figure 9. Edge detection

Edge detection plays a vital role in feature extraction. My proposal was formulated in Fig. 9 to achieve high efficient energy architecture. I recommended an edge detection algorithm that is performed by comparing the sum of absolute differences (SAD) of values applying Eq. 4 in four distinct directions with a threshold (TH) in a 3x3 area like in Eq. 5.

$$SAD = \begin{cases} 0 & if \quad P_i = P_{8-i} \\ 1 & if \quad P_i \neq P_{8-i} \end{cases} \tag{4}$$

where P_i is the value of a pixel in 3x3 window with i = (0, 1...8).

$$edge = \begin{cases} 0 & if \quad \Sigma_{i=0}^{3} SAD(P_i, P_{8-i}) < TH \\ 1 & otherwise \end{cases} \tag{5}$$

$$P_i = \begin{cases} 0 & if \quad RGB_{8bit} < TH1 \\ 1 & otherwise \end{cases} \tag{6}$$

For simplification toward hardware, the RGB 8-bit was diminished to 1-bit by considering with the other threshold (TH1) in Eq. 6. There was no edge if the sum of SAD was less than TH1; otherwise, the output was an edge. The four directions we suggested were 0°, 45°, 90°, and 135°. The hardware includes AND gate, OR gate, Comparison, and Shift circuits, which help enhance the effectiveness of the edge problem over other studies. This design can realize a parallel circuit with high speed and low resources on a field-programmable gate array (FPGA).

3.1.2 SLIT Detection

According to the principle of V1, most of the significant information is discovered in the central radius than at the periphery when observing an object. To replace the first layers, which was inspired by the primary visual cortex in the CNN model, a shift circuit in a range of 0° to 157.5° with a gradual increase every 22.5° for 4x4 window shown in Fig. 10 is proposed. The SLIT detection is executed at the base of the edge detection results.

$$ch_t = AND(\Sigma_{i=0}^{3} \Sigma_{j=0}^{3} \theta_{i,j}) \tag{7}$$

$$\theta = \frac{\triangle y}{\triangle x} \tag{8}$$

where θ is a gradually increase every 22.5°, $\theta_{i,j}$ is the E patterns shown in Fig. 9 at examining slope θ in 4x4 window. ch_t is the result of channel or S letters with t = (0, 1,...7)

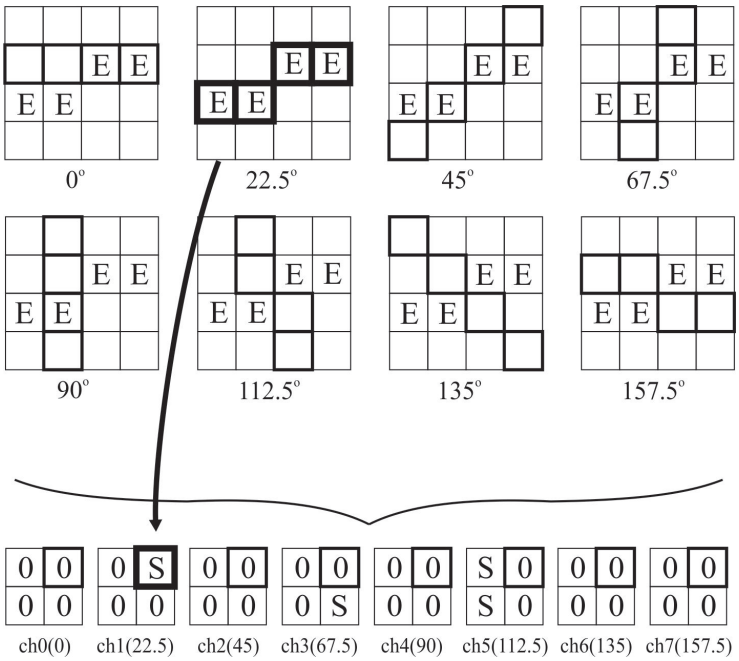


Figure 10. SLIT detection

The result of SLIT detection or ch0 to ch7 in Eq. 7 was an AND gate of 4 input values at the examining slope in Eq. 8. Each element was 1 bit. The number of elements in each channel was equivalent to that in the input image.

3.1.3 Left-Right Parallax Detection

Left-right parallax detection is the equivalent function as a stereo matching. Generally, stereo matching requires about 16x16 SAD operations in Eq. 9, which is not suitable for hardware [114].

$$SAD_{16} = \Sigma_{i=0}^{3}\Sigma_{j=0}^{3}|C_{i,j} - r_{i,j}| \tag{9}$$

where $r_{i,j}$ are the values of reference 4x4 window and $C_{i,j}$ are the values of the candidate 4x4 block.

In Fig. 11, we used the information of the SLIT function above to build the new idea for the parallax algorithm. In the range of 1/16 the width of the vision, a 3x3 window was used to analyze the level overlap of the SLIT information in the left image and right image. The ratio of overlap was the sum of the values of 1 of

AND operation from ch0 to ch7. At the same time, the SAD of the original left and right 3x3 images was obtained. Finally, the right and left parallax detections were for maximizing the SLIT coincidence and minimizing the SAD.

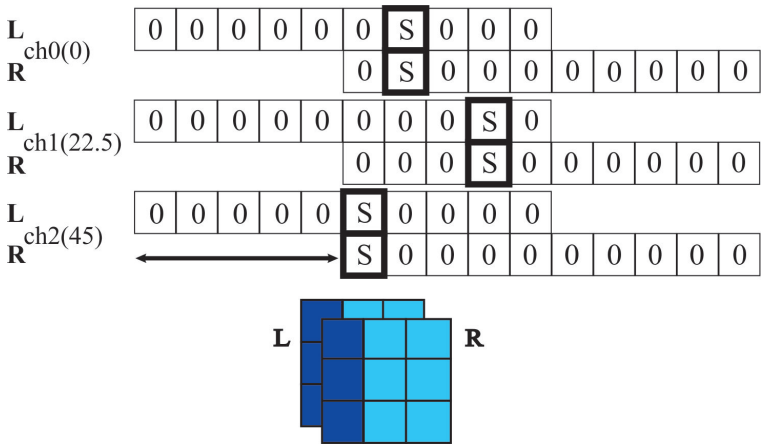


Figure 11. Left/right parallax detection

3.1.4 XY Movement Direction Detection

The Gunnar Farnebäck's optical flow algorithm was applied to determine moving object detection [115]. The horizontal and vertical components were obtained by utilizing Eq. 10 and Eq. 11.

$$H(x,y) = h_1(x,y) + h_2(x,y) \tag{10}$$

$$V(x,y) = v_1(x,y) + v_2(x,y) \tag{11}$$

$$M(x,y) = \sqrt{H^2(x,y) + V^2(x,y)} \tag{12}$$

$$N(x,y) = \frac{M(x,y) - M^{min}}{M^{max} - M^{min}} \times I_{max} \tag{13}$$

$$B(x,y) = \begin{cases} 1 & if \quad N(x,y) \geq \lambda \\ 0 & otherwise \end{cases} \tag{14}$$

where H and V are the horizontal and vertical images. (x,y) is the pixel coordinate, h and v are the horizontal and vertical optical flow components.

The magnitude M(x,y) of the horizontal and vertical optical flows, was calculated, using Eq. 12. The normalizing value N(x,y) of a pixel position (x,y) was processed by using Eq. 13. The output B(x,y) was the comparison to one threshold from Eq. 14.

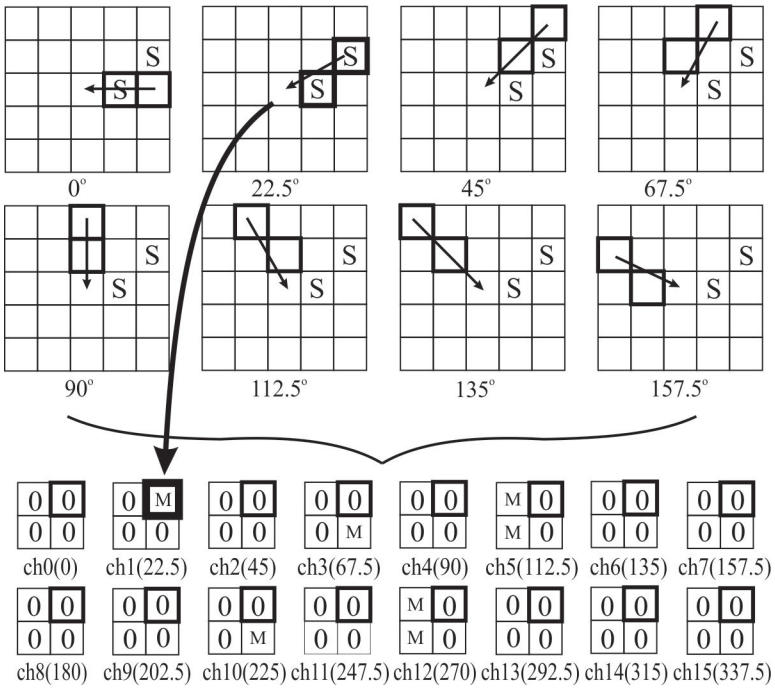


Figure 12. XY movement direction detection

According to the analyzed scheme, multiply, and square root operations are not proper for optimal parallel hardware performance. Antithetical to the above interpretation, in the design, the results of the SLIT function have been applied to determine motion. Unlike the feature detection based on spatial differentiation, the moving direction detection requires time differentiation. M characters or the results of the XY motion detection presented in Fig. 12 is a combination of SLIT detection and the concurrent comparison in which the left and right directions occur at the same moment. AND, Shift, OR, and Comparison functions are simpler than the above operations. The idea can be straightforward to build a high-speed circuit in FPGA. There were 16 output channels, and these channels

could discover the change in the vertical and horizontal directions when the left and right SLITs moved similarly.

3.1.5 Approach Detection

When the left and right SLITs are moved in the same way, the XY moving direction is detected. In contrast, the difference in movement between the left and right SLITs means the approach or separation detection. Six separate output channels in Fig. 13 are the combination of the left and right SLIT information in Fig. 10. It incorporates approach/separation for the right, center, and left eye.

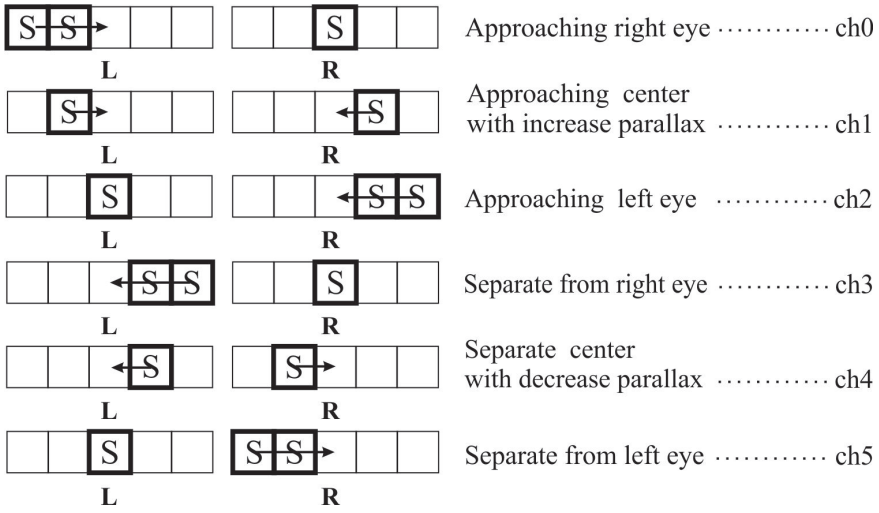


Figure 13. Approach detection

3.2 Reconfigurable Deep Neural Network Using the SLIT Function

3.2.1 SLIT Layer Architecture

In many CNNs such as VGG [116], AlexNet [117], ResNet [118]..., the CONV layer is always the first layer. This layer typically performs a whole number of sliding convolution operations due to the largest input size. The first layer requires much computation time when being compared with other layers. In the CONV layer, with a number of input channels (ICs) and the K×K filters,

there compute six consecutive loops for producing output channels (OCs) in the traditional approach.

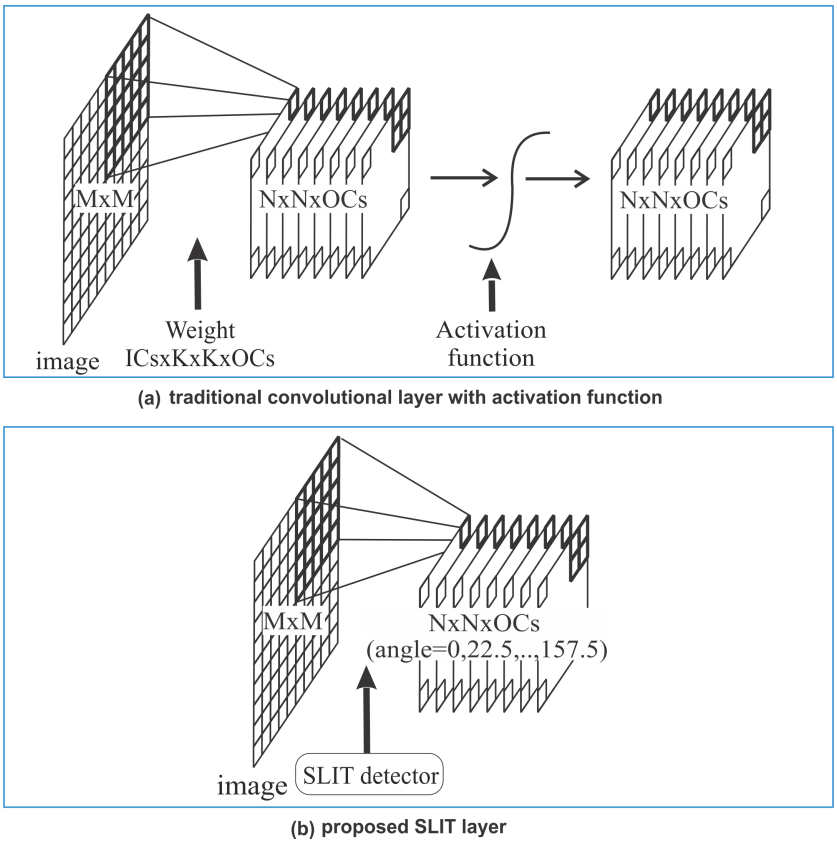


Figure 14. Proposed SLIT layer

In contrast, the proposed SLIT layer presented contains only four continuous loops. Fig. 14(a) explains how to calculate the first CONV layer with the traditional approach. The first CONV layer with a M×M input image is convoluted with ICs×K×K×OCs kernel filters to yield N×N×OCs output channels. Subsequently, the Relu activation function is employed to normalize the output values into a range between 0 and 1. In Fig. 14(b), to obtain N×N×OCs output feature maps like the first CONV layer, we leverage the SLIT layer, as explained beforehand in the motivation section. Due to the binary output, the activation function is discarded after the SLIT layer.

