HBRP PUBLICATION

Journal of VLSI Design and its Advancement
Volume 4 Issue 1

# A Review: Effective Techniques for Hardware Modelling of Machine Learning Algorithms

***Amita P. Thakare***
*Department of Electronics Engineering, Kalinga University, New Raipur, Chhattisgarh, India*
***Corresponding Author***
***E-Mail Id: amita.thakare27@gmail.com***

***ABSTRACT***
*Machine learning algorithms are complex to model on hardware. This is due to the fact that these algorithms require a lot of complex design systems, which are not easily synthesizable. Therefore, over the years, multiple researchers have developed various state-of-the art techniques, each of them has certain distinct advantages over the others. In this text, we compare the different techniques for hardware modelling of different machine learning (ML) algorithms, and their hardware-level performance. This text will be useful for any researcher or system designer that needs to first evaluate the optimum techniques for ML design, and then inspired by this, they can further extend it and optimize the system's performance. Our evaluation is based on the 3 primary parameters of hardware design; i.e.; area, energy and delay. Any design technique that can find a balance between these 3 parameters can be termed as optimum. This work also recommends certain improvements for some of the techniques, which can be taken up for further research.*

***Keywords:*** *Hardware, model, machine, learning, optimization*

## INTRODUCTION

Machine learning has become a forerunner in optimizing algorithms ranging from classification, process mapping, image processing, audio processing, etc. These operations require a series of complex tasks, that must be done with utmost accuracy in order to make the overall system accurate. As an example, the classification process, consists of the following steps Data gathering and pre-processing, wherein the data to be classified is gathered from different sensors and different datasets. The gathered data is pre-processed in order to convert it into a processing friendly format. Examples of pre-processing include, missing value removal, noise removal, etc.

Feature extraction & feature selection, in this step the gathered data is given for better numerical representation via feature extraction. If the number of features exceed a given size, then feature selection unit is deployed that can optimize the feature vector. Features are extracted using wavelet transform Fourier transform, etc., while features are selected using variance-based methods, aggregation methods, etc.

Classifier design, which basically compares these features between training and testing sets. This comparison yields in the accuracy of the classifier. The aim of any classification algorithm design must be to optimize this accuracy value.

Post-processing operations include prediction, recommendation, etc. based on the application under test.

All these operations require design of multiple kinds of hardware units including but not limited to, ALUs, fuzzy circuits, comparators, shifters, accumulators, etc. A

  

combination of these units results into the overall system development. Thus, the main aim of any hardware designer is multi-fold, wherein the designer must first ensure that all these individual components are effectively designed, and the integration of these components with each other must be done efficiently. During the design of every phase, there is a need of machine learning-based algorithm development. Thus, there is a need for every designer to first review effective techniques for design of these blocks. The next section compares algorithms ranging from segmentation, classification, feature extraction, feature selection, post-processing and more. These algorithms and their hardware modelled performance comparison will assist researchers in selecting the best possible combination of techniques for modelling. This is followed by the statistical performance comparison of these algorithms in terms of delay, area and energy consumption. Later, we recommend some techniques to further optimize the system's performance, and conclude this text with some interesting observations about the reviewed algorithms.

