

Feasibility and Design Trade-offs of Neural Network Accelerators Implemented on Reconfigurable Hardware

Quang-Kien Trinh^{1*}, Quang-Manh Duong^{1**}, Van-Thanh Nguyen², Thi-Nga Dao¹,
and Hong-Phong Nguyen¹

¹ Le Quy Don Technical University, Hanoi, Vietnam

² Posts and Telecommunications Institute of Technology

kien.trinh@lqdtu.edu.vn*, Corresponding author: manhdq@lqdtu.edu.vn**

Abstract. In recent years, neural networks based algorithms have been widely applied in computer vision applications. FPGA technology emerges as a promising choice for hardware acceleration owing to high-performance and flexibility; energy-efficiency compared to CPU and GPU; fast development round. FPGA recently has gradually become a viable alternative to the GPU/CPU platform.

This work conducts a study on the practical implementation of neural network accelerators based-on reconfigurable hardware (FPGA). This systematically analyzes utilization-accuracy-performance trade-offs in the hardware implementations of neural networks using FPGAs and discusses the feasibility of applying those designs in reality.

We have developed a highly generic architecture for implementing a single neural network layer, which eventually permits further construct arbitrary networks. As a case study, we implemented a neural network accelerator on FPGA for MNIST and CIFAR-10 dataset. The major results indicate that the hardware design outperforms by at least 1500 times when the parallel coefficient p is 1 and maybe faster up to 20,000 times when that is 16 compared to the implementation on the software while the accuracy degradations in all cases are negligible, i.e., about 0.1% lower. Regarding resource utilization, modern FPGA undoubtedly can accommodate those designs, e.g., 2-layer design with p equals 4 for MNIST and CIFAR occupied 26% and 32% of LUT on Kintex-7 XC7K325T respectively.

Keywords: Neural Network, FPGA accelerator, Data recognition.

1 Introduction

The development of the Neural Network is the motivation to improve computing capability on different platforms

In recent years, researches on the neural network have shown a significant advantage in machine learning over traditional algorithms based on handcrafted models. There has been a growing interest in the study of neural networks, inspired by the nervous system in the human brain. Owing to the high accuracy and good performance, neural

networks have been widely adopted in many applications such as image classification [1], face recognition [2], smart digital video surveillance [3], and speech recognition [4], etc. In general, neural network features a high fitting ability to a wide range of pattern recognition problems, which makes the neural network a promising candidate for many artificial intelligence applications.

Recent research on the neural network is showing great improvement over traditional algorithms, various neural network models, like Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), have been proposed. CNN [5] improves the Top-5 image classification accuracy on ImageNet [6] dataset from 73.8% to 84.7% in 2012 and further helps improve object detection [7] with its outstanding ability in feature extraction. RNN [8] achieves state-of-the-art word error rate on speech recognition.

As neural network models become larger and deeper, numerous operations and data accesses are demanded in neural network-based implementations while higher accuracy typically demands more complex models. For example, Krizhevsky et al. [9] achieved 84.7% Top-5 accuracy in Take ImageNet Large-Scale Vision Recognition Challenge (ILSVRC) with a model including 5 convolution layers and 3 fully-connected layers, they get a recognition accuracy [10] of 95.1% surpassing human-level classification (94.9% [11]) with a 22-layer model and won the ILSVRC-2015 competition for achieving an accuracy of 96.4% with a model depth of 152 [12]. Such a model can take over 11.3 billion floating-point operations (GFLOPs) for the inference procedure, and even more for training.

The operations in CNNs are computationally intensive with over billion operations per input image [13], thus requiring high-performance hardware platforms. The rapidly changing field of deep learning makes it even more difficult for a generic accelerator to match for a wide range of neural network algorithms. In this context, there is a timely need to reform the mapping strategy of neural networks to the hardware platform and to support modular and scalable hardware customization for specific applications without losing design flexibility. Choosing an appropriate computing platform for neural network-based applications is extremely essential.

FPGA, GPU, and ASIC are the widely-applied selections in addition to using the traditional CPU usage for accelerators available in the market today. For FPGAs, recently there have been major efforts from technology leaders to better integrate FPGA accelerators. There is also a growing number of GPU and ASIC accelerator solutions offered commercially, such as NVIDIA GPU and IBM PowerEN processor with edge network accelerators.

Application-Specific Standard Processor (ASSP) based approaches for Neural Network accelerators.

Neural networks are implemented on CPU and GPU platforms, i.e., currently widely adopted ASSPs; however, they are not efficient either in terms of implementation speed (CPU) or energy consumption (GPU) [14]. Indeed, a typical CNN architecture has multiple convolutional layers that extract features from the input data, followed by classification layers. This essentially requires massive parallel calculations. General-purpose processors (CPUs) rely on a few processing elements

and operate sequentially, hence they are not efficient for CNN implementation and can hardly meet the performance requirement. In contrast to CPU, GPUs can offer Giga to Tetra FLOPs [15] per second's computing speed due to their single-instruction-multiple-data (SIMD) architecture and high clock frequency, therefore they are good choices for high-performance neural network applications. However, the power consumption of typical GPUs is exceedingly high - for an NVIDIA Tesla K40 GPU, the thermal design power (TDP) is 235 W [16], thus GPUs are not suitable for embedded systems such as mobile devices, robots, etc., which are mostly powered by batteries and low power consumption becomes essential to them. Besides, both CPU and GPU have the disadvantage of poor integration capability, neither the CPU nor the GPU is specifically designed for neural network calculations so they are not optimized for neural networks, resulting in poor energy efficiency, especially in the real-time applications that require large bandwidth.

Application-Specific IC (ASIC), which is rigorously optimized for neural networks, could solve both poor performance and high energy consumption of CPU and GPU [17]. This hardware solution undoubtedly is superior to any other platform when performing calculations on the same neural network. Nonetheless, the ASIC design cycle is relatively long due to high complexity and very costly for low volume production. More important, ASIC is non-hardware-reconfigurable technology, thus, no single ASIC platform could meet the rapid improvements and the diversity of problems on the neural network application. Therefore, the implementation of ASIC for neural network accelerators, in reality, needs to be carefully considered.