In comparison with the first CONV layer on the original CNNs, the MAC

operation and activation function are eliminated in the proposal. Each input is reused across all filters of different output channels within the same layer in the CONV layer. Therefore, storing memory and power consumption have become enormous. On the other hand, since there are no parameters required for the SLIT layer during the training phase and inference step, memory access and latency are significantly reduced in the proposal. The normalization step for inputs, which are divided by 255, is also ignored in our idea. The Shift, AND, and comparator operations are used to extract feature maps. Consequently, this approach decreases many resources, latency, and energy. Total parameters ($params$) are presented in Eq. 15, and MAC operations ($MACs$) are shown in Eq. 16 are ricocheted in the way that reconstructs with the SLIT layer.

$$params = C\_in \times K \times K \times C\_out \tag{15}$$

where C_in is the number of input channel, K is the size of kernel filter and C_out is the number of output channel.

$$MACs = C\_in \times K \times K \times N\_in \times N\_in \times C\_out \tag{16}$$

where C_in is the number of the input channel, K is the size of kernel filter, N_in is the dimension of output channel, and C_out is the number of output channel.

3.2.2 Next Layer Reconfiguration

Due to the binary output of the SLIT layer, we propose a new scheme to reconfigure the second CONV, max-pooling (MP), and fully connected (FC) layers. We name SCONV, SMP, and SFC layers for the proposed second CONV, MP, and FC layers. MAC operations also occupy most computation time in the second CONV layer, directly following the first CONV layer on CNN. Many works have been investigated to optimize MAC operations, such as using XOR functions [87, 89, 93] . In contrast, we suggest the architecture that employs multiplexer (MUX) operation to determine output feature maps for the second CONV layer. Fig. 15(a) illustrates the process with a 3×3 kernel filter. To receive the second CONV output channel, there require nine multiplication operations. On the other hand, the proposal only uses 9 MUX operations to generate a feature map for the second CONV layer showed in Fig. 15(b).

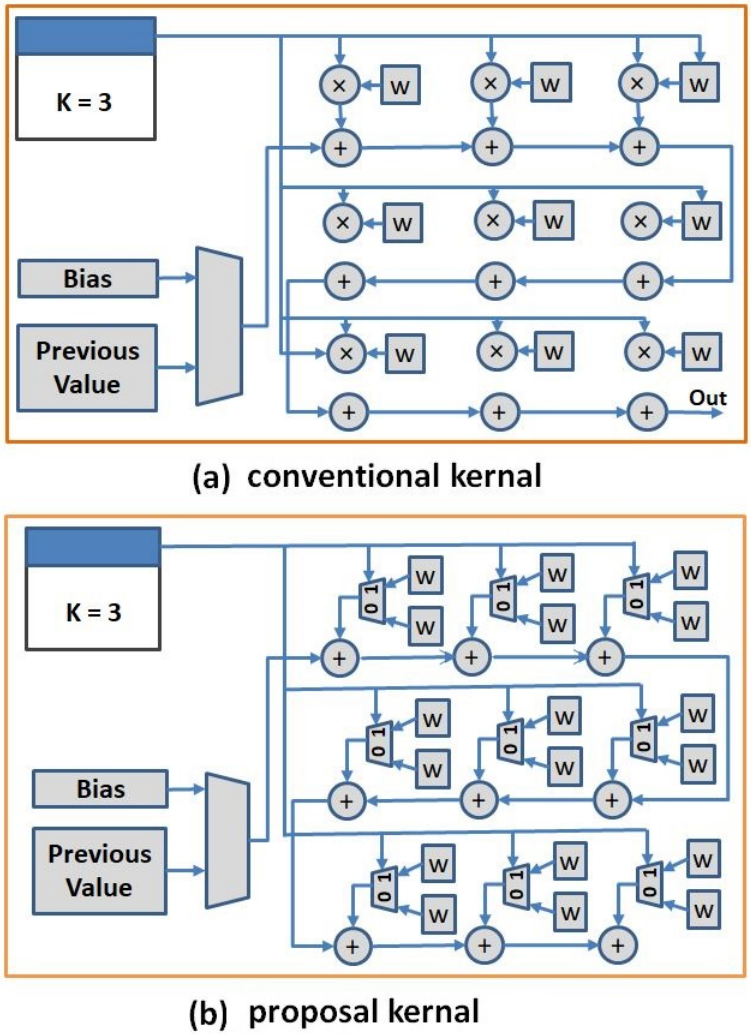


Figure 15. Proposed kernel for the second layer

The proposal excretes all or a part of multiplication operations in the second CONV layer. First, the complete replacement will affect the situation if the SLIT layer only generates eight binary output feature maps in which the input image has one channel like the MINST database. Second, a part of the replacement will take place when the input image has three channels, such as CIFAR, SVHN, or ImageNet database. The SLIT layer yields 11 channels by concatenating eight binary output feature maps of SLIT function with three original channels normalized in range 0 and 1. In this case, output channels of the second CONV layer

are determined by concatenating eight binary output channels of the SLIT layer with three feature maps of the normalized input image.

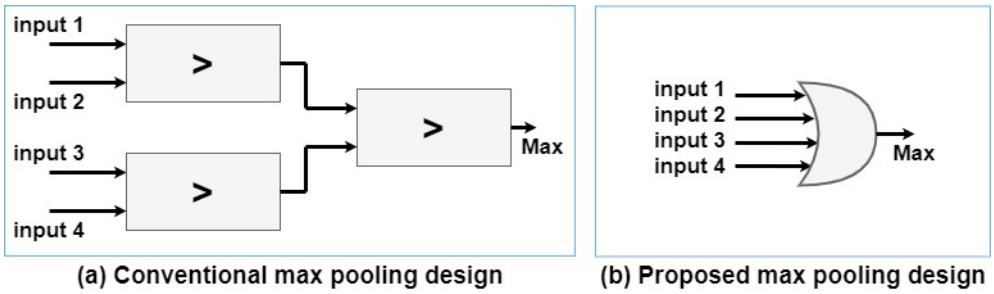


Figure 16. Proposed max pooling kernel

The max-pooling (MP) layer mentioned in Fig. 16 is optimized by utilizing an OR gate to determine the maximum value. Fig. 16(a) reveals that at least three comparator operations are required to detect the maximum value in the 2×2 window at a stride of 2 with the conventional approach. In contrast, the proposed circuit showed in Fig. 16(b) only uses the OR gate to estimate the maximum value for the MP layer. Assuming that there have eight 14×14 output channels from the previous layer, a total of 14×14×3×8 = 4704 comparator operations are expected by applying Eq. 17. On the other hand, the proposal demands 14×14×8 = 1568 OR gates with four inputs.

$$Comp = N \times N \times (K \times K - 1) \times Cout \tag{17}$$

where $Comp$ is total comparator operations required to determine the maximum value, N is the size of the previous channel, K is the kernel size, and $Cout$ is the number of output channels.

The FC proposed layer is affected by a model that consolidates the previous CONV layer and an MP layer. The basic information processing unit of one neural in the artificial network is demonstrated in Fig. 17(a). The inputs are multiplied with corresponding weights, and then outcomes are added with a bias. Fig. 17(b) shows the matrix multiplication replaced by the MUX operations, where the weight values are one. Eq. 18 defines the entire multiplication operations, which occupy most time consumption in the FC, are reduced by the multiplexer operations. Assume that there have 1024 input neurons and 1024

output neurons; by using Eq. 18, the entire 1024×1024 = 1M multiplication operations are pruned.

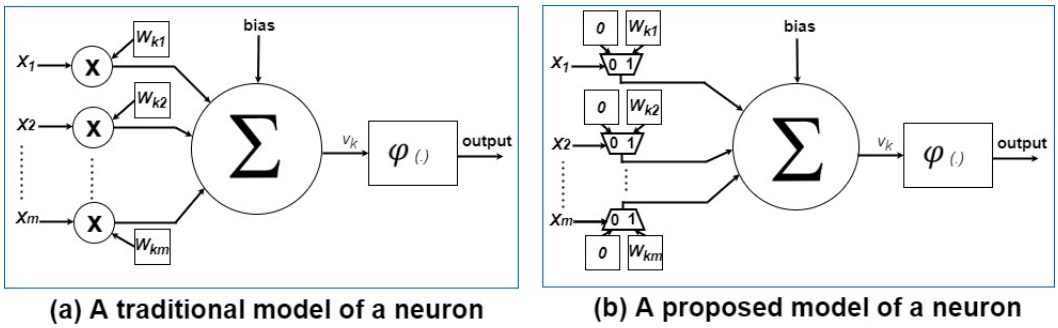


Figure 17. Proposed model of a neuron

$$Muls = Num\_in \times Num\_out \tag{18}$$

where $Muls$ is total multiplication operation to calculate one output. Num_in is the entire input neurons and Num_out is the whole output neurons.

3.2.3 Complete Proposed System

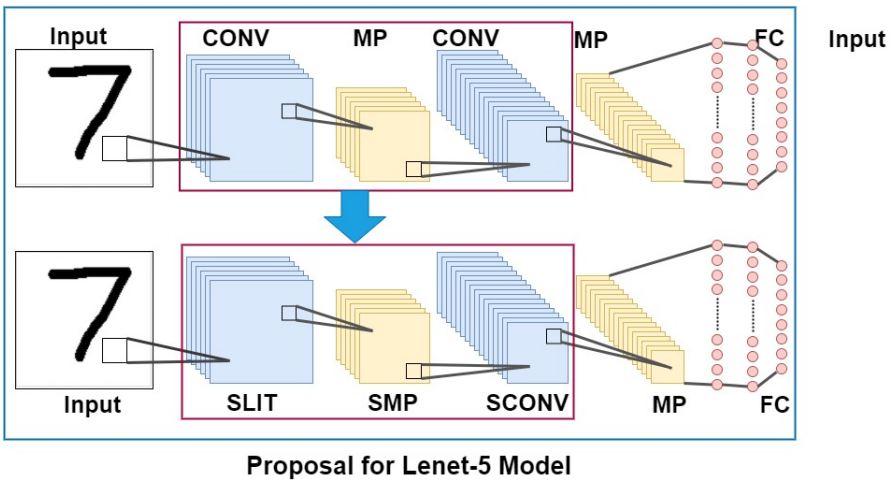


Figure 18. Proposed for Lenet-5 model

This section demonstrates how to reconstruct the first two layer or the first three layers of the proposal in practical applications. The Lenet-5 that combines one CONV + MP + another CONV layer in the model is chosen to manifest how to reconfigure with SLIT + SMP + SCONV layers. In Fig. 18, the CONV + MP + CONV layers are replaced with SLIT + SMP + SCONV layers. The remaining layers stay the same as the original model. The first two CONV layers occur in some famous models such as VGG-16, VGG-19 is reconfigured in Fig. 19. These models include two CONV layers before the MP layer. In this fashion, the first two CONV layers are changed by SLIT and SCONV layers.

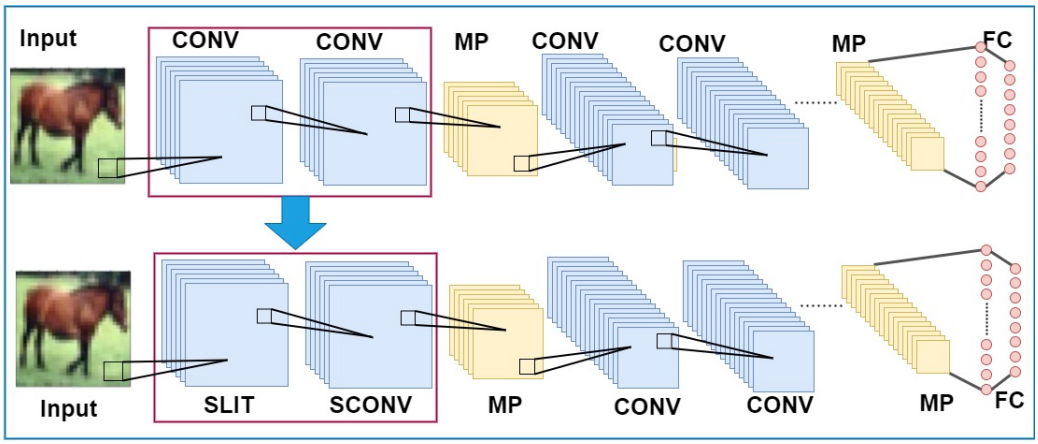


Figure 19. Proposed VGG model

4 Performance of the SLIT Function on Software Platform

4.1 Detail Performance of the SLIT Function with Simple Architecture on MNIST dataset

The network includes a 28x28 pixel input, one 5x5x8 kernel for the convolutional layer with the Relu activation function, one 8x12x12 pooling layer, and one fully connected layer. Fig. 20(a) illustrates a standard CNN network. The proposed scheme is manifested in Fig. 20(b). The SLIT function is divided in the range from 0° to 157.5° yielded 8x24x24 channels that are equivalent to the output of the convolutional layer. Due to the binary outputs of the SLIT function, the Relu activation function is eliminated in the suggested model. The purpose of the pooling layer in the CNN network reduces the feature size to minimize the multiplier-accumulator operations (MACs) in the next convolutional layer. In the proposed architecture, to get the inputs of the next layer, the pooling layer and optimize MACs are excluded by adding weight values that correspond with the previous inputs that are 1.