## LITERATURE SURVEY

Design and development of machine learning hardware can be very complex and costly due to the fact that each individual processing unit can comprise of multiple micro-processing units. For instance, in [1] researchers have implemented k-Means clustering algorithm using multi-core processing. They have used the SW26010 multi-core processor in order to do this. Due to the use of SW26010, the system is able to keep double precision and also achieve data rates up to 348 Gflops. The SW26010 consists of 4 core groups (CGs), each one of them has 1 management processing element (MPE) and internally consists of 64 individual computing processing elements (CPEs). Due to such a vividly distributed architecture, the processor is capable of performing enhanced computations which support for its usage to implement ML algorithms. Different tasks like block acquisition, block processing and finally block combination are given to different CPEs inside the processor. Due to such a distributed architecture the ML algorithm can be easily implemented with high data rate. But, due to an increase in data rate, the overall energy and area requirements of the system exponentially increase, and thus work needs to be done to optimize these parameters as well. This optimization can be done with the help of random placement of physically unclonable functions (PUFs) as proposed in [2]. Using the random placement allows for the chip to dissipate lower energy, and thereby reduce the dynamic energy consumption. This method can be used for any kind of ML chip design. An extension to the work in [1], is done in [3], wherein researchers have worked on implementation of k-Means using a hybrid processor. The processor is a combination of a soft core NIOS processor working on an Excalibur ARM combined with a Pentium III chipset. The k-Means implementation is done for clustering satellite images, wherein a performance of upto 6X is achieved when compared to the single-processor environment. Due to this, the overall system's speed is increased, but this comes with an increased energy requirement, and an increased area requirement (same as [1]). Which needs to be addressed by using techniques like [2].

When implementing ML algorithms, neural networks (NNs) and its variants are the de-facto standard. The work in [4] focuses on defining architectures for FPGAs that can be used for implementing NNs. They have reviewed techniques like Loop unrolling, Pipelining, Data compression, Eliminating FC layer, Memory-related optimizations, Double

buffering, Polyhedral optimization framework, etc. for developing FPGAs for NN implementation. They observed that implementing a distributed architecture for effective NN performance is a must when designing them on FPGAs. Moreover, multi-processor FPGAs are the de-facto choice when implementing such algorithms. Furthermore, techniques like piece-wise linear and piece-wise quadratic as defined in [5] can be further used for improving the FPGA's performance when deploying complex algorithms. A combination of these algorithms can be found in [6], wherein researchers have utilized the concept of FPGA-optimized multipliers (inspired from [5]) in order to implement deep neural nets (DNNs). They have developed reconfigurable constant coefficient multipliers or RCCMs in order to achieve 50% reduction in area, when compared with standard 8-bit quantized networks. This network is able to implement complex architectures like AlexNet with reduced area requirements, but the power requirements are increased by more than 20%. This is an area of research which needs further optimization.

Classifier design is another area of research in ML-based algorithms. The work done in [7] utilizes the concepts of Hardware Acceleration in order to achieve this task. The researchers are able to achieve a data rate of more than 200 Mbps, with a classification accuracy of 98%. Usage of hardware accelerator increases the speed, but also exponentially increases the power requirement of the system. It uses a combination of faster random-access memories (RAM), along with synchronized state machines in order to perform this task. Another SVM based implementation is defined in [8], wherein researchers have compared the performance of SVM implementation on FPGAs and GPUs. Components like FIFOs, FSMs, etc. have better efficiency on FPGAs than on GPUs, while processing components like multipliers, addition units, etc. have faster performance but higher energy consumption on GPUs. Therefore, FPGA implementation is preferred for low power and high efficiency calculations, while GPUs are preferred for a smoother and faster experience. A comparison of this performance can be seen from figure 1 as follows,

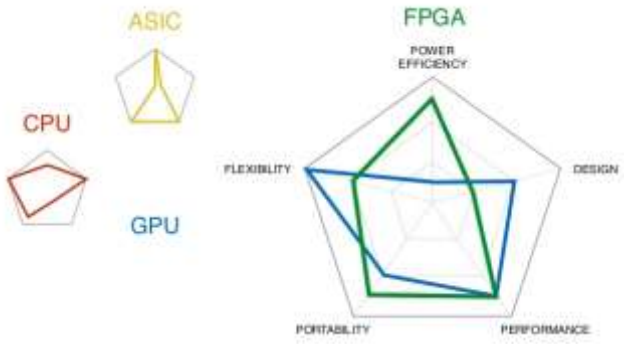


***Figure 1.*** *Comparison of FPGA and GPU Performance*

A similar implementation of SVM is done in [9], wherein CPU and FPGA performances are compared. It is found that BRAM based implementation is better than SRAM, and thus must be used for effective device utilization. FPGA delay, area and power performance are better than CPU, but the accuracy of CPU is better than FPGA. This can be improved by using optimized multipliers as suggested in [5]. The application of [9] is done in [10] and in [11], wherein FPGA