Reconfigurable hardware-based approaches for Neural Network accelerators

As a balancing approach among the mentioned ASSP platforms and ASIC, along with distinct features, FPGAs present as promising platforms for the hardware acceleration of CNNs [18]. FPGA-based neural network accelerators have become increasingly popular thanks to their high reconfigurability, fast turn-around time (compared to ASICs), high-performance, and better energy efficiency (compared to GPUs) [19].

In a particular study, Marco Bettoni *et al.* [20] implemented a CNN design on FPGA and obtained results showing that the proposed implementation is as efficient as a general-purpose 16-core CPU, and almost 15 times faster than an SoC GPU for mobile application. Research by Eriko Nurvitadhi *et al.* [21, 22] implemented on BNN and RNN networks showed that in comparison to 14-nm ASIC, GPU, and multi-core CPU, FPGA provides superior performance/watt over CPU and GPU because FPGA's on-chip BRAMs, hard DSPs, and reconfigurable fabric allow for efficiently extracting fine-grained parallelisms. Moreover, newer FPGAs with more DSPs, on-chip BRAMs, integrated hard accelerators IP cores, and higher frequency have the potential to narrow the FPGA-ASIC efficiency gap.

Nonetheless, those prior works are targeted at either high accuracy or high performance for specific architecture [20-22] and ignore the intrinsic trade-offs between resource utilization, performance, accuracy, and network architecture. Therefore, the scalability and the integrability of the neural network design has not fully explored and studied.

In this work, we developed fundamental and highly generic building blocks that allow constructing virtually any neural networks. These base components allow us to systematically study the feasibility of using FPGAs as an accelerator for neural network applications. The design trade-offs on aspects including network architecture, resource utilization, accuracy, and performance for a wide range of devices to understand the real power and limitation of the FPGAs as the reconfigurable platform for neural network implementation. These assessments will be the basis for the application of FPGAs as hardware accelerators for practical neural network applications.

The main contributions of this work are summarized as follows

- A high-performance generic design of neural network accelerator combining software (for parameters training) with the powerful capability of hardware computation (on matrix additions and multiplications). In particular, we analyze the design by theoretically deriving performance metrics including the memory size and processing latency of the FPGA-based neural network accelerator. To support the design analysis, a numerical format selection method based on trained parameter values domain on two considered datasets.
- A methodology is proposed on how to optimize parallelism strategy with different parallel coefficients for each layer to achieve high throughput.
- An in-depth discussion on the design tradeoffs between resource utilization, model accuracy, and performance of the image classification models with different parameters including numerical formats, parallel coefficients, and network architectures through the practical accelerators (for the most representative datasets (MNIST and CIFAR-10)).
- On-board demonstrations of FPGA implementation using single or multilayer neural networks and CNN that achieve mostly the same accuracy as the software implementation.

The remaining of this paper is organized as follows: Section II introduces the basic background of neural networks. Section III presents the results of data recognition performed on the software. Section IV proposes a generic design for the data recognition problem on the hardware and describes our FPGA-based implementation details upon this proposed design. Section V concludes the paper.

2 Background

2.1 Neural Network

Generally, a fully-connected neural network consists of three consecutive layers: input, hidden, and output layers as shown in Fig.1. First, input features (e.g., image data) are collected and fed into the input layer. Then, input features are fully connected to hidden layers that learn the underlying patterns of input data. Finally, hidden features are progressively propagated to the output layer which provides

predicted discrete labels (for classification models) or continuous values (for regression models).

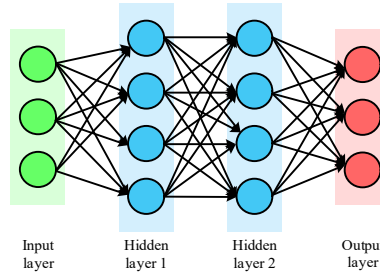


Fig. 1. The basic structure of the neural networks.

This work considers the case study of image recognition tasks on MNIST [23] and CIFAR-10 datasets [24]. In the case of the MNIST set, we aim to construct a neural network-based classifier that can understand the handwritten digits. Specifically, the classification model should output the most likely digit among 10 possible single digits with a given input image of 28x28 pixels.

However, with large and high-resolution image inputs, the fully-connected neural networks suffer from a complex network architecture, which requires a large memory size to store training parameters and high-performance computing units. Therefore, a more effective network architecture should be designed to overcome the drawback of fully-connected neural networks, and convolutional neural networks (CNN) were invented for achieving superhuman performance on complex visual tasks.

2.2 Convolutional Neural Network

Emerged from the study of the brain's visual cortex, CNNs have been widely applied in image recognition. Typically, CNNs are composed of three types of layers: convolutional layers, pooling layers, and fully-connected layers. Multiple convolutional and pooling layers are stacked one after another followed by a series of fully-connected layers. Each neuron in the convolutional layer corresponds to learn a specific pattern of a limited area by only connecting to features related to that area. The pooling layer then simply performs downsampling on activation units of the previous convolutional layer for further reducing the number of training parameters. Finally, the fully-connected layers conduct the same duties found in traditional neural networks and produce class scores from the extracted features provided by the convolutional and pooling layers.

A CNN model consists of two components: the feature extraction part and the classification part. The convolution layers and pooling layers perform feature extraction. For example, given an image, the convolution layer detects features such as two eyes, long ears, four legs, a short tail, and so on. The fully connected layers then act as a classifier on top of these features and assign a probability for the input image being a dog.

In our study, the popular CIFAR-10 dataset was selected as the case study to evaluate the implementation of the image classification task on the hardware, using the CNNs. The 32x32 pixel RGB images in the CIFAR-10 dataset are sent to the feature extraction layer to filter out the most basic characteristics of the object.

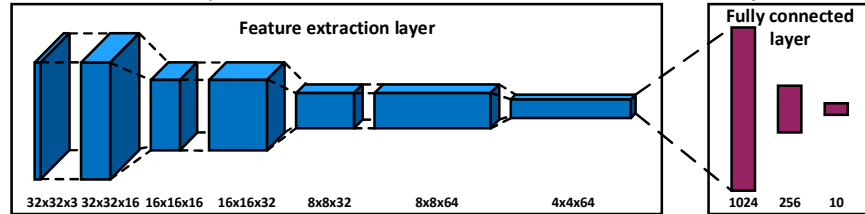


Fig. 2. Block diagram of the image recognition model on the CIFAR-10 dataset.

