High-Throughput Machine Learning Approaches for Network Attacks Detection on FPGA

Duc-Minh Ngo, Binh Tran-Thanh, Truong Dang, Tuan Tran, Tran Ngoc Thinh, and Cuong Pham-Quoc(✉)

Ho Chi Minh City University of Technology, VNU-HCM, Ho Chi Minh City, Vietnam
cuongpham@hcmut.edu.vn

Abstract. The popularity of applying Artificial Intelligence (AI) to perform prediction and automation tasks has become one of the most conspicuous trends in computer science. However, AI systems usually require heavy computational tasks and result in violating applications that need real-time interactions. In this work, we propose a system which is a combination of FPGA platform and AI to achieve a high-throughput network attacks detection. Our architecture consists of 2 well-known and powerful classification techniques, which are the Decision Tree and Neural Network. To prove the feasibility of the proposed approach, we implement a prototype on NetFPGA-10G board using Verilog-HDL. Moreover, the prototype is trained and tested with NSL-KDD dataset, the most popular dataset for network attack detection system. Our experimental results show that the Neural network core can detect attacks with speed at up to 9.86 Gbps for all packet sizes from 64B to 1500B, which is thoroughly 11x and 83x times faster than Geforce GTX 850M GPU and i5 8th generation CPU, respectively. The Neural Network classifier system can function at 104.091 MHz and achieve the accuracy at 87.3.

Keywords: Machine learning · FPGA platform · Network attacks

1 Introduction

In recent years, the capacity of a machine to imitate intelligent human behaviors called Artificial Intelligence (AI) [14] has become a prominent topic. AI has achieved several successes in practical applications such as visual perception, decision-making, speech recognition, and also object classification. Likewise, Machine learning (ML) [10] is well-known as a subset of AI with the ability to update, improve itself when exposed to more data; machine learning is flexible and does not require human intervention to make certain changes.

One of the most practical applications of ML is to solve classification problems. Many ML models such as Linear Classifiers, Logistic Regression, Naive Bayes Classifier, Support Vector Machines, Decision Trees, or Neural Networks can be used to make predictions for new data. For instance, an artificial neural

P. C. Vinh and A. Rakib (Eds.): ICCASA 2019/ICTCC 2019, LNICST 298, pp. 47–60, 2019.
https://doi.org/10.1007/978-3-030-34365-1_5

network (ANN) computation model which compose of multiple neuron layers, connections, and directions of data propagation has ability to learn features of data with multiple levels of abstraction by finding the suitable linear or non-linear mathematical manipulation to turn inputs into outputs. The learning processes, referred to as training phases, of a neural network are conducted to determine the value of parameters as well as hyperparameters (such as the number of neurons in the hidden layer, the weights apply to activation functions and the bias values) from training datasets. Based on results of these processes, each neuron will be assigned the most suitable weight value to form the trained neuron network. The entire network, then, can be used to compute corresponding outcomes for new data. This is referred to as the inference phase.

Real-time applications usually require heavy computational tasks; thus, general purpose processors (such as CPUs) are not efficient in system performance. Therefore, hardware accelerators such as Graphics Processing Units (GPUs), and Field Programmable Gate Arrays (FPGAs), have been employed to improve the throughput of ML algorithms in recent years. Although GPUs are mainly used for this purpose, they suffer from inflexibility in architecture due to hardwired configuration. Meanwhile, Field-Programmable Gate Arrays (FPGAs) play an important role in data sampling and processing industries due to its flexibility in custom hardware, high parallelism architecture, and energy-efficiency. While GPU is a good choice for the training phase of an ANN, FPGA is a promising candidate for processing inference phase [5,12].

In this work, we study on designing and implementing classification models for high-speed network attacks detection on FPGA platforms. In details, decision tree and neural network techniques are deployed into a NetFPGA-10G board to detect network attacks based on the NSL-KDD dataset [3]. The main contributions of this work are summarized as three folds.

- We design two classification models, the decision tree and neurons network, for detecting network attacks using the NSL-KDD dataset.
- We propose an architecture for implementing the models on FPGA platforms.
- We implement the first prototype version on the NetFPGA-10G board and validate the system with the NSL-KDD dataset. The experimental results shows that we can beat both Geforce GTX 850M GPU and Intel core i5 8th generation CPU in processing time.