based SVM classifier is used for facial classification. From the work it is clear that SVM's are one of the best choices for hardware implementation when it comes to classifier designs. EEG and ECG based classification systems also find their way in the area of ML based processing systems. The work done in [12] indicates the usage of FPGA implementation for ANN design. Neural network design is done on the FPGA due to high speed and low area requirements. The efficiency of the system is tuned by adding reconfiguration to the number of neurons used by the system. The system lacks in terms of energy efficiency, which can be improved with the help of Layer Multiplexing Scheme as defined in [13]. Using Layer Multiplexing Scheme, the hardware is re-used via pipelining, and thereby a reduction in number of LUTs and flip flops is observed.

Due to this reduction the area and power requirement of the system reduces, without affecting the speed of operation. The only issue faced here is the reduction in performance as the number of neurons are increased. That can be tackled by using reconfigurable multipliers as mentioned in [5]. Another NN design which uses optimized multipliers and BRAMs is defined in [14]. Researchers have claimed that BRAM based designs are better, and must be utilized for ML algorithm design. The work in [15] also confirms this point, and furthers the study by observing that weight-training can be done on a highly précised software platform like Matlab, and these weights can be used for classification by inputting them to the FPGA device. Thereby, combining best performance from both the architectures. Moreover, piece-wise linear approximation must be used to implement transfer functions of the network. This observation is implemented in [16], wherein researchers have used piece-wise linear approximation along with NIOS II for FPGA-based NN implementation. Their work also suggests software training, and hardware weight application for an effective NN classification system.

The work in [17] takes this NN implementation on the cloud, and provides FPGA as a service. They combine GPUs and FPGAs together in order to effectively process a large queue of requests. Based on their observations a single FPGA service accessed by many CPUs can achieve a throughput of 600–700 inferences per second, comparable to large batch-size GPU throughput and significantly better than small batch-size GPU throughput. Therefore, the greater number of times an FPGA is accessed, the better is its output efficiency. The work in [18] discusses concepts like Loops Unrolling, 2D-Mapping, Kernel-based implementation, Smart Memory Blocks, Independent Accelerator Architectures, Embedded Accelerator Architectures, Graph Partitioning, Overall Acceleration Parallel Frameworks, matrix-vector threshold unit, Weights Reloading, CNN Accelerator, Line Buffer Design, Dispatchers, Systolic Arrays, Reconfigurable Computing, Network Inference Accelerator and Data Routers for ML design. Based on their research, this is accomplished by using equal duplicate collect tasks limited by asset restrictions. What's more, cautious structure of information get to designs are focused to limit the memory transfer speed prerequisites using inside memory structures and boosting information reuse. This is urgent in the increasing speed process because of the enormous memory information that should be gotten to counting highlight maps (FMs) and loads. To limit the memory impression and to accomplish viable use of assets, a few methods enhance the quantity of bits used to speak to the element maps and loads with insignificant sway on precision. This is joined with the enhanced choice of the

quantity of part bits utilized for each layer.

Different methods enhance the quantity of utilized loads in the completely associated (FC) layers as they are memory-serious. Coprocessors are additionally utilized to naturally design both the product and the equipment components to completely misuse parallelism. To improve parallelization of convolution activities, a few methodologies have been endeavored. Outstanding burden examination has been attempted to decide calculations that can be organized as equal streams.