As shown in Fig. 2, the featured extraction layer consists of 7 component layers and the output of the feature extraction layer will be transformed into a one-dimensional vector, which will be the input of the fully connected layer. This input through a multi-layer perceptual algorithm is used to calculate the probability and draw conclusions: the input data belongs to which of the 10 labels of the CIFAR-10 dataset.

3 Performance Evaluation of Neural Network-based Classifier on a Software Tool

Although this work will eventually focus on hardware implementation, implementing the neural network-based classification model on a software platform is needed for the parameter learning process and architectural optimization. Parameter training should be conducted using a software tool since this phase is generally performed only once using the offline training data, we can perform parameter training on any powerful computing units. Also, this process runs highly sophisticated learning algorithms and complex activation functions that are not efficient for implementation on the FPGA. Then, the inference phase, which requires much less computational resources, can be conducted on the FPGA board for the real-time data. Therefore, to compare the neural network-based classifier performance between the software platform and an FPGA board, this section constructs an optimal neural network-based image classification model and evaluates the neural network-based classifier performance on a software platform using a variety of network hyper-parameters.

3.1 Software platform

There are many software-based approaches for modeling the neural network. Among those, Python is the most popular and widely-used programming language for evaluating neural network-based models. Most data scientists and machine learning developers (57% [25]) are currently using a variety of Python-based libraries such as

TensorFlow, Keras, Theano, Scikit-learn, PyTorch [26]. In this work, we conduct modeling neural networks using the programming language Python and TensorFlow library running on the Ubuntu 16.04 64 bit operating system (Intel Core i5 5200U, RAM 12GB, SSD 256GB) as a software platform to conduct model training and evaluation of the entire image recognition result.

3.2 Analysis of the Recognition Accuracy

Using the above-mentioned software platform, we first conduct the study on the impacts of design parameters on the accuracy of the model. We consider the following four cases to calculate the accuracy: 1 layer, 2 layers, 3 layers, and 4 layers. In these cases, the parameters to be adjusted include the learning rate, epoch, and batch size. The learning rate shows the degree of adjustment of the weight matrix value W after each learning to reduce the value of the loss function. The greater the learning rate, the faster the loss function decreases. Epoch is the number of times a model is learned in the training session. Batch size is the amount of data to be included in a training session. The image recognition results on the two sets of MNIST and CIFAR-10 databases with the presented software settings are shown in the following figure.

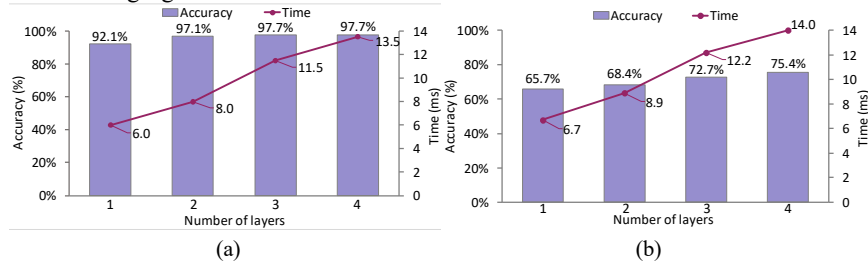


Fig. 3. Software-based recognition accuracy for the (a) MNIST dataset and the (b) CIFAR-10 dataset corresponding to different numbers of layers.

Based on the results obtained on the graph, it can be seen that image recognition accuracy in the MNIST dataset is relatively high (at least 92.1%) compared to object classification accuracy in the CIFAR-10 dataset (maximum up to 75.4%). The higher prediction accuracy on the MNIST dataset than CIFAR-10 is expected since CIFAR-10 images are undoubtedly more complex than MNIST ones. Also, the average time to recognize MNIST images is relatively lower (9.75ms) compared to CIFAR-10 images (10.45ms). This can be explained by the neural network structure for CIFAR-10 image recognition much larger than that of MNIST, therefore the average inference time for CIFAR-10 should be longer than that for MNIST pictures.

3.3 Analysis of the Number Format

Most software tools for machine learning techniques by default support floating-point arithmetic operations and floating-point training parameters, which achieves mostly

absolute calculation accuracy. However, implementing floating-point arithmetic operations on hardware is inherently complicated and area-inefficient. Therefore, we need to look for an alternative way to implement a neural network-based classification model on hardware. First, we analyze the range domain of input data and weight matrix elements values extracted from the software implementation. Then, an appropriate number format for parameters and unit values of the neural network is selected. For both MNIST and CIFAR-10 datasets, as shown in Fig.4, the weight matrix elements values are fundamentally concentrated in the range $(-0.25 \div 0.25)$.

Based on the statistical analysis, an 8-bit fixed-point for numerical representation, accuracy can be up to 2^{-6} (i.e., using 6 bits for representing the fraction) could be enough for representing the value domain of the parameters we have calculated. Compared to the floating-point number (single precision), an 8-bit fixed-point number would drastically reduce the design complexity and resource usage. Nonetheless, to evaluate the proposed numerical domain selection and the impacts of the number format on recognition accuracy, we still need to conduct the performance assessment of the neural network on the actual hardware design. This will be presented and discussed in Section 5.2.

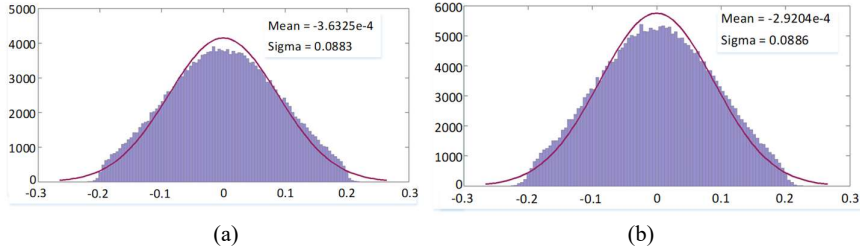


Fig. 4. The weight matrix value domain of the (a) MNIST and the (b) CIFAR-10 image recognition model.

