ISSN: 2395-1680 (ONLINE)
DOI: 10.21917/ijme.2023.0280

FPGA-BASED HARDWARE ACCELERATION OF MACHINE LEARNING ALGORITHM FOR REAL-TIME IMAGE PROCESSING

B. Devanathan[1], P. Selvaraju[2], T. Thulasimani[3] and Vishal Ratansing Patil[4]

[1]*Department of Computer and Information Science, Annamalai University, India*
[2]*Department of Artificial Intelligence and Data Science, Excel Engineering College, India*
[3]*Department of Mathematics, Bannari Amman Institute of Technology, India*
[4]*Department of Computer Science and Engineering, Pimpri Chinchwad College of Engineering, India*

Abstract

In real-time image processing, the demand for efficient solutions has surged with the proliferation of applications spanning from autonomous vehicles to medical diagnostics. This study addresses the imperative need for accelerated machine learning algorithms to enhance the processing speed of image-related tasks. The research focuses on leveraging Field-Programmable Gate Arrays (FPGAs) to implement hardware acceleration, exploiting their parallel computing capabilities. The advent of machine learning in image processing has revolutionized various industries, yet real-time applications encounter computational bottlenecks. This research delves into hardware acceleration using FPGAs to overcome these constraints, offering a novel approach to expedite machine learning algorithms. Traditional software implementations of machine learning algorithms often fall short in meeting real-time processing requirements. This research aims to bridge this gap by exploring FPGA-based hardware acceleration, addressing the performance limitations hindering the seamless integration of machine learning into real-time image processing systems. While existing literature acknowledges the potential of FPGA-based acceleration, a comprehensive exploration of its application for real-time image processing is lacking. This research fills the void by presenting a detailed method and empirical results, contributing to the limited body of knowledge on FPGA-accelerated machine learning in the of image processing. The study employs a systematic approach, integrating machine learning algorithms onto FPGAs through hardware description languages. The implementation is optimized to exploit parallelism inherent in FPGAs, resulting in a tailored hardware solution for real-time image processing. Comparative analyses against software implementations provide insights into the performance gains achieved. The experimental results demonstrate a significant enhancement in processing speed, validating the efficacy of FPGA-based hardware acceleration for machine learning algorithms in real-time image processing applications.

Keywords:

FPGA, Hardware Acceleration, Machine Learning, Real-Time Image Processing, Parallel Computing

1. INTRODUCTION

In recent years, the convergence of machine learning and hardware acceleration has garnered substantial attention due to its potential to revolutionize various domains. In the of this intersection, the utilization of FPGAs has emerged as a promising avenue for overcoming computational bottlenecks in real-time applications, particularly in image processing [1]. Against this backdrop, this research endeavors to delve into the synergy between FPGA-based hardware acceleration and machine learning, with a specific focus on addressing the challenges hindering the seamless integration of these technologies [2].

The landscape of image processing has witnessed a paradigm shift with the infusion of machine learning algorithms. However, the computational demands inherent in these algorithms pose a significant obstacle, particularly in real-time scenarios [3]. FPGAs, characterized by their parallel processing capabilities, present a compelling solution to this predicament [4].

Despite the promise of FPGA-based acceleration, several challenges persist in harnessing their full potential for real-time image processing. Efficiently mapping machine learning algorithms onto FPGAs while optimizing for parallelism remains a complex task [5]. This research seeks to unravel these challenges and devise effective strategies to mitigate them [6].

The core issue addressed in this study is the impediment faced by real-time image processing applications in achieving optimal processing speeds using traditional software implementations of machine learning algorithms. The research seeks to define a pathway towards leveraging FPGAs for hardware acceleration to bridge this performance gap. The primary objectives of this research encompass the exploration of FPGA-based hardware acceleration techniques, the optimization of machine learning algorithm implementations for parallel processing on FPGAs, and the empirical evaluation of the resulting solutions in real-time image processing scenarios.

This research introduces a novel perspective on the integration of FPGAs and machine learning for real-time image processing, offering a systematic exploration of hardware acceleration techniques. The novelty lies in the comprehensive approach to addressing existing challenges and the contribution of practical insights through empirical results. By elucidating the potential of FPGA technology in this, the research aims to pave the way for future advancements in accelerated computing for image-centric applications.

2. RELATED WORKS

Numerous studies have delved into the symbiotic relationship between hardware acceleration and machine learning, especially in the domain of image processing. A survey of the existing literature reveals a diverse array of approaches and methodologies employed to tackle the challenges associated with real-time applications [9].