The rest of this work is organized as follows. In Sect. 2, we discuss some relevant work and classification techniques used in this work. Section 3 presents our method to build and optimize machine learning models. Section 4 shows our implementation on the NetFPGA platform. We evaluate and analyze our system in Sect. 5. Finally, conclusion is discussed in Sect. 6.

2 Related Work and Background

2.1 Related Work

ID3 is a supervised learning algorithm which builds the tree based on attributes of a given set and the resulting model is used to predict the later samples. The more information it gains, the high precision the model is. However, researchers in [6] pointed out that its sensitivity on large value will yield low conditional values. C4.5 algorithm was proposed by the work in [1] to overcome the issues left by the ID3 algorithm by using information gain computation which produces measurable gain ratio. In order to increase its performance, researchers in [15] determined another alternative form of DT classification which is accelerated in the pipeline. The main idea is conducted on a binary decision tree in which input values going in the model are decided which subset will be executed instead of running through all the model at one time. After triggering the subset to execute, another input going in the model is calculated to choose the branch of the model while the previous subset is being executed.

In term of hardware-based, an implementation using FPGA approach is proposed in [15] for accelerating the decision tree algorithm. The architecture is constructed by various parallel processing nodes. In addition, the pipeline technique is applied to increase resource utilization as well as throughput. The proposed system is reported to be 3.5 times faster than the existing implementation. In recent years, classification and machine learning implementations are blockbuster research trends on FPGA platform. A hardware-based classification architecture named BV-TCAM, proposed in [16] aiming to implement a Network Intrusion Detection System (NIDS). The proposed architecture is a combination of the two algorithms, including Ternary Content Addressable Memory (TCAM) and Bit Vector (BV). This combination helps to represent data effectively as well as increasing system throughput.

There are various neural network implementations proposed on FPGA platform to take full advantages of the ability in reconfiguration, high performance and short developing time. The authors in [2] allows quickly prototyping different variants of neural networks. Other works focus on maximize resource utilization of FPGA hardware. Other works of James-Roxby [8] proposed an implementation of multi-layer perceptron (MLP) with fixed weights, which can be modified via dynamic reconfiguration with a short amount of time. A similar exploration is found in the work of [21]. On the one hand, in artificial neural networks (ANNs) FPGA-based implementations, weights are mostly represented in an integer format. Special algorithms are proposed in [9] represents weights by power-of-two integers. On the other hand, floating-point precision weights are also investigated in the work of [11]. However, there is rarely implement of floating-point weights on FPGA platform. In this paper, a MLP model is proposed with 32-bit floating-point precision weights for classification purposes on NetFPGA platform. In addition, a decision tree model is implemented for results comparison and evaluation.

2.2 Background

In this section, we introduce an overview of the two models, the decision tree and neurons network, that we use for building our high-throughput network attacks detection system. These models are used because they are efficient when implemented on FPGA.

Decision Tree. A decision tree [13] is a tree-like model for classifying data based on different parameters which are built as intermediate nodes. Each node functions as a test that provides possible answers for classifying data. The process is iterated until a leaf node is reached. The leaf nodes represent classifications of input data.

Artificial Neural Networks - ANN. ANNs [7] are computing systems that play an important role in variety of applications domain such as computer vision, speech recognition, or medical diagnosis. In ANNs, artificial neurons are connected through a directed and weighted connections and compute outputs based on the internal state and inputs (activation function). Compared to recurrent networks where neurons can be connected to other neurons in the same or previous layer, the feedforeword ones where neurons are formed a directed acyclic graph are mainly used in computing.

Back Propagation has been dominated in the neural network as its efficiency as well as its stable error-minimizing for activation functions. Since the feed-forward is computed in the usual way, the back propagation depends on the output calculated from the activation function. In FPGA, the activation function will consume a huge amount of resources from hardware because of its complicated exponential equation, instead, a simulated activation which is simpler and implementable is applied in the model. To conduct the back propagation calculation, all the results of feed-forward computation from each node are cached so that it can compute the error of the function and narrow the weights to their most accurate values.

Weights in a neural network can be treated as input going to a single node and fed to the network in feed-forward steps calculating the output of the single neuron. The main idea of back-propagation is using that output to calculate the error of the function and narrow the weights to their most accurate values. To handle the back-propagation computation, there are two values must be stored at each node:

- The output o of the node j in the feed-forward calculation
- The cumulative result of backward computation which is a back-propagated error, denote by δ

These two values are part of the gradient computation. The partial derivative of a function E respected to weight w is using the output of the neural network to calculate the impact of related weight inputs to the whole network can be express by Eq. 1.