4 Design of Neural Network Accelerator on FPGA

In this section, we first introduce a standard and fully-parameterized hardware architecture design for a single neural network layer as shown in Fig. 5. Then, this fundamental design can be used to construct the whole complex neural network with an arbitrary number of hidden layers. The generic hyperparameters of a single layer are n_i and n_{i+1} , where n_i is the number of input units and n_{i+1} is the number of output units at layer i (or the number of input units at the layer $i+1$), respectively. For each layer, there are multiple processing units including multiplier-accumulator (MAC), adder, and activation and memory buffer for input, output, and training parameters. Note that the higher number of hidden layers results in more resource utilization on the hardware. To reduce the computational complexity of hardware architecture, the training parameters including weights and biases matrices are learned using the neural network software tool and are restored in the memory of the hardware. The detailed design of these processing elements and memory buffer is presented in the following subsections.

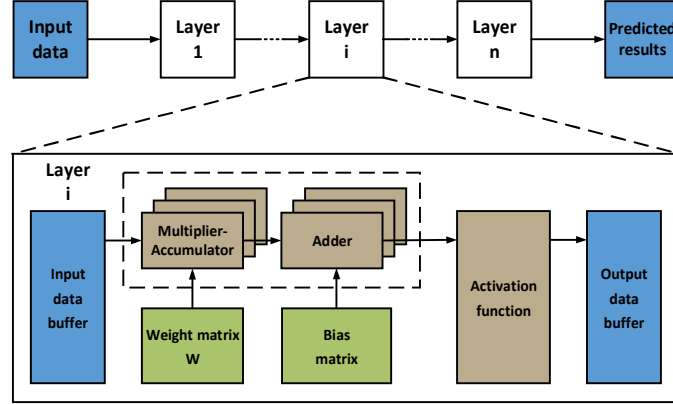


Fig. 5. Block diagram of neural network accelerator implemented on reconfigurable hardware.

4.1 Design of processing units

Multiplier-Accumulators (MACs)

Assume that each MAC unit corresponds to multiplication between two binary numbers: an input feature and a weight value. During a clock cycle, an input feature and a column of weight matrix are multiplied in parallel using n_{i+1} MACs. To complete multiplication between the input vector and the weight matrix, n_i clock cycles corresponding to n_i input features are required.

Taking inputs from the input data buffer and the weight matrix, MACs are the main processing element used to perform multiplication between input features X of $1 \times n_i$ and the weighted matrix W of $n_i \times n_{i+1}$. The MAC output, called vector c_k , is calculated as below.

$$c_k = \sum_{j=1}^{n_i} x_j w_{jk}, 1 \leq k \leq n_{i+1} \quad (1)$$

Where c_k is the k^{th} element of the output vector, x_j is the j^{th} element of the input vector, and w_{jk} is the weight element at column j and row k . Thus, the number of cumulative adders used is n_{i+1} and the number of multiplier-accumulators used is n_{i+1} .

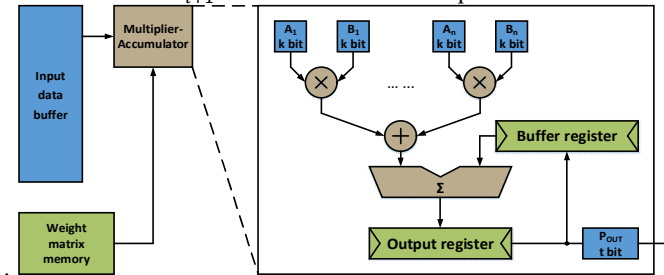


Fig. 6. Hardware design of the Multiplier-Accumulator.

Parallel coefficient

To reduce the number of clock cycles needed for matrix multiplication, it is possible to read j input units and j columns of the weight matrix at the same time. Then the required number of clock periods for the matrix multiplication can be reduced by a factor of j ; however, the number of MACs in a clock cycle will increase accordingly by $p = n_i/j$ times. Herein, the value of parameter p is called a parallel coefficient. The number of multipliers, accumulators, and clock cycles are estimated as follows:

$$N^{MAC} = pn_{i+1}; N^{add} = pn_{i+1} + 1; N_i^{clocks} = \frac{n_i}{p} \quad (2)$$

where N^{MAC} is the number of multipliers, N^{add} is the number of adders, and N_i^{clocks} is the number of clocks.

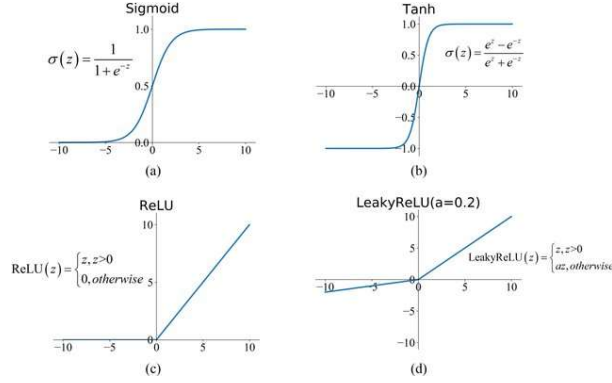


Fig. 7. Common activation functions: (a) Sigmoid, (b) Tanh, (c) ReLU, and (d) LeakyReLU.

Activation function

Among the most commonly used activation functions for neural networks, there are some options, including Sigmoid, Tanh, ReLU, or leak ReLU as shown in Fig. 7. From the hardware design point of view, we essentially selected the Rectified Linear Unit (ReLU) function because of its simplicity and feasibility for hardware implementation. As we can see in Fig. 7, the ReLU activation function is a piecewise linear function that outputs the input directly in case of the positive input value and returns zero otherwise. Using the ReLU function can accelerate the training process thanks to the fast calculation of the loss function's gradient concerning parameters. ReLU is also proven to be good enough for achieving an adequate level of accuracy for different neural network problems [27-29].

4.2 Design of data buffer

Input data buffer

The input data buffer has memory cells arranged in rows and operates under the FIFO mechanism. The FIFO width is equal to the number of bits for input representation

multiplied by the parallel coefficient p . This permits p elements that can be read or written at the same time by issuing a FIFO read and write, respectively. Regardless of the value of the parallel coefficient, the total memory required for the i^{th} layer with n_i elements, each represented by k bits, is equal to

$$Mem_i^{Input} = kn_i \text{ (bits)} \quad (3)$$

Weight matrix memory.