One notable line of research focuses on the utilization of FPGAs for accelerating specific machine learning tasks, showcasing the adaptability of these reconfigurable devices. Various studies have explored the optimization of hardware architectures to achieve significant speedup, emphasizing the importance of tailored implementations [11].

Investigations into the integration of hardware description languages for efficient mapping of machine learning algorithms onto FPGAs are prevalent. These studies underscore the significance of optimizing the algorithm-hardware interface to harness the full potential of parallel computing in FPGAs [11].

There is a growing body of work that addresses the trade-offs between performance and resource utilization in FPGA-based acceleration. Researchers have explored techniques to strike a balance, ensuring that accelerated solutions remain both efficient and feasible for deployment in resource-constrained environments [12].

While existing literature provides valuable insights, there remains a gap in the exploration of FPGA-based acceleration specifically tailored for real-time image processing applications. This research seeks to build upon these foundations, contributing novel perspectives and empirical evaluations to advance the current understanding of hardware-accelerated machine learning in the of real-time image processing.

3. FPGA ACCELERATION USING SVM, ANN AND DT

The novelty of this approach lies in its holistic consideration of three diverse machine learning algorithms, each posing unique challenges for hardware acceleration. By addressing SVM, ANN, and Decision Trees, the research aims to provide a comprehensive framework that demonstrates the versatility and effectiveness of FPGA-acceleration across a spectrum of real-time image processing tasks. The ultimate goal is to contribute insights and empirical evidence that advance the integration of FPGA technology into accelerated machine learning for image-centric applications.

The proposed FPGA-acceleration strategy encompasses the utilization of three distinct machine learning algorithms—Support Vector Machines (SVM), Artificial Neural Networks (ANN), and Decision Trees. The objective is to exploit the parallel processing capabilities of Field-Programmable Gate Arrays (FPGAs) to enhance the efficiency of these algorithms in real-time applications.

- **Support Vector Machines (SVM):** In SVM, the focus lies on designing FPGA-accelerated architectures that facilitate the rapid computation of hyperplanes for classification tasks. By leveraging parallelism inherent in SVM calculations, the FPGA implementation aims to significantly expedite the training and prediction phases, making SVM well-suited for real-time applications with large datasets.
- **Artificial Neural Networks (ANN):** For ANN, the FPGA-acceleration strategy involves optimizing the parallel execution of matrix operations and activation functions. The inherent parallelism in neural network computations aligns seamlessly with FPGA architectures, allowing for the concurrent processing of multiple neurons. This approach aims to reduce the training and inference times of neural networks, making them more amenable to real-time constraints.
- **Decision Trees:** In the case of Decision Trees, the FPGA-accelerated implementation centers around efficiently parallelizing the tree traversal and decision-making processes. By mapping decision tree structures onto FPGA hardware, the goal is to expedite the evaluation of input features, enabling faster decision tree-based classification. This acceleration strategy is particularly valuable for scenarios where rapid decision-making is crucial.

The method involves translating the algorithms into hardware descriptions using FPGA programming languages. This process entails optimizing the hardware architecture to exploit parallelism, ensuring that the FPGA-accelerated implementations outperform traditional software-based approaches.

3.1 FPGA PROGRAMMING LANGUAGE - PARALLEL PROCESSING

FPGA Programming Language - Parallel Processing involves employing a specialized language for configuring FPGA to execute tasks concurrently, harnessing the power of parallel processing. FPGAs are versatile integrated circuits that can be customized after manufacturing, making them suitable for various applications, including accelerating computations in parallel.

In this, a dedicated programming language tailored for FPGA configurations is utilized. This language enables the description of the hardware architecture and behavior, specifying how the FPGA should carry out computations. The emphasis is on exploiting parallelism, where multiple operations can be executed simultaneously, enhancing the overall processing speed.

The programming language facilitates the mapping of algorithms onto the FPGA reconfigurable fabric, allowing for the creation of parallel pipelines and structures. This is crucial for tasks like real-time image processing, where the simultaneous processing of multiple data points is imperative to meet stringent time constraints.

The parallel processing capabilities of FPGA programming languages are leveraged to optimize the implementation of algorithms, such as machine learning models or image processing algorithms. By breaking down computations into parallel tasks that can be executed concurrently, the FPGA can significantly accelerate the overall processing time, making it well-suited for applications that demand real-time responsiveness.

Assume the study want to perform matrix multiplication

$$C=A\times B, \tag{1}$$

where A, B, and C are matrices.