The roofline model based quickening agent utilizes polyhedral-based information reliance investigation to locate the ideal unrolling factor for each convolutional layer, and to completely use all FPGA computational assets through circle pipelining. To enhance execution, tiled framework augmentation is organized as a pipelined paired snake tree for performing augmentation and producing incomplete wholes. To decrease computational intricacy of CONV layers and improve asset effectiveness, various methodologies used Winograd change in performing CONV tasks as this decreases the computational multifaceted nature by around half. To augment throughput, a few methods, for example, have utilized different CONV layer processors (CLPs) rather than utilizing a solitary CLP that is enhanced for all CONV layers. This pipeline the activity of the different CLPs accomplishing layer-level parallelism which augments asset use and improves execution in correlation with utilizing a solitary CLP. Since the computational prerequisite of FC layers is fundamentally not as much as that of CONV layers, to improve execution, and amplify asset use, a number of strategies, for example, make clumps by gathering distinctive information FMs and handling them together in FC layers. Complex access examples and information area are utilized in Deep Burning instrument for better information reuse. In [18], the creators investigated problem areas profiling to decide the computational parts that should be quickened to improve the presentation. Increasing speed is cultivated by lessening the memory data transmission necessities. Strategies proposed misuse information reuse to decrease off-chip memory interchanges. Circle changes have likewise been utilized by decreasing tiling parameters to improve information area, and to decrease excess correspondence activities to amplify the information sharing/reuse. Productive buffering, where the weight cushions are utilized to guarantee the accessibility of CONV and FC layers' loads prior to their calculation, just as to cover the exchange of FC layer loads with its calculation, helps in improving execution [18]. In the Catapult venture, FPGA sheets were coordinated into server farm applications and accomplished speedup. Microsoft Research's Catapult used multi-banked input support and portion weight cushion to give an effective buffering plan of highlight maps and loads, separately. To limit the off-chip memory traffic, a particular system on-chip was intended to redistribute the yield include maps on the multi-banked input support as opposed to moving them to the outer memory [18]. To additionally lessen memory impression and transfer speed prerequisite, ideal partial length for loads and highlight maps in each layer are utilized. Solitary worth deterioration (SVD) has additionally been applied to the weight grid of FC layer so as to decrease memory impression at this layer [18]. Tiling strategies have been proposed where huge scope input information is parceled into little subsets or tiles whose size is arranged to use the exchange off between the equipment cost and the speedup [18]. Computerization apparatuses have been built up that naturally manufacture neural systems with enhanced execution [18]. They utilize pre-

developed register move level (RTL) module library that holds equipment (counting sensible and number juggling tasks) and design contents. Deep Burning, for instance, produces the equipment portrayal for neural system contents.

Another modularized RTL compiler, ALAMO, coordinates both the RTL better level streamlining what's more, the adaptability of elevated level blend (HLS) to produce proficient Verilog parameterized RTL contents for ASIC or FPGA stage under the accessible number of equal figuring assets (i.e., the quantity of multipliers) [18]. Speeding up is accomplished by utilizing circle unrolling procedure for CONV layer tasks. A portion of the evaluated procedures likewise help limit the size of FPGA on-chip recollections to enhance vitality and territory use [18]. To improve usage of FPGAs in CNNs increasing speed also, to amplify their adequacy, we suggest the advancement of a system that incorporates an easy to use interface that permits the client to effortlessly indicate the CNN model to be quickened.