Recall that the weight matrix is optimized during the training process implemented by the software tool since parameter learning consumes a lot of hardware resources. The values of the weight matrix can be represented by the k bits binary number. Similar to the organization of the input buffer memory, the data width of the weight memory has to be matched the designed parallel coefficient. The size of the weight matrix W is $n_i \times n_{i+1}$. To represent the address for the n_i registers we need to take up to the following hardware resources:

$$Mem_i^{Weight} = kn_in_{i+1} \text{ (bits)} \quad (4)$$

Considering the analysis of number format in subsection 3.3, which shows that trained weight values are real numbers with a limited value range, we design the format of weight values as follows. Those values in the actual hardware design if remapped to convenient fixed-point representation values, in turn, can be treated as the equally scaled-down of the integer values. This can be done first by multiplied by a scale-up factor m (m is a non-negative number) and then is rounded to a signed integer number. Therefore, the actual hardware multiplier eventually is the just an integer multiplier, which is much simpler than the real-number multiplier as can be shown in Fig. 8.

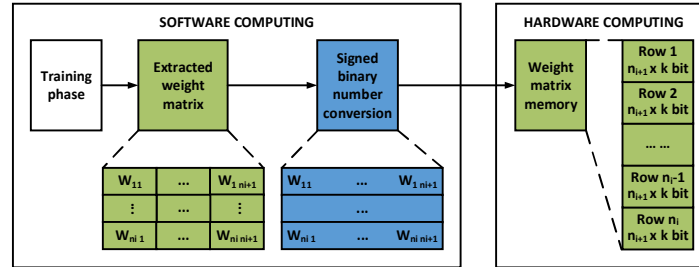


Fig. 8. Hardware design of the weight matrix.

Bias vector memory

The bias matrix extracted from the training phase of the classification model is a vector with n_{i+1} elements. Similarly to weight values, each bias-element needs k bits to represent. The resource occupied by the hardware for the bias vector is estimated as below:

$$Mem_i^{Bias} = kn_{i+1} \text{ (bits)} \quad (5)$$

Output data buffer

The output data buffer with the FIFO mechanism is designed to store the results of the multiplier-accumulator (vector of n_{i+1}). To overcome the overflow phenomenon when performing the scalar product between n_i -element vectors (each element occupies k bits), the output result should be represented by $(\log_2 n_i + 2k)$ (bits). Theoretically, the number of bits for an output unit is at least 2 times higher than that for an input unit, which can cause the memory shortage especially in case of a large number of hidden layers. Therefore, we can reduce the number of bits occupied by each output value by m bits. More specifically, before storing in the buffer, the output value is divided by 2^m . Dividing the real-number output value by 2^m can be simply implemented in the case of binary numbers by removing the m lowest bits of the binary data. The output data buffer consists of n_{i+1} registers for n_{i+1} output features and each register is represented by $(\log_2 n_i + 2k - m)$ (bits) and requires $add_{b_out} = \log_2 n_i$ (bits) to specify a register address. Finally, the amount of memory resources on the hardware for the output data buffer at layer $i + 1$ equals:

$$Mem_i^{Output} = n_{i+1}(\log_2 n_i + 2k - m) \text{ (bits)} \quad (6)$$

4.3 Hardware Utilization and Processing Latency

We have derived the resource utilization on hardware for the hidden layer $i + 1$ including input data buffer, weight matrix, bias vector, output data buffer. If there are L consecutive hidden layers, the number of matrix multiplication blocks are $L + 1$. Then, the total amount of memory occupied is estimated as below:

$$Mem^{total} = \sum_{i=1}^{L+1} (Mem_i^{Input} + Mem_i^{Weight} + Mem_i^{Bias}) + Mem_{i+1}^{Output} \text{ (bits)} \quad (7)$$

Given the parallel coefficient p , the total number of MACs is equal to $N^{MAC} = \sum_{i=1}^{L+1} N_i^{MAC} = p \sum_{i=1}^{L+1} n_{i+1}$ and similarly, the total number of adders is $N^{Add} = \sum_{i=1}^{L+1} N_i^{Add} = p \sum_{i=1}^{L+1} n_{i+1}$. The processing latency (or the number of clock cycles) N^{clocks} for the neural network with parallel coefficient, p is calculated as:

$$N^{clocks} = \sum_{i=1}^{L+1} N_i^{clocks} = \sum_{i=1}^{L+1} \frac{n_i}{p} \quad (8)$$

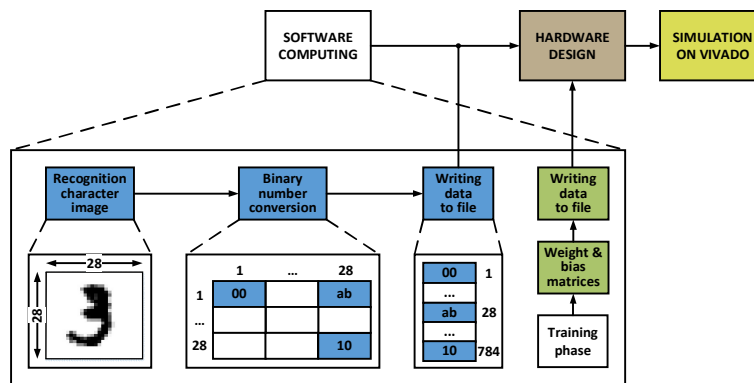
We can infer from hardware consumption and processing delay that the neural network-based hardware architecture requires hardware resource that is linearly proportional to the parallel coefficient. Meanwhile, the processing delay is linearly reduced by a factor of p . The selection of parallel coefficients should be considered based on the FPGA memory, the number of given processing units, and the required delay of a specific application. The actual performance metrics and resource utilization will be presented and discussed in the subsequent section.

5 Performance Evaluation of Neural Network Accelerator on FPGA

5.1 Experimental Setup

In this subsection, we introduce two considered datasets and describe the neural network architecture for each data. Then, the performance metrics and network parameters are also presented. Based on the generic model described in the previous section, we have implemented the hardware models targeted for MNIST and CIFAR-10 datasets. Those models are fully described using synthesizable VHDL optimized for FPGA. Those case studies will be further used for evaluating the performance and other design aspects.