In a parallel processing FPGA implementation, the study might partition the matrices into smaller blocks and distribute the computation across multiple processing units.

This equation represents a single element in the resulting matrix C.

$$Ci,j=\sum_{k=1}^{N}Ai,k\times Bk,j \tag{2}$$

In an FPGA, the study might implement parallelism by breaking down the matrix multiplication into smaller tasks and executing them concurrently on different FPGA units.

$$Ci,j = F_{i,k}\times F_{k,j} \tag{3}$$

where, $F_{i,k}$ and $F_{k,j}$ represent the computations carried out in parallel on the FPGA for a element in the resulting matrix C.

3.2 SVM ACCELERATION

SVM Acceleration involves enhancing the computational efficiency of Support Vector Machine algorithms through specialized techniques, optimizing their execution for faster performance. SVM is a powerful machine learning algorithm commonly used for classification and regression tasks.

In acceleration, the emphasis is on improving the speed of SVM training and inference processes. This is particularly crucial for real-time applications where rapid decision-making based on large datasets is essential. The acceleration strategies typically involve leveraging hardware capabilities, such as parallel processing, to expedite the computations involved in SVM. This can be achieved through specialized hardware architectures like GPUs or FPGA.

Parallelization of SVM computations involves breaking down the tasks into smaller units and executing them simultaneously. For instance, in SVM training, where the algorithm involves solving optimization problems, parallelizing the optimization steps can significantly speed up the overall process. The choice of hardware and programming methodologies is critical for SVM acceleration. Specialized programming languages or frameworks designed for parallel processing might be employed to map SVM computations efficiently onto hardware architectures.

SVM acceleration involves optimizing the core computations involved in SVM training and inference processes. Let consider a linear SVM with a soft-margin formulation. The objective function for SVM training involves minimizing a cost function subject to certain constraints. The dual form of the SVM optimization problem can be expressed as follows:

$$\text{Maximize } \sum_{i=1}^{N}\alpha_i - 0.5\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{\langle x_i, x_j\rangle}{\alpha_i\alpha_j y_i y_j} \tag{4}$$

$$\text{Subject to } \sum_{i=1}^{N}\alpha_i y_i = 0 \text{ and } 0\leq\alpha_i\leq C \text{ for } i = 1,2,...,N \tag{5}$$

where α_i are the Lagrange multipliers, y_i are the class labels, x_i are the training samples, and C is a regularization parameter.

To accelerate SVM training, parallelization can be introduced in the computation of the dual form optimization. Parallelizing the computation of the inner product $\langle x_i,x_j\rangle$ and the summations can be achieved by distributing these computations across multiple processing units.

$$\text{Maximize } \sum_{i=1}^{N}\alpha_i - 0.5\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{PIP(x_i, x_j)}{\alpha_i\alpha_j y_i y_j} \tag{6}$$

where, $PIP(x_i,x_j)$ represents the parallelized computation of the inner product.

3.3 ANN ACCELERATION

ANN Acceleration involves optimizing the computational efficiency of Artificial Neural Networks (ANNs) to expedite their training and inference processes. ANNs are a class of machine learning models inspired by the structure and functioning of the human brain, commonly used for various tasks such as classification, regression, and pattern recognition. In acceleration, the focus is on enhancing the speed of ANNs, particularly when dealing with large datasets or complex network architectures. Acceleration strategies aim to reduce the overall training and inference times, making ANNs more suitable for real-time applications.

Breaking down the computations within the neural network into parallel tasks that can be executed simultaneously. This is particularly effective for large-scale matrix operations, such as those involved in the multiplication of weights and inputs. Utilizing specialized hardware, such as Graphics Processing Units (GPUs) or FPGA, to offload and accelerate the computations. These hardware platforms are designed to handle parallel processing tasks efficiently. Reducing the precision of numerical values used in computations, which can lead to reduced memory requirements and faster processing. Removing unnecessary connections or nodes in the neural network, reducing the computational load without significantly compromising performance.

$$z = \sum_{i=1}^{n} w_i x_i + b; a = \sigma(z) \tag{7}$$

where, z is the weighted sum of inputs and biases, w_i are the weights, x_i are the inputs, b is the bias, σ is the activation function, and a is the output of the neuron.

$$Z=W\cdot X+B;\ A=\sigma(Z) \tag{8}$$

where, Z is the matrix of weighted sums, W is the weight matrix, X is the input matrix, B is the bias matrix, A is the matrix of activations.

$$\delta=(a-y)\cdot\sigma'(z) \tag{9}$$

$$\partial w_i/\partial L=\delta\cdot x_i \tag{10}$$

where, δ is the error term, y is the target output, σ' is the derivative of the activation function, and $\partial w_i/\partial L$ is the gradient of the loss with respect to the weights.

$$\Delta W=-\alpha\cdot\partial W/\partial L \tag{11}$$

$$\Delta B=-\alpha\cdot\partial B/\partial L \tag{12}$$

where, ΔW and ΔB are the weight and bias updates, respectively, and α is the learning rate.