$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_j \tag{1}$$

We use Eq. 2 for calculating back-propagated errors, there are differences of finding at output layer and hidden layer. With the back-propagated error at output layer, the output target is required to compute using delta rule.

$$\delta = (target - output) * output * (1 - output) \tag{2}$$

With the back-propagated error at other layers, instead of finding difference between target activate value and actual output to calculate δ, they requires the total of multiplied back-propagated error of all nodes in the next layer and the respected weight since all single nodes of current layer connect to all node of the next layer.

$$\delta = (\sum \delta(nextlayer) * w) * output * (1 - output) \tag{3}$$

Once the gradient is computed in Eq. 3, the change of weight ($\triangle w$) can be calculated in Eq. 4 by multiplying it with the learning rate γ. Learning rate is a hyperparameter that controls how much weight it is adjusted in the network with respect to the loss gradient. The lower the learning rate, the slower travel on the slope of updating weight. It also means that it will take more time to get coverage.

$$\Delta w_{ij} = -\gamma o_i \delta_j \tag{4}$$

Finally, new weight are calculated by using current weight of *j-th* node adding the coverage of gradient respected to that weight in Eq. 5.

$$w_{new} = w_{old} + \Delta w_{ij} \tag{5}$$

3 Methodology

Our first prototype system on FPGA is developed to detect attacks on recorded network data. We choose NSL-KDD dataset [3] to construct and evaluate our design. Besides, the design of the FPGA-based approach which is parallel processing hardware is quite different from the software-based approach. With FPGA, hardware resources and tasks scheduling should be considered; thus, we try to optimize and find suitable machine learning models by using software-based before applying into FPGA. Furthermore, we can easily evaluate machine learning models which are built on software then using these results to compare with hardware in the same experiments (speed, accuracy test).

NLS-KDD [3] is chosen as the dataset for training and inference phases. For running with Weka tool [18], the dataset must be changed to the `.arff` format (ARFF stands for Attribute-Relation File Format). It is an ASCII text file that describes a list of instances sharing a set of attributes. There are 41 features

Table 1. The 6 features descriptions

Feature name	Description
duration	Length (number of seconds) of the connection
protocol_type	Type of the protocol, e.g. tcp, udp, etc.
src_bytes	Number of data bytes from source to destination
dst_bytes	Number of data bytes from destination to source
count	Number of connections to the same host as the current connection in the past two seconds
srv_count	Number of connections to the same service as the current connection in the past two seconds

in the data-set, however based on the hardware resource constraints, the 6 outstanding features [17] are selected due to their high impacts on the classification accuracy. The 6 features descriptions are shown in Table 1.

We have trained the system using 6 out of 41 features of NSL-KDD dataset as mentioned above to balance between accuracy and model size. The generated models are also tested with NSL-KDD dataset.

4 FPGA Implementation

In this section, we introduce our implementation of the proposed system, where a number of classification techniques are deployed on FPGA platform. Figure 1 illustrates the overview architecture of our system that can be partitioned into two layers, including CPU for running a software-based monitor tool and a device for deploying the FPGA-based architecture.

The CPU layer consists of monitor tools as interfaces for communication between administrators at the software level and the FPGA-based device. The FPGA-based device accommodates our proposed classification techniques in other to detect abnormal behaviors, including the following blocks:

1. The `Classifier` block is used to deploy classification techniques either decision tree or neural network. This block receives processed input features from the `Pre-processor` module to extract necessary features of incoming packets.
2. The `FIFO` memory buffers raw packets to increase the system throughput because of time-intensive of the Classifier block. This memory block is directly connected to Packet Pre-processor and Packet Controller.
3. The `Packet Controller` module receives results from Classifier and processes packets in the FIFO memory as well as sends alert signals based on decisions to administrators.