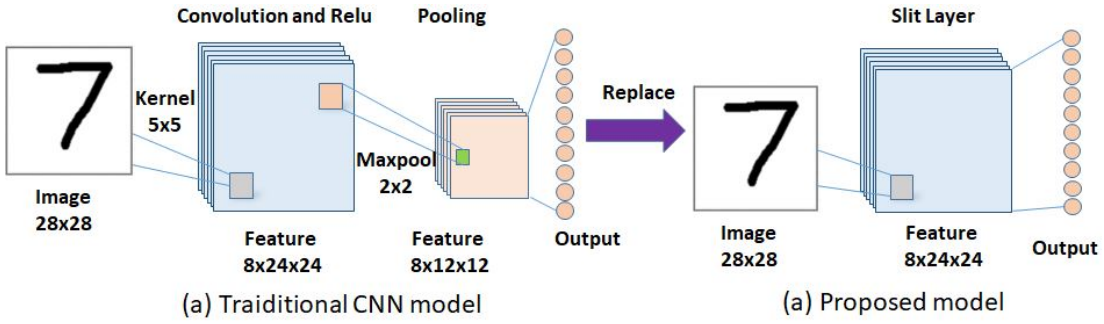


Figure 20. Apply SLIT function on small CNN architecture using MNIST database

This design is verified with the MNIST database [119] with 60,000 images for learning and 10,000 images for evaluation. The batch size consists of 100 images, and the number of epochs is 20. C programming language and the matrix calculation library Atlas are utilized to evaluate this method. Intel(R) Xeon(R)

Gold 6144 CPU @ 3.50GHz is used to measure the execution time. As represented in Table 1, this model evaluated by software using C programming language shows that the SLIT function only takes 15.3 seconds to extract features of the input images.

Table 1. Execution time and point convergence measurement using C programming language

Metrics	**Traditional CNN**	**Proposal**
The first layer	190 sec (Conv)	15.3 sec (SLIT)
Forward	231.8 sec	121 sec
Conv.backward	446.9 sec	0 sec
Backward	511.5 sec	102 sec
Training time	743.5 sec	239 sec
Inference time	38.7 sec	22.8 sec
Point convergence	13^{th}	5^{th}

Conv: Convolutional layer, **SLIT**: SLIT function
sec: Seconds, **th**: n^{th} within 20 epochs.

In comparison, it requires 190 seconds for the convolutional layer. Notably, execution time is approximately 446.9 seconds for the convolutional layer in back-propagation, which needs no time in the proposal. The total forward time is smaller by a factor of 1.91 when being compared to the conventional CNN. Furthermore, it also reduces the feed-backward time from 511.5 seconds to 102 seconds. The learning time is decreased by 3.1 times from our model equaled to CNN. In brief, total time consumption on CNN is extensive because it takes much time for feed-backward. The identification consumes 22.8 seconds, which is smaller than 38.7 seconds of the CNN model. The proposed scheme convergence is achieved after only five epochs, while the CNN model requires 13 epochs during 20 iterations. Analyzing the execution time analysis of the two models shows that the SLIT function efficiently replaces the convolutional operand. The suggested architecture has significantly reduced the time demand for the training phase and inference step in the deep neural network.

The fundamental idea in the proposed method is the SLIT function with input

as the result of the edge function. We propose three different designs for the SLIT function. The result of the 4x4 window gains the highest performance of all three models. The V1 region of the brain extracts the feature properties of objects from the eye, and it corresponds with the first layers of CNN, mimicking V1 [2, 113]. The 4x4 window in Fig. 10 rotates in a resolution from 0° to 157.5°, which perfectly coincides with the eye vision. It is considered that they are eight different combinations in this range.

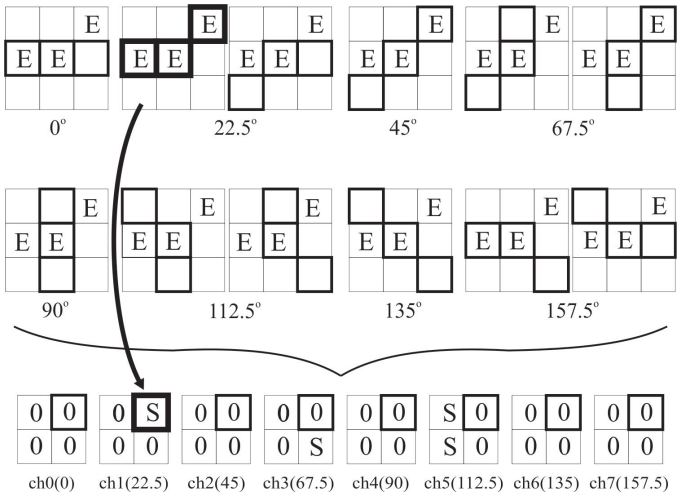


Figure 21. SLIT function using 3×3 window

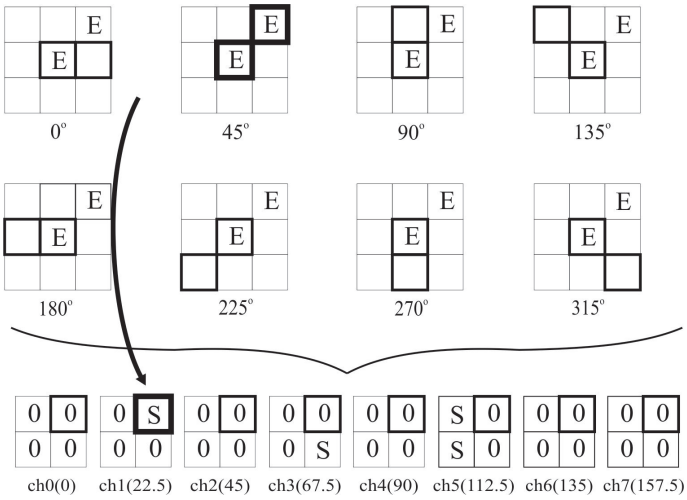


Figure 22. SLIT function using 2×2 window

For the 3x3 window in Fig. 21, the resolution is still within the eye vision, but there are many overlapping slopes at 22.5°, 67.5°, 112.5°, and 157.5°. Therefore, its result is nearly the same as the 4x4 kernel. The 2x2 window in Fig. 22 scans with an angle from 0° to 315° with a gradual increase every 45°, but with only four cases of necessary information contribute to the network, and the remaining cases pass the eye's perspective. It is assumed that we choose the 5x5 window and reduce the step in the range of 0° to 157.5° to extract more features, the overfitting problem and hardware resources are a trade-off. For balancing resources and precision, the 4x4 window is the best candidate.

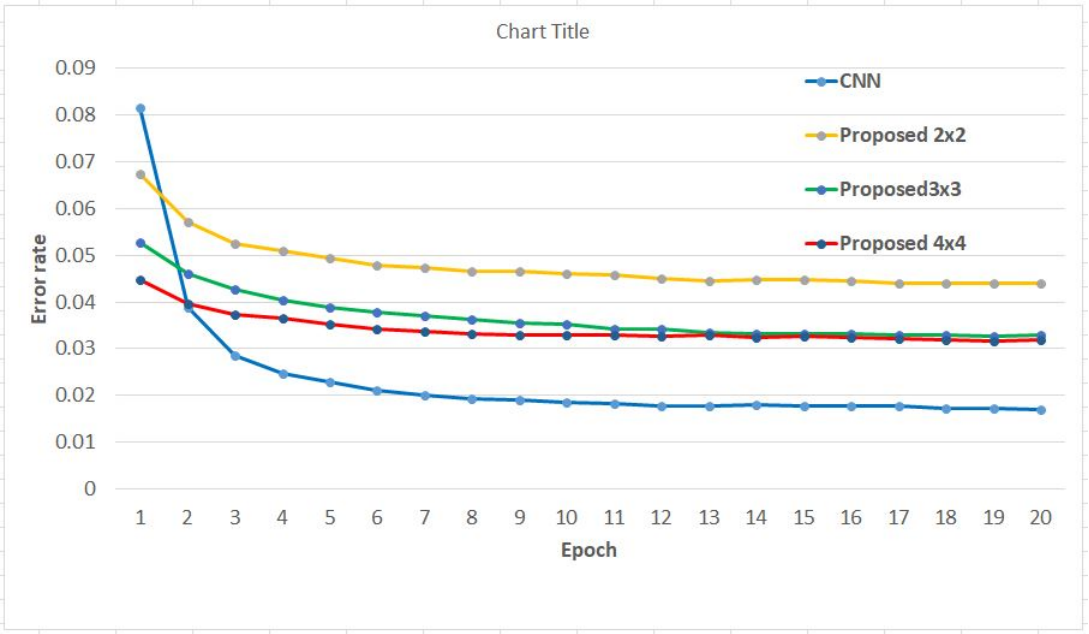


Figure 23. Comparison error rate between SLIT and CNN on MNIST database

Fig. 23 summarizes the comparison of three suggested concepts for SLIT function with conventional CNN. Comparison of three proposals, the window 4×4 gets better results than others. In short, the accuracy of our model with the proposed 4×4 window is slightly decreased from 97.89% to 97.34% when comparing with that of the CNN model.

4.2 Performance of Reconfigurable Deep Neural Network

4.2.1 Software Configuration

In this section, the utilization of the SLIT layer in various models is investigated. I conduct extensive experiments on the standard Lenet-5, VGG-16, and VGG-19 prototypes with the MNIST [119], SVHN [120], CIFAR-10, CIFAR-100 [121], and ImageNet [122] datasets. Tensorflow, Keras [123], and Pytorch [124] platforms are used to build the models. Training time and accuracy of the MNIST, CIFAR, and SVHN databases are analyzed by using the Intel(R) Core(TM) i7-3970X CPU @ 3.50GHz. The GeForce GTX 1080 is used for training the ImageNet dataset. To determine hyper-parameter values in the proposed model, first, training the examining database on the traditional model to estimate the hyper-parameters at an acceptable accuracy like benchmarks. Then, during the training phase of the proposed model, I increase or decrease appropriately the value of the reference hyper-parameters extracted from the conventional model. Finally, the hyper-parameters of the proposal are determined when the over-fitting and under-fitting phenomena disappear, and the model converges with the highest accuracy. In comparison with the traditional model at the same database, the hyper-parameters are nearly identical between the two models. Therefore, I have used the same values when training conventional and proposed models. I evaluate latency, hardware resources, and power consumption of the inference phase on the chip ZC7Z020-1CLG484C FPGA.

Handwritten digits (MNIST): The MNIST database [119] consists of 28×28 gray images of the handwritten digits "0" through "9". A total of 60000 images are provided for training, and 10000 images leave for testing. In the reported experiment, training images are sliced further into a training set (50000 images) and a validation set (10000 images), equal to the distribution of digit classes. Fig. 24(a) shows samples of the MNIST dataset. The Lenet-5 paradigm is considered for performance analyses. From the original Lenet-5 model which combines CONV(6) + MP(2) + CONV(16) + MP(2) + FC(120) + FC(84) + FC(10), the design that mixes of SLIT(8) + SMP(2) + SCONV(16) + MP(2) + FC(120) + FC(84) + FC(10) is proposed. The simulation is carried out using a batch size of 100 images, 20 epochs, and the stochastic gradient descent (SGD)

optimization function with a learning rate of 0.1.

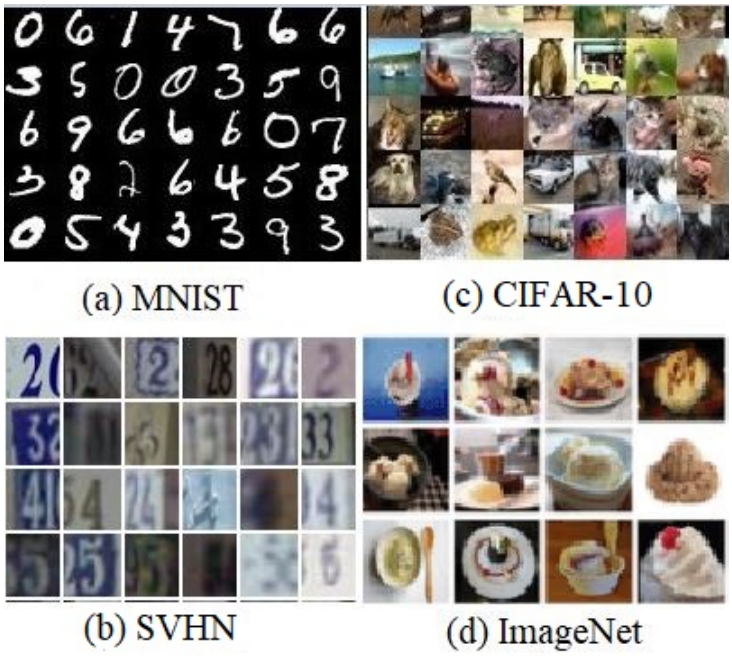


Figure 24. Examples of MNIST, CIFAR, SVHN and ImageNet databases

SVHN dataset: the SVHN dataset [120] has three channels in an image is also examined in this proposal. SVHN is collected from house numbers in Google Street View images. It includes 73257 images for training and 26032 images for testing. Examples of the SVHN dataset are displayed in Fig. 24(b). I investigate the SVHN dataset with the model that combines 2CONV(32) + MP(2) + 2CONV(64) + MP(2) + FC(512) + FC(10). In this manner, two CONV(32) layers are replaced with SLIT(11) and SCONV(32) layers. The model is trained with a batch size of 128 images, 20 epochs, and the SGD optimization function with a learning rate of 0.01.