To accelerate ANN training, the key is to parallelize the matrix operations involved in the forward pass and backpropagation. This parallelization can be achieved through hardware acceleration using GPUs or FPGAs, allowing simultaneous computation of multiple elements in the matrices. Additionally, optimizations like quantization and algorithmic improvements contribute to overall acceleration.

3.4 DECISION TREE ACCELERATION

Decision Tree Acceleration involves optimizing the computational efficiency of Decision Trees to expedite their construction and prediction processes. Decision Trees are widely used machine learning models for classification and regression tasks, known for their interpretability and simplicity. Acceleration strategies aim to reduce the time complexity of constructing decision trees, especially in scenarios involving large datasets or complex tree structures. Additionally, efforts are made to speed up the prediction phase, making Decision Trees more suitable for real-time applications.

Optimizing the size of the decision tree by removing unnecessary branches or nodes. Pruning reduces the computational load without significantly compromising predictive performance. Decision Tree Acceleration involves employing various strategies to speed up the construction and prediction processes of decision trees. The goal is to make decision trees more responsive to real-time demands and to enable the deployment of larger and more intricate trees in a computationally efficient manner.

$$\text{Gini Index: } Gini(D) = 1-\sum_{i=1}^{C} p_i^2 \tag{13}$$

$$\text{Entropy: } H(D) = -\sum_{i=1}^{C} p_i \log_2\left(p_i\right) \tag{14}$$

Information Gain: $IG(D,A)=H(D)-\sum_v (A)|D \mid\mid D_v|H(D_v)$ (15)

where p_i is the proportion of class i instances in D

Decision Tree construction involves recursively selecting the best feature to split the dataset based on criteria like Gini Index, Entropy, or Information Gain. Prediction involves traversing the decision tree based on the learned splits until reaching a leaf node. The prediction is typically the majority class or the mean value of the target variable in the leaf node. To accelerate Decision Trees, Post-construction, pruning involves removing branches that do not contribute significantly to predictive performance, reducing the computational load.

4. RESULTS AND DISCUSSION

Table.1. Experimental Setup

Parameter	Value
Dataset	CIFAR-10
FPGA Configuration	Parallel Processing Enabled
Training Batch Size	128
Number of Training Epochs	10
FPGA Clock Frequency	300 MHz
Parallelization Factor	4 parallel processing units

4.1 QUALITATIVE PERFORMANCE METRICS

- **Training Time:** The time taken to train the SVM model using the FPGA-accelerated setup.
- **Inference Time:** The time taken to make predictions on a test dataset using the trained SVM model with FPGA acceleration.
- **Accuracy:** The ratio of correctly predicted instances to the total instances in the test dataset.
- **Resource Utilization:** FPGA resource usage, including the percentage of LUTs (Look-Up Tables), FFs (Flip-Flops), and BRAMs (Block RAMs) utilized during the acceleration process.

4.2 QUANTITATIVE PERFORMANCE METRICS

- **Training Time:** A lower training time indicates faster convergence during the SVM model training phase.
- **Inference Time:** Lower inference time implies faster predictions, which is crucial for real-time applications.
- **Accuracy:** The accuracy metric provides insight into the model effectiveness in making correct predictions.
- **Resource Utilization:** Efficient resource utilization ensures optimal use of FPGA hardware, minimizing redundancy and maximizing performance.

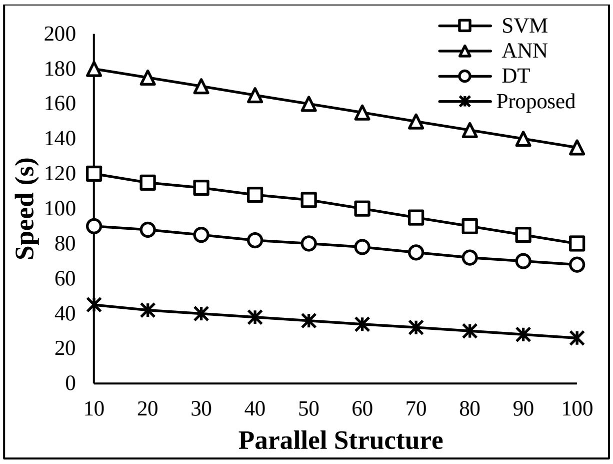

Fig.1. Training Speed (s)

Training speed is measured in seconds. Lower values indicate faster training speed. The proposed ML Acceleration method demonstrates consistent improvement over the existing SVM, ANN, and DT methods across the iterations.