For the MNIST dataset, each input image will be converted to a 1×784 binary matrix. Input matrix and the network parameters (i.e., weights and biases values) obtained from the training phase are fed into the Xilinx Vivado Simulator [30] to collect performance metrics including computational speed and recognition accuracy.



(a)

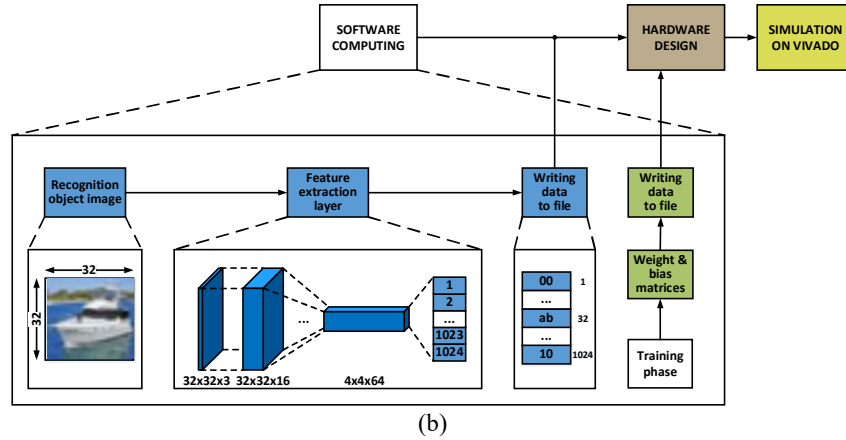


Fig. 9. Design model of image recognition on hardware for the (a) MNIST dataset and the (b) CIFAR-10 dataset.

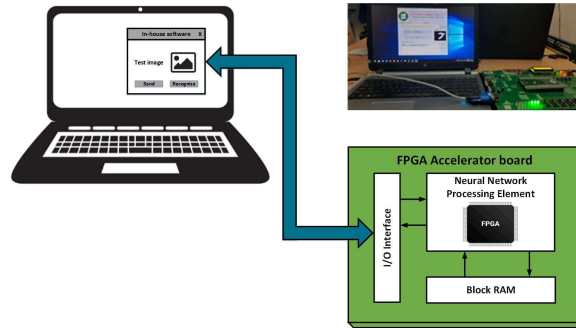


Fig. 10. Experiments on the FPGA board.

In cases of the CIFAR-10 dataset, images are larger and more complex than those in the MNIST database. To temporarily simplify the hardware design, we only implement fully-connected layers on the hardware while feature extraction layers are pre-processed. Note that the implementation of those layers follows the classical image classifier and does not cost much in cases of the small filter kernel [31]. In the first fully-connected layer, the number of input units is 1024, the number of output features is 256; in the second fully-connected layer, the number of output neurons is 10. Therefore, the size of the weight matrix for the first layer is 1024×256 , for the second layer is 256×10 ; the size of the bias vectors for the first layer is 256×1 , for the second layer is 10×1 .

To examine the performance of the hardware design of the neural network accelerator, we have collected the performance of the image classification model on a real FPGA board as can be seen in Fig. 10.

5.2 Experimental Results of Neural Network Accelerator on FPGA

Our entire generic design presented in section 4 has been described by a hardware description language (HDL), where the number format and parallel coefficient are considered as design parameters and can be set to desired values. The hardware architecture of NN is evaluated on the Vivado Simulator. The parameters are converted into a fixed-point number format (with 2, 4, 8, or 16 bits) by an in-house software on C++. The featured parameters of the built neural networks are trained and extracted using TensorFlow. The performance evaluation is conducted on 1,000 samples in the MNIST dataset.

After designing the image recognition model on the hardware, experiments are conducted to evaluate the hardware architecture. First, the extracted image data from the text files are put into the designed block and processed by the hardware simulator Xilinx Vivado. The classified label which is the output of the image classification model is then compared with the true label and the number of correctly recognized images will be recorded in the counter. When the last image in the test set is classified, the classification accuracy is obtained.

The impacts of the number formats and network architectures

Table 1. Accuracy of MNIST image recognition implemented on the hardware with different hidden layers and number formats for 1,000 samples.

Number of hidden layers	2-bit	4-bit	8-bit	16-bit
1	16.8%	88.6%	89%	89%
2	16%	96.8%	96.9%	96.9%

In this subsection, we focus on analyzing the dependence of recognition accuracy on two main parameters: the number of hidden layers and number format to represent the values of the weight matrix. The results of MNIST image recognition accuracy with different parameters are shown in Table 1. It can be seen that the classification accuracy depends more on the number format than the number of hidden layers. When the number of bits used to represent the weight matrix is too small, the recognition accuracy is very low (e.g., 16% accuracy for 2-bit number format). Meanwhile, the accuracy sharply improves if the number of bits changes from 2 to 4 or more. However, there is no significant improvement in cases of using more than 4 bits for each training parameter. E.g., the accuracy converges to 89% and 96.9% with 1 and 2 hidden layers, respectively. From Table 1, we observe that the 8-bit format can be used to save hardware resources while ensuring accurate performance on image classification.

Hardware utilization on different chips and with different parallel coefficients

Recall that we have derived the hardware resources required for MACs, adders, and memory in Eq. (2). The number of MACs, the number of adders, and memory size increase proportionally to the parallel coefficient and the number of hidden layers. Indeed, using a higher parallel coefficient p results in short prediction time but

a significant increase in the system resources demand. At the same time, multilayer neural networks can produce higher accuracy while demanding more computing resources. Therefore, it is necessary to study the relationship between these two factors (recognition time and resource demand) for a better selection of neural network architectures in reality.

To understand the feasibility of FPGA for the neural network application, we first implemented and compared the designs on some representative FPGA devices from Xilinx. Then, with the HDL designed and fully logical verified, we have implemented on the actual FPGAs. The main results are presented in Fig. 11.