CIFAR database: The proposal is interpreted in detail on the CIFAR-10 and CIFAR-100 datasets [121]. These datasets are composed of 60000 samples from ten categories for CIFAR-10 and 100 categories for CIFAR-100. Fig. 24(c) shows examples of the CIFAR-10 dataset. This experiment utilizes 45000 images for training, 5000 images for validation, and the last 10000 images for testing, and augment the database by exerting flip and shift operators. The Lenet-5, VGG-16,

and VGG-19 models are employed for measuring performance. These models are assessed with a batch size of 128 samples, 200 epochs, and the SGD optimization function with learning rate change from 0.1 in a range of 0 to 100 epochs, 0.01 in a range of 100 to 150 epochs, and 0.001 for larger than 150 epochs. Since the CIFAR dataset has three input channels, I have concatenated eight output feature maps of the SLIT function with three input channels normalized in a range of 0 and 1 to create the SLIT layer. For the Lenet-5 design, we validate with SLIT(11) + SMP(2) + SCONV(16) + MP(2) + FC(120) + FC(84) + FC(10) as the equivalence of the conventional Lenet-5 scheme which stacks up of CONV(6) + MP(2) + CONV(16) + MP(2) + FC(120) + FC(84) + FC(10). In the VGG-16 and VGG-19 forms, two first CONV layers with 64 output channels are switched by SLIT(11) + SCONV(64) layers.

ImageNet database: The ILSVRC2012 ImageNet dataset [122] has also been chosen as a target to assess our topology in a complicated case. ImageNet includes approximately 1.2M training images with 1K classes and 50K validation images. This dataset covers natural images with reasonably high resolution compared to the CIFAR, MNIST, and SVHN datasets, which have relatively small images. The examples of the ImageNet database are shown in Fig. 24(d). The image classification performance has been conducted to report Top-1 and Top-5 accuracy. The VGG-16 architecture is adopted as the base proposal. Two first CONV layers of the VGG-16 model is reconstructed with SLIT(11) and SCONV layers. The design is simulated with a batch size of 16 samples, 100 epochs, and the SGD optimization function at a learning rate of 0.001.

4.2.2 Software Results

Figure. 25 shows the accuracy of the MNIST, CIFAR-10, CIFAR-100, SVHN, and ImageNet datasets. A small decrease in accuracy of 0.27% from 99.07% to 98.8% with the MNIST database has been observed when being compared between the Lenet-5 proposal and the conventional Lenet-5 paradigm. Moreover, it slightly decreases from 0.5% to 1.5% with CIFAR-10 and CIFAR-100 datasets and around 2.2% on the ImageNet. It also remarks efficiently on the small CNN model that is experimented on the SVHN dataset. With complicated models such as VGG-16 and VGG-19, the loss of accuracy ranges from 0.5% to 2.2%.

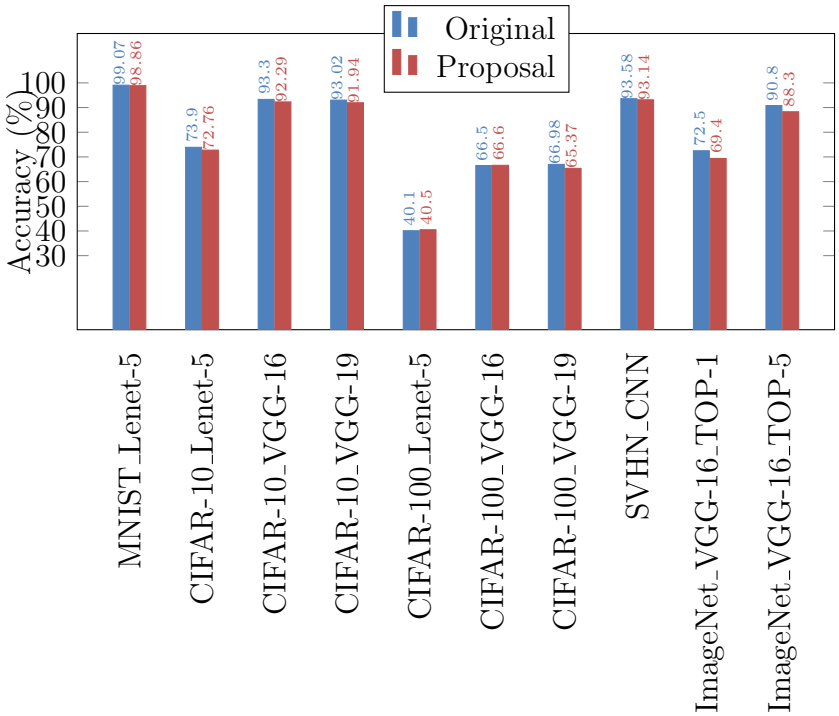


Figure 25. Comparing accuracy between the original model and the proposed model

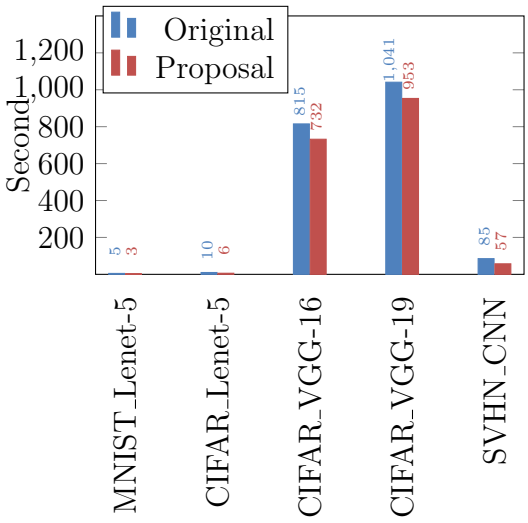


Figure 26. Comparing training time of one epoch between the original model and the proposed model

A training time reduction shown in Fig. 26 compensates for the loss of accuracy when the proposal is applied. A decrease training time in a range of 10% to 40% has been verified from small scheme to complex topology models. Total training time is diminished by 40%, 40%, and 32%, corresponding with MNIST, CIFAR, and SVHN databases on Lenet-5 and CNN models. It also decreases by approximately 10% on larger paradigms such as VGG-16 and VGG-19 with the CIFAR database. Because the model verified with the VGG-16 on ImageNet takes a long time for training one epoch, this case is not revealed in Fig. 26.

Table 2. Comparison parameters on Lenet-5 and VGG-16 model

Layer	**Kernal**	**Original**	**Optimized [50]**	**Proposal**
Lenet-5 model				
CONV1	1×5×5×6	150	336	0
CONV2	6×5×5×16	2400	2752	2400
VGG-16 model				
CONV1	3×3×3×64	1728	41K	0
CONV2	64×3×3×64	2400	49K	2400

Table 3. Comparison operations on Lenet-5 and VGG-16 model

Layer	**Dimension**	**Original**	**Optimized [50]**	**Proposal**
Lenet-5 model				
CONV1	1×28×28	117.6K (a)	225K (a)	0
MP	6×24×24	27.6K (b)	27.6K (b)	9216 (c)
CONV2	6×12×12	345.6K (a)	419.5K (a)	345.6K (d)
VGG-16 model				
CONV1	3×32×32	1.77M (a)	37.6G (a)	0
CONV2	64×32×32	37.7M (a)	50.3G (a)	37.7M (d)

a: MAC, b: Comparator, c: OR, d: Multiplexer

Ordinarily, the first and second CONV layers contribute 92.4% of MAC operations, while the FC layers offer 7.6% of MAC operations on the Lenet-5 model.

Parameter and operation reduction highlight the contribution of the proposed model for the training phase on CNNs. Results indicate a considerable efficiency obtained on the proposed Lenet-5 model. Table 2 and Table 3 show that a total of $1\times5\times5\times6 = 150$ parameters and 463K MAC operations are pruned in the proposal when evaluated with the MNIST database. Remarkably, with the CIFAR dataset that has three channels, $3\times5\times5\times8 = 450$ parameters and a total of $3\times5\times5\times32\times32\times6 + 6\times5\times5\times16\times16\times16 = 1.07$G MAC operations are excluded. The proposal decreases approximately by 90% MAC operations and leads to training time reduction during the training phase on the Lenet-5 model. An entirety of 1728 parameters and 1.77M MAC operations are also eliminated in the first layer on the VGG-16 model. In short, to compare with the original approach and reference, my model illustrates better MAC operation optimization on the Lenet-5 and VGG-16 models.

5 Performance of the SLIT Function on Hardware Platform

5.1 Hardware Setup

Among the various available tools for implementing hardware designs of CNNs on different FPGAs, Xilinx Vivado ® High-Level-Synthesis (Vivado_HLS) is commonly used in literature for the sake of productivity at the cost of hardware efficiency and performance [50, 105, 106, 107, 108]. Hence, I leverage the Vivado_HLS and Vivado_IDE (v2018.3) tools to realize hardware circuits. The FPGA synthesis is executed with chip ZC7Z020-1CLG484C for the property with benchmarks in comparison. I use Vivado_HLS to compare hardware resources and latency of the IP core between the original approach and the proposal with a 32-bits floating-point and 16-bits fixed point at a frequency of 100 Mhz. My IP core is conducted into an embedded system to verify area and power on real FPGA at 115MHZ of the frequency with 24-bits fixed point.

First, the SLIT layer is evaluated in two cases with eight binary output feature maps and eleven output channels that concatenate eight binary channels with three input channels. Second, I stack another CONV layer after the first CONV layer to assess how to compose the SCONV in the proposed topology. There have two CONV layers in the primary, but the proposal is concatenated SLIT and SCONV layers. Third, the MP proposal on CNNs is analyzed by appending one MP after the first CONV layer. I explain two cases: the first case is the structure having CONV, MP layers and the second is CONV, MP, CONV layers in the model. I handle the SLIT, SMP layers, and the SLIT, SMP, SCONV layers to compare with the data obtained from the conventional scheme. Next, the FC proposal is studied by linking the SLIT, SMP, and SFC layers. Finally, I investigate the architecture of the Lenet-5 and VGG-16 models as the analyzed standard of the proposed networks on hardware to compare with state-of-the-art. How to replace the first three layers is showed in the Lenet-5, and VGG-16 represents how to reconstruct the first two layers on the deep neural networks.

5.2 Comparison Hardware Resources and Latency with Vivado_HLS

Table 4. Comparing hardware resources and latency for the first layer

Metrics	CONV(8)	SLIT(8)	CONV(11)	SLIT(11)
Floating-point	**32-bits**	**32-bits**	**32-bits**	**32-bits**
LUT	724	241	815	549
FF	960	233	1073	657
DSP48E	8	0	8	3
BRAM	2	1	8	1
Latency (ms)	13.47	0.427	40.17	1.04

Table 4 exposes the reduction of hardware resources and performance for the first layer. The SLIT layer employs fewer LUTs, FFs, BRAM and DSP48E blocks than the CONV layer. Especially, the DSP48E blocks are humbled eight times in the case of SLIT(8). Latency achieves a $13.47/0.427 = 31.5\times$ reduction compared with the CONV(8) layer and a factor of $38\times$ decrease with the CONV(11) layer.

Table 5. Comparing hardware resources and latency for the second layer

Metrics	CONV(8)+ CONV(16)	SLIT(8)+ SCONV(16)	CONV(11)+ +CONV(16)	SLIT(11)+ SCONV(16)
Floating-point	**32-bits**	**32-bits**	**32-bits**	**32-bits**
LUT	1303	736	1425	1223
FF	1649	769	1811	1399
DSP48E	13	2	13	8
BRAM	18	2	40	10
Latency (ms)	160.7	96.5	266.3	210.9

Table 5 reveals the hardware resources and latency for the second layer. By replacing all MAC operations with the MUX function in case SLIT(8) +

SCONV(16), hardware utilization is notably reduced. For example, 2 DSP48E blocks are proportional to 13 DSP48E blocks in the standard design. BRAM blocks are lessened $18/2 = 9\times$ between two models. Latency is also decreased remarkably in our proposal when being compared with the traditional topology.

Table 6. Comparing hardware resources and latency for the max pooling layer

Metrics	**CONV(8) +MP(2)**	**SLIT(8) +SMP(2)**	**CONV(11) +MP(2) +CONV(16)**	**SLIT(11) +SMP(2) +SCONV(16)**
Floating-point	**32-bits**	**32-bits**	**32-bits**	**32-bits**
LUT	1028	341	1686	1497
FF	1255	334	2071	1673
DSP48E	8	0	13	8
BRAM	18	2	48	13
Latency (ms)	13.6	0.46	96.9	53.6

Table 6 reveals the max-pooling layer performance. By replacing three comparator operations with an OR gate, latency or speed is reduced from 13.6 ms to 0.46 ms. Hardware resources that estimate the IP core area extremely decrease on DSP48E and BRAM blocks. A factor of approximately 92% hardware resource reduction is observed when our SMP layer is compared to the second traditional MP layer. In a more complicated case like SLIT(11) + SMP(2) + SCONV(16), the proposed reconfiguration not only reduces significant hardware resources but also demand 53.6 ms, a reduction from 96.9 ms as in the case of CONV(11) + MP(2) + CONV(16).