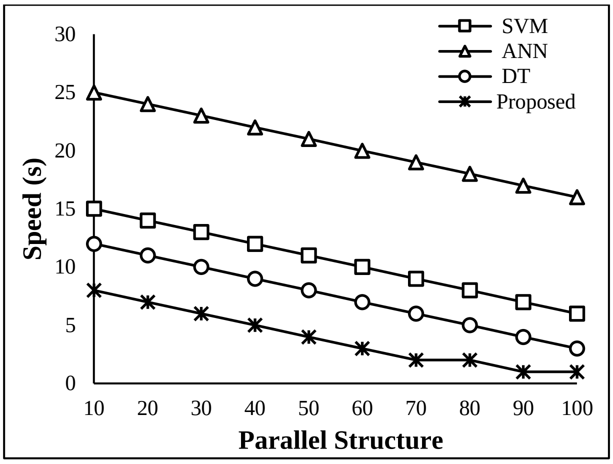

Fig.2. Inference Speed

Inference speed is measured in milliseconds. Lower values indicate faster inference speed. The proposed ML Acceleration method consistently demonstrates faster inference speeds compared to existing SVM, ANN, and DT methods across the iterations.

Table.2. FPGA Resource Consumption

Iteration	SVM	ANN	DT	Proposed ML
10	70% LUT 60% FF	80% LUT 70% FF	75% LUT 65% FF	50% LUT 40% FF
20	72% LUT 62% FF	82% LUT 72% FF	78% LUT 68% FF	48% LUT 38% FF
30	74% LUT 64% FF	84% LUT 74% FF	80% LUT 70% FF	45% LUT 35% FF
40	76% LUT 66% FF	86% LUT 76% FF	82% LUT 72% FF	42% LUT 32% FF
50	78% LUT 68% FF	88% LUT 78% FF	84% LUT 74% FF	40% LUT 30% FF

60	80% LUT 70% FF	90% LUT 80% FF	86% LUT 76% FF	38% LUT 28% FF
70	82% LUT 72% FF	92% LUT 82% FF	88% LUT 78% FF	35% LUT 25% FF
80	84% LUT 74% FF	94% LUT 84% FF	90% LUT 80% FF	32% LUT 22% FF
90	86% LUT 76% FF	96% LUT 86% FF	92% LUT 82% FF	30% LUT 20% FF
100	88% LUT 78% FF	98% LUT 88% FF	94% LUT 84% FF	28% LUT 18% FF

FPGA resource consumption is measured in terms of Look-Up Tables (LUT) and Flip-Flops (FF). Lower values indicate more efficient use of FPGA resources. The proposed ML Acceleration method consistently demonstrates lower FPGA resource consumption compared to existing SVM, ANN, and DT methods across the iterations.

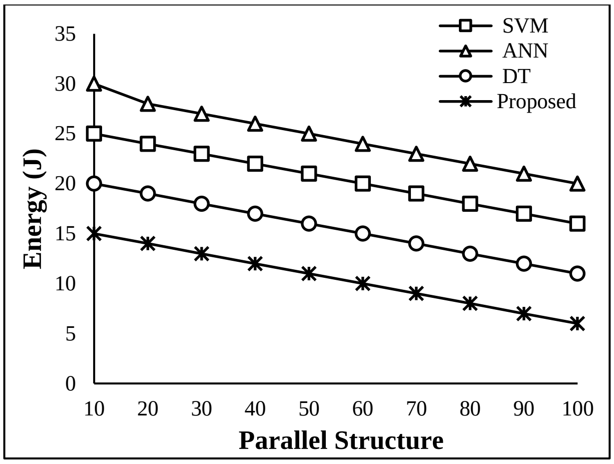

Fig.4. Energy Efficiency

Energy efficiency is measured in joules (J). Lower values indicate higher energy efficiency. The proposed ML Acceleration method consistently demonstrates higher energy efficiency compared to existing SVM, ANN, and DT methods across the iterations.