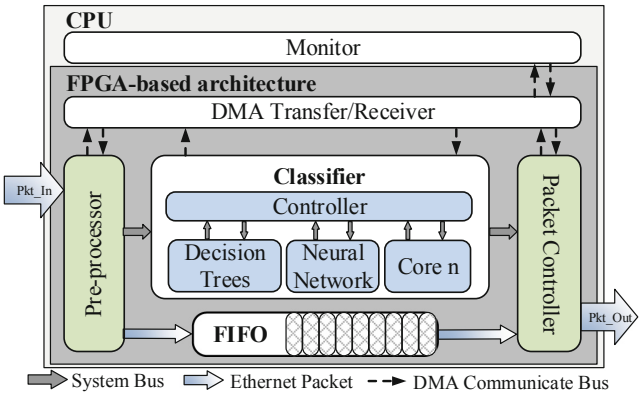

Fig. 1. First prototype system for applying classification techniques on FPFA

4.1 Decision Tree

The block diagram of the decision tree is represented in Fig. 2. There are 5 blocks in the architecture, including an input block, an output block, a recursive decision tree (sub-tree), a left-hand-side block, and a right-hand-side block. The input block is responsible for providing inputs to the recursive decision tree block while the output block gets the predictions from it. The recursive decision tree block decides which tree branch is enabled for making a prediction based on the combination of inputs. The left-hand-side tree branch is implemented as the LHS block while the right-hand-side tree branch is implemented as the RHS block.

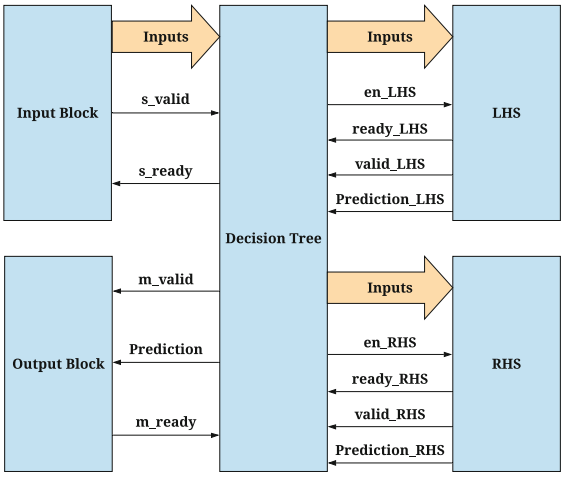

Fig. 2. Decision tree block diagram

4.2 Artificial Neural Network

In this section, we introduce our implementation for the proposed neuron-network core.

Feedforward Phase. Figure 3 illustrates the general model of a fully connected multiple layer neural network which is implemented on FPGA platform. The neural network is constructed from 4 layers, including one input layer, two hidden layers, and one output layer. Moreover, comparator and FIFO are added for outputs estimating and storing purposes.

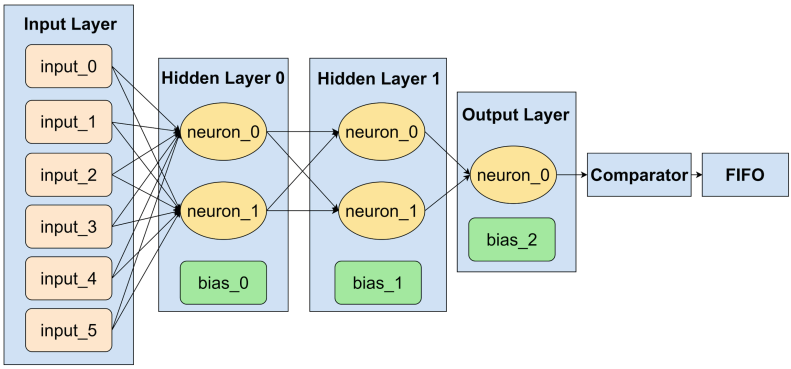

Fig. 3. Neuron network overview

There are 2 neurons in each hidden layer while only one neuron is implemented in the output layer. In addition, each hidden and output layer has dedicated configurable bias value for fitting different dataset. Furthermore, weight values in the two hidden and output layer are also adjustable for changing dataset or updating (neural network) model purposes. The block diagram of the multiple layer neural network is shown in Fig. 4.

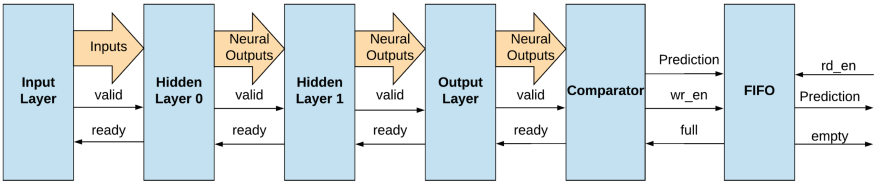

Fig. 4. Neuron network block diagram

For supporting asynchronous communication between modules, the handshaking mechanism is used in the neural network model. “Inputs” are passed

through hidden layers and output layer for producing "Prediction" basing on weights and biases. These predictions are then written into a FIFO, waiting to be read. Moreover, the pipeline technique is used for increasing throughput of the system.