The FC proposal is analyzed by combining SLIT, SMP, and SFC layers. By replacing the multiplication matrix with MUX functions, Table 7 proves that my suggestion also works better than the traditional process in terms of hardware resource utilization and execution time requirements. In short, four loops in the SLIT layer, OR gate in the SMP layer, and MUX operation in the SCONV layer result in enormous hardware resources and latency reduction.

Table 7. Comparing hardware resources and latency for the fully connected layer

Metrics	**CONV(8) +MP(2)) +FC(512)**	**SLIT(8) +SMP(2) +FC(512)**	**CONV(11) +MP(2) +FC(1024)**	**SLIT(11) +SMP(2) +FC(1024)**
Floating-point	**32-bits**	**32-bits**	**32-bits**	**32-bits**
LUT	1531	741	1549	1359
FF	1924	829	2006	1578
DSP48E	13	2	13	8
BRAM	22	3	48	13
Latency (ms)	105.5	60.5	454.3	379.9

Table 8. Comparing hardware resources and latency on Lenet-5 and VGG-16 models

Metrics	**Proposal Lenet-5**		**Proposal VGG-16**	
	Traditional	**Proposal**	**Traditional**	**Proposal**
Floating-point	**32-bits**	**32-bits**	**32-bits**	**32-bits**
LUT	4568	3854	43442	43140
FF	4371	3405	11276	10852
DSP48E	24	16	62	57
BRAM	17	8	480	354
Latency (ms)	34.3	20.78	49820	44503

Compared with the traditional CNNs on Lenet-5 and VGG models, our scheme replaces the first three layers on the conventional Lenet-5 model and the first two layers on VGG. By synthesizing with the Vivado_HLS tool, Table 8 shows the proposal has consumed less than 52.9% BRAM and 33.3% DSP48E blocks compared with the traditional Lenet-5 model. Moreover, my scheme achieves about 1- 20.78/34.3 = 0.394 or 39% latency reduction without using the optimized methods such as #parama HLS_PIPElINE or #parama HLS_UNROLL. I have also investigated hardware resources and latency for the VGG-16 scheme. The hardware resources also degrade 26% in BRAM blocks and 8% in DSP28E blocks

for the complete proposed VGG-16 design. The latency reduces about 10% as comparing the conventional approach.

5.3 Comparison Hardware Resources and Power Consumption on IP Core with Orther Researhes

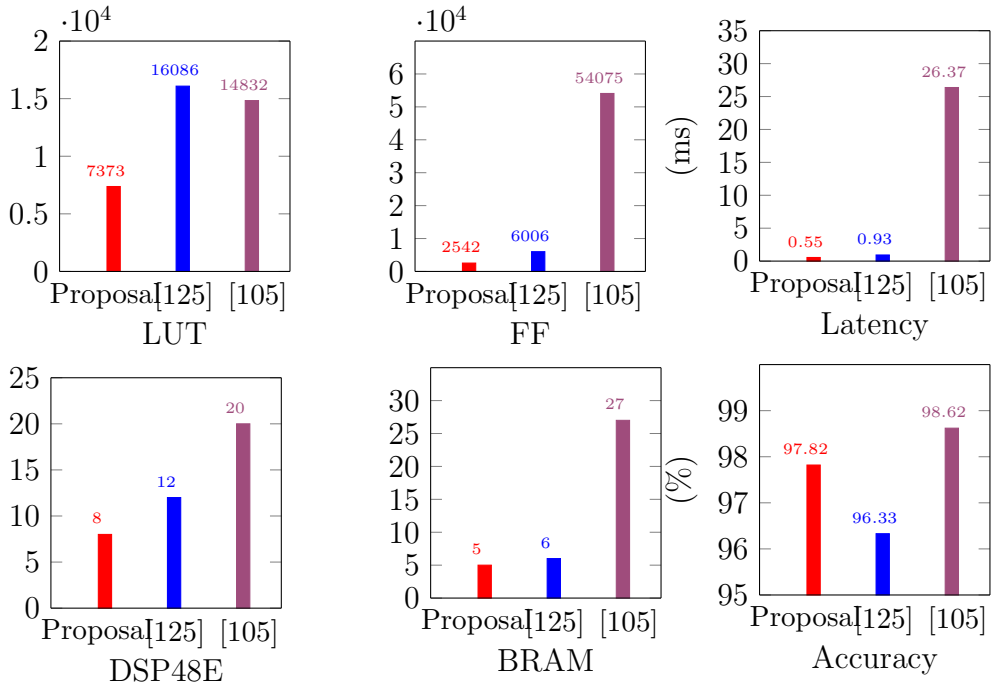


Figure 27. Comparison hardware resources and latency of our IP core proposal with other works on Lenet-5 model at 100 MHz using Vivado_HLS tool

For the IP core comparison between the proposal and existing state-of-the-art using the MNIST database, the network combining CONV(8) + MP(2) + CONV(8) + MP(2) + FC(10) is proved. The proposed model consists of SLIT(8) + SMP(2) + SCONV(8) + MP(2) + FC(10). In addition to constructing the first three layers, I also use #parama HLS_PIPELINE and #parama HLS_UNROLL technologies to improve the hardware design performance. I utilize a 16-bits fixed point while still maintaining accuracy. Fig. 23 reveals that the proposal demands a smaller number of hardware resources than previous works at higher accuracy. Especially, the latency achieves a 40.8% reduction over the work, and a factor

of 26.3/0.55 = 47× decrease compared with the result reported in the previous study. Besides, the hardware resource is lower with 1 BRAM block, 4 DSP48E blocks, 6006/2542 = 2.3× FFs, and 16086/7373 = 2.18× LUTs as compared with the highest current performance. Moreover, the proposal maintains 97.82% accuracy higher than 96.33% in reference.

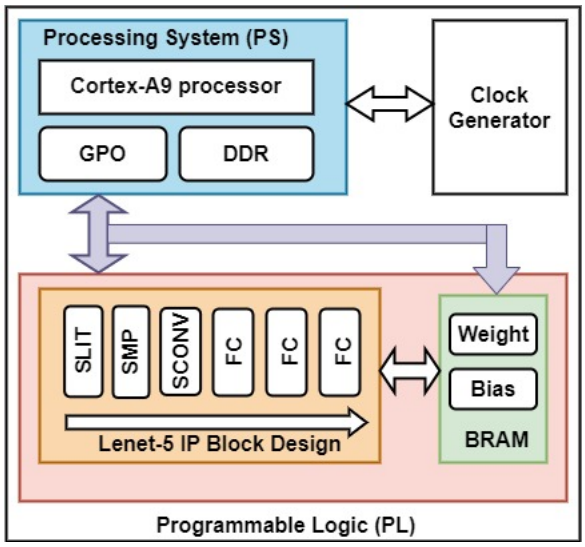


Figure 28. System on chip implementation of the Lenet-5 model on zynq7020 FPGA

As shown in Fig. 28, the CNN accelerator design includes ARM, AXI, BRAM, and my IP core. The IP core is called in an ARM CPU-based embedded system to analyze the effectiveness of the proposed optimization technique. Table 9 exposes the comparison between my model and the previous works in hardware resources used to estimate area and power consumption. Due to the binary calculation on SLIT, SMP, and SCONV layers, the DSP48E blocks are extremely reduced in the proposal. To fairly assess, I convert 16 DSP48E blocks into 1003 LUTs, and 537 FFs in the way reference [108] measurement and estimate equivalently one BRAM into 256 LUTs as references [126, 127]. As a result, my topology utilizes the same LUTs with a 72.5% reduction in FFs compared with the other works. Moreover, the proposal also employs a 0.456 W power consumption lower than works [106, 107, 108].

Table 9. Comparing resource utilization and power consumption on chip zynq7020 FPGA for Lenet-5 model

Parameter	**[106]** 24-bits fixed point	**[107]** 32-bits floating-point	**[108]** 8-bits fixed point	**Proposal** 24-bits fixed point
Frequency	166 MHZ	100 MHZ	100 MHZ	115 MHZ
LUT	38836	14659	39898	6853
FF	23408	14172	25161	6378
DSP48E	95	125	0	16
BRAM	92	119.5	24	127
Power (W)	3.32	1.8	1.758	0.456

5.4 DPU architecture for SLIT+CNN on Vitis AI platform

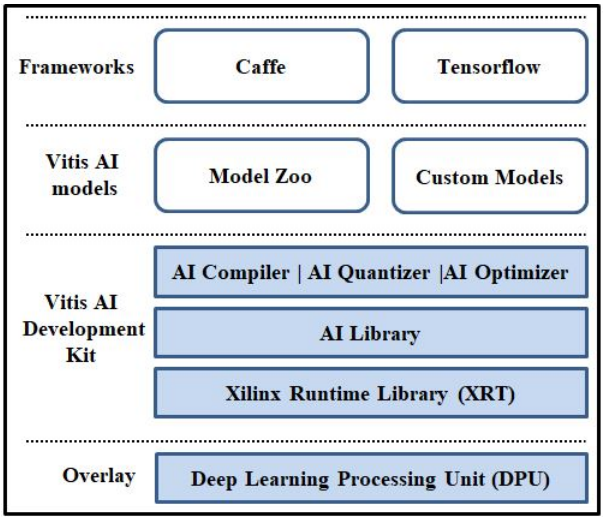


Figure 29. Flow vitis AI platform[128]

Vitis AI supporting AI inference acceleration is an integral part of Vitis from the Xilinx platform [128]. The Vitis AI produces high-throughput CNN inference

engines for FPGAs and ASICs. It offers a complete series of tools and APIs for pruning, quantizing, optimizing, and compiling pre-trained models to reach the highest AI inference performance on Xilinx platforms. This program is a foundation for the gap between deep learning frames such as Tensorflow, Keras, Caffe, and hardware circuits. Vitis AI permits software developers to exploit FPGA or ASIC acceleration benefits without demanding HDL progress and low-level circuit expertise. The overall flow chat of Vitis AI is summarized in Fig. 29.

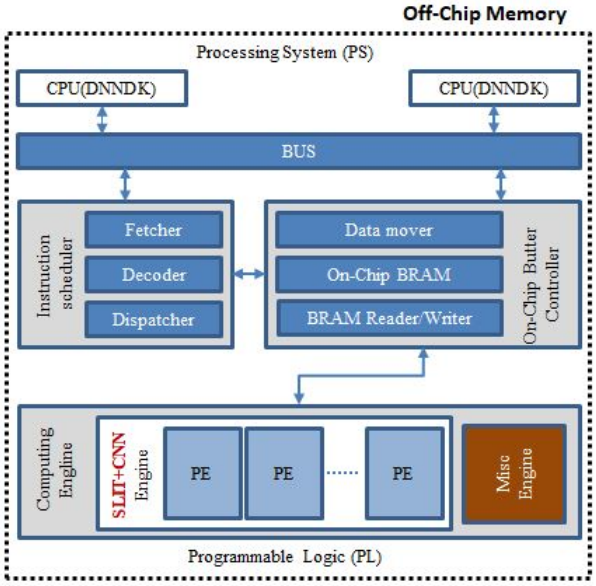


Figure 30. The proposal SLIT + CNN on the DPU architecture

In this research, the SLIT layer is mixed with conventional CNNs on the DPU structure to reduce the latency of the inference step on the Xilinx ZCU_102 FPGA board. The Xilinx® Deep Learning Processing Unit (DPU) [129] is an optimized programmable tool for deep neural networks. It comprises an instruction fetch unit, a high-performance scheduler module, a global memory pool module, and a hybrid computing array. A specific instruction set conceding for the satisfactory implementation of CNNs is integrated into the DPU. Fig. 30 shows the proposal of the SLIT layer on DPU architecture in the deep neural network topology. The first layer of CNNs replaced by the SLIT layer. After training the hybrid network, the parameters are quantized and optimized to measure the accuracy and latency of

the inference stage on the ZC7Z020-1CLG484C FPGA. The MNIST and CIFAR-10 datasets are employed to validate the execution. Due to hardware resource limitations, I only assess the schemes, for example, Lenet-5, VGG-11, and VGG-13.