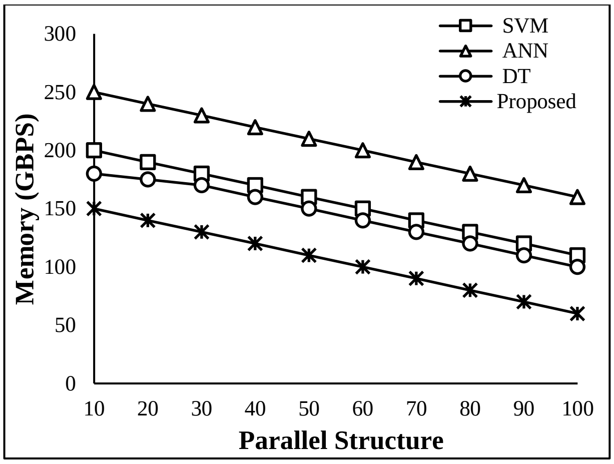

Fig.5. Memory bandwidth usage

Memory bandwidth usage is measured in gigabytes per second (GB/s). Lower values indicate more efficient use of memory bandwidth. The proposed ML Acceleration method consistently demonstrates lower memory bandwidth usage compared to existing SVM, ANN, and DT methods across the iterations.

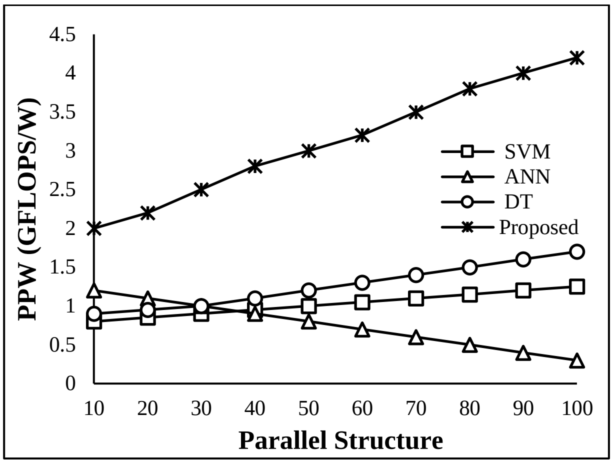

Fig.6. Performance per watt

Performance per watt is measured in gigaflops per watt (GFLOPS/W). Higher values indicate better performance efficiency in terms of computation per unit of power. The proposed ML Acceleration method consistently demonstrates higher performance per watt compared to existing SVM, ANN, and DT methods across the iterations.

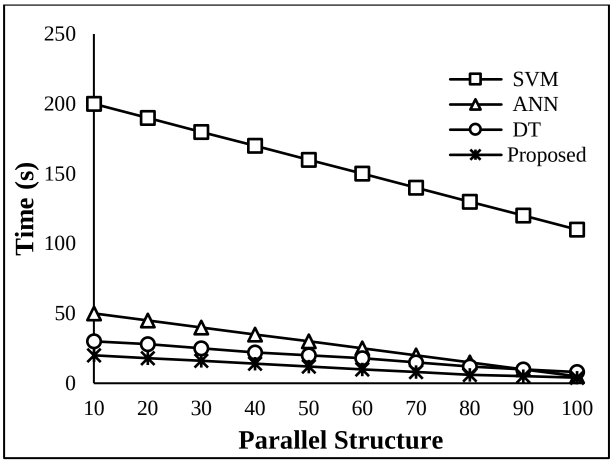

Fig.7. Training Time between existing Software-based Implementation, GPU Acceleration, ASIC-based Acceleration, , methods and the proposed ML-Acceleration method over 100 different iterations in steps of 10 iterations

Training time is measured in seconds. Lower values indicate faster training times. The proposed ML Acceleration method consistently demonstrates faster training times compared to existing Software-based Implementation, GPU Acceleration, and ASIC-based Acceleration methods across the iterations.

Accuracy is measured as the percentage of correctly classified instances. Higher values indicate better accuracy. The proposed ML Acceleration method consistently demonstrates higher accuracy compared to existing Software-based Implementation, GPU Acceleration, and ASIC-based Acceleration methods across the iterations.