This incorporates determining the CNN model parameters regarding number of convolution layers also, their sizes, and number of completely associated layers along with other transitional tasks. The predefined CNN model loads will be perused from a record. Moreover, the client ought to have the choice of indicating the FPGA stage that will be utilized for executing the CNN quickening agent what's more, the greatest fair blunder, alongside the determination of a library from a lot of utilizations to be utilized for model enhancement and assessment. The structure at that point ought to perform enhancements to locate the base number of bits that should be utilized for speaking to the loads and highlight maps and the quantity of division bits to be utilized for each layer. Likewise, advancement of completely associated layers is performed to limit the memory prerequisites. All such enhancements are completed limited by the most extreme blunder indicated by the client for the predefined application library. The system ought to be planned dependent on the advancement of an adaptable equipment design that works for any given FPGA stage and accomplishes higher speed up with the accessibility of higher assets. In view of the accessible assets, indicated by the FPGA stage, the instrument will perform enhancements to expand parallelism and information reuse, given the asset restrictions. The instrument will at that point consequently create the CNN model that will fit on the given FPGA stage and will permit the client to assess the exhibition dependent on the picked application library. This will permit the client to assess the exhibition gains while assessing distinctive FPGA stages with various assets. The device ought to have the choice to create execution estimates dependent on various execution measurements as chose by the client, for example, number of edges handled every second or number of activities performed every second. What's more, the device will report other plan measurements such as asset use, memory sizes and data transfer capacity, and power dissemination. Moreover, it is wanted to have the alternative for the client to determine the ideal execution for a given CNN model also, have the device perform vital investigation and assessment also, prescribe to the client competitor FPGA stages for accomplishing the ideal execution levels. This will require the improvement of sensibly exact systematic models that will gauge the required assets for accomplishing the wanted execution. The client would then be able to pick the suggested FPGA stage and perform total assessment to check that the ideal execution levels are met. Similar assessments are made in [19], wherein

systolic architectures are used for implementation of self-organizing map NNs (SOM NNs). Using the SOMPE architecture the overall energy requirement and the delay have been reduced. This architecture is further utilized in [20], wherein a high-speed CNN is proposed. Due to the usage of on-chip memories, the data rate has been increased by 2X, and the power requirement is reduced by 1.2X. Such an optimized architecture can be used for effective CNN implementation. Similar implementations are mentioned in [21], [22], and [23], wherein accelerators and sparseness utilization of CNN is done in order to obtain a high speed and low energy requirement network.

Other techniques like Naïve Bayes (NB), Decision Tree (DT), Random Forest (RF), k-Nearest Neighbour (kNN), and Support Vector Machine (SVM) are discussed in [24] and [25], these techniques can be implemented by the architectures mentioned in [19-23], in order to check their performance, and optimize the same. Out of these classifiers, readers can find the implementation of Naïve Bayes (NB) classifier using FPGAs in [26, 27, 28], wherein BRAMs are used for effective computations, as suggested by our previous study. Moreover, work in [26] also suggests the use of accelerators for improved performance of the system. In the next section we describe the effect of different ML-implementation architectures on the area, power and delay of the hardware design.

## STATISTICAL ANALYSIS

Observing the kind of architectures used by researchers over the years, we formulated the effect of using them on various parameters of hardware design. The following table 1 indicates the usage and effect of the architectures on area, power, delay and efficiency of the FPGA-based system. These ranges are defined in terms of requirement of a particular parameter. For example, a low power requirement indicates that the energy consumption of the given architecture is low, and thus can be used for real time implementation of FPGA-based ML systems.

***Table 1.*** *Comparison of Parameters for Different Architectures.*

| Architecture used | Area | Power | Delay | Accuracy |
|---|---|---|---|---|
| Loops [2] | L | L | H | M |
| Unrolling [3] | L | L | M | M |
| 2D-Mapping [3] | H | M | M | M |
| Kernel-based implementation [6] | H | H | M | M |
| Smart Memory [7] | L | L | M | H |
| Blocks Independent Accelerator Architectures [18] | M | M | L | M |
| Embedded Accelerator Architectures [18] | M | M | L | H |
| Graph Partitioning [18] | H | H | M | M |
| Overall Acceleration [18] | M | L | L | H |
| Matrix-vector threshold unit [18] | L | M | M | M |
| Weights Reloading [18] | M | M | L | M |
| CNN Accelerator [18] | H | H | M | H |
| Line Buffer Design Dispatchers [18] | M | M | L | M |
| Systolic Arrays [18] | M | M | M | H |
| Reconfigurable Computing [18] | L | M | L | M |
| Network Inference Accelerator [18] | L | L | M | H |
| Data Routers [18] | M | M | M | L |
| Parallel Frameworks [18] | M | H | L | M |

From the given table, we can observe that for a higher level of accuracy, with moderate delay, and low power & area requirements the network interface accelerator should be used. As it is the best combination of accelerators, and uses

BRAM based designs for optimum performance. Moreover, other delay aware architectures like overall-acceleration, and blocks independent acceleration can be explored and further optimized for real-time application.