Update Weights Phase. As it can be seen in Fig. 5, there are two main elements in update weight implementation called delta calculator and weight calculator. The delta calculator must be executed in serial while the weight calculator can be started when the delta calculation is finished. The delta calculating flow top-down is ordered from the output layer back to the input layer.

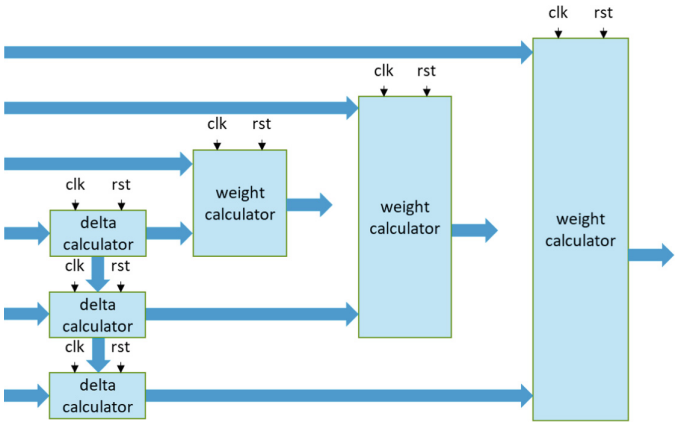

Fig. 5. Update weight block diagram

Delta calculator consists of three types. The delta calculator at output layer is the implementation of function in Eq. 2 while the other two is the implementation of function in Eq. 3. The weight calculator is the implementation of function in Eq. 5.

- Delta calculator at the output layer demands two inputs: result calculated from the output layer and the expected output from testing samples. This sub-module performs the back-propagated error calculation based on Eq. 2.
- Delta calculator at second hidden layer consists of the following inputs: the result calculated by the current neuron and all pairs of delta and coordinated weight of the right side layer that the neuron connects to. Because there are three neurons in the hidden layer so three instances of this module are required.
- Delta calculator at first hidden layer consists of the following inputs: the result calculated by the current neuron and all pairs of delta and coordinated weight of the right side layer that the neuron connects to. Because there are three neurons in the hidden layer so three instances of this module are required.

Weight calculator consists of three inputs which are the delta from the previous sub-module, the result from activation function and the weight respected to the output of activation function. The function of this module is that it will calculate the new weight based on the update function and the implementation is the same in all three layers.

5 Evaluation

In this section, we discuss the synthesis results, then present our experimental results for the proposed system. The classification cores are evaluated in two different experiments which are throughput and accuracy test.

5.1 Synthesis Results

Our proposed system is deployed into NetFPGA-10G board including Xilinx Virtex-5 xc5vtx240t device, which has a combination of 149,760 Registers, 149,760 LUTs, 324 BlockRAMs, and 37,440 Slice hardware resources in total. The system is synthesized with the Xilinx XPS 13.4 [20] and optimized using Xilinx PlanAhead toolchain [19]. Because of the limitation in hardware resources, we can not integrate both classification cores at the same time. Table 2 shows the resources usage of the system with either decision tree or Neural Network core.

Table 2. Resource usage

Resources	Decision tree	Neural network
Register	74,049 *(49.45%)*	117,078
LUT	67,552 *(45.11%)*	107,036 *(71.47%)*
BlockRAM	181 *(55.86%)*	181 *(55.86%)*
Maximum frequency	100.675 MHz	104.091 MHz

The results shows that the system with decision tree consumes nearly a haft hardware resources of xc5vtx240t device and the minimum frequency is 100.675 MHz. Meanwhile, the system with neural network consumes up to 78.18% Registers of this device with minimum frequency at 104.091 MHz. This result is one evidence that we have to separate decision tree from neural network in other to satisfy the availability of hardware resources.

5.2 Experimental Setup

The testing model in Fig. 6 is used to evaluate our system. The test uses two boards NetFPGA-10G with Xilinx xc5vtx240t device integrated into CPUs. Each board functions as a high-speed network transfer/receiver:

- One board NetFPGA-10G is installed OSNT (Open Source Network Tester) [4] which is responsible for sending packets at line-rate speed (up to 9.87 Gbps on each port) to our proposed system.
- Another board is our network attack detection system in which either the neural network or decision tree core is integrated The system is directly connected by network cable to the OSNT using SFP+ interface.