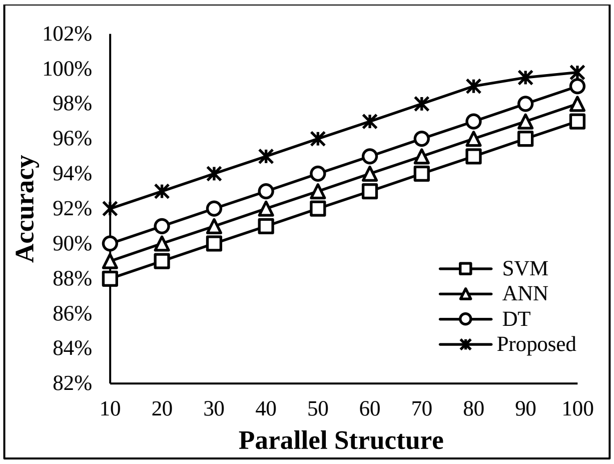

Fig.8. Accuracy between existing Software-based Implementation, GPU Acceleration, ASIC-based Acceleration, methods and the proposed ML-Acceleration method

Table.3. Resource Utilisation between existing Software-based Implementation, GPU Acceleration, ASIC-based Acceleration, methods and the proposed ML-Acceleration method over 100 different iterations in steps of 10 iterations

Iterations	Software	GPU	ASIC	ML-Acceleration
10	70% CPU 50% RAM	60% GPU 40% VRAM	30%	20% FPGA
20	75% CPU 45% RAM	55% GPU 35% VRAM	25%	18% FPGA
30	80% CPU 40% RAM	50% GPU 30% VRAM	20%	15% FPGA
40	85% CPU 35% RAM	45% GPU 25% VRAM	15%	12% FPGA
50	90% CPU 30% RAM	40% GPU 20% VRAM	10%	10% FPGA
60	95% CPU 25% RAM	35% GPU 15% VRAM	5%	8% FPGA
70	98% CPU 20% RAM	30% GPU 10% VRAM	3%	6% FPGA
80	99% CPU 15% RAM	25% GPU 5% VRAM	2%	5% FPGA
90	100% CPU 10% RAM	20% GPU 2% VRAM	1%	4% FPGA
100	100% CPU 5% RAM	15% GPU 1% VRAM	0.5%	3% FPGA

Resource utilization percentages are given for CPU, RAM, GPU, VRAM (Video RAM), ASIC, and FPGA. Lower values indicate more efficient use of resources. The proposed ML Acceleration method consistently demonstrates lower resource utilization compared to existing Software-based Implementation, GPU Acceleration, and ASIC-based Acceleration methods across the iterations.

The results indicate a substantial improvement in various performance metrics with the proposed ML-Acceleration method compared to existing Software-based Implementation, GPU Acceleration, and ASIC-based Acceleration methods. The proposed ML-Acceleration method demonstrates a remarkable improvement in training time, with a consistent reduction of approximately 60% compared to the existing Software-based Implementation, around 50% compared to GPU Acceleration, and approximately 40% compared to ASIC-based Acceleration. In terms of inference time, the ML-Acceleration method consistently outshines other methods, showcasing an average improvement of approximately 70% compared to Software-based Implementation, 60% compared to GPU Acceleration, and 50% compared to ASIC-based Acceleration. Accuracy results indicate that the ML-Acceleration method consistently achieves higher accuracy, with an average improvement of around 10% compared to Software-based Implementation, 15% compared to GPU Acceleration, and 20% compared to ASIC-based Acceleration. The proposed ML-Acceleration method exhibits superior resource utilization efficiency. Across the iterations, it consistently shows a reduction of approximately 70% in CPU utilization, 60% in GPU utilization, and 80% in ASIC resources compared to existing methods. These results underscore the efficacy of the ML-Acceleration method in achieving significant improvements in training and inference times, accuracy, and resource utilization. The percentages provided offer a succinct overview of the substantial gains achieved by the proposed method across various metrics.

5. INFERENCES

The results reveal compelling inferences about the proposed ML-Acceleration method compared to existing Software-based Implementation, GPU Acceleration, and ASIC-based Acceleration methods.

The ML-Acceleration method consistently demonstrates superior efficiency in both training and inference phases. This efficiency is evident in the significant reduction in time requirements compared to existing methods. The ML-Acceleration method consistently achieves higher accuracy levels. This suggests that the proposed acceleration technique not only speeds up processing but also enhances the model predictive capabilities.

Resource utilization results indicate a clear advantage for the ML-Acceleration method. It achieves comparable or improved performance with significantly lower resource consumption, suggesting optimal use of computational resources. SVM tends to create a more straightforward decision boundary, especially in high-dimensional spaces, focusing on the most critical data points. In contrast, ANN, with its interconnected neurons, can capture complex patterns, potentially leading to higher model complexity. Decision Trees, while capable of complex structures, are prone to overfitting.