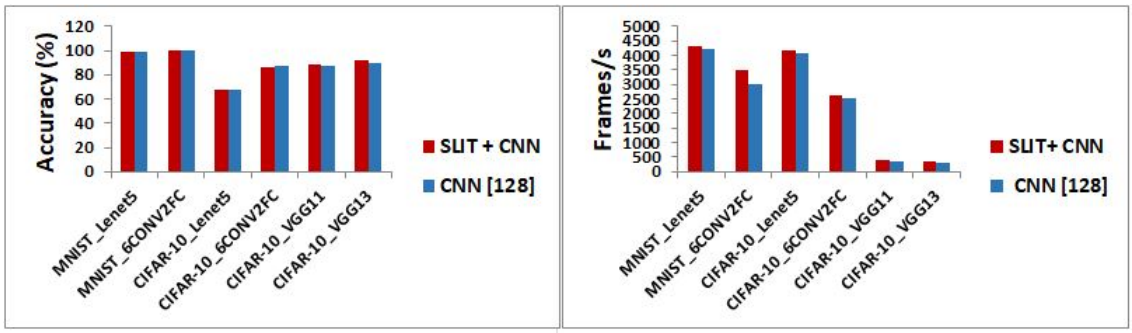
Table 10. Comparison resource utilization on DPU platform

Parameter Size (MB)		
	SLIT + CNN	**CNN [129]**
MNIST_Lenet5	**0.04**	0.04
MNIST_6Conv2fc	**0.86**	0.86
CIFAR10_Lenet5	**0.06**	0.06
CIFAR10_6Conv2fc	**1.09**	1.09
CIFAR10_VGG11	**9.74**	9.8
CIFAR10_VGG13	**9.94**	9.97
MAC (MOPs)		
	SLIT + CNN	**CNN [129]**
MNIST_Lenet5	**0.56**	0.72
MNIST_6Conv2fc	**41.89**	43.61
CIFAR10_Lenet5	**1**	2.29
CIFAR10_6Conv2fc	**43.89**	58.14
CIFAR10_VGG11	**1330.56**	1451.45
CIFAR10_VGG13	**1744.3**	1810.36
I/O Memory space		
	SLIT + CNN	**CNN [129]**
MNIST_Lenet5	**0.008 (KB)**	2.02 (KB)
MNIST_6Conv2fc	**0.03 (MB)**	0.05 (MB)
CIFAR10_Lenet5	**0.01(KB)**	5.23(KB)
CIFAR10_6Conv2fc	**0.04 (MB)**	0.07 (MB)
CIFAR10_VGG11	**0.19 (MB)**	0.2 (MB)
CIFAR10_VGG13	**0.3 (MB)**	0.31 (MB)

The board FPGA Zybo Zynq-7000 (XC7Z010) containing 240KB block RAM, 28K logic cells, and 80 DSP slices is handled to measure the performance. The

target FPGA board includes the DPU B4096 built at a frequency of 325 MHz. The parameter size is utilized to save weight and bias in MB, KB, or bytes for the DPU kernel. The computation workload in the unit of MOPs for the DPU kernel is called workload MACs (MACs). The required DPU memory space in MB, KB, or bytes for the intermediate feature map is named I/O memory space.

Table 10 compares the resource utilization outcomes from the FPGA accelerator on the DPU platform. Due to optimizing the first layer in traditional CNN architecture, the parameter size, workload MACs, and I/0 memory space are reduced from 1% to 10% on the new topology. The accuracy depicted in Fig. 31(a) has remained the same as the conventional CNN when comparing the previous work [128]. Furthermore, an enhancement throughput from 2.6% to 16% is demonstrated in Fig. 31(b) when comparing with the outcomes reported in the reference [128].



(a) Comparison accuracy between SLIT + CNN and CNN

(b) Comparison throughput between SLIT + CNN and CNN

Figure 31. Comparison accuracy and throughput between SLIT + CNN and CNN on DPU platform

6 Apply SLIT Function into Spiking Neural Networks on Adversarial Attack Application

6.1 Overview Spiking Neural Network and Adversarial Attack

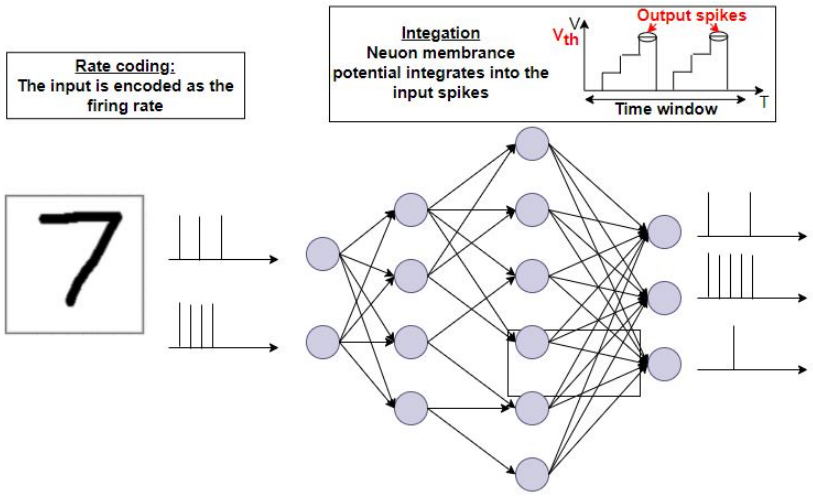


Figure 32. Overview of Spiking Neural Network

An illustration of the SNN diagram is portrayed in Fig. 32. The input data has to be suitably coded using encoding techniques such as rate coding, temporal coding. Coding schemes can be based on the number of spikes, the delay between consecutive spikes, or the latency between starting the stimulus to the first spike. Rate encoding is chosen as the most common mechanism. In rate coding, the activation intensity corresponds to the mean firing rate over a determined time window. A time window represents the measurement period in which the SNN takes the same input. An incoming spike is multiplied by its associated synaptic weight and integrated into the membrane potential V, following Eq. 19, when it arrives at the input of the neuron.

$$V = \sum_{i=1}^{N} w_i \cdot s_i \tag{19}$$

The leaky integrate and fire (LIF) model [130, 131] is the most universally utilized SNN model. The presence of each LIF neuron can be concisely expressed as Eq. 20

$$\begin{cases} \alpha \frac{dv(t)}{dt} = -v(t) + \sum_j w_j o_j t \\ \begin{cases} o(t) = 1 \quad \& \quad v(t) = v_0, \quad if \quad v(t) \geq v_{th} \\ o(t) = 0, \quad if \quad v(t) < v_{th} \end{cases} \end{cases} \tag{20}$$

where t means the time step, α is a time constant, and v and o express the membrane potential and produce output spike, respectively. o_j is the output spike of the j^{th} pre-neuron. w_j is the synaptic weight between the j^{th} pre-neuron and the current neuron. v^{th} is the considered firing threshold, and v_0 is the reset potential applied after firing a spike. Note that a spike should be fashioned as the Dirac delta function in the continuous-time domain; otherwise, it cannot increase the potential.

The adversarial attacks have been inspected on SNNs. The sensitivity of SNN w.r.t. different encoding types when subjected to white-box adversarial attacks was researched by Bagheri et al. [132]. Applying black-box adversarial attacks to DNNs and SNNs, the comparison showed that the SNNs studied by Marchisio et al. [133] were more robust. Sharmin et al. [134] also suggested a methodology to make the adversarial attack on (non-spiking) DNNs by applying the DNN-to-SNN conversion. The adversarial accuracy of SNNs trained by mixing inference latency and leak factors in leaky integrate-and-fire (LIF) spiking neurons was too analyzed. But, this study did not examine the influence of the membrane voltage threshold along with the time window. Recently, Massa et al. [135] tuned the threshold voltage and the time window to minimize the accuracy loss in the DNN-to-SNN conversion. However, this work did not consider the robustness of adversarial attacks. DIET-SNN [136] was introduced to tailor the membrane threshold and membrane leak to optimize the accuracy and the latency. Liang et al. [137] proposed a gradient-based adversarial attack methodology for SNNs and pointed the impact of the adversarial attack success rate on the loss function and threshold voltage types. The effect of structural parameters, i.e., membrane threshold and time window, was analyzed in work [138].

6.2 Apply SLIT Layer into Adversarial Attacks with Spike Compatible Gradient

Algorithm 1 Algorithm adversarial attack based on inherent structural parameters

Data: Combining SLIT layer and orginal data
Membrane voltage thresholds: $V\_th = V_i/i \in [1, n]$
Spiking time window: $T = T_j/j \in [1, m]$
Noise budget: $\epsilon = \epsilon_k/k \in [1, p]$
SLIT+SNN Architecture: $S_{ij} = SLIT + SNN(V_i, T_j)$
Label test set: $L = (X_t, L_t)$
Accuracy threshold: A_{th}

for $i \leftarrow 1$ to n **do**
  for $j \leftarrow 1$ to m **do**
    Train $S_{ij} = SLIT + SNN(V_i, T_j)$
    if Accuracy $(S_{i,j}) \geq A_{th}$ **then**
      // S_{ij} learns
      for $k \leftarrow 1$ to p **do**
        Adv = 0
        for $X_t \leftarrow 1$ to L do **do**
          // Adversarial attack
          $X_t^* = PGD(S_{ij}, \epsilon_k, X_t)$
          if $S_{ij}(X_t^*) \neq L_t$ **then**
            Adv++
          else
            NOP
          end if
        end for
        $\epsilon_k = 1 - Adv/L$
      end for
    else
      NOP
    end if
  end for
end for

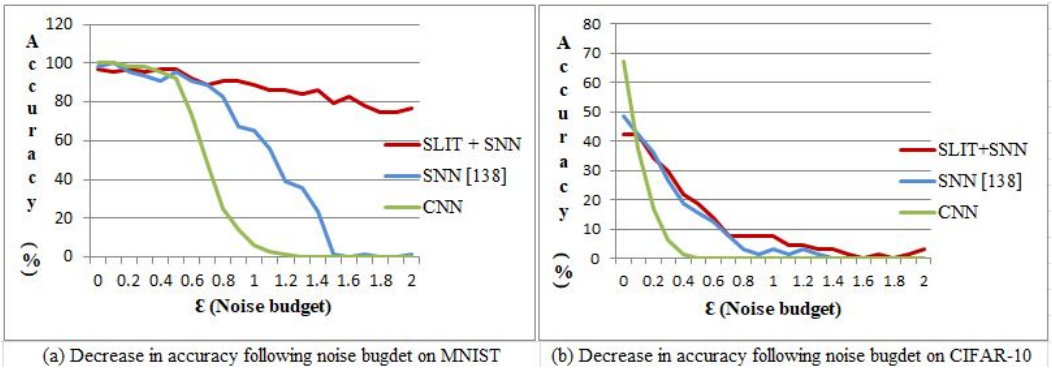
The suggested idea in the reference [138] is used to verify the performance of the SLIT layer on the adversarial attack application. The robustness exploration details in Algorithm 1. The input data is a concatenation of the SLIT layer with original data. The n threshold voltages and m time windows are browsed in Line 1 and 2. When the training stage is launched in Line 3, I treat the SLIT + SNN training analysis for the given combination of spiking threshold voltage V_th and time window boundary T. Being shown in Line 4, the learnability is quantitatively tested by fixing a minimum baseline accuracy level below to consider the SLIT+SNN learning vulnerable diagram. The security study begins from Line 6; it generates adversarial examples with different noise budgets (ϵ) to fool the SNN.

The noise budget models the aggressiveness allowed within the attack generation; the higher the noise budget, the more aggressive the attack is considered. The counter of successful attack generation states is initialized at Line 7. Then, at Line 8, the dataset L is browsed to generate the adversarial attacks. The PGD method at Line 10 is employed to assess the performance. Afterward, the algorithm verifies if the generated example can fool the SNN from Lines (11 - 17). Accordingly, increment the adversarial success counter if the attack forces the output to a wrong label. The robustness is then evaluated for every ϵ value as the attack rate where the adversary failed to generate an adversarial example that fools the victim SNN at Line 17. Therefore, by following the accuracy slope, we can compare the robustness of each topology to adversarial attacks.

The proposed investigations are conducted using the Norse library based on the PyTorch platform with primitives bio-inspired neural components. This library supports training and running SNNs in the spiking domain. The adversarial attacks are performed by employing the Foolbox v3.1.1 [139]. The SNN architecture is adapted from the Lenet-5 architecture to the spiking domain and trained on the MNIST and CIFAR-10 databases. The LIF neuron is used and run the experiments on the Nvidia GeForce GTX 1080. Preprocessing input data before implementing the adversarial attack application is proposed in this research. The input MNIST dataset is modified by concatenating eight channels of the SLIT layer with one original feature. For a complicated case dataset like CIFAR, eight channels of the SLIT layer are combined with three initial channels to produce

the input data. The detailed hyper-parameters are initialed as the work [138].

6.3 Improve Accuracy with SLIT+SNN for Adversarial Attack Application



(a) Decrease in accuracy following noise bugdet on MNIST (b) Decrease in accuracy following noise bugdet on CIFAR-10

Figure 33. Comparison decrease in accuracy on MNIST and CIFAR-10 between SLIT+SNN, SNN and CNN at V_th=0.25, T=80

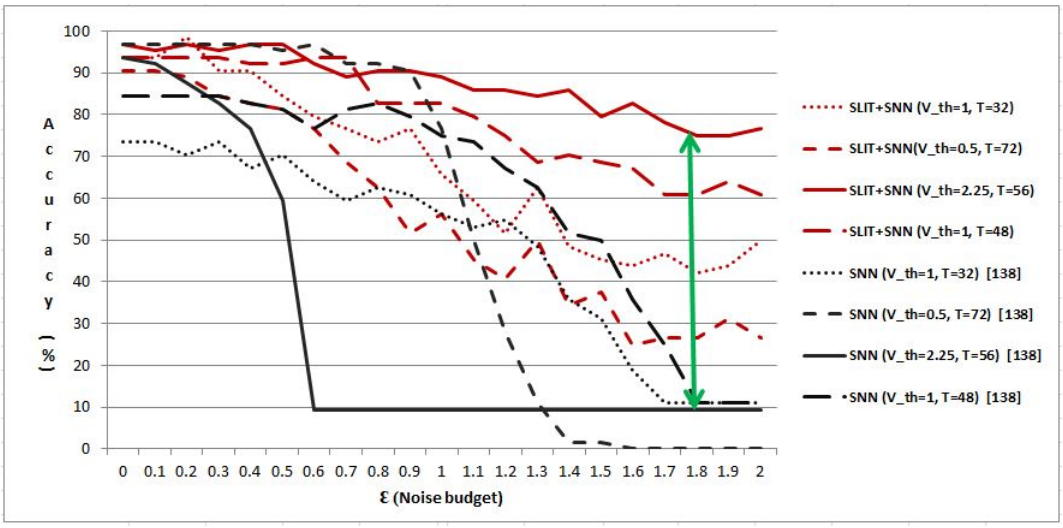


Figure 34. Comparison decrease in accuracy between SLIT+SNN and SNN tested on MNIST with different V_th and T parameters

Figure. 33 have illustrated the accuracy variation w.r.t the noise budget ϵ with the white-box PGD attack on MNIST and CIFAR-10 datasets. In Fig. 33(a), the result reports on the MNIST database that the combination SLIT+SNN keeps the same accuracy as SNN and higher than CNN at a low noise margin ϵ from 0 to 1.0. In the region of $\epsilon > 1$, the slope of SLIT+SNN has just slightly decreased; however, the slope of SNN decreases so fast, and the accuracy of CNN is nearly zero. With complicated datasets like CIFAR-10, Fig. 33(b) also reveal that SLIT+SNN looks better than SNN and CNN at every point of $\epsilon > 1$. Fig. 34 is a comparison of SLIT+SNN and SNN. This figure exhibits the impact of the concatenation of SLIT+SNN in the inherent structural parameters on SNN security. For example, combinations of (V_th, T) = (1.0, 32), (2.25, 56), (1.0, 48) have higher accuracy of 10%, 70%, 20% respectively than what is compared with SNN. With the case (V_th, T) = (0.5, 72), there is a decreased accuracy of around 20% in the SLIT+SNN at the region of ϵ from 0 to 1.2, but an increased accuracy on the proposal is nearly 28% at $\epsilon > 1.2$ when compared with SNN.