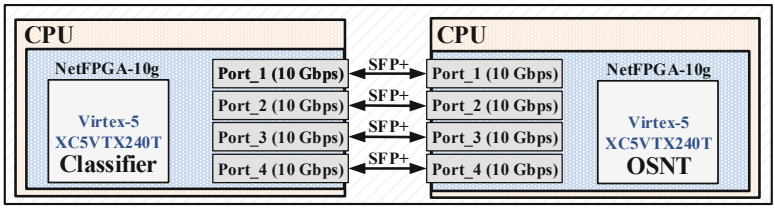

Fig. 6. Testing setup

Data using to test the system are attacking packets from NSL-KDD dataset with different lengths: 64B, 128B, 256B, 512B, 1024B and 1500B. To evaluate throughput of the system, we increase the sending rate of the OSNT board until it exceeds the maximum responding speed of the system.

5.3 Experimental Results

The throughput is recorded by the OSNT monitor tool with the setup in Fig. 6. Thank to the parallel architecture, decision tree on FPGA can achieve maximum throughput by up to 9.86 Gbps, as shown in Fig. 7. The throughput of the decision tree module is closed to throughput of incoming packets without any packet loss. Besides, the throughput of the neural network module gradually increases in the first three types of packet, from 1.28 to 3.40 and 6.96 Gbps with 64B, 128B, and 256B packets, respectively. With larger packets, 512B, 1024B, and 1500B, the processing throughput of both neural network and decision tree approximate to throughput of incoming packets without any packet loss.

For evaluating the accuracy of the neural network and decision tree core, we measure the percentage of correct predictions over total packets received. The testing set of NSL-KDD is used in this scenario. We also compare our cores on FPGA with the same models in different platforms which are GPU GeForce GTX 850M and CPU i5 8th generation with 16 GB RAM (Ubuntu 14.04). The results are shown in Table 3.

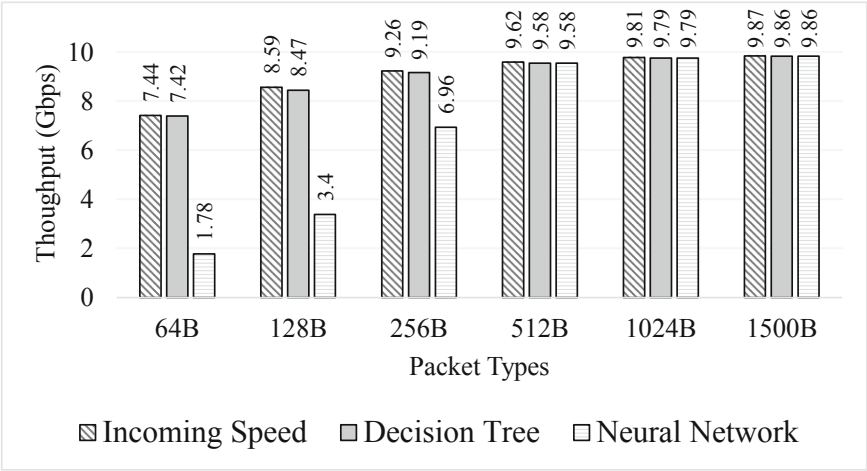

Fig. 7. Testing throughput

Table 3. Comparisons of three platforms in term of accuracy and processing time

Platform	Accuracy (%)		Processing time (s)	
	NN	DT	NN	DT
GPU GeForce GTX 850M	91.230	97.550	0.140	0.050
CPU i5 8300 with 16 GB RAM	87.300	95.102	1.007	0.405
FPGA with xc5vtx240t device	87.300	95.102	0.012	0.004

The accuracy values reported in Table 3 show that FPGA produces the same results as CPU does but faster than CPU for both cores. While GPU needs 0.14 s to finish the NN computation, the FPGA only need 0.012 s. In other words, speed-up of 11.6× is achieved when compared to GPU.