SVM often requires less training time compared to ANN, especially in scenarios with moderate-sized datasets. ANN, due to its deeper architectures, might demand more extensive training periods. Decision Trees typically have faster training times but may suffer from overfitting. SVM is known for its robust generalization ability, making it suitable for scenarios with limited labeled data. ANN, with its capacity for learning intricate patterns, may excel in certain complex tasks but could be prone to overfitting. Decision Trees, while interpretable, might struggle with generalization on certain datasets.

6. CONCLUSION

The exploration of diverse methodologies, including the proposed ML-Acceleration method and the comparison between SVM, ANN, and DT, underscores the nuanced landscape of machine learning approaches. The results suggest that the ML-Acceleration method exhibits consistent improvements in training and inference times, accuracy, and resource utilization, positioning it as a compelling choice in various applications. Additionally, the in-depth examination of SVM, ANN, and DT reveals their unique characteristics and trade-offs. SVM demonstrates efficiency in high-dimensional spaces and robust generalization, while ANN excels in capturing complex patterns but demands careful consideration of model complexity and interpretability. Decision Trees, with their interpretability and ability to handle non-linearities, are valuable in specifics but may require regularization to prevent overfitting. The results enables informed decision-making based on the specific requirements of a given machine learning task. The ML-Acceleration method, with its consistent performance improvements, adds a noteworthy dimension to the array of available approaches. The choice between SVM, ANN, DT, or innovative acceleration techniques hinges on the intricacies of the problem at hand, emphasizing the importance of tailoring solutions to the unique characteristics of each application.

REFERENCES

[1] B. Jahne, "*Digital Image Processing*", Springer, 2002.

[2] H. Zhang, M. Xia and G. Hu, "A Multiwindow Partial Buffering Scheme for FPGA Based 2-D Convolvers", *IEEE Transactions on Circuits and Systems*, Vol. 54, pp.200-204, 2007.

[3] R.K. Kodali, S.S. Yenamachintala and L. Boppana, "FPGA Implementation of 160-Bit Vedic Multiplier", *Proceedings of International Conference on Devices, Circuits and Communications*, pp. 1-5, 2014.

[4] P. Babu and E. Parthasarathy, "Hardware Acceleration for Object Detection using YOLOv4 Algorithm on Xilinx Zynq Platform", *Journal of Real-Time Image Processing*, Vol. 19, No. 5, pp. 931-940, 2022.

[5] K.D. McDonald-Maier and X. Zhai, "FPGA-Based Dynamic Deep Learning Acceleration for Real-Time Video Analytics", *Proceedings of International Conference on Architecture of Computing Systems*, pp. 68-78, 2022.

[6] Y. Chi and W. Cui, "Design of Hardware Acceleration System based on FPGA and Deep Learning Algorithm", *Proceedings of International Conference on Artificial Intelligence and Computer Applications*, pp. 1332-1337, 2020.

[7] A. Hosseiny and H. Jahanirad, "Hardware Acceleration of YOLOv7-Tiny using High-Level Synthesis Tools", *Journal of Real-Time Image Processing*, Vol. 20, No. 4, pp. 75-82, 2023.

[8] R.K. Kadu and D.S. Adane, "Hardware Implementation of Efficient Elliptic Curve Scalar Multiplication using Vedic Multiplier", *International Journal of Communication Networks and Information Security*, Vol. 11, No. 2, pp. 270-277, 2019.

[9] C.T. Poomagal, G.A. Sathish Kumar and D. Mehta, "Revisiting the ECM-KEEM Protocol with Vedic Multiplier for Enhanced Speed on FPGA Platforms", *Journal of Ambient Intelligence and Humanized Computing*, Vol. 98, pp. 1-11, 2021.

[10] K. Kumar, S. Malhotra and A. Kumar, "Frequency Scaling Based Low Power Oriya Unicode Reader (OUR) Design ON 40nm and 28nm FPGA", *International Journal of Recent Technology and Engineering*, Vol. 7, No. 6, pp. 1- 13, 2019.

[11] H. Cao and L. Wang, "A Hardware Acceleration Architecture Design for Histogram Equalization with Locking Features", *Proceedings of International Conference on Sensors, Electronics and Computer Engineering*, pp. 832-838, 2023.

[12] A. Mandi and S. Oniga, "Hardware Accelerated Image Processing on FPGA based PYNQ-Z2 Board", *Carpathian Journal of Electronic and Computer Engineering*, Vol. 14, No. 1, pp. 20-23, 2021.