7 Conclusion

In this research, the SLIT layer that imitated the primary visual cortex principle and replaced the first layer of conventional CNN has proved efficiency. The backpropagation step of the proposed scheme requires no execution time, while it takes approximately 446.9 seconds in the traditional CNN model on the small network that includes one CONV and one FC layer. Training time is decreased by 40%, 40%, and 32%, respectively, with MNIST, CIFAR, and SVHN databases on Lenet-5 and CNN topologies. It also reduces by about 10% on larger paradigms such as VGG-16 and VGG-19 with the CIFAR database. Accuracy of the proposal has just slightly degraded, for example, a factor of 0.27% on Lenet-5 with MNIST dataset, approximately 1.5% on VGG-16 and VGG-19 with CIFAR dataset, 2.2% on VGG-16 with ImageNet database, and remained the same with the SVHN database. The innovative reconfigurations for the Lenet-5 scheme have achieved a 70% discount in hardware resources and an improvement of 39% latency at a power consumption of 0.456 W for the inference phase on FPGA. The entire convolution operations in the first two convolutional layers of the traditional CNN models are removed efficiently. This architecture is relevant for real-time applications, especially due to a significant reduction in latency. A latency enhancement in the range of 2.6% to 16% has been confirmed on the DPU platform. The SLIT layer is also discovered actively in adversarial attack applications on the third generation network that is plausible for human brain functionality. The accuracy boosts nearly 70% on scheme SLIT+SNN.

Future Work The proposed method is elastic to concatenate with various conventional models at high efficient energy and minimum hardware resources on FPGA. Hence, it gives a new inspiration toward combining our proposal with BinaryConnect or SqueezeNet method to obtain higher hardware design optimization. In the future, I plan to study a more extensive and scalable CNN accelerator that will integrate our scheme with other optimization approaches. I further aim to develop the SLIT layer on a neuromorphic hardware platform. I expect this proposal will resolve the obstacle when utilizing the traditional datasets in a new network topology. I also plan to explore the performance of different functions from the proposed model.

Acknowledgements

Now, at the end of my doctor course program, I feel the duty to spend a few words to show my appreciation to all the good and caring people who have been and still are friendly to me. First and foremost, I would like to express my honest thankfulness to Professor Yasuhiko Nakashima for his ongoing and enthusiastic supervision during the three years of my doctoral course. His kind advice and guidance helped me realize my weaknesses and improve myself to become a better student and researcher. I have been and will continually be looking up to him as a great professor. I would like to thank Professor Yuichi Hayashi, Associate Professor Renyuan Zhang, Visitor Assistant Professor Tran Thi Hong, for their helpful instructions and valuable comments on my research.

Furthermore, I want to acknowledge all the Computing Architecture Lab students for their help and memorable time. Thanks to the Vietnamese friends at NAIST for all the spiritual support and pleasant time together. Besides, I want to express my thanks to all staff members in the Division of Information Science in NAIST. Especially members of the International Student Affairs Section, for all their enthusiastic supports to me. Last but not least, my sincere gratitude to my family, especially my husband and my son, for their encouragement and immense support during my doctor course.

References

[1] G Leuba and R Kraftsik. Changes in volume, surface estimate, three-dimensional shape and total number of neurons of the human primary visual cortex from midgestation until old age. *Anatomy and Embryology*, 190(4):351–366, 1994.

[2] E. Zavitz, M. G.P. Rosa, and N. S.C. Price. Primate visual cortex. *The Curated Reference Collection in Neuroscience and Biobehavioral Psychology*, (December 2015):753–773, 2016.

[3] Jesus L. Lobo, Javier Del Ser, Albert Bifet, and Nikola Kasabov. Spiking neural networks and online learning: An overview and perspectives. *Neural Networks*, 121:88–100, 2020.

[4] Dileep George and Jeff Hawkins. A hierarchical bayesian model of invariant pattern recognition in the visual cortex. *Proceedings of the International Joint Conference on Neural Networks*, 3:1812–1817, 2005.

[5] Mitsuo Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9(6):718–727, 1999.

[6] Hinton, G. E., Osindero, S., and Teh, Y. W. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[7] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. *arXiv*, pages 5918–5926, 2017.

[8] Jinil Chung, Woong Choi, Jongsun Park, and Swaroop Ghosh. Domain wall memory-based design of deep neural network convolutional layers. *IEEE Access*, 8:19783–19798, 2020.

[9] Kai HUANG, Ximeng LIU, Shaojing FU, Deke GUO, and Ming XU. A lightweight privacy-preserving cnn feature extraction framework for mobile sensing. *IEEE Transactions on Dependable and Secure Computing*, 18(3):1441, 2020.

[10] Aidin Ferdowsi, Ursula Challita, and Walid Saad. Deep learning for reliable mobile edge analytics in intelligent transportation systems: An overview. *IEEE Vehicular Technology Magazine*, 14(1):62–70, 2019.

[11] Fasih Ud Din Farrukh, Tuo Xie, Chun Zhang, and Zhihua Wang. Optimization for efficient hardware implementation of CNN on FPGA. *Proceedings of 2018 IEEE International Conference on Integrated Circuits, Technologies and Applications, ICTA 2018*, pages 88–89, 2018.

[12] Sicheng Li, Wei Wen, Yu Wang, Song Han, Yiran Chen, and Hai Helen Li. An FPGA design framework for CNN sparsification and acceleration. *Proceedings - IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2017*, page 28, 2017.

[13] Hidetoshi Ando, Yuki Niitsu, Masaki Hirasawa, Hiroaki Teduka, and Masao Yajima. Improvements of classification accuracy of film defects by using GPU-accelerated image processing and machine learning frameworks. *Proceedings - NICOGRAPH International 2016, NicoInt 2016*, pages 83–87, 2016.

[14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. pages 675–678, 2014.

[15] *A survey of FPGA-based accelerators for convolutional neural networks*, volume 32. Springer London, 2020.

[16] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. Going deeper with embedded FPGA platform for convolutional neural network. *FPGA 2016 - Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 26–35, 2016.

[17] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. *FPGA 2015 - 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170, 2015.

[18] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae Sun Seo. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 45–54, 2017.

[19] Jun Iwamoto, Yuma Kikutani, Renyuan Zhang, and Yasuhiko Nakashima. Daisy chained systolic array and reconfigurable memory space for narrow memory bandwidth. *IEICE Transactions on Information and Systems*, E103D(3):578–589, 2020.

[20] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Third generation neural networks: Spiking neural networks. *Advances in Intelligent and Soft Computing*, 61 AISC:167–178, 2009.

[21] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10(NOV):508, 2016.

[22] Mazdak Fatahi, Mahmood Ahmadi, Mahyar Shahsavari, Arash Ahmadi, and Philippe Devienne. evt_mnist: A spike based version of traditional mnist. *arXiv preprint arXiv:1604.06751*, pages 1–5, 2016.

[23] Pierre Falez. *Improving spiking neural networks trained with spike timing dependent plasticity for image recognition*. Theses, Université de Lille, October 2019.

[24] Maxence Bouvier, Alexandre Valentian, Thomas Mesquida, Francois Rummens, Marina Reyboz, Elisa Vianello, and Edith Beigne. Spiking neural networks hardware implementations and challenges: A survey. *ACM Journal on Emerging Technologies in Computing Systems*, 15(2):1–35, 2019.

[25] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[26] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: a neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

[27] Stephenie C Lemon, Jason Roy, Melissa A Clark, Peter D Friedmann, and William Rakowski. Classification and regression tree analysis in public health: Methodological review and comparison with logistic regression. *Annals of behavioral medicine*, 26(3):172–181, 2003.

[28] Odd O Aalen. A linear regression model for the analysis of life times. *Statistics in medicine*, 8(8):907–925, 1989.

[29] Wenlong Fu, Kaixuan Shao, Jiawen Tan, and Kai Wang. Fault diagnosis for rolling bearings based on composite multiscale fine-sorted dispersion entropy and svm with hybrid mutation sca-hho algorithm optimization. *IEEE Access*, 8:13086–13104, 2020.

[30] Eva Ostertagová. Modelling using polynomial regression. *Procedia Engineering*, 48:500–506, 2012.

[31] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[32] Shruti R Kulkarni and Bipin Rajendran. Spiking neural networks for handwritten digit recognition supervised learning and network optimization. *Neural Networks*, 103:118–127, 2018.

[33] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[34] Diem Tran, Thi To, Thuan Huynh, and Phuong Nguyen. Designing a harware accelerator for face recognition using vector quantization and principal

component analysis as a component of sopc. In *2010 Fifth IEEE International Symposium on Electronic Design, Test & Applications*, pages 82–86. IEEE, 2010.

[35] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, pages 1–7, 2017.

[36] Yongliang Yang, Kyriakos G Vamvoudakis, and Hamidreza Modares. Safe reinforcement learning for dynamical games. *International Journal of Robust and Nonlinear Control*, 30(9):3706–3726, 2020.

[37] Frank L Lewis and Draguna Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE circuits and systems magazine*, 9(3):32–50, 2009.

[38] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. *Advances in neural information processing systems*, 20:833–840, 2007.

[39] Jure Zupan. Introduction to artificial neural network (ann) methods: what they are and how to use them. *Acta Chimica Slovenica*, 41:327–327, 1994.

[40] George N Reeke Jr and Olaf Sporns. Behaviorally based modeling and computational approaches to neuroscience. *Annual Review of Neuroscience*, 16(1):597–623, 1993.

[41] Conrad D James, James B Aimone, Nadine E Miner, Craig M Vineyard, Fredrick H Rothganger, Kristofor D Carlson, Samuel A Mulder, Timothy J Draelos, Aleksandra Faust, Matthew J Marinella, et al. A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic computing applications. *Biologically Inspired Cognitive Architectures*, 19:49–64, 2017.

[42] David Daniel Cox and Thomas Dean. Neural networks and neuroscience-inspired computer vision. *Current Biology*, 24(18):R921–R929, 2014.

[43] Matteo Carandini, Jonathan B Demb, Valerio Mante, David J Tolhurst, Yang Dan, Bruno A Olshausen, Jack L Gallant, and Nicole C Rust. Do we know what the early visual system does? *Journal of Neuroscience*, 25(46):10577–10597, 2005.

[44] Thomas Serre. Hierarchical models of the visual system. *Encyclopedia of computational neuroscience*, 6:1–12, 2014.

[45] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the national academy of sciences*, 111(23):8619–8624, 2014.

[46] MP Uddin, MA Mamun, and MA Hossain. Feature extraction for hyperspectral image classification. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 379–382. IEEE, 2017.

[47] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

[48] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[49] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size. pages 1–13, 2016.

[50] Muluken Hailesellasie, Syed Rafay Hasan, Faiq Khalid, Falah Aw Wad, and Muhammad Shafique. Fpga-based convolutional neural network architecture with reduced parameter requirements. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[51] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for ef-

ficient evaluation. *Advances in Neural Information Processing Systems*, 2(January):1269–1277, 2014.

[52] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A framework for fast, scalable binarized neural network inference. *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74, 2017.

[53] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. Accelerating binarized convolutional neural networks with software-programmable FPGAs. *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 15–24, 2017.

[54] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: imagenet classification using binary convolutional neural networks. pages 525–542, 2016.

[55] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Sérot, and François Berry. Accelerating CNN inference on FPGAs: A survey. *arXiv*, (January), 2018.

[56] Jason Cong and Bingjun Xiao. Minimizing computation in convolutional neural networks. In *International conference on artificial neural networks*, pages 281–290. Springer, 2014.

[57] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae Sun Seo, and Yu Cao. Throughput-optimized openCL-based FPGA accelerator for large-scale convolutional neural networks. *FPGA 2016 - Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 16–25, 2016.

[58] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. pages 1–6, 2006.

[59] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, and Gordon R. Chiu. An OpenCLTM deep learning accelerator on Arria 10. *arXiv*, pages 55–64, 2017.