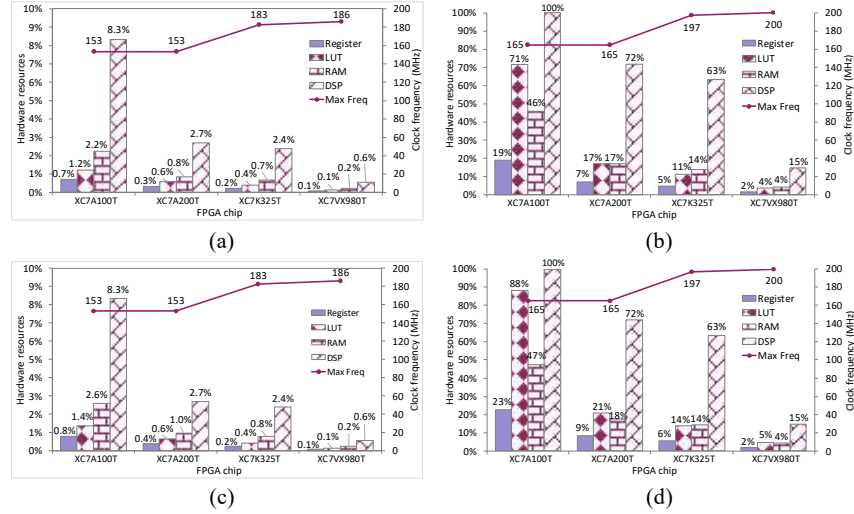


Fig. 11. Hardware utilization and corresponding maximum clock frequency (a) 1-layer MNIST, (b) 2-layer MNIST, (c) 1-layer CIFAR, and (d) 2-layer CIFAR implementations using an 8-bit fix-point number format targeted for different Xilinx FPGA devices.

In this work, we have chosen some representative and active FPGA families from Xilinx [32], including the low-cost (Artix-7), the best price/performance (Kintex-7), the performance-optimized (Virtex-7) solutions with different resource capabilities. In terms of resources, except for the Artix XC7A100T FPGA, where all DSP is fully utilized, the remaining FPGA devices are considered large enough for accommodating 2-layer neural networks in either CIFAR or MNIST. The actual resource utilization of slice registers, LUT, RAM only accounts for a small proportion of the total availability (e.g., less than 15% LUT for Kintex 7 XC7K325T, or less than 5% LUT for Virtex 7 XC7V980). This is the strong validation for the feasibility of the implementation of the more sophisticated recognition and classification engines (e.g., up to 5 hidden layers with more complex activation functions) on the next generation FPGA. The other high-end devices such as Ultra-scale [33] and Ultra-scale plus [34] with their extremely large logic and computing resource are essentially capable not only for neural networks but complete AI system implementations. In terms of performance, the achievable clock frequencies are technically dependent mainly on

the latency of MAC (i.e., FPGA DSP macro). Therefore, the reported clock frequencies range from 153MHz (Artix 7) to ~200 MHz (Virtex 7).

Furthermore, we evaluated the impacts of the parallel coefficient on the resource utilization and bandwidth of the hardware architecture. Fig. 12 presents the resource utilization and bandwidth of MNIST implementation on the Xilinx Kintex-7 FPGA series XC7K325T.

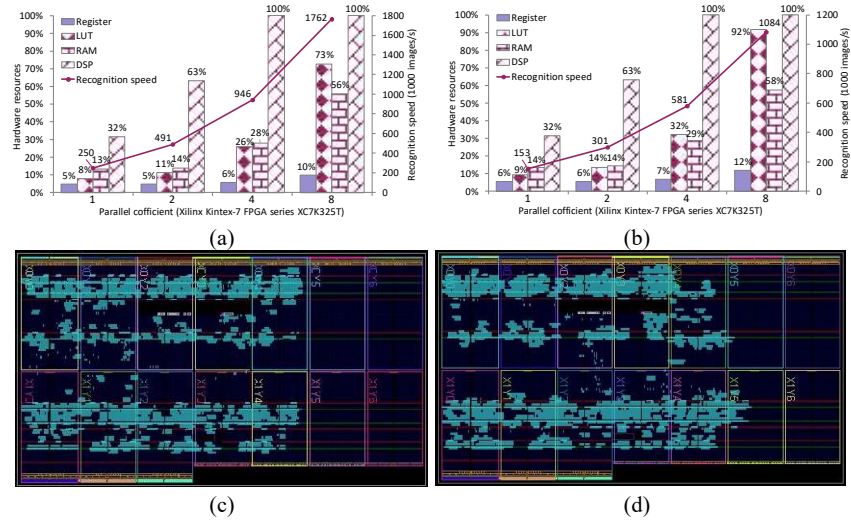


Fig. 12. Resource utilization and recognition speed on Xilinx Kintex-7 FPGA series XC7K325T of 2-layer neural network for (a) MNIST, (b) CIFAR implementations with different parallel coefficients and images depicting physical layout on the FPGA chip for (c) MNIST, (d) CIFAR implementations when parallel coefficient equals 4.

Based on the results obtained, we found that the image recognition speed also increases almost proportionally to the parallel coefficient p . Specifically, when the parallel computing is not applied (i.e., $p = 1$), the implemented network can process 250,000 images image recognition per second, and when this coefficient increases to 8, the image recognition speed reaches 1,762,000 images per second for the MNIST dataset. Similarly, for the CIFAR-10 dataset, the recognition speed is 153,000 and 1,084,000 images per second, respectively. The practical performance hence increases by more than 7X in either implementation when changing p from 1 to 8. This is explained by a slight degradation in maximum clock frequencies when the design becomes larger. There is an inevitable trade-off between increasing image recognition speed and the cost in logical resource, and this is explicitly shown by the dependency of the growth rate of the resource utilization and the parallelism level.

Performance comparison between FPGA-based neural network accelerator and neural network using the software tool

From the simulation results of the software design and the implementation results of the design on the hardware, we can see that the achievable accuracy of the image recognition on the hardware is as good as achievable accuracy by software. Meanwhile, hardware implementation is more beneficial in terms of detection speed than that on software.

Table 2. Comparison of software and hardware performance in MNIST image recognition.

Platform	Software				Hardware								
Number of layers	1	2	1	1	1	1	1	1	2	2	2	2	2
$p^{(*)}$	N/A	N/A	1	2	4	8	16	1	2	4	8	16	
$T^{(**)}$ (μ s)	6000	8000	4	2	1.1	0.6	0.3	5.3	2.7	1.4	0.8	0.4	

(*) Parallel coefficient, (**) Recognition time

Full comparative figures between image recognition on hardware and software are given in Tables 2. From this table, we observe that the time to recognize images when performing on hardware is faster than image recognition on software at least 1500 times when the parallel coefficient equals 1 (6000 μ s vs 4 μ s, using one hidden layer for recognition), and maybe faster up to 20,000 times when the parallel coefficient is 16 (8000 μ s vs 0.4 μ s with two hidden layers).