## CONCLUSION

In order to design ML algorithms on FPGAs, a lot of complicated design issues have to be handled. These issues include, but are not limited to area requirements, power requirements, delay requirements and accuracy requirements. From this research we can observe that accelerator-based models outperform other models in terms of most of the requirements. Moreover, systolic architectures can be used for further optimizations.

We would recommend the combination of network interface accelerator with systolic architecture for an optimum balance between area, power, delay and efficiency. Researchers can implement such a system for real-time ML-based solutions that are highly flexible, have low area requirements, can optimize power, have good speed of operation and finally produce sufficiently accurate results.

## REFERENCES

1. Li, M., Yang, C., Sun, Q., Ma, W. J., Cao, W. L., & Ao, Y. L. (2019). Enabling highly efficient k-means computations on the SW26010 many-core processor of Sunway TaihuLight. *Journal of Computer Science and Technology*, *34*(1), 77-93.
2. Chauhan, A. S., Sahula, V., & Mandal, A. S. (2019). Novel Randomized Placement for FPGA Based Robust ROPUF with Improved Uniqueness. *Journal of Electronic Testing*, *35*(5), 581-601.
3. Gokhale, M., Frigo, J., Mccabe, K., Theiler, J., Wolinski, C., & Lavenier, D. (2003). Experience with a hybrid processor: K-means clustering. *The Journal of Supercomputing*, *26*(2), 131-148.
4. Mittal, S. (2020). A survey of FPGA-based accelerators for convolutional neural networks. *Neural computing and applications*, *32*(4), 1109-1139.
5. Tolba, M. F., Saleh, H., Mohammad, B., Al-Qutayri, M., Elwakil, A. S., & Radwan, A. G. (2020). Enhanced FPGA realization of the fractional-order derivative and application to a variable-order chaotic system. *Nonlinear Dynamics*, 1-12.
6. Faraone, J., Kumm, M., Hardieck, M., Zipf, P., Liu, X., Boland, D., & Leong, P. H. (2019). Addnet: Deep neural networks using fpga-optimized multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *28*(1), 115-128.
7. Saurav, S., Singh, S., Saini, R., & Saini, A. K. (2016). Hardware accelerator for facial expression classification using linear SVM. In *Advances in signal processing and intelligent recognition systems* (pp. 39-50). Springer, Cham.
8. Pietron, M., Wielgosz, M., Zurek, D., Jamro, E., & Wiatr, K. (2013, February). Comparison of GPU and FPGA implementation of SVM algorithm for fast image segmentation. In *International Conference on Architecture of Computing Systems* (pp. 292-302). Springer, Berlin, Heidelberg.
9. Majolo, M., & Balbinot, A. (2019). Proposal of a Hardware SVM Implementation for Fast sEMG Classification. In *XXVI Brazilian Congress on Biomedical Engineering* (pp. 381-386). Springer, Singapore.
10. Carletti, V., Greco, A., Saggese, A., & Vento, M. (2020). An effective real time gender recognition system for smart cameras. *Journal of Ambient Intelligence and Humanized Computing*, 11(6), 2407-2419.