[60] Roberto Di Cecco, Griffin Lacey, Jasmina Vasiljevic, Paul Chow, Graham Taylor, and Shawki Areibi. Caffeinated FPGAs: FPGA framework for convolutional neural networks. *Proceedings of the 2016 International Conference on Field-Programmable Technology, FPT 2016*, pages 265–268, 2017.

[61] Chi Zhang and Viktor Prasanna. Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system. *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 35–44, 2017.

[62] Jong Hwan Ko, Burhan Mudassar, Taesik Na, and Saibal Mukhopadhyay. Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation. *Proceedings - Design Automation Conference*, Part 12828, 2017.

[63] Stylianos I. Venieris and Christos Savvas Bouganis. FpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. *Proceedings - 24th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2016*, pages 40–47, 2016.

[64] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. From high-level deep neural models to fpgas. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[65] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. A high performance fpga-based accelerator for large-scale convolutional neural networks. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–9. IEEE, 2016.

[66] Giuseppe Natale, Marco Bacis, and Marco Domenico Santambrogio. On how to design dataflow FPGA-based accelerators for convolutional neural networks. *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, 2017-July:639–644, 2017.

[67] K. Abdelouahab, M. Pelcat, J. Serot, C. Bourrasset, and F. Berry. Tactics to directly map CNN graphs on embedded FPGAs. *IEEE Embedded Systems Letters*, 9(4):113–116, 2017.

[68] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. Design space exploration of FPGA-based deep convolutional neural networks. *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, 25-28-Janu:575–580, 2016.

[69] Paolo Meloni, Gianfranco Deriu, Francesco Conti, Igor Loi, Luigi Raffo, and Luca Benini. Curbing the roofline: A scalable and flexible architecture for CNNs on FPGA. *2016 ACM International Conference on Computing Frontiers - Proceedings*, pages 376–383, 2016.

[70] Mohammad Motamedi, Philipp Gysel, and Soheil Ghiasi. PLACID: A platform for FPGA-based accelerator creation for DCNNs. *ACM Transactions on Multimedia Computing, Communications and Applications*, 13(4):1–21, 2017.

[71] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. *Proceedings - Design Automation Conference*, Part 12828:1–6, 2017.

[72] Murugan Sankaradas, Venkata Jakkula, Srihari Cadambi, Srimat Chakradhar, Igor Durdanovic, Eric Cosatto, and Hans Peter Graf. A massively parallel coprocessor for convolutional neural networks. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 53–60. IEEE, 2009.

[73] Clément Farabet, Cyril Poulet, Jefferson Y Han, and Yann LeCun. Cnp: An fpga-based processor for convolutional networks. In *2009 International Conference on Field Programmable Logic and Applications*, pages 32–37. IEEE, 2009.

[74] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural

networks. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 247–257, 2010.

[75] Clément Farabet, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. Neuflow: A runtime reconfigurable dataflow processor for vision. In *CVPR 2011 workshops*, pages 109–116. IEEE, 2011.

[76] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello. A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 682–687, 2014.

[77] Atul Rahman, Jongeun Lee, and Kiyoung Choi. Efficient fpga acceleration of convolutional neural networks using logical-3d compute array. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1393–1398. IEEE, 2016.

[78] Yufei Ma, Naveen Suda, Yu Cao, Jae-sun Seo, and Sarma Vrudhula. Scalable and modularized rtl compilation of convolutional neural networks onto fpga. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2016.

[79] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer cnn accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[80] Jack B Dennis and David P Misunas. A preliminary architecture for a basic data-flow processor. In *Proceedings of the 2nd annual symposium on Computer architecture*, pages 126–132, 1974.

[81] Lin Li, Tiziana Fanni, Timo Viitanen, Renjie Xie, Francesca Palumbo, Luigi Raffo, Heikki Huttunen, Jarmo Takala, and Shuvra S Bhattacharyya. Low power design methodology for signal processing systems using lightweight dataflow techniques. In *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 82–89. IEEE, 2016.

[82] Chung-Ching Shen, William Plishker, Hsiang-Huang Wu, and Shuvra S Bhattacharyya. A lightweight dataflow approach for design and implementation of sdr systems. In *Proceedings of the Wireless Innovation Conference and Product Exposition*, pages 640–645. Citeseer, 2010.

[83] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, (2015):1–17, 2017.

[84] Tomoya Fujii, Simpei Sato, Hiroki Nakahara, and Masato Motomura. An FPGA realization of a deep convolutional neural network using a threshold neuron pruning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10216 LNCS:268–280, 2017.

[85] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. *32nd International Conference on Machine Learning, ICML 2015*, 3:1737–1746, 2015.

[86] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. 1(1):1–13, 2016.

[87] Matthieu Courbariaux, Jean Pierre David, and Yoshua Bengio. Training deep neural networks with low precision multiplications. *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, (Section 5):1–10, 2015.

[88] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi. Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1604.03168*, pages 1–8, 2016.

[89] Matthieu Courbariaux, Yoshua Bengio, and Jean Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in Neural Information Processing Systems*, 2015-Janua:3123–3131, 2015.

[90] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights. *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, 2016-Septe:236–241, 2016.

[91] Rui Zhao, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. Saliency detection by multi-context deep learning. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:1265–1274, 2015.

[92] Hyeonuk Sim and Jongeun Lee. A new stochastic computing multiplier with application to deep convolutional neural networks. *Proceedings - Design Automation Conference*, Part 12828:1–6, 2017.

[93] Vincent T Lee, Armin Alaghi, John P Hayes, Visvesh Sathe, and Luis Ceze. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. pages 13–18, 2017.

[94] Sebastian Vogel, Christoph Schorn, Andre Guntoro, and Gerd Ascheid. Efficient stochastic inference of bitwise deep neural networks. (Nips):1–6, 2016.

[95] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. *Proceedings - Design Automation Conference*, 05-09-June(1):1–6, 2016.

[96] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. *ACM SIGPLAN Notices*, 52(4):405–418, 2017.

[97] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1131–1135. IEEE, 2015.

[98] Arash Ardakani, François Leduc-Primeau, Naoya Onizawa, Takahiro Hanyu, Senior Member, Warren J Gross, and Senior Member. VLSI implementation of deep neural network. *IEEE Transaction on very large scale integration systems*, 25(10):2688–2699, 2017.

[99] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, pages 2849–2858. PMLR, 2016.

[100] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[101] Shu-Chang Zhou, Yu-Zhi Wang, He Wen, Qin-Yao He, and Yu-Heng Zou. Balanced quantization: An effective and efficient approach to quantized neural networks. *Journal of Computer Science and Technology*, 32(4):667–682, 2017.

[102] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[103] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 806–814, 2015.

[104] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, pages 1–9, 2015.

[105] Sina Ghaffari and Saeed Sharifian. Fpga-based convolutional neural network accelerator design using high level synthesize. In *2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)*, pages 1–6. IEEE, 2016.

[106] Gan Feng, Zuyi Hu, Song Chen, and Feng Wu. Energy-efficient and high-throughput fpga-based accelerator for convolutional neural networks. In *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 624–626. IEEE, 2016.

[107] Dai Rongshi and Tang Yongming. Accelerator implementation of lenet-5 convolution neural network based on fpga with hls. In *2019 3rd International Conference on Circuits, System and Simulation (ICCSS)*, pages 64–67. IEEE, 2019.

[108] Mengxing Zhao, Xiang Li, Shunyi Zhu, and Li Zhou. A method for accelerating convolutional neural networks based on fpga. In *2019 4th International Conference on Communication and Information Systems (ICCIS)*, pages 241–246. IEEE, 2019.

[109] Jialiang Zhang and Jing Li. Improving the performance of opencl-based fpga accelerator for convolutional neural network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 25–34, 2017.

[110] Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A comprehensive performance comparison of cuda and opencl. In *2011 International Conference on Parallel Processing*, pages 216–225. IEEE, 2011.

[111] François Serre and Markus Püschel. A dsl-based fft hardware generator in scala. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 315–3157. IEEE, 2018.

[112] Jing Pu, Steven Bell, Xuan Yang, Jeff Setter, Stephen Richardson, Jonathan Ragan-Kelley, and Mark Horowitz. Programming heterogeneous systems from an image processing dsl. *ACM Transactions on Architecture and Code Optimization (TACO)*, 14(3):1–25, 2017.

[113] Hui Wei and Hu Li. Shape description and recognition method inspired by the primary visual cortex. *Cognitive Computation*, 6(2):164–174, 2014.

[114] Raad H.Thaher and Zaid K. Hussein. Stereo vision distance estimation employing SAD with canny edge detector. *International Journal of Computer Applications*, 107(3):38–43, 2014.

[115] Paulo A. S. Mendes and A. Paulo Coimbra. Movement detection and moving object distinction based on optical flow for a surveillance system. *Transactions on Engineering Technologies*, 0958:143–158, 2021.

[116] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

[117] By Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2012.

[118] Guangyong Zeng, Yi He, Zongxue Yu, Xi Yang, Ranran Yang, and Lei Zhang. Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 91(8):2322–2330, 2016.

[119] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[120] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, pages 1–9, 2011.

[121] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, pages 32–33, 2009.

[122] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bern-

stein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[123] Jojo Moolayil, Jojo Moolayil, and Suresh John. *Learn keras for deep neural networks*. Springer, 2019.

[124] Vishnu Subramanian. *Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch*. Packt Publishing Ltd, 2018.

[125] Tsung Han Tsai, Yuan Chen Ho, and Ming Hwa Sheu. Implementation of FPGA-based accelerator for deep neural networks. *Proceedings - 2019 22nd International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2019*, 7:73–76, 2019.

[126] Giacinto Paolo Saggese, Antonino Mazzeo, Nicola Mazzocca, and Antonio GM Strollo. An fpga-based performance analysis of the unrolling, tiling, and pipelining of the aes algorithm. In *International Conference on Field Programmable Logic and Applications*, pages 292–302. Springer, 2003.

[127] Lin Li, Shaoyu Lin, Shuli Shen, Kongcheng Wu, Xiaochao Li, and Yihui Chen. High-throughput and area-efficient fully-pipelined hashing cores using bram in fpga. *Microprocessors and Microsystems*, 67:82–92, 2019.

[128] Xilinx. Vitis AI user guide. 1414:1–157, 2019.

[129] Zynq Dpu. Zynq DPU v3.2. 338:1–57, 2020.

[130] Sangya Dutta, Vinay Kumar, Aditya Shukla, Nihar R Mohapatra, and Udayan Ganguly. Leaky integrate and fire neuron by charge-discharge dynamics in floating-body mosfet. *Scientific reports*, 7(1):1–7, 2017.

[131] Dibyendu Chatterjee and Anil Kottantharayil. A cmos compatible bulk finfet-based ultra low energy leaky integrate and fire neuron for spiking neural networks. *IEEE Electron Device Letters*, 40(8):1301–1304, 2019.

[132] Alireza Bagheri, Osvaldo Simeone, and Bipin Rajendran. Training probabilistic spiking neural networks with first-to-spike decoding. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2986–2990. IEEE, 2018.

[133] Alberto Marchisio, Giorgio Nanfa, Faiq Khalid, Muhammad Abdullah Hanif, Maurizio Martina, and Muhammad Shafique. Is spiking secure? a comparative study on the security vulnerabilities of spiking and deep neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[134] Saima Sharmin, Priyadarshini Panda, Syed Shakib Sarwar, Chankyu Lee, Wachirawit Ponghiran, and Kaushik Roy. A comprehensive analysis on adversarial robustness of spiking neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[135] Riccardo Massa, Alberto Marchisio, Maurizio Martina, and Muhammad Shafique. An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor. In *2020 International Joint Conference on Neural Networks, IJCNN 2020*, page 9207109. Institute of Electrical and Electronics Engineers Inc., 2020.

[136] Nitin Rathi and Kaushik Roy. Diet-snn: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv preprint arXiv:2008.03658*, pages 1–11, 2020.

[137] Ling Liang, Xing Hu, Lei Deng, Yujie Wu, Guoqi Li, Yufei Ding, Peng Li, and Yuan Xie. Exploring adversarial attack in spiking neural networks with spike-compatible gradient. *arXiv preprint arXiv:2001.01587*, pages 1–18, 2020.

[138] Rida El-Allami, Alberto Marchisio, Muhammad Shafique, and Ihsen Alouani. Securing deep spiking neural networks against adversarial attacks through inherent structural parameters. *arXiv preprint arXiv:2012.05321*, pages 1–6, 2020.

[139] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox v0. 8.0: A python toolbox to benchmark the robustness of machine learning models. corr abs/1707.04131 (2017). *arXiv preprint arXiv:1707.04131*, pages 1–6, 2017.

List of Publication

1. TRAN, Thi Diem; KIMURA, Mutsumi; NAKASHIMA, Yasuhiko. Primary Visual Cortex Inspired Feature Extraction Hardware Model. In: 2020 4th International Conference on Recent Advances in Signal Processing, Telecommunications & Computing (SigTelCom). IEEE, p. 20-24, 2020. [Correspond to Chapter 3 and 4]

2. TRAN, Thi Diem; NAKASHIMA, Yasuhiko. SLIT: An Energy-Efficient Reconfigurable Hardware Architecture for Deep Convolutional Neural Networks. IEICE Transactions on Electronics, p. 319-329, 2020. [Correspond to Chapter 3, 4 and 5]

3. TRAN, Thi Diem; NAKASHIMA, Yasuhiko. Exploring Versatility of Primary Visual Cortex Inspired Feature Extraction Hardware Model through Various Network Architectures. In: 2021 4th International Conference on Computing, Electronics & Communications Engineering (iCCECE2021). IEEE, p. 53-58, 2021. [Correspond to Chapter 5 and 6]