6 Conclusion

In this work, we have proposed a generic design for neuromorphic computing upon one layer, that allows us to construct any other sophisticated neural networks. Along with the generic design, a systematic study has been conducted on the resource utilization, performance, and accuracy of the neural network models and their dependencies on the design hyper-parameters. From the statistical study, we have practically proven that using a fixed-point number for hardware implementation could greatly reduce the complexity and resource for the hardware implementation while still maintaining mostly the same level of accuracy compared to the software implementation.

As a case study, we implemented the hardware models for MNIST and CIFAR-10 datasets on a reconfigurable hardware platform. Regarding the resource utilization, a Xilinx Virtex 7 device (XC7VX980) can handle the 2-layer CIFAR-10 implementation with spending less than 5% of LUT and 15% in DSP. Furthermore, at iso-accuracy, the FPGA-based neural network implementations are notably faster in recognition speed. If no parallel computing is considered, the proposed hardware accelerator is 1,500 times quicker than the baseline software implementation and could reach 20,000 with a higher degree of parallelism. Though our initial design in this work is limited for the neural networks, the impressive results proved the potential of reconfigurable devices and FPGA as the flexible and powerful platform for neuromorphic computing and AI applications in general.

Acknowledgment

This research is funded by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01-2018.310.

References

1. A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural networks. In NIPS 25, pages 1106-1114. Curran Associates, Inc., 2012.
2. Y. Sun, Yi, X. Wang, and X. Tang, "Deep Learning Face Representation by Joint Identification-Verification," in Neural Information Processing Systems, 2014, pp. 1988-1996.
3. S. Ji and W. Xu, "3D Convolutional Neural networks for Automatic Human Action Recognition," in Pattern Analysis & Machine Intelligence, 2013, 35(1), pp. 221-31.
4. O. Abdel-Hamid, "Convolutional Neural networks for Speech Recognition," in Audio Speech & Language Processing, 2014.
5. [n. d.]. <https://github.com/Xilinx/chaidnn>. ([n. d.]). Accessed Mar 31, 2020.
6. [n. d.]. https://www.xilinx.com/support/documentation/white_papers/wp504-accel-dNeural_networks.pdf. ([n. d.]). Accessed Mar 31, 2020.
7. [n. d.]. <http://www.deephi.com/technology/dnndk>. ([n. d.]). Accessed Mar 31, 2020.
8. Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale ML on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016).
9. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional Neural networks," in Neural Information Processing Systems, 2012, pp. 1097-1105.
10. K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on Imagenet classification, in Proceedings of the IEEE International Conference on Computer Vision, 2015a, pp. 1026-1034.
11. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211-252.
12. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770-778.
13. C. Szegedy, et al. Going deeper with convolutions. In CVPR, 1-9, 2015.
14. Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, Huazhong Yang, "A Survey of FPGA-Based Neural Network Accelerator", 2017, arXiv:1712.08934v3.
15. Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, Shaojun Wei, "FP-BNN: Binarized neural network on FPGA", Elsevier, 2017, Available online 18 October 2017: <https://doi.org/10.1016/j.neucom.2017.09.046>.
16. NVIDIA, Tesla K40 GPU Active Accelerator, NVIDIA, 2013.
17. Y.-H. Chen, et al. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural networks. In IEEE Int. Solid-State Circuits Conf. (ISSCC), 2016.
18. Ovtcharov K, Ruwase O, Kim J-Y, Fowers J, Strauss K, Chung ES (2015) Accelerating deep CNNs using specialized hardware. Microsoft Research Whitepaper vol 2, no 11
19. A. Putnam, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In Int. Symp. on Computer Architecture (ISCA), 1324, 2014.
20. M. Bettoni, G. Urgese, Y. Kobayashi, E. Macii, and A. Acquaviva, "A Convolutional Neural network Fully Implemented on FPGA for Embedded Platforms," 2017 New Generation of CAS (NGCAS), Genova, 2017, pp. 49-52, DOI: 10.1109/NGCAS.2017.16.

21. E. Nurvitadhi, D. Sheffield, Jaewoong Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating Binarized Neural networks: Comparison of FPGA, CPU, GPU, and ASIC," 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, 2016, pp. 77-84, DOI: 10.1109/FPT.2016.7929192.
22. E. Nurvitadhi, Jaewoong Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, 2016, pp. 1-4, DOI: 10.1109/FPL.2016.7577314.
23. [n. d.]. <http://yann.lecun.com/exdb/mnist/>. ([n. d.]). Accessed Mar 31, 2020.
24. A. Krizhevsky, "CIFAR-10 AND CIFAR-100 DATASETS," 2009. [Online]. Available: <https://www.cs.toronto.edu/~kriz/cifar.html>.
25. [n. d.]. <https://becominghuman.ai/best-languages-for-machine-learning-in-2020-6034732dd24>. ([n. d.]). Accessed Mar 31, 2020.
26. [n. d.]. <https://opensource.com/article/18/5/top-8-open-source-ai-technologies-machine-learning>. ([n. d.]). Accessed Mar 31, 2020.
27. Feng, Junxi & He, Xiaohai & Teng, Qizhi & Ren, Chao & Chen, Honggang & Li, Yang. (2019). Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. *Physical Review E*. 100. 10.1103/PhysRevE.100.033308.
28. Jie Hu and Li Shen and Gang Sun. Squeeze-and-Excitation Networks. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018.
29. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*. May 2017.
30. [n.d.]. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug937-vivado-design-suite-simulation-tutorial.pdf. ([n. d.]). Accessed Jun 06, 2020.
31. Aurelien Gron. 2017. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (1st. ed.). O'Reilly Media.
32. [n.d.]. <https://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf>. ([n. d.]). Accessed Jun 06, 2020.
33. [n.d.]. <https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf>. ([n. d.]). Accessed Jun 06, 2020.
34. [n.d.]. https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf. ([n. d.]). Accessed Jun 06, 2020.