11. Porcello, J. C. (2019, March). Designing and Implementing SVMs for High-Dimensional Knowledge Discovery using FPGAs. In *2019 IEEE Aerospace Conference* (pp. 1-8). IEEE.
12. Zairi, H., Talha, M. K., Meddah, K., & Slimane, S. O. (2019). FPGA-based system for artificial neural network arrhythmia classification. *Neural Computing and Applications*, 1-16.
13. Ortega-Zamorano, F., Jerez, J. M., Gómez, I., & Franco, L. (2016). Deep Neural Network Architecture Implementation on FPGAs Using a Layer Multiplexing Scheme. In *Distributed Computing and Artificial Intelligence, 13th International Conference* (pp. 79-86). Springer, Cham.
14. Sahin, S., Becerikli, Y., & Yazici, S. (2006, October). Neural network implementation in hardware using FPGAs. In *International Conference on Neural Information Processing* (pp. 1105-1112). Springer, Berlin, Heidelberg.
15. Zhu, J., & Sutton, P. (2003, September). FPGA implementations of neural networks–a survey of a decade of progress. In *International Conference on Field Programmable Logic and Applications* (pp. 1062-1066). Springer, Berlin, Heidelberg.
16. Atibi, M., Boussaa, M., Bennis, A., & Atouf, I. (2020). Real-time implementation of artificial neural network in FPGA platform. In *Embedded Systems and Artificial Intelligence* (pp. 3-13). Springer, Singapore.
17. Duarte, J., Harris, P., Hauck, S., Holzman, B., Hsu, S. C., Jindariani, S., ... & Wu, Z. (2019). FPGA-accelerated machine learning inference as a service for particle physics computing. *Computing and Software for Big Science*, *3*(1), 1-15.
18. Shawahna, A., Sait, S. M., & El-Maleh, A. (2018). FPGA-based accelerators of deep learning networks for learning and classification: *A review. IEEE Access*, 7, 7823-7859.
19. Ben Khalifa, K., Blaiech, A. G., & Bedoui, M. H. (2019). A Novel Hardware Systolic Architecture of a Self-Organizing Map Neural Network. *Computational intelligence and neuroscience*, 2019.
20. Li, J., Long, X., Hu, S., Hu, Y., Gu, Q., & Xu, D. (2019). A novel hardware-oriented ultra-high-speed object detection algorithm based on convolutional neural network. *Journal of Real-Time Image Processing*, 1-12.
21. Hassan, R. O., & Mostafa, H. (2020). Implementation of deep neural networks on FPGA-CPU platform using Xilinx SDSOC. *Analog Integrated Circuits and Signal Processing*, 1-10.
22. Shawahna, A., Sait, S. M., & El-Maleh, A. (2018). FPGA-based accelerators of deep learning networks for learning and classification: *A review. IEEE Access*, 7, 7823-7859.
23. Nakahara, H., Sada, Y., Shimoda, M., Sayama, K., Jinguji, A., & Sato, S. (2019, September). FPGA-based training accelerator utilizing sparseness of convolutional neural network. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)* (pp. 180-186). IEEE.
24. Celin, S., & Vasanth, K. (2018). ECG signal classification using various machine learning techniques. *Journal of medical systems*, 42(12), 1-11.
25. Kovačević, Ž., Pokvić, L. G., Spahić, L., & Badnjević, A. (2020). Prediction of medical device performance using machine learning techniques: infant incubator case study. *Health and Technology*, 10(1), 151-155.
26. Xue, Z., Wei, J., & Guo, W. (2020). A Real-Time Naive Bayes Classifier

Accelerator on FPGA. *IEEE Access*, 8, 40755-40766.

27. Meng, H., Appiah, K., Hunter, A., & Dickinson, P. (2011, June). Fpga implementation of naive bayes classifier for visual object recognition. In *CVPR 2011 WORKSHOPS* (pp. 123-128). IEEE.
28. Xue, Z., Wei, J., & Guo, W. (2020). A Real-Time Naive Bayes Classifier Accelerator on FPGA. *IEEE Access*, 8, 40755-40766.