6 Conclusion

In this paper, we design and implement high-throughput machine learning techniques which are decision tree and neural network on FPGA platform to detect network attacks. The proposed system not only could examine network packets to detect various types of network attacks but also are flexible when different core can changed to adapt attacking types. We implement our proposal in Xilinx xc5vtx240t device with two detection cores: decision tree and neural network. The implemented neural network can detect attacks at 1.78 Gbps and up to 9.86 Gbps with packets size from 64B to 1500B, which thoroughly faster than Geforce GTX 850M GPU and i5 8th generation CPU 11x and 83x times, respectively. The neural network detection core can function at 104.091 MHz and

achieve the accuracy at 87.3% while these numbers are 100.675 MHz and 95.1% for the decision tree core.

Acknowledgements. This research is funded by Ho Chi Minh City University of Technology - VNU-HCM, under Grant number T-KHMT-2018-25.

References

1. RuleQuest Research 2017: Is See5/C5.0 Better Than C4.5? https://rulequest.com/see5-comparison.html. Accessed 27 June 2019
2. Cox, C.E., Blanz, W.E.: GANGLION-a fast field-programmable gate array implementation of a connectionist classifier. IEEE J. Solid-State Circuits **27**(3), 288–299 (1992)
3. Canadian Institute for Cybersecurity: NSL-KDD dataset. https://www.unb.ca/cic/datasets/nsl.html. Accessed 27 June 2019
4. Github: OSNT 10G Home. https://github.com/NetFPGA/OSNT-Public/wiki/OSNT-10G-Home. Accessed 27 June 2019
5. Guo, K., Zeng, S., Yu, J., Wang, Y., Yang, H.: [DL] a survey of FPGA-based neural network inference accelerators. ACM Trans. Reconfigurable Technol. Syst. **12**(1), 21–226 (2019). https://doi.org/10.1145/3289185
6. Hssina, B., Merbouha, A., Ezzikouri, H., Erritali, M.: A comparative study of decision tree ID3 and C4.5. Int. J. Adv. Comput. Sci. Appl. **4**(2), 13–19 (2014)
7. Jain, A.K., Mao, J., Mohiuddin, K.: Artificial neural networks: a tutorial. Computer **3**, 31–44 (1996)
8. James-Roxby, P., Blodget, B.: Adapting constant multipliers in a neural network implementation. In: Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No. PR00871), pp. 335–336. IEEE (2000)
9. Marchesi, M., Orlandi, G., Piazza, F., Uncini, A.: Fast neural networks without multipliers. IEEE Trans. Neural Netw. **4**(1), 53–62 (1993)
10. Mitchell, T.M.: Machine Learning, 1st edn. McGraw-Hill Inc., New York (1997)
11. Nichols, K.R., Moussa, M.A., Areibi, S.M.: Feasibility of floating-point arithmetic in FPGA based artificial neural networks. In: CAINE. Citeseer (2002)
12. Nurvitadhi, E., et al.: Can FPGAs beat GPUs in accelerating next-generation deep neural networks? In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2017, pp. 5–14. ACM, New York (2017). https://doi.org/10.1145/3020078.3021740
13. Quinlan, J.: Simplifying decision trees. Int. J. Man-Mach. Stud. **27**(3), 221–234 (1987). https://doi.org/10.1016/S0020-7373(87)80053-6. http://www.sciencedirect.com/science/article/pii/S0020737387800536
14. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall Press, Upper Saddle River (2009)
15. Saqib, F., Dutta, A., Plusquellic, J., Ortiz, P., Pattichis, M.S.: Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF). IEEE Trans. Comput. **64**(1), 280–285 (2013)
16. Song, H., Lockwood, J.W.: Efficient packet classification for network intrusion detection using FPGA. In: Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays, pp. 238–245. ACM (2005)

17. Tang, T.A., Mhamdi, L., McLernon, D., Zaidi, S.A.R., Ghogho, M.: Deep learning approach for network intrusion detection in software defined networking. In: 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), pp. 258–263. IEEE (2016)
18. T.U. of Waikato: Weka 3: Machine Learning Software in Java. https://www.cs.waikato.ac.nz/ml/weka/. Accessed 27 June 2019
19. Xilinx: PlanAhead Design and Analysis Tool. https://www.xilinx.com/products/design-tools/planahead.html. Accessed 02 Aug 2018
20. Xilinx: Xilinx Platform Studio (XPS). https://www.xilinx.com/products/design-tools/xps.html. Accessed 02 Aug 2018
21. Zhu, J., Milne, G.J., Gunther, B.: Towards an FPGA based reconfigurable computing environment for neural network implementations (1999)